International Journal of Computer Applications (0975 - 8887)
Volume 154 - No.4, November 2016

Public Key Encryption with 'Fixed and Short Length' Keyword Search

Fairouz Sher Ali
[1] School of Computer Science and Technology,
Huazhong University of Science and Technology,
Wuhan, Hubei 430074, P.R.China
[2] Kufa University
Kufa, Iraq

ABSTRACT

Cloud computing has emerged as a new technology that aims to provide unlimited virtualized resources to clients and enterprises. As services and huge sensitive data are being moved to the cloud server, a new challenge arises as to how to use the migrated data in a way that preserves privacy. Due to privacy concerns, important data should be encrypted before uploading onto cloud servers, so that only authenticated clients can access the data. Searchable encryption techniques allow the clients to search the encrypted data. Public key encryption with keyword search (PEKS) is a scheme of searchable encryption using a public key solution. In our scheme, we present a novel public key encryption with the 'fixed and short length' keyword search which reduce the size of the keyword space and get keywords with a fixed and short length. Further, we employe the Bloom filters (BFs), which can accelerate the search process with a large amount of keywords. We also analyse the security of our construction in the random oracle model.

Keywords

Cloud Computing, Searchable Encryption, Public key encryption, Bloom filter, Bilinear pairing

1. INTRODUCTION

1.1 Basic Concepts

Cloud computing [1, 2, 3, 4, 5] is one of today's most attractive technologies. In it, cloud service providers (CSP) offer an efficient and secure data storage and computing service to consumers and enterprises. A connected terminal is the only requirement for a user. Cloud computing migrates the data to the cloud storage, where the management of the data and services may not be fully reliable. Many papers are focusing on the security of cloud data storage, which is regarded as an important feature of the quality of service. A currently standard solution to protect data in cloud computing is applying cryptographic techniques to sensitive data. There are very many cryptographic primitives that we are able to apply in this manner. In recent years, many attempts have been made to propose efficient and effective schemes to enable searching over encrypted data.

Searchable Encryption is a cryptographic primitive that allows clients to perform keyword-based searches on an encrypted database [6, 7, 8, 9]. Searchable encryption can be done securely in its full generality using the scheme in Ref. [10] on software protection depending on oblivious RAMs [10, 11]. Searchable encryption can be classified into two fields: symmetric searchable encryption (SSE) and asymmetric searchable encryption (ASE)

Symmetric Searchable Encryption (SSE): this type enables a data owner to outsource the storage of its data to a remote server in a private form, while maintaining the ability to conditionally search over it. All prior techniques on SSE tackled the setting where only the owner of the data can send search queries. In Ref. [6], the authors consider the natural extension where an arbitrary group of users other than the owner can send search queries. They suggest sharing the secret key for database searching among all users.

Asymmetric Searchable Encryption (ASE): in this scheme, the user who encrypt the data and sends it to the server is usually different from the owner of the decryption secret key. In a classic public-key scheme, a user publishes a public key while multiple senders send their files to the server [12].

1.2 Related Work and Our Contributions

Song, Wagner, and Perrig [9] first proposed the notion of searchable encryption for a single user. They presented a scheme in the symmetric key setting, which encrypts each word of a file separately. However, this scheme can be applied to the private-key setting for the user who owns his data and needs to outsourced it to a non-trusted third party database. They also gave out some practical solutions in which the searching overhead is linear in the file's length. On the other hand, this scheme is not suitable for many practical applications, such an outsourced database, an email routing system, etc. [13]. Goh [8] introduced an efficient secure index construction based on pseudo-random functions and Bloom filters that requires *O(n)* search time, where n is the number of files in the collection; it results in false positives. Unlike Goh, Chang and Mitzenmacher [7] developed a new construction also with *O(n)* search time but without false positives.

Boneh, Kushilevitz, Ostrovsky and Skeith [14] also employed a Bloom filter that allows the users to keep the space that is used to store the extra information "small". The technique is similar to Goh's use of Bloom filter [8]; the essential difference is that in their

technique they consider a public-key solution, while Goh [8] supports a private-key solution. Wang, Chena, Lic, Zhaod and Shene [15] proposed a novel verifiable search technique for outsourced database based on an invertible Bloom filter (IBF).
The public key encryption with keyword search (PEKS) scheme was first proposed by Boneh, Crescenzo, Ostrovsky, and Persiano [12] This scheme enables searching keywords within encrypted messages. PEKS is desirable for mobile devices, e.g. accessing encrypted email messages through the mobile Internet. Golle, Staddon, and Waters [16] presented schemes that allow for conjunctive keyword queries on encrypted data. Boneh and Waters [17] developed [12] to support conjunctive, subset, and range comparisons over the keywords.
Despite the efficiency of PEKS [12], there are some important cases relating to the use of PEKS, which were studied in [18]. One of these cases is where a server that has received the trapdoors can save them in its memory and use them to retrieve all future emails within that category. The scheme does not determine what happens if the remote server memorizes the trapdoor, and also does not discuss protection against this situation. The user can solve this problem by using different keywords in different search operations, but this solution is impractical, especially when the user needs to reuse their keywords.
Our proposed solution to solve the above problem is to modify the repeatedly used keywords by attaching them with the number of their use times. For example, the keyword $w =$ *Science* now becomes $\acute{w} =$ *Science*$||j$, where $j \in [1, \infty]$. In another words, if $j = 10$, that means the keyword *'Science10'* has been used 10 times in search operations. Our above solution basically makes the size of the keyword space unlimited, hence, we propose a provably secure scheme called *Public key Encryption with 'Fixed and Short Length Keyword' Search* **(PEKS-FSL)** for an outsourced unstructured database based on Bloom filters (BFs). By using a new algorithm, which will be illustrated in Section 3.3, we can reduce the size of the keyword space and get keywords with a fixed and short length. Further more, the scheme enables the server to participate in the encryption process, thus a data owner could pay less computational cost for encryption, without leaking any information about the plaintext.
Also, we show that our scheme is secure against adaptive chosen-keyword attacks in the random oracle model ROM under the Bilinear Inverse Diffie–Hellman problem (BIDHP).

