Mathematical Biosciences and Engineering

http://www.aimspress.com/journal/MBE

MBE, 16(5): 3914–3935.
DOI: 10.3934/mbe.2019193
Received: 15 February 2019
Accepted: 07 April 2019
Published: 05 May 2019

Research article

Public key encryption with temporary and fuzzy keyword search

Nyamsuren Vaanchig*and Zhiguang Qin

School of Information and Software Engineering, University of Electronic Science and Technology of China, No. 4, North Jianshe Road, Chenghua District, Chengdu, Sichuan 610054, China

* **Correspondence:** Email: nyamsuren.v@gmail.com.

Abstract: Public Key Encryption with Keyword Search (PEKS) is a desirable technique to provide searchable functionality over encrypted data in public key settings, which allows a user to delegate a third party server to perform the search operation on encrypted data by means of keyword search trapdoor without learning about the data. However, the existing PEKS schemes cannot be directly applied to practice due to keyword guessing attack or the absence of a mechanism to limit the lifetime of a trapdoor. By addressing these issues at the same time, this paper presents a Public Key Encryption Scheme with Temporary and Fuzzy Keyword Search (PETFKS) by using a fuzzy function and an encryption tree. The proposed PETFKS scheme is proven adaptively secure concerning keyword confidentiality and backward and forward secrecy in the random oracle model under the Bilinear Diffie-Hellman assumption. Moreover, it is also proven selectively secure with regard to the resistance of keyword guessing attack. Furthermore, the security and efficiency analyses of the proposed scheme are provided by comparing to the related works. The analyses indicate that the proposed scheme makes a threefold contribution to the practical application of public key encryption with keyword search, namely offering secure search operation, limiting the lifetime of a trapdoor and enabling secure time-dependent data retrieval.

Keywords: public key encryption with keyword search; searchable encryption; temporary and fuzzy keyword search

1. Introduction

Public Key Encryption with Keyword Search (PEKS) is a desirable tool to provide searchable functionality over encrypted data for public key cryptosystems, and the concept of PEKS was first put forward by Boneh et al. [1]. PEKS allows a user to delegate a third party server to perform the search operation on encrypted data without learning about the data. Consider, for example, secure data outsourcing system, where data outsourced to third party storage server should be protected from any unauthorized parties including the storage server since it may contain sensitive information [2, 3, 4, 5]. Suppose, Alice wants to send/share data to/with Bob through the secure data outsourcing system. To do so, Alice should encrypt the data as well as the keyword extracted from the data and then store them on the storage server. Later, to selectively retrieve the data from the storage server, Bob firstly generates a trapdoor for a certain keyword which is included in one of the searchable ciphertexts stored along with the data ciphertext Alice outsourced. Then he sends the trapdoor to the server to delegate it to perform a search operation on behalf of him.

Although PEKS [1] was the first realization of searchable encryption in public key settings, it has been proven to be insecure under Keyword Guessing Attack (KGA) [6, 7, 8] so that the keyword will be compromised by a malicious server. Subsequently, Xu et al. [9] proposed Public Key Encryption with Fuzzy Keyword Search (PEFKS) by enhancing the PEKS with fuzzy keyword search operation in order to tackle the insecurity under KGA. However, this sacrifices the efficiency for the security so that it doubles the overhead in a PEKS scheme in terms of communication cost of the search result from server to user as well as computational cost of search operation on user side. Later on, Abdalla et al. [10] introduced Public Key Encryption with Temporary Keyword Search (PETKS) scheme as a generalization of PEKS by limiting the time period in which the trapdoor can be used. The temporary property of the PETKS limits the period in which the trapdoor could be used such that the server is only allowed to test whether ciphertexts generated in the period, for which the trapdoor is issued, contain the keyword. That is to say that the server only can perform the search operation on a subset of ciphertexts generated in the time period of the given trapdoor, not on all ciphertexts. Although the PETKS offers more efficient performance on search operation by limiting the size of ciphertext set on which search operation to be performed, it also remains vulnerable to KGA as the PEKS does. Therefore, it is an open problem to propose a public key encryption scheme with keyword search that offers at least the same security level with the PEFKS while limiting the lifetime of keyword trapdoor to be used. Moreover, in some contexts, it might be a desire for users who want to securely retrieve data created only in a limited period of time besides containing a particular keyword. To the best of our knowledge, there is no solution which deals with the open problem and fulfills the desire at the same time.

1.1. Our contributions

By addressing the above-mentioned problem and desire simultaneously, we propose a Public Key Encryption Scheme with Temporary and Fuzzy Keyword Search(PETFKS). To accomplish the goal of this study, we encounter the following major challenges: (a) how to prevent the KGA while offering more efficient performance on search operation, (b) how to achieve a mechanism that limits the lifetime of a trapdoor, (c) how to enable time-dependent data retrieval. In the proposed scheme, we thus overcome these challenges by using a fuzzy function –which maps two or more keywords to the same fuzzy value to prevent KGA by third party server from learning the exact keyword contained in the data– and an encryption tree –which is represented by a keyword and time structure to limit the lifetime of trapdoor. Intuitively, the proposed PETFKS scheme enriches the PEFKS scheme with temporary property. Main contributions of this work are summarized as follows.

- First, we formalize an unambiguous concept of Public Key Encryption Scheme with Temporary and Fuzzy Keyword Search. In PETFKS, the third party server filters out the most non-matched exact keyword searchable ciphertexts by performing a search operation on temporary and fuzzy keyword searchable ciphertexts of the searchable ciphertexts by means of temporary and fuzzy keyword trapdoor that contains a fuzzy keyword and a time interval, and it returns a small size of a matched exact keyword searchable ciphertext set to the data receiver. After receiving the set of the matched exact keyword searchable ciphertexts, the data receiver can locate a subset of the matched exact keyword searchable ciphertexts by locally performing an exact search operation on the matched set sent by the server by means of exact keyword trapdoor that contains an exact keyword.

- Next, we define a rigorous notion of PETFKS-IND-CKA(Indistinguishability under Chosen Keyword Attack) and PETFKS-IND-sCKA-KGA(Indistinguishability under selectively Chosen Keyword Attack and Keyword Guessing Attack) securities for the proposed scheme by concerning the security requirements of keyword confidentiality, forward and backward secrecy, and keyword guessing resistance.
- Then, we provide a concrete construction of PETFKS using the bilinear pairing, an encryption tree, and a fuzzy function.
- Furthermore, we prove that the proposed scheme is adaptively secure in the random oracle model under the BDH assumption concerning the security requirements of keyword confidentiality and backward and forward secrecy as well as it is selectively secure in the random oracle model under the BDH assumption concerning the resistance of keyword guessing attack.
- Moreover, we conduct the experimental simulation of the proposed scheme and the related works.
- Finally, we provide security and efficiency analyses by comparing the proposed scheme to the related works. The analyses indicate that the proposed scheme makes a threefold contribution to the practical application of public key encryption with keyword search, namely offering secure search operation, limiting the lifetime of trapdoor and enabling secure time-dependent data retrieval.

1.2. Related work

There has been many works [11, 12, 13, 14] concerning the fact that data must be stored and transmitted in encrypted form such that if the storage server is compromised or the communication channel is intercepted the data disclosure will be limited. In recent years, PEKS draws the widespread attention of researchers. Since the first introduction of PEKS by Boneh et al. [1], many works have been proposed to improve the functionality of PEKS. In order to overcome the limitation of PEKS that performs a search operation for a certain keyword, some public key encryption with conjunctive keyword search schemes have been proposed in [15, 16, 17], by enabling search operation for boolean combinations of several keywords. By limiting the time period in which the trapdoor can be used, public key encryption with temporary keyword search scheme is proposed as a generalization of PEKS. Shi et al. [18] proposed public key encryption that focuses on multi-dimensional comparisons and range queries. The concept of public key encryption with delegated keyword search [19] is proposed to enable users to delegate the server to check whether encrypted data is infected by malware by giving the server a delegated master trapdoor that does not reveal users' private key. Yu et al. [20] proposed efficient public key encryption with revocable keyword search scheme to restrict the search power of the server by means of trapdoor that performs a search operation for the keyword in a certain time. Public key encryption with authorized keyword search scheme that allows the authority to authorize users to search different keyword sets is proposed by Peng et al. [21].

By enhancing the security of PEKS, some works have been proposed. Baek et al. [22] extended the PEKS scheme by removing a secure channel between users and a server. With concerning the offline keyword guessing attack in PEKS, Tang et al. [23] and Yang et al. [24] are proposed public key encryption with registered keyword search and public key encryption with keyword scheme not using pairing, respectively. Public key encryption with fuzzy keyword search scheme [9] is also proposed to address the keyword guessing attack in PEKS by means of a novel fuzzy function. In this paper, we proposed a Public Key Encryption Scheme with Temporary and Fuzzy Keyword Search. Our proposed offers secure search operation on server side and more efficient search operation on user side at the same time. Furthermore, it limits the lifetime of trapdoor as well as fulfills the desire for secure time-dependent data retrieval. To the best of our knowledge, our scheme overcomes the drawbacks in public key encryption with keyword search, which are not able to be tackled by the existing works.

1.3. Paper organization

We organize the rest of the paper as follows. In Section 2, we review preliminaries. In Section 3 and Section 4, we define the system model and the algorithm definitions of the proposed scheme, separately. Security requirements and security models are described in Section 5. The proposed Public Key Encryption Scheme with Temporary and Fuzzy Keyword Search (PETFKS) is presented in Section 6. The security and efficiency analyses of the proposed scheme are provided in Section 7. Finally, a conclusion is given in Section 8.

2. Preliminaries

Now we review the bilinear map which plays important role in the construction of the proposed PETFKS scheme and Bilinear Diffie-Hellman assumption on which the security of the proposed PETFKS scheme is based.

2.1. Bilinear map

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two multiplicative groups of prime order p. Let g be a generator of $\mathbb{G}_1$ and e be a bilinear map, $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ with the following properties [25, 26]:

1. *Bilinearity*: for all $u, v \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$.
2. *Non-degeneracy*: $e(g, g) \neq 1$.
3. *Computability*: There is an efficient algorithm to compute $e(u, v)$ for $\forall u, v \in \mathbb{G}_1$.

2.2. Bilinear diffie-Hellman (BDH) assumption

The BDH problem[25] in $\langle \mathbb{G}_1, \mathbb{G}_2, p, g\rangle$ is as follows: Given $g, g^\alpha, g^\beta, g^\gamma \in \mathbb{G}_1$ as input, compute $e(g, g)^{\alpha\beta\gamma} \in \mathbb{G}_2$. We say that BDH is hard if all polynomial time algorithms have a negligible advantage in solving the BDH problem.

3. System model

We consider a scenario illustrated in Figure 1 as an application of public key cryptosystem with temporary and fuzzy keyword search (PETFKS), in which a *data receiver*(DR) retrieves data from a *cloud server*(CS) storing ciphertexts published by *data senders*(DS). More precisely, the system is described as the below.