1.3 Paper Organization

The rest of this paper is organized as follows. Section 2 introduces the preliminaries. Then we provide the outline of the proposed work, the problem formulation, the semantic security and construction of the PEKS-FSL scheme in Section 3. Section 4 gives the security analysis. Section 5 shows its performance and comparisons. Finally, Section 6 introduces the brief conclusions.

2. PRELIMINARIES

2.1 PEKS algorithm

PEKS consists of three parties: the 'sender', the 'receiver', and the 'server'. The sender creates and sends encrypted keywords (PEKS ciphertexts). The server receives PEKS ciphertexts and performs a search after receiving trapdoors from the receiver. The receiver creates the trapdoors and sends them to the server to retrieve the required data.

Definition 1. *A public key encryption with keyword search (PEKS) scheme consists of four polynomial-time algorithms as follows:*

- A Key Generation Algorithm KeyGenReceiver(k): Taking a security parameter k as input, this algorithm creates a private and public key pair (sk_R, pk_R) of the receiver.
- A PEKS Algorithm PEKS(w, pk_R): Taking a keyword w and a receiver's public key pk_R as input, this algorithm produces a PEKS ciphertext S= PEKS(pk_R, w) which is a searchable encryption of w.
- A Trapdoor Generation Algorithm Trapdoor(w, sk_R): Taking a keyword w and a receiver's private key sk_R as input, this algorithm produces a trapdoor Tw for the keyword w.
- A Test Algorithm Test(S, Tw, pk_R): Taking a PEKS ciphertext S = PEKS(pk_R, w'), a trapdoor Tw for a keyword w and a receiver's public key pk_R, this algorithm returns "yes" if $w = w'$ and "no" otherwise.

The authors in Ref.[12] defined a security notion for PEKS schemes indistinguishability of PEKS against chosen keyword attack (IND-CKA).

IND-CKA game:

- KeyGen: The challenger $\mathcal{C}$ calls the KeyGen(k) algorithm to produce (pk_R, sk_R). pk is given to the attacker $\mathcal{AT}$ and sk_R is kept secret from $\mathcal{AT}$.
- Phase 1: The attacker $\mathcal{AT}$ can adaptively ask the challenger $\mathcal{C}$ for the corresponding trapdoor *Tw* for any keyword w of his choice.
- Challenge: At some point, the attacker $\mathcal{AT}$ sends the challenger two words w_0 and w_1 on which it wants to be challenged. The only restriction is that $\mathcal{AT}$ did not previously ask for the trapdoors Tw_0 or Tw_1 . $\mathcal{C}$ picks a random $b \in \{0, 1\}$ and creates PEKS ciphertext C = PEKS(pk, w_b), and returns it to $\mathcal{AT}$ as the challenge PEKS ciphertext.
- Phase 2: $\mathcal{AT}$ can continue to ask for trapdoors Tw for any keyword w of his choice as long as $w \neq w_0, w_1$.
- Guess: Finally, the attacker outputs its guess $b' \in \{0, 1\}$ and wins the game if $b = b'$.
  $\mathcal{AT}$'s advantage in attacking the scheme ϵ is defined as the following function of the security parameter k:

$$|Adv_{\epsilon,A}(k) = |Pr[b = b'] - \frac{1}{2}|.$$

DEFINITION 2. *The PEKS scheme is said to be IND-CKA secure if for any polynomial time attacker $\mathcal{AT}$ we have that $Adv_{\epsilon,A}(k)$ is a negligible function.*

2.2 Bilinear Pairing

DEFINITION 3. *(Bilinear Pairing[19]) Let $\mathcal{G}_1$, $\mathcal{G}_2$ be two groups of order p for some large prime p. Let α and β be elements of Z_p. A bilinear pairing is a map $\hat{e} : \mathcal{G}_1 \times \mathcal{G}_1 \rightarrow \mathcal{G}_2$ between these two group with the following properties:*

1. Bilinear: for all $X, Y \in \mathcal{G}_1$ and $\alpha, \beta \in \mathcal{Z}_p$, $e(\alpha X, \beta Y) = e(X, Y)^{\alpha\beta}$.
2. Non-degenerate: if X is a generator of $\mathcal{G}_1$ then $e(X, X)$ is a generator of $\mathcal{G}_2$.

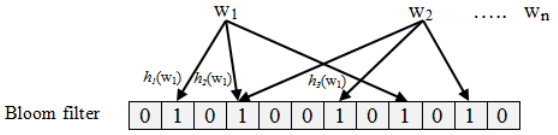

Fig. 1. A Toy Example of Bloom Filter

3. Computable: given $X, Y \in \mathcal{G}_1$ there is an efficient polynomial time algorithms to compute $e(X, Y) \in \mathcal{G}_2$.

We now review the definition of the Bilinear Diffie-Hellman (BDH)[19] problem and the Bilinear Inverse Diffie-Hellman (BDHI)[20] [21] [22] problem associated with the bilinear pairings.