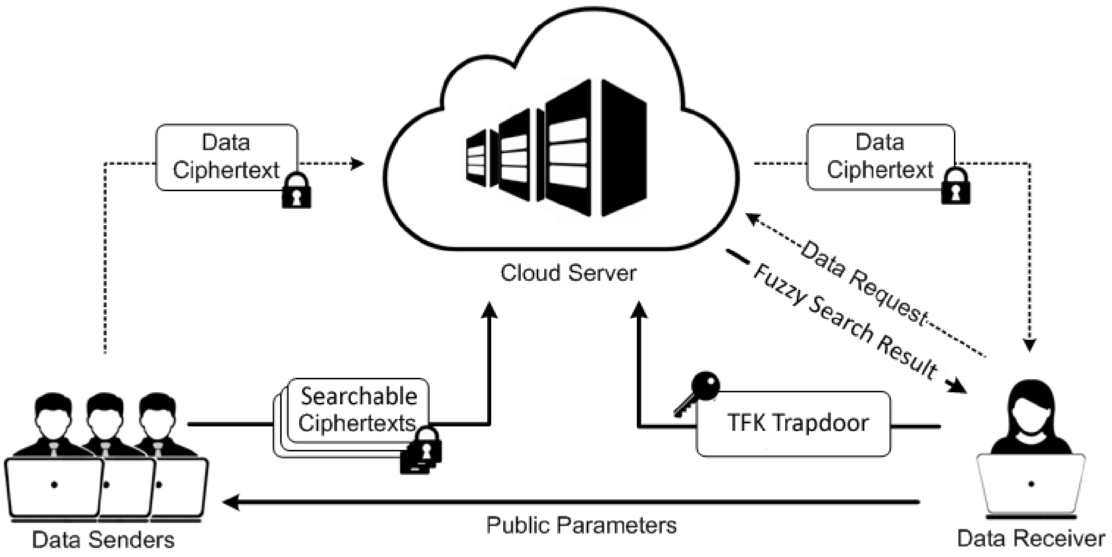


Figure 1. The Architecture of PETFKS System.

The DR sets up the system and publishes the public parameters publicly. Each DS who wants to send/share data to/with the DR generates a data ciphertext by encrypting the data under the DR's identity, as that in IBE does, and also generates a set of searchable ciphertexts from a set of keywords extracted from the data as well as a time slot defined by the DS himself/herself. Each searchable ciphertext consists of a temporary and fuzzy keyword searchable (TFKS) ciphertext and an exact keyword searchable (EKS) ciphertext. That is to say that each TFKS ciphertext contains a fuzzy value of a keyword and is created for the time slot, and each EKS ciphertext contains a keyword. Then the DS sends the data ciphertext along with its searchable ciphertexts to the CS. The CS stores each data ciphertext along with its searchable ciphertexts. To retrieve the data ciphertext that created in a time period and also contains a keyword, the DR first generates a pair of trapdoors, one is a temporary and fuzzy keyword (TFK) trapdoor and another is an exact keyword (EK) trapdoor. Then the DR sends only the TFK trapdoor to the CS. Upon receiving the TFK trapdoor, the CS performs a fuzzy search operation on TFKS ciphertexts of searchable ciphertexts and returns a set of EKS ciphertexts of which corresponding TFKS ciphertexts are matched with the TFK trapdoor as the fuzzy search result to the DR. After receiving the fuzzy search result, the DR performs an exact search operation on the fuzzy search result by using the EK trapdoor in order to selectively find the ciphertext that contains the keyword. With the result of the exact search, the DR is able to request the CS to retrieve the data ciphertext he/she wants. After receiving the requested data ciphertext from the CS, the DR decrypts it with his/her secret key to obtain the data, as that in IBE does.

4. Formal definition of PETFKS

Let $\mathcal{W}$ be a uniformly distributed keyword space and $\mathcal{T}$ be a time space, respectively. The PETFKS system consists of the following algorithms:

Setup$(\lambda, \mathcal{W}, \mathcal{T}) \rightarrow (PP, mk)$. This algorithm takes as input a security parameter λ, a uniformly distributed keyword space $\mathcal{W}$ and a time space $\mathcal{T}$. It then outputs public parameters PP and the master key mk. Here PP contains a fuzzy function $Fuz(w_i, W)$ that takes a keyword $w_i \in \mathcal{W}$ and outputs a fuzzy value.

Trapdoor$(PP, mk, w_i, s, e) \rightarrow (TD^F_{w_i}, TD^E_{w_i})$. This algorithm takes as input the public parameters PP, the master key mk, a keyword $w_i \in \mathcal{W}$, a time period defined by two time slots s, e, where $s, e \in \mathcal{T}$ and $s \leq e$. This algorithm then outputs a pair of trapdoors: a temporary and fuzzy keyword (TFK) trapdoor $TD^F_{w_i}$ and an exact keyword (EK) trapdoor $TD^E_{w_i}$.

PETFKS$(PP, w_i, t) \rightarrow CT$. This algorithm takes as input the public parameters PP, a keyword $w_i \in \mathcal{W}$ and a time slot $t \in \mathcal{T}$, and then it outputs a searchable ciphertext CT that consists of a temporary and fuzzy keyword searchable (TFKS) ciphertext CT_F and an exact keyword searchable (EKS) ciphertext CT_E.

FuzzTest$(PP, CT, TD^F_{w_i}) \rightarrow 1|0$. This algorithm takes as input the public parameters PP, a searchable ciphertext CT and a TFK trapdoor $TD^F_{w_i}$, and it returns 1 meaning "yes" or 0 meaning "no".

ExactTest$(PP, CT_E, TD^E_{w_i}) \rightarrow 1|0$. This algorithm takes as input the public parameters PP, an EKS ciphertext CT_E and an EK trapdoor $TD^E_{w_i}$, and it returns 1 meaning "yes" or 0 meaning "no".

Correctness. A PETFKS system is *correct* if for all (PP, mk) generated by **Setup**$(\lambda, \mathcal{W}, \mathcal{T})$, CT computed by **PETFKS**(PP, w_i, t) and $(TD^F_{w'_i}, TD^E_{w'_i})$ obtained **Trapdoor**(PP, mk, w'_i, s, e), where $\{s, t, e\} \in \mathcal{T}$ and $\{w_i, w'_i\} \in \mathcal{W}$,

1. **FuzzTest**$(CT, TD^F_{w'_i})$ returns 1 as long as $Fuz(w_i, \mathcal{W}) = Fuz(w'_i, \mathcal{W})$ and $t \in [s..e]$.
2. **ExactTest**$(CT_E, TD^E_{w'_i})$ returns 1 as long as $w_i = w'_i$.

5. Security requirements and models

In our system, we consider that the CS is semi-trusted to delegate it to perform fuzzy search operations on encrypted data on behalf of the DS. Although the CS performs the search operation honestly, it may try to learn as much information as possible while performing fuzzy search operations on encrypted data. In addition, any other unauthorized entities are assumed to be dishonest and malicious so that they might attempt to learn about data to which they are not authorized to access. Considering the potential attacks from the adversaries to our system, as discussed above, we define the following security requirements for our system:

- **Keyword Confidentiality**: Any searchable ciphertext must not reveal any information about the corresponding keyword. In other words, given a searchable ciphertext, any adversary including the CS cannot know that the searchable ciphertext is the encryption of which keyword among a set of keywords in keyword space.
- **Forward and Backward Secrecy**: Any TFK trapdoor of a keyword and a time period should not reveal the information about the searchable ciphertexts containing the same keyword but not included in the time period. That is say, given a TFK trapdoor created for a keyword and a time period, the CS should not test positively the searchable ciphertexts containing the same keyword but not included in the time period of the given TFK trapdoor.
- **Keyword Guessing Resistance**: Given a TFK trapdoor of a keyword which is likely to be concealed in any of the searchable ciphertexts, the CS cannot determine which searchable ciphertext

is the encryption of which keyword among the keywords having the same fuzzy value.

Now we define formal notions of security for our system by considering the security requirements defined above. Capturing the keyword confidentiality and backward and forward secrecy requirements defined above, we first define PETFKS-IND-CKA (Indistinguishability under Chosen Keyword Attack) security model for a PETFKS system by using the following game between a challenger and the adversary $\mathcal{A}$. PETFKS-IND-CKA security game:

Setup. The challenger takes a security parameter λ and runs Setup algorithm. Then it gives the public parameters PP to the adversary and keeps the system master key to itself.

Phase 1. The adversary adaptively makes TFK trapdoor queries $(w_1, s_1, e_1), ..., (w_m, s_m, e_m)$ to Trapdoor oracle. The challenger responds by generating the TFK trapdoors $TD^F_{w_i}$ corresponding to the keyword w_i and the time period $[s_i..e_i]$.

Challenge. Once the adversary decides that Phase 1 is over, it sends the challenger two keywords w^*_0, w^*_1 and a time slot t^* on those it wishes to be challenged. The only constraint is that the adversary did not query for the TFK trapdoors for either of the pair of a keyword w^*_0 and a time period containing t^* or the pair of a keyword w^*_1 and a time period containing t^*. The challenger picks a random bit $b \in \{0, 1\}$ and sets the challenge searchable ciphertext $CT^* =$PETFKS(PP, w^*_b, t^*). Then it sends CT^* as the challenge to the adversary.

Phase 2. The $\mathcal{A}$ can continue making TFK trapdoor queries $(w_{m+1}, s_{m+1}, e_{m+1}), ..., (w_n, s_n, e_n)$ to Trapdoor oracle as long as each of the queries does not violate the constraint on the challenge.

Guess. Finally, the adversary outputs $b' \in \{0, 1\}$ and it wins the game if $b = b'$.

We refer to such an adversary $\mathcal{A}$ as a PETFKS-IND-CKA adversary, and we define the $\mathcal{A}$'s advantage in breaking PETFKS system as:

$$Adv_{\mathcal{A}_{\text{PETFKS-IND-CKA}}}(\lambda) = |Pr[b = b'] - \tfrac{1}{2}|.$$

Definition 1. *A PETFKS system is said to be secure against an adaptively chosen keyword attack if any polynomial time PETFKS-IND-CKA adversary has a negligible advantage in the above security game.*

Note that, in the PETFKS-IND-CKA security model, we make use of sufficient condition for key-privacy [27] to achieve keyword confidentiality in our scheme.

Now, we describe PETFKS-IND-sCKA-KGA (Indistinguishability under selectively Chosen Keyword Attack and Keyword Guessing Attack) security model, by capturing the resistance to keyword guessing requirement defined above, for a PETFKS system by the following game between a challenger and the adversary $\mathcal{A}$. PETFKS-IND-sCKA-KGA security game:

Init. The adversary $\mathcal{A}$ declares two keywords, w^*_0 and w^*_1, that have the same fuzzy value, $Fuz(w^*_0, \mathcal{W}) = Fuz(w^*_1, \mathcal{W})$, on those it wishes to be challenged.

Setup. This phase is the same as Setup in the PETFKS-IND-CKA security game.

Phase 1. This phase is the same as Phase 1 in the PETFKS-IND-CKA security game.

Challenge. The adversary $\mathcal{A}$ sends the challenger a time slot t^*. The challenger picks a random bit $b \in \{0, 1\}$ and sets the challenge searchable ciphertext $CT^* =$PETFKS(PP, w^*_b, t^*). Then it sends CT^* as the challenge to the adversary $\mathcal{A}$.

Phase 2. The $\mathcal{A}$ can continue making queries to the Trapdoor oracle for the TFK trapdoors $TD^F_{w_i}$ for any keywords w_i (even the challenge keywords w^*_0, w^*_1) and any time periods $[s..e]$ (even time periods containing the challenge time slot t^*).

Guess. The adversary $\mathcal{A}$ outputs $b' \in \{0,1\}$ and wins the game if $b = b'$.

We refer to such an adversary $\mathcal{A}$ as a PETFKS-IND-sCKA-KGA adversary, and we define $\mathcal{A}$'s advantage in attacking PETFKS system as:

$$Adv_{\mathcal{A}_{\text{PETFKS-IND-sCKA-KGA}}}(\lambda) = |Pr[b = b'] - \tfrac{1}{2}|.$$

Definition 2. *A PETFKS system is said to be secure against a selectively chosen keyword attack and keyword guessing attack if any polynomial PETFKS-IND-sCKA-KGA adversary has a negligible advantage in the above PETFKS-IND-CKA-KGA security game.*

6. Construction

Setup$(\lambda, \mathcal{W}, \mathcal{T})$. This algorithm takes as input a security parameter λ, a uniformly distributed keyword space $\mathcal{W}$ and a time space $\mathcal{T}$. Let $BGen(1^\lambda)$ be the bilinear map generator, $Fuz(w_i, \mathcal{W})$ be a fuzzy function which maps two or more keywords to the same fuzzy value, and $N : \mathbb{N} \to \mathbb{N}$ be a polynomially bounded function. The size of $\mathcal{T}$ is polynomial in the security parameter and it is defined by $N(\lambda) = 2^{l(\lambda)}$ for some $l(\lambda) = O(\log(\lambda))$. For simplicity, let l denote $l(\lambda)$. Our scheme represents time slots with a full binary tree of height $l+1$ as in Katz's scheme [28]. That is to say, our construction makes use of a full binary tree with the root node associated with a keyword and all the lower level nodes represent the time structure. For simplicity, we call this full binary tree an encryption tree. In the encryption tree, the root node is labeled with a keyword $w_i \in \mathcal{W}$ and its left child and right child are labeled with "0" and "1", respectively, and other nodes are recursively labeled as follows: if a node is labeled with x its children are labeled with "$x0$" and "$x1$", corresponding to its left child and right child. In such a way, for each time slot $t \in [0..N(\lambda)-1]$ there is a leaf node labeled with binary representation of t, which is l-bit string as $t_1...t_l$. Hence, from an encryption tree with the root node associated with a keyword $w_i \in \mathcal{W}$ and each leaf node associated with each time slot $t \in [0..N(\lambda)-1]$, keyword-time tuples are constructed for TFK trapdoors as well as for TFKS ciphertexts. A keyword-time tuple is a vector of strings $(w_i, t_1, ..., t_l)$, and $t|_k = (t_1, ..., t_k)$ denotes a vector containing the first k bits of the binary representation of the time slot t, where $1 \le k \le l$. The algorithm runs as follows:

1. runs $BGen$ on the security parameter λ to generate a prime p, two groups $\mathbb{G}_1, \mathbb{G}_2$ of order p, and an admissible bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$.
2. chooses an arbitrary generator $g \in \mathbb{G}_1$.
3. chooses a random $a \overset{R}{\leftarrow} \mathbb{Z}_p$ and sets $P = g^a$.
4. defines a fuzzy function $Fuz(w_i, \mathcal{W})$ which takes a keyword $w_i \in \mathcal{W}$ as input and outputs a fuzzy value as follows by depending on whether the size of the uniformly distributed keyword space $|\mathcal{W}|$ is even or not:
   - if $|\mathcal{W}|$ is even, the output of $Fuz(w_i, \mathcal{W})$ is

$$\begin{cases} w_{i-1}||w_i, & \text{if } i \text{ is even,} \\ w_i||w_{i+1}, & \text{if } i \text{ is odd.} \end{cases}$$

- otherwise, the output of $Fuz(w_i, \mathcal{W})$ is

$$\begin{cases} w_{|\mathcal{W}|-2}||w_{|\mathcal{W}|-1}||w_{|\mathcal{W}|}, & \text{if } i \geq |\mathcal{W}| - 2, \\ w_{i-1}||w_i, & \text{if } i \text{ is even,} \\ w_i||w_{i+1}, & \text{if } i \text{ is odd,} \end{cases}$$

where || denotes concatenation. Note that the description of the fuzzy function follows the work [9].

5. chooses two cryptographic hash functions $H_1 : \{0, 1\}^* \to \mathbb{G}_1, H_2 : \mathbb{G}_2 \to \{0, 1\}^n$ for some $n \in \mathbb{N}$.
6. outputs the public parameters $PP = (p, g, e, \mathbb{G}_1, \mathbb{G}_2, P, \mathcal{W}, \mathcal{T}, H_1, H_2, Fuz)$ and the master key $mk = a$.

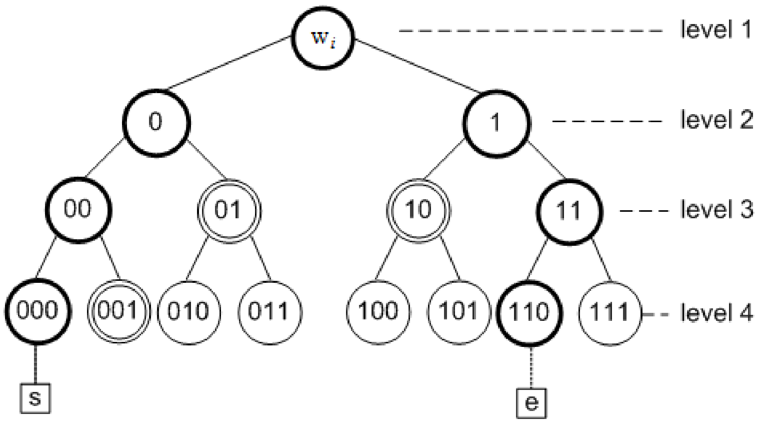


Figure 2. A sample of an encryption tree.

Trapdoor(PP, mk, w_i, s, e). This algorithm takes as input the public parameters PP, the master key mk, a keyword $w_i \in \mathcal{W}$, and a time period defined by two time slots s, e, where $0 \leq s \leq e \leq N(\lambda) - 1$. This algorithm then outputs a pair of trapdoors, a TFK trapdoor $TD^F_{w_i}$, and an EK trapdoor $TD^E_{w_i}$ as follows:

1. computes $S_F = H_1(Fuz(w_i, \mathcal{W}))^a$.
2. builds an encryption tree with a root node associated the keyword w_i and each leaf node associated with each time slot in the time space $\mathcal{T}$.
3. constructs a set of keyword-time tuples T, from the encryption tree, for the keyword w_i, and the given time period represented by s and e as follows:
   - if $s \neq e$, then let k be the smallest index representing the highest level in the encryption tree so that $k \neq 1$ and $s_k \neq e_k$. Thus, the set T is constructed in a way that containing the keyword-time tuples $(w_i, s_1, ..., s_l), (w_i, e_1, ..., e_l)$, and the keyword-time tuples in which time bits are the right siblings of all nodes on the path from $(w_i, s_1, ..., s_{k+1})$ to $(w_i, s_1, ..., s_l)$ and the left siblings of all nodes on the path from $(w_i, e_1, ..., e_{k+1})$ to $(w_i, e_1, ..., e_l)$. Here, $(s_1, ..., s_l)$ are time bits representing s, $(e_1, ..., e_l)$ are time bits representing e, s_i and e_i $(1 \leq i \leq k+1)$ denote the i-th bit of the binary representation of the time slot s and e, respectively. For example, given $\mathcal{T} = 16$, suppose $s = 1$ and $e = 7$, then we build an encryption tree with the root node associated with a keyword w_i as shown in Figure 2, and obtain the binary representations of

s and e as "000" and "110", respectively. Since the $k = 2$ as we can see from an encryption tree in Figure 2, the set T is constructed as $\{(w_i, 0, 0, 0), (w_i, 1, 1, 0), (w_i, 0, 1), (w_i, 0, 0, 1), (w_i, 1, 0)\}$, containing 5 keyword-time tuples.

- if $s = e$, then let k be the smallest index representing the highest level in the encryption tree so that $k = 1$ and $s_k = e_k$. Thus, the set T is constructed in a way that only containing a keyword-time tuple $(w_i, e_1, ..., e_l)$, where e_i is the i-th bit of the binary representation of the time slot e. For instance, suppose $s = e = 7$ when $\mathcal{T} = 16$, then the binary representation of s and e is "110". From an encryption tree in Figure 2, we can see that the $k = 1$ such that the set T is constructed as $\{(w_i, 1, 1, 0)\}$, only containing a keyword-time tuple.

4. computes for each keyword-time tuple $j \in T$ as follows:
   - chooses $s_{j,1}, ..., s_{j,r} \xleftarrow{R} \mathbb{Z}_p$, where r is the size of j and $1 \leq r \leq l$
   - computes $TD_j = (\widehat{D}_j = S_F \prod_{k=1}^{r} H_1(t_{j,|k})^{s_{j,k}}, \{D'_{j,k} = g^{s_{j,k}}\}_{k=1,...,r})$. Here, we denote the k bits of the binary representation of the time slot $t_{j,1}, ..., t_{j,k}$ with $t_{i,|k}$, where k=1,...r.
5. outputs $TD^F_{w_i} = \{TD_j\}_{\forall j \in T}$ and $TD^E_{w_i} = H_1(w_i)^a$.

PETFKS(PP, w_i, t). This algorithm takes as input the public parameters PP, a keyword $w_i \in \mathcal{W}$ and a time slot $t \in \mathcal{T}$, and then it outputs a searchable ciphertext CT. This algorithm runs as follows:

1. constructs a keyword-time tuple $(w_i, t_1, ..., t_l)$ for w_i and t from the encryption tree.
2. chooses $\widetilde{C} \in \{0, 1\}^n$.
3. computes $Q_F = H_1(Fuz(w_i, \mathcal{W}))$ and $Q_E = H_1(w_i)$.
4. chooses $r_1, r_2 \xleftarrow{R} \mathbb{Z}_p$, and computes $\widehat{C}_F = g^{r_1}, \widehat{C}_E = g^{r_2}$ and $\{C'_k = H_1(t_{|k})^{r_1}\}_{k=1,...,l}$
5. computes $C_F = \widetilde{C} \oplus H_2(e(Q_F, P^{r_1}))$ and $C_E = \widetilde{C} \oplus H_2(e(Q_E, P^{r_2}))$.
6. forms the TFKS ciphertext as $CT_F = (C_F, \widehat{C}_F, \{C'_k\}_{k=1,...,l})$ and the EKS ciphertext as $CT_E = (\widetilde{C}, C_E, \widehat{C}_E)$.
7. outputs a searchable ciphertext as $CT = (\widetilde{C}, CT_F, CT_E)$.