DEFINITION 4. *Bilinear Diffie-Hellman Problem (BDH) Let $\mathcal{G}_1, \mathcal{G}_2$ be two groups of prime order p. Let X be a generator of $\mathcal{G}_1$ and $\hat{e} : \mathcal{G}_1 \times \mathcal{G}_1 \rightarrow \mathcal{G}_2$ be an admissible bilinear map. The BDH problem in $(\mathcal{G}_1, \mathcal{G}_2, \hat{e})$ is as follows: Given $(X, \alpha X, \beta X, \gamma X)$ for some $\alpha, \beta, \gamma \in \mathcal{Z}_p$ compute $W = \hat{e}(X, X)^{\alpha\beta\gamma} \in \mathcal{G}_2$. An algorithm $\mathcal{AT}$ solves BDH problem with the probability ϵ in solving BDH in $(\mathcal{G}_1, \mathcal{G}_2, \hat{e})$ if $Pr[A(X, \alpha X, \beta X, \gamma X) = \hat{e}(X, X)^{\alpha\beta\gamma} \geq \epsilon$*

DEFINITION 5. *Bilinear Inverse Diffie-Hellman (BIDH): Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be a finite cyclic groups of same order p, and X is a generator of $\mathcal{G}_1$. Let $\alpha, \beta \in Z_p$, the BIDH problem is to compute the value of bilinear pairing $e(X, X)^{\alpha^{-1}\beta}$, when given $X, \alpha X, \beta X \in \mathcal{G}_1$.*

2.3 Bloom Filter

The Bloom filter (BF) was proposed by Burton Bloom in the 1970s for database applications, but recently it has been used in many network processing applications. BF is a space-efficient probabilistic data structure based on hash functions. It is used to support membership queries by providing a representation of a set. In BF, the data is treated as a string (e.g. an IP address, a name, an email, etc.) and encoded in the filter by taking r hash functions having digests within the filter size x, and by setting the related positions in the bit array BF. A Bloom filter for representing a set $W = w_1, w_2, ..., w_n$ of n items is described by a bit array of x bits, denoted by BF[1]...BF[x], initially all bits set to 0. BF uses r independent hash functions $h_1, h_2, ..., h_r$ to map each item of W to a random number ranging from 1 to x. To insert an element w of the set W, we set all the BF[$h_i(w)$]$(1 \leq i \leq r)$ bit to 1. Once the set W is represented as a Bloom filter, in order to query whether a data item w belongs to the set W, it suffices to check whether all BF[$h_i(w)$]$(1 \leq i \leq r)$, taken over the considered data, are set to 1 in the filter. If not, w is not in W. If all bits are set to 1, this means that w is a member of W [23].
A simple example is illustrated in Fig. 1, where *r* = 3, w_1 and w_2 are encoded by three hash functions, and the three corresponding items for each word are set to one. Note that a bit of the vector may be set to one multiple times.
In BF, a false positive might happen, because of the collision of the hash function, when a query for a data item not stored in the BF nevertheless 'hits' all bits already set to 1. Suppose that the hash functions are completely random. The the probability of a false positive for a non-member can be computed in a simple way. Let *Pr* be the probability that a random bit of a Bloom filter is 0. After all the n items are hashed into the BF, the probability that a specific bit remain 0 is

$$Pr = (1 - \frac{1}{x})^{r*n} \tag{1}$$

The false positive probability is

$$fp = (1 - Pr)^r \approx (1 - e^{\frac{-rn}{x}})^r \tag{2}$$

The probability of a false positive *fp* can be minimized by choosing the proper values for x and r. It is a well known result that the minimum *fp* is attained for

$$x = \frac{-n \ln(FPR)}{(\ln 2)^2} \tag{3}$$

$$r = \frac{x}{n} ln2 \tag{4}$$

The amount of space required to store a Bloom file is significantly less compared to data structures, such as self-balancing binary search trees, hash tables, or simple arrays or linked lists, etc. The time required to either add elements or to check whether an element is in the set or not is completely independent of the number of elements already in the set. We just need to find the r indices using r hash functions. In hardware implementation, the Bloom filter is regarded as a perfect scheme because its r lookups are independent and can be parallelized.

document id	keywords
1	w_2, w_5, w_7
2	w_1, w_2, w_4, w_6, w_8
...	...
n	w_2, w_5, w_6

Fig. 2. Example of an unencrypted forward index

2.4 Forward Index

Informally, a forward index structure refers to the indexing technique for files where each file's unique id points to a set of keywords contained in that file, as shown in Fig. 2. Hence the search for a keyword using the forward index technique would require sequentially scanning each file and comparing against all unique keywords in each file. This structure requires *O(n*m)* comparisons, where *n* is the number of files and *m* is the number of unique keywords in a file. But one disadvantage of this kind of index is that it leaks some information, e.g. how many keywords are in the index. The applicability of a forward index structure is inefficient in terms of the space needed to store the data and the number of computations needed to perform a search.
In Section (5.2), we will compare the efficiency of the Bloom filter with that of the forward index in terms of the search time required to retrieve the relevant data.

3. SYSTEM DESIGN

3.1 Problem Formulation

In our scheme, we have the data owner *DO*, the data user *DU* and the cloud server *S*. Let *F* be a file collection consisting of

n files, the collection of identifiers of files F where ID_i is a unique file identifier. The data owner *DO* encrypts a collection of files $F = \{f_1, f_2, ..., f_n\}$ and gets the corresponding encrypted files $Enc_F = \{ef_1, ef_2, ..., ef_n\}$. Data owner construct a secure searchable index IDX_{f_i} from a set of *m* different keywords $\mathcal{W}_{f_i} = \{w_1, w_2, ..., w_m\}$ extracted from each file in *F* as a per file index. After that, *DO* encrypts each index using public key encryption and sends it with the encrypted files to *S*.
When *DU* wants to retrieve the file ID_i that has a keyword query *q*, he generates a trapdoor *Tw* for the queried keywords, and submits it to the server *S*.
Upon receiving the trapdoor *Tw* from *DU*, the server participate in the encryption operation by constructing a Bloom filter for each searchable index IDX_{f_i}, then *S* tests the Bloom filter against the trapdoor and retrieves the associated matched file $Enc_{RF} = \{ef_1, ef_2, ..., ef_r\}$ to the *DU*, $Enc_{RF} \subseteq Enc_F$. Finally, *DU* decrypts Enc_{RF} received through the access control mechanism. Note that how to rank the encrypted files is outside the scope of this paper; some excellent work on this problem can be found in [24, 25, 26, 27, 28]

3.2 Semantic Security of the PEKS-SFL scheme

The proposed scheme is semantically secure (indistinguishability) against an adaptive chosen keyword attack IND-CKA if every *PPT* (Probabilistic Polynomial Time) attacker has a negligible advantage. We define the security of the PEKS-SFL scheme as follows:

- Given the security parameter λ, the challenger $\mathcal{B}$ calls the key generation algorithm *KeyGenerator*(λ) to generate a secret key and public key DU_{pub}, then he sends U_{pub} to $\mathcal{AT}$ and keeps the secret key to himself.
- Let $\mathcal{AT}$ be an adversary that can adaptively ask the challenger for the trapdoor *Tw* for any keyword $\mathcal{W} \in \{0,1\}^*$ of its choice.
- Firstly, $\mathcal{AT}$ chooses two sets of keywords $\mathcal{W}_0^* = \mathcal{W}_0||J$ and $\mathcal{W}_1^* = \mathcal{W}_1||J$, which are not to be asked for the trapdoors T_{W_0} or T_{W_1} previously, and sends them to the challenger. Then $\mathcal{B}$ picks a random $\mu \in \{0,1\}$ and creates the secure index IDX_{W_μ} using the *BuildIndex* algorithm and gives the attacker $C_{\mathcal{W}_\mu} = \{DU_{pub}, IDX_{W_\mu^*}\}$. $\mathcal{AT}$ can continue to ask for trapdoors *Tw* for any keyword $\mathcal{W}$ of his choice as long as $\mathcal{W} \neq \mathcal{W}_0^*, \mathcal{W}_1^*$. Finally, $\mathcal{AT}$ outputs a guess $\mu' \in \{0,1\}$ and wins the game if $\mu = \mu'$.

We define $\mathcal{AT}'s$ advantage in breaking the PEKS-SFL scheme as

$$|Adv_{\mathcal{AT}}(\lambda) = |Pr[\mu = \mu'] - \frac{1}{2}|.$$

3.3 PEKS-SFL procedure

As mentioned before, there are some important cases relating to the use of PEKS, which were not considered in [12]. One of these cases is when a server that has received the trapdoors can save them in its memory and may use them to retrieve all future emails within that category. To solve this problem, we propose a scheme of public key encryption with fixed and short length keyword search, PEKS-SFL, to modify repeatedly used keywords by attaching to each one of them the number of its use time; our solution makes the keywords useless for the server to keep trapdoors. On the other hand, our solution makes the size of the keyword space unlimited. Therefore, we propose a new algorithm *FixShoLen(q)* where *q* is the query or corresponding keyword. With its low computational overhead, this algorithm reduces the length of all trapdoors and corresponding keywords to six characters.

Algorithm 1 illustrates our proposed procedure, PEKS-SFL, for making all trapdoors and corresponding keywords a fixed and short length.

Algorithm 1- FixShoLen (q)

- $q = q\|J$, where *J* is the number of use times of the keyword(*q*).
- split *q* into two parts of characters as follows:
  * P_1: this part starts from the first character to $(k/2)^{th}$ character if $k \bmod 2 = 0$, else the part starts from the first character to $(k-1)/2^{th}$ character. Where k is the length of the keyword.
  * P_2: this part starts from $((k/2)+1)^{th}$ character to k^{th} if $k \bmod 2 = 0$, else the part starts from $((k-1)/2)+1^{th}$ character to k^{th} character.
    Split each part into three subparts as follows:
    - $F_{P_{1,2}}$: the first character in each part.
    - $L_{P_{1,2}}$: the last character in each part.
    - $M_{P_{1,2}}$: the other characters (middle characters) in each part. For this part,
  * merge the characters as one character, the merge process includes summing the Ascii code of the mentioned characters
- recombine all parts as: $x = F_{P_1} M_{P_1} L_{P_1} \| F_{P_2} M_{P_2} L_{P_2}$.
- output $x = h(x)$, where h is a cryptographic hash functions like SHA-1.

Our scheme consists of another five algorithms: {a parameter generation algorithm *ParamGenerator*, a key generation algorithm *KeyGenerator*, an index construction algorithm *BuildIdx*, a trapdoor generation algorithm *TrapdoorGen*, and a searchable index algorithm *SearchIdx*}. These are scattered between three phases: the Sender phase, Receiver phase, and Server phase.

3.4 Sender Phase

This phase includes three algorithms as detailed below:
Algorithm 2- ParamGenerator(λ)
Given a security parameter $\lambda \in Z^+$ which determines the size of $\mathcal{G}$ and $\mathcal{G}_T$, the algorithm works as follows:

1: generate a prime *p*, and select a random generator X of $\mathcal{G}$.
2: choose two cyclic groups $(\mathcal{G}, +)$, $(\mathcal{G}_T, .)$ of order *p*, and construct a bilinear map $\hat{e} : \mathcal{G} \times \mathcal{G} \to \mathcal{G}_T$.
4: specify two hash functions $H_1 : \{0,1\}^* \to Z_p^*, H_2 : \mathcal{G}_T \to \{0,1\}^{log_p}$, where H_1 and H_2 are random oracles, and select r hash functions for the bloom filter $H_{bloom} = \{h_1, ..., h_r\}$.
5: return a common parameter *CP*= $\{\mathcal{G}, \mathcal{G}_T, p, X, H_1, H_2, H_{bloom}, \hat{e}\}$,

Algorithm 3- KeyGenerator(CP)
This algorithm randomly selects the following elements:

1: a random $\beta \in Z_p^*$ as a *DU* private key, calculate the corresponding public $DU_{pub} = \beta X$.
2: a random $\alpha \in Z_p^*$ as a *DO* private key, calculate the corresponding public $DO_{pub} = \alpha X$.