FuzzTest$(PP, CT, TD^F_{w_i})$. This algorithm takes as input the public parameters PP, a searchbable ciphertext CT and a TFK trapdoor $TD^F_{w_i}$ and tests if $\widetilde{C} = C_F \oplus H_2(e(\widehat{D}_j, \widehat{C}_F)/\prod_{k=1}^{r} e(D'_{j,k}, C'_k))$ for any $TD_j \in TD^F_{w_i}$. If so for any $TD_j \in TD^F_{w_i}$, it returns 1; If not, it returns 0. The test computes the following :

1. $\forall TD_j \in TD^F_{w_i}$:
$$K = e(\widehat{D}_j, \widehat{C}_F)/\prod_{k=1}^{r} e(D'_{j,k}, C'_k) = e(S_F \prod_{k=1}^{r} H_1(t_{j,|k})^{s_{j,k}}, g^{r_1})/\prod_{k=1}^{r} e(g^{s_{j,k}}, H_1(t_{|k})^{r_1})$$
$$= e(S_F, g^{r_1}) e(\prod_{k=1}^{r} H_1(t_{j,|k})^{s_{j,k}}, g^{r_1})/\prod_{k=1}^{r} e(g^{s_{j,k}}, H_1(t_{|k})^{r_1})$$
$$= e(S_F, g^{r_1}) e(H_1(t_{j,|k}), g)^{r_1 \sum_{k=1}^{r} s_{j,k}}/e(g, H_1(t_{|k}))^{r_1 \sum_{k=1}^{r} s_{j,k}} = e(S_F, g^{r_1}) = e(H_1(Fuz(w_i, \mathcal{W}))^a, g^{r_1})$$
2. $$\widetilde{C} = C_F \oplus H_2(K) = \widetilde{C} \oplus H_2(e(Q_F, P^{r_1})) \oplus H_2(e(H_1(Fuz(w_i, \mathcal{W}))^a, g^{r_1}))$$
$$= \widetilde{C} \oplus H_2(e(H_1(Fuz(w_i, \mathcal{W})), (g^a)^{r_1})) \oplus H_2(e(H_1(Fuz(w_i, \mathcal{W}))^a, g^{r_1}))$$
$$= \widetilde{C} \oplus H_2(e(H_1(Fuz(w_i, \mathcal{W})), g)^{ar_1}) \oplus H_2(e(H_1(Fuz(w_i, \mathcal{W})), g)^{ar_1})$$

ExactTest$(PP, CT_E, TD^E_{w_i})$. This algorithm takes as input the public parameters PP, an EKS ciphertext CT_E and an EK trapdoor $TD^E_{w_i}$, and tests if $\widetilde{C} = C_E \oplus H_2(e(\widehat{C}_E, TD^E_{w_i}))$. If so, it returns 1; If not, it returns 0. The test computes the following: $\widetilde{C} = C_E \oplus H_2(e(\widehat{C}_E, TD^E_{w_i})) = \widetilde{C} \oplus H_2(e(Q_E, P^{r_2})) \oplus H_2(e(g^{r_2}, H_1(w_i)^a))$
$$= \widetilde{C} \oplus H_2(e(H_1(w_i), (g^a)^{r_2})) \oplus H_2(e(g^{r_2}, H_1(w_i)^a)) = \widetilde{C} \oplus H_2(e(H_1(w_i), g)^{ar_2}) \oplus H_2(e(g, H_1(w_i))^{ar_2})$$

7. Analysis

7.1. Security analysis

Before giving security proofs of the proposed PETFKS scheme, we give a comparison between the proposed PETFKS scheme and the related works by considering security requirements defined in Section 5, as shown in Table 1. As can be seen in Table 1, PEFKS [9] and our PETFKS schemes overcome insecurity of PEKS [1] and PETKS [10] schemes.

Table 1. Comparison of security property.

Security Properties	PEKS [1]	PEFKS [9]	PETKS [10]	PETFKS
Keyword Confidentiality	Yes	Yes	Yes	Yes
Backward and Forward Secrecy	NA	NA	Yes	Yes
Keyword Guessing Resistance	No	Yes	No	Yes

We first show the PETFKS-IND-CKA security of our PETFKS scheme by the following theorem.

Theorem 1. *Suppose the hash functions H_1, H_2 are random oracles. Then our PETFKS scheme is PETFKS-IND-CKA-secure in the random oracle model assuming BDH is intractable in groups generated by $BGen(1^\lambda)$.*

Proof. Assume there is an adversary $\mathcal{A}$ that has advantage ϵ in breaking PETFKS-IND-CKA security of the PETFKS scheme. Suppose $\mathcal{A}$ makes at most q_{H_2} hash function queries to H_2 and at most q_T TFK trapdoor queries to Trapdoor oracle. We build a simulator $\mathcal{B}$ that is able to solve the BDH problem with advantage at least $2\epsilon/e(1+q_T)q_{H_2}$, where e is the base of the natural logarithm. Before the game starts, the challenger sets the BDH parameters $\langle p, \mathbb{G}_1, \mathbb{G}_2, e\rangle$ generating by $BGen(1^\lambda)$ and a random instance $\langle g, g^\alpha, g^\beta, g^\gamma\rangle$ of the BDH problem for a random $g \in \mathbb{G}_1$ and $\alpha, \beta, \gamma \in \mathbb{Z}_p$, where p is order of the $\mathbb{G}_1, \mathbb{G}_2$ and gives them to the simulator $\mathcal{B}$. Let $D = e(g,g)^{\alpha\beta\gamma} \in \mathbb{G}_2$ be the solution to this BDH problem. The simulator's goal is to output D. The $\mathcal{B}$ acts as the adversary $\mathcal{A}$'s challenger in the PETFKS-IND-CKA game and interacts with the $\mathcal{A}$ as follows:

Setup. The $\mathcal{B}$ sets $P = g^\beta$ and gives the public parameters $PP = (p, g, e, \mathbb{G}_1, \mathbb{G}_2, P, \mathcal{W}, \mathcal{T}, H_1, H_2, Fuz)$ to the $\mathcal{A}$.

The simulator $\mathcal{B}$ will program the random oracles H_1 and H_2 as follows.

- H_1-queries. The adversary $\mathcal{A}$ can query the random oracle H_1 at any time. In order to respond the queries, the $\mathcal{B}$ maintains a list called L_1. Each entry in the list is a tuple of form ⟨*Keyword-time-tuple, Point-tuple, Scalar-tuple, Secret-tuple, Coin-tuple*⟩. For the j-th query, the tuple is formed as $\langle\{v_0^{(j)}, ..., v_r^{(j)}\}, \{P_0^{(j)}, ..., P_r^{(j)}\}, \{x_0^{(j)}, ..., x_r^{(j)}\}, \{s_0^{(j)}, ..., s_r^{(j)}\}, \{c_0^{(j)}, ..., c_r^{(j)}\}\rangle$, where $0 \le r \le l$. Initially, this list is set as empty. Whenever the $\mathcal{A}$ queries the random oracle H_1 for any keyword-time-tuple $(v_0^{(j)}, ..., v_r^{(j)})$, the $\mathcal{B}$ responds as follows:
  1. let h be the largest integer $0 \le h \le r$ such that $\{v_0^{(j)}, ..., v_h^{(j)}\} = \{v_0^{(y)}, ..., v_h^{(y)}\}$ for some tuple $\langle\{v_0^{(y)}, ...v_r^{(y)}\}, \{P_0^{(y)}, ..., P_r^{(y)}\}, \{x_0^{(y)}, ..., x_r^{(y)}\}, \{s_0^{(y)}, ..., s_r^{(y)}\}, \{c_0^{(y)}, ..., c_r^{(y)}\}\rangle$ already appears on the L_1, or let $h = -1$ if such a tuple doesn't exist.
  2. for $0 \le k \le h$, the $\mathcal{B}$ sets $P_k^{(j)} = P_k^{(y)}, x_k^{(j)} = x_k^{(y)}, s_k^{(j)} = s_k^{(y)}, c_k^{(j)} = c_k^{(y)}$.

3. for $h < k \leq r$,
   (a) chooses a random $x_k^{(j)} \xleftarrow{R} \mathbb{Z}_p$;
   (b) if $k = 0$, then generates a random coin $c_0^{(j)} \in \{0, 1\}$ so that $Pr[c_0^{(j)} = 0] = \delta$ for some δ that will be determined later. If $c_0^{(j)} = 1$, then sets $P_0^{(j)} = g^{\alpha}$; else sets $P_0^{(j)} = g^{x_0^{(j)}}$. It also sets $s_0^{(j)} = \perp$;
   (c) else if $c_{k-1}^{(j)} = 1$, then randomly chooses $s_k^{(j)} \xleftarrow{R} \mathbb{Z}_p$ and generates a random coin $c_k^{(j)} \in \{0, 1\}$ so that $Pr[c_k^{(j)} = 0] = \delta$ for some δ that will be determined later. If $c_k^{(j)} = 1$, sets $P_k^{(j)} = g^{x_k^{(j)}}$; else sets $P_k^{(j)} = g^{x_k^{(j)}} - (g^{\alpha} + \sum_{z=1}^{k-1} g^{s_z^{(j)} x_z^{(j)}})^{1/s_k^{(j)}}$;
   (d) else if $c_{k-1}^{(j)} = 0$, chooses a random $s_k^{(j)} \xleftarrow{R} \mathbb{Z}_p$ and sets $c_k^{(j)} = 0$ and $P_k^{(j)} = g^{x_k^{(j)}}$.
4. adds $\langle\{v_0^{(j)}, ...v_r^{(j)}\}, \{P_0^{(j)}, ..., P_r^{(j)}\}, \{x_0^{(j)}, ..., x_r^{(j)}\}, \{s_0^{(j)}, ..., s_r^{(j)}\}, \{c_{i,0}^{(j)}, ..., c_{i,r}^{(j)}\}\rangle$ to L_1 and returns $\{P_0^{(j)}, ..., P_r^{(j)})\}$ to the $\mathcal{A}$.

- H_2-queries. The $\mathcal{A}$ can make queries to the random oracle H_2 at any time. In order to respond to these queries, the $\mathcal{B}$ keeps track of all the queries by maintaining a list called L_2. Each entry in the list is a tuple of the form $\langle K_j, V_j\rangle$. The list H_2 is initially empty. To respond to query $H_2(K_j)$, the $\mathcal{B}$ proceed as follows:
  1. if the query K_j already appears on the list L_2 in a tuple $\langle K_j, V_j\rangle$ then respond with $H_2(K_j) = V_j$.
  2. otherwise, the $\mathcal{B}$ picks a random value $V_j \in \{0, 1\}^n$ for $H_2(K_j)$ and adds the tuple $\langle K_j, V_j\rangle$ to the L_2. It responds to $\mathcal{A}$ with $H_2(K_j) = V_j$.

Phase 1. w and $[s..e]$ be a pair of keyword and time period. When the $\mathcal{A}$ issues a TFK trapdoor query for w, s, e to Trapdoor oracle, the $\mathcal{B}$ responds as follows:

1. constructs a set T exactly as Trapdoor algorithm does. Let a set T be $\{(w, t_1, ..., t_r), ...\}$, where $1 \leq r \leq l$.
2. for each keyword-time-tuple $i \in T$, does the following:
   (a) runs the algorithm for responding to H_1-queries to obtain a tuple $\langle\{v_{i,0}, ..., v_{i,r}\}, \{P_{i,0}, ..., P_{i,r}\}, \{x_{i,0}, ..., x_{i,r}\}, \{s_{i,0}, ..., s_{i,r}\}, \{c_{i,0}, ..., c_{i,r}\}\rangle$ in L_1 such that $v_{i,0} = Fuz(w_i, \mathcal{W})$ and $\{v_{i,k} = (t_{i,|k})\}_{1\leq k\leq r}$.
   (b) if $c_{i,r} = 1$, then $\mathcal{B}$ aborts.
   (c) for $0 \leq k \leq r$, does the following:
      - if $k = 0$ and $c_{i,0} = 1$, then sets $\widehat{D_i} = \perp$.
      - if $k = 0$ and $c_{i,0} = 0$, then computes $\widehat{D_i} = (g^{\beta})^{x_{i,0}}$.
      - if $k > 0$ and $c_{i,k-1} = 0$ and $c_{i,k} = 0$, then computes $\widehat{D_i} = \widehat{D_i} + P_{i,k}^{s_{i,k}}$ and $D'_{i,k} = g^{s_{i,k}}$.
      - if $k > 0$ and $c_{i,k-1} = 1$ and $c_{i,k} = 1$, then computes $\widehat{D_i} = \widehat{D_i} + \perp$ and $D'_{i,k} = (g^{\beta})^{s_{i,k}}$.
      - if $k > 0$ and $c_{i,k-1} = 1$ and $c_{i,k} = 0$, then computes $\widehat{D_i} = \widehat{D_i} + (g^{\beta})^{x_{i,k}s_{i,k}}$ and $D'_{i,k} = (g^{\beta})^{s_{i,k}}$.
      - constructs $TD_i = (\widehat{D}_{i,k}, \{D'_{i,k}\}_{k=1,...,r})$.
   (d) constructs $TD_w^F = \{TD_i\}_{\forall i\in T}$.
   (e) runs the algorithm for responding to H_1-queries to obtain a tuple $\langle v_0, P_0, x_0, s_0, c_0\rangle$ such that $v_0 = w$ be the corresponding tuple on the L_1.
   (f) sets $TD_w^E = (g^{\beta})^{x_0}$ for w.
3. gives TD_w^F to algorithm $\mathcal{A}$

Challenge. Once the adversary $\mathcal{A}$ decides that Phase 1 is over, it outputs two keywords w_0^*, w_1^* and a time slot t^*, on those it wishes to be challenged. The $\mathcal{B}$ responds as follows:

1. constructs two keyword-time-tuples from w_0^*, w_1^*, and t^* as $(w_0^*, t_1^*, ..., t_l^*)$ and $(w_1^*, t_1^*, ..., t_l^*)$, Note that these two tuples differ on level 1, but are otherwise equal.
2. runs the algorithm for responding to H_1-queries twice to obtain tuples for $(w_0^*, t_1^*, ..., t_l^*)$ and $(w_1^*, t_1^*, ..., t_l^*)$. For $i = 0, 1$, let $\langle\{v_{i,0}^*, ..., v_{i,l}^*\}, \{P_{i,0}^*, ..., P_{i,l}^*\}, \{x_{i,0}^*, ..., x_{i,l}^*\}, \{s_{i,0}^*, ..., s_{i,l}^*\}, \{c_{i,0}^*, ..., c_{i,l}^*\}\rangle$ such that $v_{i,0}^* = Fuz(w_i^*, \mathcal{W})$ and $\{v_k^* = (t_{i,|k}^*)\}_{k=1,...,l}$ and $\langle v_{i,0}^*, P_{i,0}^*, x_{i,0}^*, s_{i,0}^*, c_{i,0}^*\rangle$ such that $v_{i,0}^* = w_i^*$ be the corresponding tuples on the L_1.
3. if both or either of $c_{0,l}^* = 0$ and $c_{1,l}^* = 0$, then $\mathcal{B}$ aborts;
4. otherwise, the $\mathcal{B}$ randomly picks $b \in \{0, 1\}$ and $\widetilde{C}^* \in \{0, 1\}^n$, and sets $CT_F^* = (\widehat{C}_F^* = g^\gamma, \{C_k'^* = (g^\gamma)^{x_k^*}\}_{k=1,...,l}, C_F^* \in \{0, 1\}^n)$. Note that C_F^* is implicitly defined as $\widetilde{C}^* \oplus H_2(e(H_1(w_b^*), P^\gamma)) = \widetilde{C}^* \oplus H_2(e(g^\alpha, (g^\beta)^\gamma))$. It also chooses a random $r_2 \overset{R}{\leftarrow} \mathbb{Z}_p$ $CT_E^* = (\widetilde{C}^*, \widehat{C}_E^* = g^{r_2}, C_E^* = \widetilde{C}^* \oplus H_2(e(g^{x_0^*}, P^{r_2}))$.
5. constructs the challenge searchable ciphertext $CT^* = (\widetilde{C}^*, CT_F^*, CT_E^*)$
6. returns CT^* to the $\mathcal{A}$.

Phase 2. The $\mathcal{A}$ can continue issue TFK trapdoor queries for any keywords $w^{(j)}$ and any time periods $s^{(j)}, e^{(j)}$, where only restriction is that $t^* \notin [s^{(j)}..e^{(j)}]$. The $\mathcal{B}$ responds to these queries as before.

Guess. Eventually, the $\mathcal{A}$ outputs its guess $b' \in \{0, 1\}$ indicating the challenge CT^* is the result of PETFKS(PP, w_0^*, t^*) or PETFKS(PP, w_1^*, t^*). At this point, the $\mathcal{B}$ picks a random tuple (K, V) from L_2 and outputs K as its solution to the BDH problem.

To complete the proof of Theorem 1, we first analyze the probability that the $\mathcal{B}$ does not abort during the simulation.

Claim 1.1. *The probability that the $\mathcal{B}$ does not abort during Phase 1, Phase 2 and Challenge phases is* $1/e(1 + q_T)$.

Proof of Claim 1.1. Suppose the $\mathcal{A}$ issues a total of q_T TFK trapdoor queries. Then the probability that the $\mathcal{B}$ does not abort as a result of TFK trapdoor queries is δ^{q_T} and the probability that the $\mathcal{B}$ does not abort during the challenge phase is $1 - \delta$. Therefore, the probability that the $\mathcal{B}$ does not abort during the simulation is $\delta^{q_T}(1 - \delta)$. This value is maximized at $\delta_{opt} = 1 - 1/(q_T - 1)$. Using δ_{opt}, the probability that $\mathcal{B}$ does not abort during simulation is at least $1/e(1 + q_T)$. □

Now we show that the $\mathcal{B}$ outputs the correct answer D with probability at least $2/q_{H_2}$ by the following two claims. The proof is based on comparing $\mathcal{A}$'s behavior in the simulated game and the real PETFKS-IND-CKA attack game.

Let $\mathcal{E}$ be the event that the $\mathcal{A}$ queries for the oracle H_2 for D. Let $Pr_R[\cdot]$ and $Pr_{\mathcal{B}}[\cdot]$ denote the probability that an event occurs in the real attack and the simulation, respectively.

Claim 1.2. $Pr_{\mathcal{B}}[\mathcal{E}] = Pr_R[\mathcal{E}]$ *as long as the $\mathcal{B}$ does not abort.*

Proof of Claim 1.2. Let $\mathcal{E}_i$ be the event that the $\mathcal{A}$ issues a query for $H_2(D)$ within its first i queries to the H_2 oracle. We prove by induction on i that $Pr_R[\mathcal{E}_i] = Pr_{\mathcal{B}}[\mathcal{E}_i]$ for all $i \geq 0$. Clearly, $Pr_R[\mathcal{E}_0] = Pr_{\mathcal{B}}[\mathcal{E}] = 0$. Now assume that, for some $i > 0$, we have that $Pr_R[\mathcal{E}_{i-1}] = Pr_{\mathcal{B}}[\mathcal{E}_{i-1}]$. So, we show that

$Pr_R[\mathcal{E}_i] = Pr_{\mathcal{B}}[\mathcal{E}_i]$. We know that

$$\begin{aligned} Pr_R[\mathcal{E}_i] &= Pr_R[\mathcal{E}_i|\mathcal{E}_{i-1}]Pr_R[\mathcal{E}_{i-1}]Pr_R[\mathcal{E}_i|\neg\mathcal{E}_{i-1}]Pr_R[\neg\mathcal{E}_{i-1}] \\ &= Pr_R[\mathcal{E}_{i-1}]Pr_R[\mathcal{E}_i|\neg\mathcal{E}_{i-1}]Pr_R[\neg\mathcal{E}_{i-1}] \end{aligned} \tag{7.1}$$

We argue that $Pr_R[\mathcal{E}_i|\neg\mathcal{E}_{i-1}] = Pr_{\mathcal{B}}[\mathcal{E}_i|\neg\mathcal{E}_{i-1}]$. To see this, observe that as long as the $\mathcal{A}$ does not make a query for $H_2(D)$, its view during the simulation is identical to its view in the real attack. Moreover, the public parameters and the challenge are distributed as in the real attack. Similarly, all responses to H_2-queries are uniform and independent in $\{0,1\}^n$. Therefore, $Pr_{\mathcal{B}}[\mathcal{E}_i|\neg\mathcal{E}_{i-1}] = Pr_R[\mathcal{E}_i|\neg\mathcal{E}_{i-1}]$. It follows by the (1) and the inductive hypothesis that $Pr_R[\mathcal{E}_i] = Pr_{\mathcal{B}}[\mathcal{E}_i]$. By induction on i, we obtain that $Pr_R[\mathcal{E}] = Pr_{\mathcal{B}}[\mathcal{E}]$. □

We show that $Pr_{\mathcal{B}}[\mathcal{E}] \geq 2\epsilon$. This will prove that the $\mathcal{B}$ outputs D with probability at least $2\epsilon/q_{H_2}$.