Algorithm 4- BuildIdx(CP, F)
The data owner *DO* executes this algorithm to encrypt the keywords $\mathcal{W}_{f_i}$ and create a searchable encrypted index IDX_{f_i} as follows:

1: for each file $f_i \in F$
2: encrypt the ID_i with any semantically secure encryption technique where $i \in [1, n]$
3: compute the length x of the Bloom filter BF_{ID_i} as in equation (3)

4: extract the set of keywords $\mathcal{W}_{f_i}$ from f_i
5: for each keyword $w_j \in \mathcal{W}_{f_i}$ for $j \in [1, m]$ do
6: apply *FixShoLen* algorithm on w_j as: ws_j=*FixShoLen*(w_j)
7: encrypt the modified keyword ws_j under the *DU*'s public key and *DO*'s private key $Enc_{w_j} = \alpha H_1(ws_j)X + \alpha DU_{pub}$
8: Add the encrypted keyword Enc_{w_j} to the $Encw_{f_i}$ list
9: compute $E = H_2(\hat{e}(X, X)^{\alpha})$
10: store $\{Encrypted - ID_i, Encw_{f_i}\}$ in IDX_{f_i}
11: encrypt collection file *F* using a standard symmetric encryption algorithm(AES)
12: send E and the index IDX_{f_i} as the index for f_i.

3.5 Receiver phase

This phase includes one algorithm as detailed below:
Algorithm 5: TrapdoorGen(*q, CP*)
The algorithm is executed by the data user to create a trapdoor as follows:

1: apply *FixShoLen* algorithm on q as: $qw = FixShoLen(q)$
2: compute trapdoor under *DU*'s private key β as: $Tw = (H_1(qw) + \beta)^{-1}X$
3: send the generated trapdoor Tw to the server.

3.6 Server phase

This phase includes one algorithm as detailed below:
Algorithm 6: SearchIdx$(IDX_{f_i}, Encw_{f_i}, Tw, CP, E, x)$: Upon receiving the trapdoor *Tw*, the server *S* executes this algorithm to determine whether a given Index IDX_{f_i} contains the provided keyword as follows:

1: create the Bloom filter BF_{ID_i} of x zero-bits
2: for each encrypted keyword $Enc_{w_j} \in Encw_{f_i}$ for $j \in [1, m]$ do
3: compute $T = H_2(\hat{e}(Enc_{w_j}, Tw))$
4: for $y = 1$ to r do
5: calculate independent hash functions: $b_y = h_y(T)$
6: set $BF_{ID_i}[b_y] = 1$
7: calculate independent hash functions: $h_1(E), h_2(E), ..., h_r(E)$
8: if all r locations of all independent hash functions in BF_{ID_i} are 1, then return the relevant encrypted file to DU
9: decrypt the relevant files Enc_{RF} using (AES) algorithm.

4. SECURITY ANALYSIS

THEOREM 1. *The proposed PEKS-FSL scheme is semantically secure against CKA in the random oracle model under the the Bilinear Inverse Diffie–Hellman Problem (BIDHP).*

PROOF. Let $\mathcal{AT}$ be an attacker that wins the security game (breaking the PEKS-FSL scheme) with an ϵ advantage. Suppose $\mathcal{AT}$ makes q_{H_1} and q_{H_2} hash function queries and q_T trapdoor queries. Suppose further there is an algorithm $\mathcal{B}$ that breaks the BIDH problem with advantage at least ϵ', and suppose $\mathcal{B}$ is given $(p, \mathcal{G}, \mathcal{G}_T, \hat{e}, X, \beta X)$. Then $\mathcal{B}$'s goal is to solve a BIDHP. $\mathcal{B}$ interacts with the forger $\mathcal{AT}$ in the security game as follows:
At any time, algorithm $\mathcal{AT}$ queries the random oracle H_1 or H_2.

- H_1**-queries**: For responding to this type of query, $\mathcal{B}$ maintains H_1, a list $< W_j, \nu_j, L_j >$. The list is initially empty.
When $\mathcal{AT}$ queries W_i from the random oracle H_1, and algorithm $\mathcal{B}$ responds as follows:
if W_i already appears in the list H_1, then $\mathcal{B}$ responds with ν_i. Otherwise, $\mathcal{B}$ generates a random coin $L_i \in \{0, 1\}$. If $L_i = 0$, then $\mathcal{B}$ computes $\nu_i = bX$ for a randomly selected $b \in Z_p^*$; otherwise, $L_i = 1$, $\mathcal{B}$ picks a random element a and computes $\nu_i = a^{-1}X$.
In both cases, $\mathcal{B}$ adds the tuple $< W_i, \nu_i, L_i >$ to the list H_1 and responds with $H_1(W_i) = \nu_i$.
When $\mathcal{AT}$ requests an encryption of keyword $W||J$, Algorithm $\mathcal{B}$ calls the above algorithm to respond to H_1-queries to get $\nu_i \in \mathcal{G}$. Then he searches H_1 for the keyword $W_i||J$. If $L_i = 1$, then $\mathcal{B}$ aborts. Otherwise, $H_1(W_i) = bX$, and then $\mathcal{B}$ computes

$$\begin{aligned} Enc^*_{W_b} &= \alpha H_1(W_i||J)X + \alpha DU_{pub} \\ &= \alpha(bX)X + \alpha DU_{pub} \\ &= \alpha(bX)X + \alpha\beta X \\ &= \alpha X(bX + \beta). \end{aligned}$$

$\mathcal{B}$ can build a searchable index *IDX* by executing the algorithm *BuildIdx(CP, F)*. Then it returns the index *IDX* to $\mathcal{AT}$.

- H_2**-queries**: $\mathcal{B}$ maintains H_2, a list (ϑ, δ). The list is initially empty. At any time, $\mathcal{AT}$ issues a query ϑ to H_2. $\mathcal{B}$ checks whether ϑ appears in H_2 in a pair (ϑ, δ). If not, $\mathcal{B}$ selects a random value δ, adds the pair (ϑ, δ) to H_2, and answers $\mathcal{AT}$ with $H_2(\vartheta) = \delta$.