Claim 1.3. *We have $Pr_R[\mathcal{E}] \geq 2\epsilon$.*

Proof of Claim 1.3. In the real attack, if the $\mathcal{A}$ never issues a query for $H_2(e(H_1(w_b^*, P^{r_1})))$, then the last component C_F^* of the ciphertext is independent of the $\mathcal{A}$'s view (since $H_2(D)$ is independent of $\mathcal{A}$'s view). Therefore, in the real attack, $Pr[b = b'|\neg\mathcal{E}] = 1/2$. By definition of the $\mathcal{A}$, we know that $|Pr_R[b = b'] - 1/2| \geq \epsilon$. We show that these two facts imply that $Pr_R[\mathcal{E}] \geq 2\epsilon$. To do so, we first derive simple upper and lower bounds on $Pr_R[b = b']$:

$$\begin{aligned} Pr_R[b = b'] &= Pr_R[b = b'|\neg\mathcal{E}]Pr_R[\neg\mathcal{E}] + Pr[b = b'|\mathcal{E}]Pr[_R\mathcal{E}] \\ &\leq Pr_R[b = b'|\neg\mathcal{E}]Pr_R[\neg\mathcal{E}] + Pr_R[\mathcal{E}] \\ &= \frac{1}{2}Pr_R[\neg\mathcal{E}] + Pr_R[\mathcal{E}] = \frac{1}{2} + \frac{1}{2}Pr_R[\mathcal{E}], \end{aligned}$$

$$Pr_R[b = b'] \geq Pr_R[b = b'|\neg\mathcal{E}]Pr_R[\neg\mathcal{E}] = \frac{1}{2} - \frac{1}{2}Pr_R[\mathcal{E}]$$

It follows that $\epsilon \leq |Pr_R[b = b'] - 1/2| \leq \frac{1}{2}Pr_R[\mathcal{E}]$. Therefore, $Pr_R[\mathcal{E}] \geq 2\epsilon$. □

Since $Pr_R[\mathcal{E}] = Pr_{\mathcal{B}}[\mathcal{E}]$ by Claim 1.2 and $Pr_R[\mathcal{E}] \geq 2\epsilon$ by Claim 1.3, we know that $Pr_{\mathcal{B}}[\mathcal{E}] \geq 2\epsilon$. Hence, at the end of the simulation, D appears in some tuple on the L_2 with probability at least 2ϵ. It follows that the $\mathcal{B}$ outputs the correct answer with probability at least $2\epsilon/q_{H_2}$ as required.

With combining Claims 1.1, 1.2, and 1.3, the $\mathcal{B}$'s advantage in solving the BDH problem is at least $2\epsilon/e(1 + q_T)q_{H_2}$, from which the Theorem 1 follows. □

Now we show the PETFKS-IND-sCKA-KGA security of our PETFKS scheme by the following theorem.

Theorem 2. *Suppose the hash functions H_1, H_2 are random oracles. Then our PETFKS scheme is PETFKS-IND-sCKA-KGA-secure in the random oracle model assuming BDH problem is hard in groups generated by $BGen(1^\lambda)$.*

Proof. Suppose there is an adversary $\mathcal{A}$ that has advantage ϵ in attacking PETFKS-IND-sCKA-KGA security of the PETFKS scheme. Suppose $\mathcal{A}$ makes at most q_{H_2} hash function queries to H_2. We construct a simulator $\mathcal{B}$ that is able to solve the BDH problem with advantage at least $2\epsilon/q_{H_2}$. Before the game starts, the challenger sets the BDH parameters as them in previous security proof and gives them to the simulator $\mathcal{B}$. The $\mathcal{B}$ acts as the adversary $\mathcal{A}$'s challenger in the PETFKS-IND-sCKA-KGA game and interacts with the $\mathcal{A}$ as follows:

Init. The $\mathcal{B}$ runs the $\mathcal{A}$ and receives w_0^* and w_1^*, that have the same fuzzy value, $Fuz(w_0^*, \mathcal{W}) = Fuz(w_1^*, \mathcal{W})$.

Setup. The $\mathcal{B}$ sets $P = g^{\beta}$ and gives the public parameters $PP = (p, g, e, \mathbb{G}_1, \mathbb{G}_2, P, \mathcal{W}, \mathcal{T}, H_1, H_2, Fuz)$ to the $\mathcal{A}$.

The simulator $\mathcal{B}$ will program the random oracles H_1 and H_2 as follows.

- H_1-queries. The $\mathcal{A}$ can query the random oracle H_1 at any time. In order to respond the queries, the $\mathcal{B}$ maintains a list called L_1. Each entry in the list is a tuple of form $\langle$*Keyword-time-tuple, Point-tuple, Scalar-tuple, Secret-tuple*$\rangle$. For j-th query, the tuple is formed as $\langle\{v_0^{(j)}, ..., v_r^{(j)}\}, \{P_0^{(j)}, ..., P_t^{(j)}\}, \{x_0^{(j)}, ..., x_r^{(j)}\}, \{s_0^{(j)}, ..., s_r^{(j)}\}, \{c_0^{(j)}, ..., c_r^{(j)}\}\rangle$, where $0 \leq r \leq l$. Initially, L_1 is set as empty. Whenever the $\mathcal{A}$ queries the random oracle H_1 for any value $\{v_0^{(j)}, ..., v_r^{(j)}\}$, the $\mathcal{B}$ responds as follows:
  1. let h be the largest integer $0 \leq h \leq r$ such that $\{v_0^{(j)}, ..., v_h^{(j)}\} = \{v_0^{(y)}, ..., v_h^{(y)}\}$ for some tuple $\langle\{v_0^{(y)}, ...v_r^{(y)}\}, \{P_0^{(y)}, ..., P_r^{(y)}\}, \{x_0^{(y)}, ..., x_r^{(y)}\}, \{s_0^{(y)}, ..., s_r^{(y)}\}, \{c_0^{(y)}, ..., c_r^{(y)}\}\rangle$ already appears on the L_1, or let $h = -1$ if such a tuple doesn't exist.
  2. for $0 \leq k \leq h$, $\mathcal{B}$ sets $P_k^{(j)} = P_k^{(y)}, x_k^{(j)} = x_k^{(y)}, s_k^{(j)} = s_k^{(y)}, c_k^{(j)} = c_k^{(y)}$.
  3. for $h < k \leq r$,
     (a) chooses a random $x_k^{(j)} \overset{R}{\leftarrow} \mathbb{Z}_p$;
     (b) if $k = 0$ and $v_0^{(j)}$ is equal to w_0^* or w_1^*, then sets $P_0^{(j)} = g^{\alpha}$ and $x_0^{(j)} = \perp$; else sets $P_0^{(j)} = g^{x_0^{(j)}}$. It also sets $s_0^{(j)} = \perp$;
     (c) else chooses a random $s_k^{(j)} \overset{R}{\leftarrow} \mathbb{Z}_p$ and sets $P_k^{(j)} = g^{x_k^{(j)}}$;
  4. adds $\langle\{v_0^{(j)}, ...v_r^{(j)}\}, \{P_0^{(j)}, ..., P_r^{(j)}\}, \{x_0^{(j)}, ..., x_r^{(j)}\}, \{s_0^{(j)}, ..., s_r^{(j)}\}, \{c_0^{(j)}, ..., c_r^{(j)}\}\rangle$ to L_1 and returns $\{P_0^{(j)}, ..., P_r^{(j)})\}$ to the $\mathcal{A}$.
- H_2-queries. The $\mathcal{B}$ responds to H_2 queries in the same way it does in the PETFKS-IND-CKA security game.

Phase 1. Let w and $[s..e]$ be a pair of keyword and time period. When the $\mathcal{A}$ issues a TFK trapdoor query for w, s, e to Trapdoor oracle, the $\mathcal{B}$ responds as follows:

1. constructs a set T exactly as Trapdoor algorithm does. Let a set T be $\{(w, t_1, ..., t_r), ...\}$, where $1 \leq r \leq l$.
2. for each keyword-time-tuple $i \in T$, does the following:
   (a) runs the algorithm for responding to H_1-queries to obtain a tuple $\langle\{v_{i,0}, ..., v_{i,r}\}, \{P_{i,0}, ..., P_{i,r}\}, \{x_{i,0}, ..., x_{i,r}\}, \{s_{i,0}, ..., s_{i,r}\}\rangle$ in L_1 such that $v_{i,0} = Fuz(w_i, \mathcal{W})$ and $\{v_{i,k} = (t_{i,|k})\}_{1 \leq k \leq r}$.
   (b) computes $\widehat{D}_i = (g^{\beta})^{x_{i,0}} + \sum_{k=1}^{r} P_{i,k}^{s_{i,k}}$ and $\{D'_{i,k} = g^{s_{i,k}}\}_{k=1,...,r}$.
   (c) constructs $TD_i = (\widehat{D}_{i,k}, \{D'_{i,k}\}_{k=1,...,r})$.
   (d) constructs $TD_w^F = \{TD_i\}_{\forall i \in T}$.

(e) runs the algorithm for responding to H_1-queries to obtain a tuple $\langle v_0, P_0, x_0, \perp, c_0\rangle$ such that $v_0 = w$ be the corresponding tuple on the L_1.

(f) sets $TD_w^E = (g^\beta)^{x_0}$ for $w \neq w_0^*$ and $w \neq w_1^*$.

3. gives TD_w^F to the $\mathcal{A}$

Challenge. Once the $\mathcal{A}$ decides that Phase 1 is over, it outputs a time slot t^*. The $\mathcal{B}$ responds as follows:

1. constructs two keyword-time-tuples from w_0^*, w_1^*, and t^* as $(w_0^*, t_1^*, ..., t_l^*)$ and $(w_1^*, t_1^*, ..., t_l^*)$, Note that these two tuples differ on level 1, but are otherwise equal.
2. randomly picks $b \in \{0, 1\}$.
3. runs the algorithm for responding to H_1-queries twice to obtain tuples for $(w_b^*, t_1^*, ..., t_l^*)$ and w_b^*. Let $\langle\{v_0^*, ..., v_l^*\}, \{P_0^*, ..., P_l^*\}, \{x_0^*, ..., x_l^*\}, \{s_0^*, ..., s_l^*\}\rangle$ such that $v_0^* = Fuz(w_b^*, \mathcal{W})$ and $\{v_k^* = (t_{b,|k}^*)\}_{k=1,...,l}$ and $\langle v_0^*, P_0^*, \perp, \perp\rangle$ such that $v_0^* = w_b$ be the corresponding tuples on the L_1.
4. chooses a random $r_1 \overset{R}{\leftarrow} \mathbb{Z}_p$ and $\widetilde{C}^* \in \{0, 1\}^n$, and sets $CT_F^* = (\widehat{C}_F^* = g^{r_1}, \{C_k^{'*} = (P_k^*)^{r_1}\}_{k=1,...,l}, C_F^* = \widetilde{C}^* \oplus H_2(e(g^{x_0^*}, P^{r_1}))$ and $CT_E^* = (\widetilde{C}^*, \widehat{C}_E^* = g^\gamma, C_E^* \in \{0, 1\}^n$. Note that C_E^* is implicitly defined as $\widetilde{C}^* \oplus H_2(e(H_1(w_b^*), P^\gamma)) = \widetilde{C}^* \oplus H_2(e(g^\alpha, g^{\beta\gamma}))$.
5. constructs $CT^* = (\widetilde{C}, CT_F^*, CT_E^*)$
6. returns CT^* to $\mathcal{A}$.

Phase 2. The $\mathcal{A}$ can continue issue TFK trapdoor queries for any keywords $w^{(j)}$ and any time periods $[s^{(j)}..e^{(j)}]$ to Trapdoor oracle. The $\mathcal{B}$ responds to these queries as before.

Guess. Eventually, the $\mathcal{A}$ outputs its guess $b' \in \{0, 1\}$ indicating the challenge CT^* is the result of PETFKS(PP, w_0^*, t^*) or PETFKS(PP, w_1^*, t^*). At this point, the $\mathcal{B}$ picks a random tuple (K, V) from L_2 and outputs K as its solution to the BDH problem.

This completes the description of the simulator $\mathcal{B}$. Since the $\mathcal{A}$ is allowed to issue TFK trapdoor queries to Trapdoor oracle for any pair of keyword and time period including the challenge ones, the $\mathcal{B}$ does not abort during the simulation as a result of the $\mathcal{A}$'s TFK trapdoor queries. It remains to show that the $\mathcal{B}$ correctly outputs D with probability at least $2\epsilon/q_{H_2}$.

Claim 2.1. *Suppose that in the real attack the $\mathcal{A}$ is given the public parameters $PP = (p, g, e, \mathbb{G}_1, \mathbb{G}_2, P = g^\beta, \mathcal{W}, \mathcal{T}, H_1, H_2, Fuz)$ and asks to be challenged on w_0^*, w_1^* and a time slot t^*. In response, the $\mathcal{A}$ is given a challenge $CT^* = \langle\widetilde{C}, CT_F^* = (\widehat{C}_F^* = g^{r_1}, \{C_k^{'*} = (P_k^*)^{r_1}\}_{k=1,...,l}, C_F^* = \widetilde{C}^* \oplus H_2(e(g^{x_0^*}, P^{r_1})), CT_E^* = (\widetilde{C}^*, \widehat{C}_E^* = g^\gamma, C_E^* \in \{0, 1\}^n)\rangle$. Then in the real attack game the $\mathcal{A}$ issues query for either for $H_2(e(H_1(w_0^*), P^{r_2})$ or $H_2(e(H_1(w_1^*), P^{r_2})$ with probability al least 2ϵ.*

Proof of Claim 2.1. Let $\mathcal{H}$ be the event that in the real attack the $\mathcal{A}$ issues a query for either of $H_2(e(H_1(w_0^*), P^{r_2})$ or $H_2(e(H_1(w_0^*), P^{r_2})$. Then, when $\mathcal{H}$ takes place we know that the bit $b \in \{0, 1\}$ indicating whether C_E^* is an EKS ciphertext of w_0^* or w_1^* is independent of the $\mathcal{A}$'s view. Therefore, the $\mathcal{A}$'s output b' will satisfy $b = b'$ with probability at most $1/2$. By definition of the $\mathcal{A}$, we know that in the real attack $|Pr[b = b'] - 1/2| \geq \epsilon$. We show that these two facts imply that $Pr[\mathcal{H}] \geq 2\epsilon$. To do so, we first derive simple upper and lower bounds on $Pr[b = b']$:

$$\begin{aligned} Pr[b = b'] &= Pr[b = b'|\neg\mathcal{H}]Pr[\neg\mathcal{H}] + Pr[b = b'|\mathcal{H}]Pr[\mathcal{H}] \\ &\leq Pr[b = b'|\neg\mathcal{H}]Pr[\neg\mathcal{H}] + Pr[\mathcal{H}] \end{aligned}$$

$$= \frac{1}{2}Pr[\neg\mathcal{H}] + Pr[\mathcal{H}] = \frac{1}{2} + \frac{1}{2}Pr[\mathcal{H}],$$

$$Pr[b = b'] \geq Pr[b = b'|\neg\mathcal{H}]Pr[\neg\mathcal{H}] = \frac{1}{2} - \frac{1}{2}Pr[\mathcal{H}]$$

It follows that $\epsilon \leq |Pr[b = b'] - 1/2| \leq \frac{1}{2}Pr[\mathcal{H}]$. Therefore, in the real attack, $Pr[\mathcal{H}] \geq 2\epsilon$.

Now, assuming the $\mathcal{B}$ does not abort, we know that $\mathcal{B}$ simulates a real attack game perfectly up to the moment when the $\mathcal{A}$ issues a query for either $H_2(e(H_1(w_0^*), P^\gamma)$ or $H_2(e(H_1(w_1^*), P^\gamma)$. Therefore, at the end of the simulation, D appears in some tuple on the L_2 with probability at least 2ϵ. It follows that the $\mathcal{B}$ outputs the correct answer with probability at least $2\epsilon/q_{H_2}$ as required. □

By Claim 2.1, the $\mathcal{B}$'s advantage in solving the BDH problem is at least $2\epsilon/q_{H_2}$, from which the Theorem 2 follows. □

7.2. *Efficiency analysis*

Now we give efficiency analysis of our scheme by comparing with the other related public key encryption with keyword search schemes, PEFKS[9] and PETKS[10], as these schemes support the properties, namely fuzzy keyword search and temporary property, as our scheme does. The comparisons are made in terms of storage overhead, communication cost, and computational efficiency. Now we describe the notations used in the comparisons. Let $|\mathbb{G}_1|$, $|\mathbb{G}_2|$, and $|\mathbb{Z}_p|$ denote the size of an element in the source group, the target group, and the field Z_p, respectively. In addition, let t denote the size of the system time space and s be the number of time slots included in the time period of the trapdoor, which is $s < t$. Moreover, let n denote the total number of searchable ciphertexts stored in the cloud server and m be the total number of searchable ciphertexts including the keyword in the search query for each time slot in the system time space.

Storage Overhead. From Table 2, we can see the storage overhead on each entity in the schemes, namely data receiver (DR), data sender (DS) and cloud server (CS). The public parameters and master key contribute the storage overhead on DS and DR, respectively. In our scheme, storage overhead on DR and DS is equal to the ones in PEFKS [9] and PETKS [10]. The main storage overhead on CS comes from searchable ciphertexts. Since our scheme supports both fuzzy keyword search and temporary property, the storage overhead on CS in our scheme is quite more than that in PEFKS [9] due to the components representing the temporary property, and it is slightly more than that in PETKS [10] due to the EKS ciphertexts.

Table 2. Comparison of storage overhead.

Entity	Our Scheme	PEFKS [9]	PETKS [10]
DR	$\lvert\mathbb{Z}_p\rvert$	$\lvert\mathbb{Z}_p\rvert$	$\lvert\mathbb{Z}_p\rvert$
DS	$2\lvert\mathbb{G}_1\rvert$	$2\lvert\mathbb{G}_1\rvert$	$2\lvert\mathbb{G}_1\rvert$
CS	$n((2 + log(T))\lvert\mathbb{G}_1\rvert + 2\lvert\mathbb{G}_2\rvert)$	$n(2\lvert\mathbb{G}_1\rvert + 2\lvert\mathbb{G}_2\rvert)$	$n((1 + log(T))\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_2\rvert)$

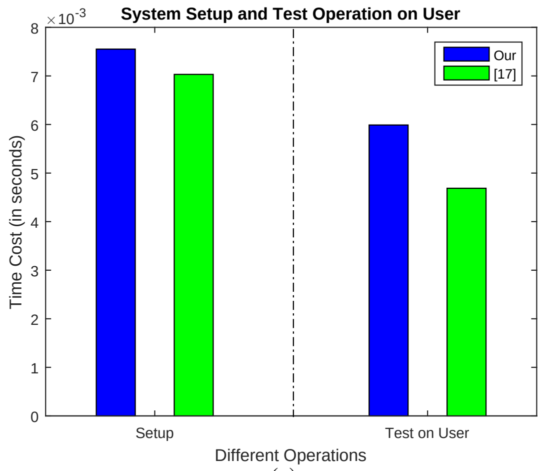
Communication Cost. In Table 3, we discuss communication cost between the entities in the schemes. In all the schemes, the communication cost from DR to DS comes from public parameters, and it is equal in all the schemes. The communication cost between DR and CS comes from the transmission of a trapdoor, and smaller in PEFKS [9] and it is higher in PETKS [10] comparing to our scheme. The size of a searchable ciphertext results in communication cost from DS to CS, and in our scheme, it is much higher than that in PEFKS [9] owing to the components representing the temporary property , and slightly higher than that in PETKS [10] because of the EKS ciphertexts. Due to the resistance of KGA, both our scheme and PEFKS [9] bring communication cost from CS to DR before transmitting the data ciphertexts, and it much less than in our scheme by means of temporary property.

Table 3. Comparison of communication cost.

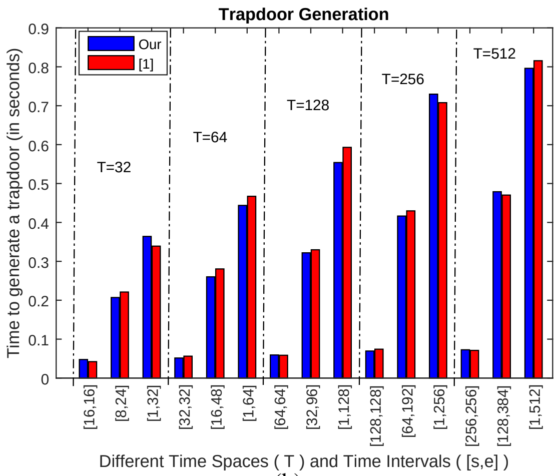
Entity	Our Scheme	PEFKS [9]	PETKS [10]
DR to DS	$2\|\mathbb{G}_1\|$	$2\|\mathbb{G}_1\|$	$2\|\mathbb{G}_1\|$
DS to CS	$(2 + log(T))\|\mathbb{G}_1\| + 2\|\mathbb{G}_2\|$	$2\|\mathbb{G}_1\| + 2\|\mathbb{G}_2\|$	$(1 + log(T))\|\mathbb{G}_1\| + \|\mathbb{G}_2\|$
DR to CS	$s((1 + log(T))\|\mathbb{G}_1\|)$	$\|\mathbb{G}_1\|$	$s((1 + log(T))\|\mathbb{G}_1\| + \|\mathbb{Z}_p\|)$
CS to DR	$sm(\|\mathbb{G}_1\| + \|\mathbb{G}_2\|)$	$tm(2\|\mathbb{G}_1\| + 2\|\mathbb{G}_2\|)$	NA

Computational Efficiency. To evaluate the computational efficiency of our scheme, we implemented our scheme and other related schemes, PEFKS [9] and PETKS [10] . The implementations use the PBC library [29] and 160-bit elliptic curve group based on the supersingular curve $y^2 = x^3 + x$ over a 512-bit finite field. All the experiments were carried out on 2.50GHz, Intel(R) Core(TM) i5, 8GB memory and running Ubuntu 16.04. All the experiment results are the average of 20 trials.