- **Trapdoor queries:** When $\mathcal{AT}$ issues a query for the trapdoor of the keyword $W_i||J$, $\mathcal{B}$ calls the above algorithm to respond to H_1-queries to get $\nu_i \in \mathcal{G}$, then searches the H_1-list for the query. If $L_i = 1$, then $\mathcal{B}$ aborts. Otherwise, $H_1(W_i||J) = bX$, and then $\mathcal{B}$ computes

$$\begin{aligned} Tw &= (H_1(W_i||J) + \beta)^{-1}X \\ &= (bX + \beta)^{-1}X, \end{aligned}$$

where $\mathcal{B}$ does know β, and answers $\mathcal{AT}$ with *Tw*.

- **Challenge**. Algorithm $\mathcal{AT}$ selects and sends a pair of keywords $\mathcal{W}_0 = \{w_0||J\}$ and $\mathcal{W}_1 = \{w_1||J\}$ to $\mathcal{B}$ on which it wishes to be challenged, and $\mathcal{AT}$ must not have asked previously for the trapdoors of any of the words $\mathcal{W}_0$ or $\mathcal{W}_1$. After receiving $(\mathcal{W}_0, \mathcal{W}_1)$ from the attacker, $\mathcal{B}$ calls the above algorithm twice to respond to H_1-queries to get $(\mathcal{W}_0, \nu_0, L_0)$ and $(\mathcal{W}_1, \nu_1, L_1)$. If both L_0 and L_1 are 0, then $\mathcal{B}$ reports failure and terminates. Otherwise, $\mathcal{B}$ randomly picks $\mu \in \{0, 1\}$ such that $L_\mu = 1$, and then $\mathcal{B}$ responds with the challenge ciphertext $Enc^*_{w_\mu}$ and E^* where $E^* \in \{0, 1\}^{logp}$ $C_\mu = (Enc^*_{w_\mu}, E^*)$. The decryption of the ciphertext is

$$\begin{aligned} Dec^*_{W_\mu} &= H_2(\hat{e}(Tw^*_\mu, Enc_{W^*_\mu})) \\ &= H_2(\hat{e}(bX, a^{-1}X)) \\ &= H_2(\hat{e}(X, X)^{a^{-1}b}) \end{aligned}$$

by the definition $Dec^*_{W_\mu} = E^*$.

$\mathcal{B}$ creates the secure index IDX_{f_μ} by executing the algorithm *BuildIdx*(CP, F) and sends IDX_{f_μ} as a challenge to $\mathcal{AT}$.

- **More queries**. After the above challenge query, $\mathcal{AT}$ can perform additional trapdoor queries, with same restriction that $W_i \neq W_0, W_1$, $\mathcal{B}$ answers these queries as before.

- **Output**. Finally, $\mathcal{AT}$ outputs its guess $\mu' \in \{0, 1\}$ for μ.

To complete the proof of the above theorem, we use the same procedure as in [12] to analyse the probability that $\mathcal{B}$ does not abort during the Trapdoor and Challenge queries. We define the following two events:

- Eve_1: algorithm $\mathcal{B}$ does not abort during the Trapdoor queries.
- Eve_2: algorithm $\mathcal{B}$ does not abort during the Challenge queries.

We suppose that both events Eve_1 and Eve_2 occur with sufficiently high probability. In terms of the first event Eve_1, the probability of Eve_1 is $(1-1/(mq_T+1))^{mq_T} \geq 1/e$, where $1/(q_T+1)$ is the probability that a trapdoor query makes $\mathcal{B}$ abort.
For the second event Eve_2, the algorithm $\mathcal{B}$ does not abort during the challenge phase if one of L_0 and L_1 is 0. By the definition of the H_1-list, $Pr[L_\mu = 0] = 1/(q_T + 1)$ where $\mu \in \{0, 1\}$ and the two values are independent of one another. So, we have that both $Pr[L_0 = L_1 = 1] = 1 - 1/q_T \geq (1 - 1/(q_T + 1))^2$. Hence, $Pr[Eve_2]$ is at least $1/q_T$. Consequently, the probability that $\mathcal{B}$ does not abort during the entire simulation is $Pr[Eve_1 \wedge Eve_2] \geq 1/(eq_T)$.
As a result, if the advantage of $\mathcal{AT}$ against the proposed scheme is ϵ, the probability of success of $\mathcal{B}$ against the BIDH challenge is at least $\epsilon/(e(q_T + 1)$. □

5. PERFORMANCE ANALYSIS

Here, we show the preponderance of our PEKS-FSL scheme from complexity analysis and experimental results.

5.1 Complexity analysis and comparison

In term of time complexity, the PEKS-FSL scheme uses the Bloom filter to greatly reduce the time of a query search in the server side. We analyse the time complexity in two aspects: the data owner side and the server side. In the data owner side, the computation time complexity of both our scheme and the schemes which have been supported with a forward index, like [29, 30], for creating indexes of keywords, are *O(n*m)*, where *n* is the number of files, and *m* is the number of keywords per file. In the server side, as mentioned above, we propose an indexing method that has a single Bloom filter index for each file in the file collection. Our Bloom filter index requires *O(n)* time to search for all files that contain a keyword, while the schemes using a forward index have to test all files' forward indexes. In other words, these schemes require *O(n*m)* time for a search.
In terms of space complexity, our scheme using a Bloom filter requires negligible space compared to the entire data set. For large file sets, our scheme is preferable to a forward index if some false positives are allowed.

5.2 Experimental Evaluation

In this section, we conducted a thorough experimental evaluation of the proposed scheme on a real-world dataset: Request for comments database (RFC) [31]. We evaluate the Bloom filter index used in our scheme and compare its performance with that of other schemes using a forward index, by implementing the search systems using MATLAB on a Windows 8 server with an Intel(R) Core(TM) i5-3032M CPU at 2.60GHz.

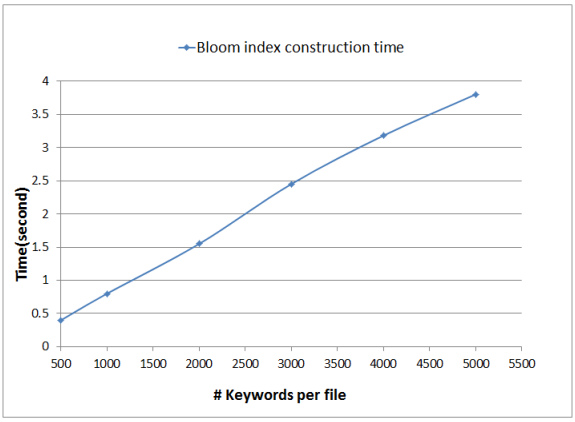

Fig. 3. The Bloom filter and forward index construction time for a single file v.s. # of the keywords

I- Index Construction: Intuitively, the secure index construction operation is a one-time computation. In our scheme, the index construction time take less computational overhead, because the server participates in the index construction operation, where the computation mainly comes from the calculation of the hash function (we can use a BKDR hash function with different seeds to encode the keyword in the filter). Fig. 3 shows the times needed for the index construction operation. The construction time increases linearly with the number of inserted keywords.

II- Search over Encrypted Index: Intuitively, the search operation is executed at the cloud server side. Fig. 4 shows the times spent by a server to perform a search query on the Bloom filter and on the forward index. Obviously, with an increasing number of keywords, the efficiency of a search using a Bloom filter is higher than that using the forward index method. Fig. 5 shows the results of another time metric. In this set of experiments, we set the number of keywords in the index to 1000, and the number of querying keywords varies from 1 to 10. The figure shows the times consumed for the search process using a Bloom filter and using a forward index, versus the number of querying keywords. Additionally, Fig. 6 shows the trapdoor construction time versus the number of querying keywords.

6. CONCLUSION

In this paper, we constructed a new trapdoor, which is modified every time, where an adversary cannot differentiate between two trapdoors even when they come from the same keyword. In other words, such a technique ensures that if the same keyword is encrypted multiple times, it will create different trapdoors, which incorporates the advantages of trapdoor indistinguishability. Further more, the proposed scheme allows the server S to participate in the encryption operation, thus a data owner could pay less computational cost for encryption, without leaking any information about the plaintext. Via a thorough security analysis and experimental evaluation of the use of a Bloom filter versus a forward index, we have shown the suitability and efficiency of our scheme for practical use in a cloud environment.

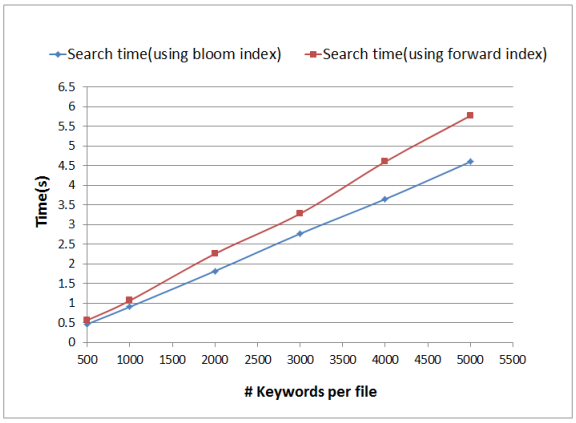

Fig. 4. The time consumed for the search process using Bloom filter and forward index v.s. # of the keywords per file

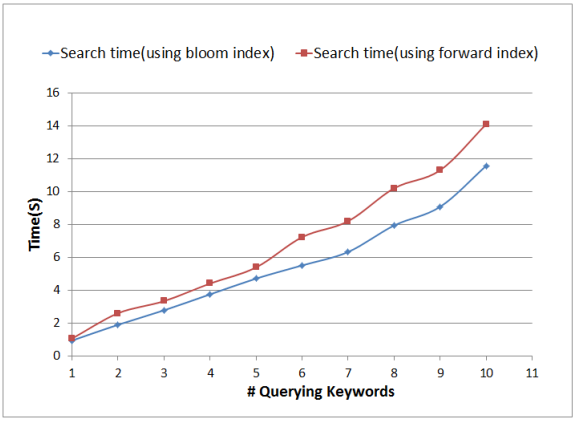

Fig. 5. The time consumed for the search process using Bloom filter and forward index v.s. # of the querying keywords

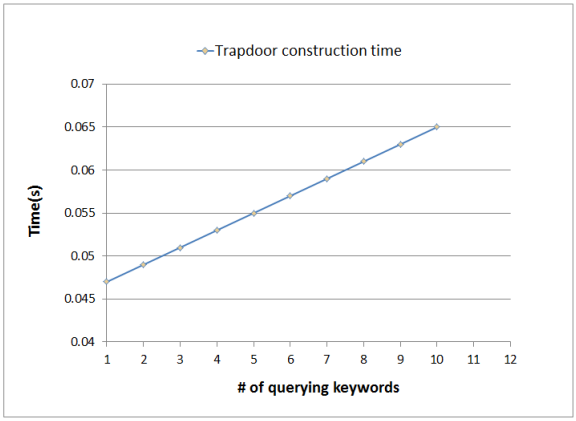

Fig. 6. Trapdoor construction time respect to the number of querying keywords

7. REFERENCES

[1] Armbrust, M., Fox, A., Griffith, R., Joseph, AD. , Katz, RH., Konwinski, A. , et al. 2009. Above the clouds: a berkeley view of cloud computing [Technical report], In: Berkeley: EECS Department, University of California, [Tech. Rep. UCB/EECS-2009-28].

[2] Qian, L., Luo, Z., Du, Y. and Guo, L. 2009. Cloud computing: an over view, in: Proceedings of the 1st International Conference on Cloud Computing Cloud Com' 09, Springer- Verlag, pp. 626-631.

[3] Rimal, B. , Choi, E. and Lumb, I. 2009. A taxonomy and survey of cloud computing systems, in: IEEE Fifth International Joint Conference on INC, IMS and IDC, pp. 44-51.

[4] Zhang, Q., Cheng, L. and Boutaba, R. 2010. Cloud computing: state-of-the-art and research challenges, in: Journal of Internet Services and Applications 1, pp. 7-18.