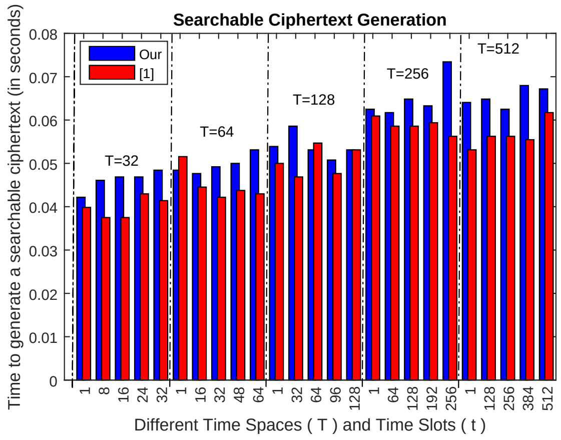
We compare the computational efficiency of our scheme and PEFKS [9] regarding system setup and test operation on DR, and the computational efficiency of our scheme and PETKS [10] with regard to trapdoor generation, searchable ciphertext generation and test operation on CS decryption, as shown in Figure 3. The computational efficiency of the system setup in PETKS [10] is equal to that of our scheme, and the PETKS scheme [10] does not perform test operation on DR. Hence, we show the system setup time and test operation time on DR in our scheme by comparing to the ones in PEFKS [9] by excluding PETKS [10], and the computational costs in both the schemes are nearly close, as shown in Figure 3(a). In order to test the computational cost of trapdoor generation, searchable ciphertext generation and test operation on CS, we run our code for several choices of time spaces $T = \{32, 64, 128, 256, 512\}$, and several choices of time intervals $[s, e]$ and time slots t depending on the chosen time space. As we can see from Figure 3(b), 3(c) and 3(d), the time space is divided into 4 equal parts as 1, $\frac{1}{4}T$, $\frac{1}{2}T$, $\frac{3}{4}T$, T and use these time slots to test trapdoor generation time, searchable ciphertext generation time and test operation time on CS. For example, when time space is $T = 32$, the choices of time intervals $[s, e]$ for trapdoor generation are $[8, 8]$, $[8, 24]$, and $[1, 32]$ and time slots t for searchable ciphertext generation and positive result of test operation on CS are $1, 8, 16, 24, 32$. From the choices of the time intervals and time slots, we can see the interval for trapdoor is growing from a time slot $[8, 8]$ to whole time space $[1, 32]$. Given a trapdoor generated on $[1, 32]$, the time slot for the positive search result of a test operation on CS is also growing by the same size. Figure 3(b), 3(c), and 3(d) show that even though the cost of each operation scales linearly with the size of the time space, they remain similar in both schemes for the given time space. Overall, although our scheme is a little less efficient than other schemes, it makes a great effort on decreasing the communication cost from CS to DR such that computational cost of the test operation on DR is brought down. Therefore, we conclude the performance of our scheme is practical since our scheme provides secure keyword search



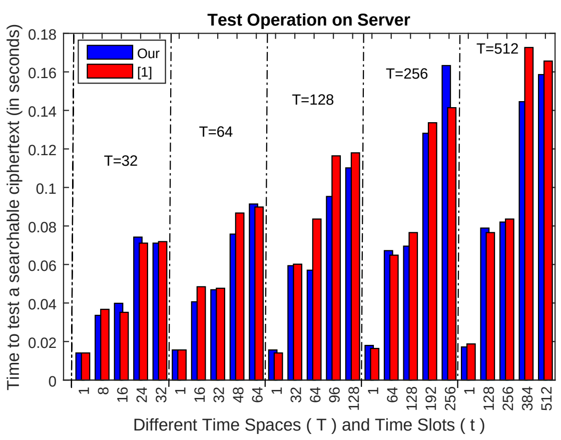
(a)



(b)



(c)



(d)

Figure 3. Comparison of computational efficiency

and temporary property at the same time.

8. Conclusion

In this paper, we proposed a Public Key Encryption Scheme with Temporary and Fuzzy Keyword Search (PETFKS). It offers secure search operation on server side and more efficient search operation on user side at the same time, limits the lifetime of trapdoor as well as fulfills the desire for secure time-dependent data retrieval, which are not able to be tackled by the existing works. Furthermore, the PETFKS-IND-CKA and PETFKS-IND-sCKA-KGA securities of the PETFKS scheme are formalized concerning the security requirements of keyword confidentiality, forward and backward secrecy and keyword guessing resistance. Moreover, the rigorous security proof is given to show that the proposed scheme is PETFKS-IND-CKA and PETFKS-IND-sCKA-KGA secure in the random oracle model under the BDH assumption. Based on the analyses conducted with regard to security and efficiency of the proposed scheme and related works, we demonstrate that our proposed scheme is a practical solution to the open problem and an effective response to the gap in public key encryption with keyword search.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (No.61672135), the National Science Foundation of China-Guangdong Joint Foundation (No.U1401257), the Sichuan Science-Technology Support Plan Program (No.2018GZ0236 and No.2017FZ0004), and the Fundamental Research Funds for the Central Universities (No.2672018ZYGX2018J057). The authors gratefully acknowledge the anonymous reviewers for their valuable comments.

References

1. D. Boneh, G. Di Crescenzo, R. Ostrovsky, et al., Public key encryption with keyword search, *Advances in Cryptology - EUROCRYPT 2004 (Lecture Notes in Computer Science)*, **3027** (2004), 506–522.
2. H. Xiong, H. Zhang and J. Sun, Attribute-based Privacy-Preserving Data Sharing for Dynamic Groups in Cloud Computing, *IEEE Syst. J.*, (2018), 1–22.
3. H. Xiong, Q. Mei and Y. Zhao, Efficient and Provably Secure Certificateless Parallel Key-Insulated Signature without Pairing for IIoT Environments, *IEEE Syst. J.*, (2018).
4. Q. Jiang, Y. Qian, J. Ma, et al., User centric three-factor authentication protocol for cloud-assisted wearable devices, *Int. J. Commun. Syst.*, **e3900**, (2018).
5. J. Sun, Y. Bao, X. Nie, et al., Attribute-hiding Predicate Encryption with Equality Test in Cloud Computing, *IEEE Access*, **6**, (2018), 31621–31629.
6. J. W. Buyn, H. S. Rhee, H. A. Park, et al., Off-line keyword guessing attacks on recent keyword search schemes over encrypted data *Proc. SDM'06,* Seoul, Korea, (2006), 75–83.
7. I. R. Jeong, J. O. Kwon, D. Hong, et al., Constructing PEKS schemes secure against keyword guessing attacks is possible? *Comput. Commun.*, **32**, (2009), 394–396.
8. C. Chen, B. Xiang, Y. Liu, et al., A Secure Authentication Protocol for Internet of Vehicles, *IEEE Access*, (2019).
9. P. Xu, H. Jin, Q. Wu, et al., Public-key encryption with fuzzy keyword search: A provably secure scheme under keyword guessing attack, *IEEE T. Comput.*, **62**, (2013), 2266–2277.
10. M. Abdalla, M. Bellare, D. Catalano, et al., Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions, *Advances in Cryptology-CRYPTO 2005 (Lecture Notes in Computer Science)*, **3621** (2005), 205–222.
11. H. Xiong, Y. Zhao, L. Peng, et al., Partially policy-hidden attribute-based broadcast encryption with secure delegation in edge computing, *Future Gener. Compu. Sy.*, **97**, (2019), 453–461.
12. H. Xiong, K. R. Choo and A. V. Vasilakos, Revocable Identity-Based Access Control for Big Data with Verifiable Outsourced Computing, *IEEE T. Big Data*, (2017).

13. H. Xiong and J. Sun, Comments on Verifiable and exculpable outsourced attribute-based encryption for access control in cloud computing, *IEEE T. Depend. Secure Comput.*, **14**, (2017), 461–462.

14. K. H. Yeh, A Secure Transaction Scheme With Certificateless Cryptographic Primitives for IoT-Based Mobile Payments, *IEEE Syst. J.*, **12**, (2018), 2027–2038.

15. P. Golle, J.Staddon and B. Waters, Secure Conjunctive Keyword Search over Encrypted Data, *Applied Cryptography and Network Security - ACNS 2004 (Lecture Notes in Computer Science)*, **3089**, (2004), 31–45.

16. D. J. Park, K. Kim and P. J. Lee Guo, Public Key Encryption with Conjunctive Field Keyword Search, *Information Security Applications-WISA 2004 (Lecture Notes in Computer Science)*, **3325**, (2004), 73–86.

17. Y. H. Hwang and P. J. Lee, Public Key Encryption with Conjunctive Keyword Search and Its Extension to a Multi-user System, *Pairing-Based Cryptography-Pairing 2007 (Lecture Notes in Computer Science)*, **4575** (2007), 2–22.

18. E. Shi, J. Bethencourt, T. H. Chan, et al., Multi-Dimensional Range Query over Encrypted Data, in *2007 IEEE Symposium on Security and Privacy (SP '07),* Berkeley, CA, (2007), 350–364.

19. L. Ibraimi, S. Nikova, P. Hartel, et al., Public Key Encryption with Authorized Keyword Search, *Applied Cryptography and Network Security-ACNS 2011 (Lecture Notes in Computer Science)*, **6715** (2011), 532–549.

20. J. Zhang and J. Mao, Efficient public key encryption with revocable keyword search in cloud computing, *Cluster Comput.*, **19** (2016), 1211–1217.

21. P. Jiang, Y. Mu, F. Guo, et al., Public Key Encryption with Authorized Keyword Search, *Information Security and Privacy-ACISP 2016 (Lecture Notes in Computer Science)*, **9723** (2016), 170–186.

22. J. Baek, R. Safavi-Naini and W. Susilo, Public Key Encryption with Keyword Search Revisited, *Proc. ICCSA '08,* Perugia, Italy, (2008), 1249–1259.

23. Q. Tang and L. Chen, Public-Key Encryption with Registered Keyword Search, *Public Key Infrastructures, Services and Applications-EuroPKI 2009 (Lecture Notes in Computer Science)*, **6391** (2009), 163–178.

24. H. Yang, C. Xu and H. Zhao, An Efficient Public Key Encryption with Keyword Scheme Not Using Pairing, in *'11 First International Conference on Instrumentation, Measurement, Computer, Communication and Control,* Beijing, China, (2011), 900–904.

25. D. Boneh and M. Franklin, Identity-Based Encryption from the Weil Pairing, *Advances in Cryptology CRYPTO 2001(Lecture Notes in Computer Science)*, **2139** (2001), 213–229.

26. H. Xiong and Z. Qin, Revocable and Scalable Certificateless Remote Authentication Protocol with Anonymity for Wireless Body Area Networks, *IEEE T. Inf. Foren. Sec.*, **10** (2015), 1442–1455.

27. S. Halevi, PBC (Pairing-Based Cryptography) library, *IACR Cryptology ePrint Archive*,(2005), Available from: `https://eprint.iacr.org/2005/005.pdf`.

28. J. Katz, A Forward-Secure Public-Key Encryption Scheme, *IACR Cryptology ePrint Archive*, (2002), Available from: `https://eprint.iacr.org/2002/060.pdf`.

29. B. Lynn, A sufficient condition for key-privacy, Available from: https://crypto.stanford.edu/pbc/, Accessed on: Sep. 15, 2018.