[5] Stanoevska-Slabeva, K. and Wozniak, T. 2010. Grid and Cloud Computing-A Business Perspective on Technology and Applications, in: Springer-Verlag, Berlin, Heidelberg.

[6] Curtmola, R., Garay, J., Kamara, S. and Ostrovsky R. 2006. Searchable symmetric encryption: Improved definitions and efficient constructions, in: Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS06), pp. 79-88.

[7] Chang, Y.-C. and Mitzenmacher, M. 2005. Privacy preserving keyword searches on remote encrypted data, in: Proc. of ACNS05.

[8] Goh, E-J. 2003. Secure indexes, Technical Report 2003/216, in: IACR ePrint Cryptography Archive. See http://eprint.iacr.org/2003/216.

[9] Song, D., Wagner, D. and Perrig, A. 2000. Practical techniques for searches on encrypted data, in: Proc. IEEE International Symposium on Security and Privacy (S and P00), Nagoya, Japan, Jan. pp. 44-55.

[10] Goldreich, O. and Ostrovsky, R. 1996. Software protection and simulation on oblivious RAMs, in: Journal of the ACM, 43(3):pp. 431-473.

[11] Ostrovsky, R. 1992. Software protection and simulations on oblivious RAMs, in: Proceedings of 22nd Annual ACM Symposium on Theory of Computing, 1990. MIT Ph.D.Thesis.

[12] Boneh, D., Crescenzo, G. Di, Ostrovsky, R. and Persiano G. 2004. Public key encryption with keyword search, in: Proc. EUROCRYPT 04, pp. 506-522.

[13] Hwang, Y. H. and Lee, P. J. 2007. Public Key Encryption with Conjunctive Keyword Search and Its Extension to a Multiuser System, Lecture Notes in Computer Science, Volume 4575/2007, pp. 2-22.

[14] Boneh, D., Kushilevitz, E., Ostrovsky, R. and Skeith W. 2006. Public-key encryption that allows PIR queries, in: Unpublished Manuscript.

[15] Wang, J., Chena, X. and Lic, J. 2016. Zhaod J. and Shene J., Towards achieving flexible and verifiable search for outsourced database in cloud computing. Future Generation Computer Systems.

[16] Golle, P., Staddon, J. and Waters, B. 2004. Secure conjunctive keyword search over encrypted data. In: Jakobsson, M., Yung, M. (eds.) ACNS 2004. LNCS.Springer, Heidelberg. 3089: pp. 31-45.

[17] Boneh, D. and Waters, B. 2007. Conjunctive, subset, and range queries on encrypted data. In: Vadhan,S.P. (ed.) TCC 2007. LNCS, Vol. 4392, pp. 535-554. Springer, Heidelberg.

[18] Baek, J., Safavi Naini, R. and Susilo, W. 2008. Public key encryption with keyword search revisited. In: Gervasi, O.,Murgante, B., LaganRa, A., Taniar, D., Mun, Y., Gavrilova, M.L. (eds.) ICCSA (1). Lecture Notes in Computer Science, Vol. 5072, pp. 1249-1259. Springer.

[19] Boneh, D. and Franklin M. 2001. Identity-Based Encryption from the Weil Pairing, In CRYPTO 2001,LNCS 2139, pp. 213-229, Springer-Verlag.

[20] Boneh, D. and Boyen, X. 2004. Efficient Selective ID Secure Identity Based Encryption without Random Oracles. Advances In Cryptology-Eurocrypt 2004, LNCS 3027, pp. 223-238, Springer-Verlag.

[21] Zhang, F., Safavi-Naini, R. and Susilo, W. 2004. An efficient signature scheme from bilinear pairings and its applications. In PKC' 2004, LNCS 2947, pp. 277-290. Springer-Verlag.

[22] Gu, C. and Zhu, Y. 2010. New efficient Searchable Encryption Schemes from Bilinear Pairings. International Journal of Network Security. 2010, Vol.10, No.1, pp. 25-31.

[23] Bloom, B. H. 1970. Space/time trade-offs in Hash Coding with Allowable Errors, in: Communications of the ACM, http:// portal.acm.org/ citation.cfm? doid=362686.362692,Vol. 13, Issue 7.

[24] Cao, N., Wang, C. ,Li M. , Ren, K. and Lou, W. 2014. Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data, in: IEEE transactions on parallel and distributed systems, Vol. 25, No. 1.

[25] Wang, C., Cao, N., Li, J., Ren, K. and Lou, W. 2010. Secure Ranked Keyword Search over Encrypted Cloud Data, in: International Conference on Distributed Computing Systems. pp. 253-262.

[26] Swaminathan, A., Mao, Y., Su, G.-M. and Gou, H. 2007. Varna, A., L., He, S., Wu, M., Oard, D., W., Confidentiality-preserving rank-ordered search, in: Proceedings of the 2007 ACM Workshop on Storage security and Survivability (StorageSS07). New York, NY, USA: ACM, pp. 7-12.

[27] Wang, B., Song, W., Lou, W. and Hou, Y., T. 2015. Inverted Index Based Multi-Keyword Public-key Searchable Encryption with Strong Privacy Guarantee, in: INFOCOM, 2015 Proceedings IEEE, pp. 2092-2100.

[28] Sun, W., Wang, B., Cao, N., Li, M., Lou, W., Hou, Y. T. and Li, H. 2013. Privacy preserving multi-keyword text search in the cloud supporting similaritybased ranking, in: Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, ser. ASIA CCS13. New York, NY, USA: ACM, pp. 71-82.

[29] Orencik, C. and Savas, E. 2014. An efficient privacy-preserving multi keyword search over encrypted cloud data with ranking, in Springer Distributed and Parallel Databases, pp. 119-160.

[30] Kapase, J., G. and Shinde, S., M. 2014. User-Friendly Keyword Based Search on XML Data. International Journal of Science and Research (IJSR), Vol. 3, Issue 6, pp. 2751-2756.

[31] RFC, Request For Comments Database, http://www.ietf.org/rfc.html.