ISSN 2277-3061
Volume 15 Number 14
INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY

Public Key Encryption with Conjunctive Field Free Keyword Search Scheme

Fairouz Sher Ali[1,2] , Song Feng Lu[1]
[1]School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074, P.R.China
[2]Kufa University, Kufa, Iraq
Fairouz_sherali@yahoo.com
lusongfeng@hust.edu.cn

ABSTRACT

Searchable encryption allows a remote server to search over encrypted documents without knowing the sensitive data contents. Prior searchable symmetric encryption schemes focus on single keyword search. Conjunctive Keyword Searches (CKS) schemes improve system usability by retrieving the matched documents. In this type of search, the user has to repeatedly perform the search protocol for many times. Most of existent (CKS) schemes use conjunctive keyword searches with fixed position keyword fields; this type of search is not useful for many applications, such as unstructured text. In our paper, we propose a new public key encryption scheme based on bilinear pairings, the scheme supports conjunctive keyword search queries on encrypted data without needing to specify the positions of the keywords where the keywords can be in any arbitrary order. Instead of giving the server one trapdoor for each keyword in the conjunction set, we use a bilinear map per a set of combined keywords to make them regarded as one keyword. In another meaning, the proposed method will retrieve the data in one round of communication between the user and server. Furthermore, the search process could not reveal any information about the number of keywords in the query expression. Through analysis section we determine how such scheme could be used to guarantee fast and secure access to the database.

Indexing terms/Keywords

Searchable Encryption; Public Key Encryption; Conjunctive Keyword Search; Keyword Field Free; Bloom Filter.

Academic Discipline And Sub-Disciplines

Computer Science, Computer Networks, Cloud Computing

SUBJECT CLASSIFICATION

Information Security, Cryptography

TYPE (METHOD/APPROACH)

A novel public key encryption scheme based on bilinear pairings

1. INTRODUCTION

Cloud computing has become the most common phenomenon in the recent years. More and more cloud services have flourished all around the world such as computing resource, storage space outsourcing and different kinds of software applications. For many reasons like low cost, efficiency, convenience, better connectivity and etc., user often stores his data on a remote server. Since more servers are public, there exist a lot of risks for the data in the transition process, the user ensures the privacy of his data by storing it in encrypted form, and then he can search the encrypted data and retrieve it. The first scheme of searching encrypted data by keyword was tackled by Song et al. [1]. To securely search through encrypted data, searchable encryption schemes have been introduced in recent years [2,3,4,5,6,7,8], which can be divided into two schemes: symmetric searchable encryption (SSE) and asymmetric searchable encryption (ASE). To perform a search on a dataset, an user creates an index of keywords listed in the documents and later on executes the search on the index in a way that allows the server to retrieve the documents contain a certain keyword instead of retrieving all the encrypted documents back which is fully impractical solution in cloud computing scenarios. Recent refinements and extensions to this scheme are given in[7,8].

The drawback of all the follow-up works is that they only allow the remote server to retrieve the documents that match a specific keyword, but they do not allow for Boolean combinations, conjunctive and disjunctive, of such queries.

Most classical searchable encryption works focus only on single keyword search [4,5,6,7,1] or multiple keyword search [9,10,11,12,13]. In the symmetric key schemes, recently some solutions have been introduced for general Boolean queries on encrypted files [14,15], and there are only two related works in the public key setting [16,17].

There are many Boolean operations, like disjunction, conjunction and negation. In the disjunctive search, the user can search for encrypted documents containing: w_1 or w_2 or w_n. While in the conjunctive search, the user can search for the encrypted documents containing: w_1 and w_2 and w_n and finally in the negative search, the user can search for all encrypted documents which do not contain particular words.

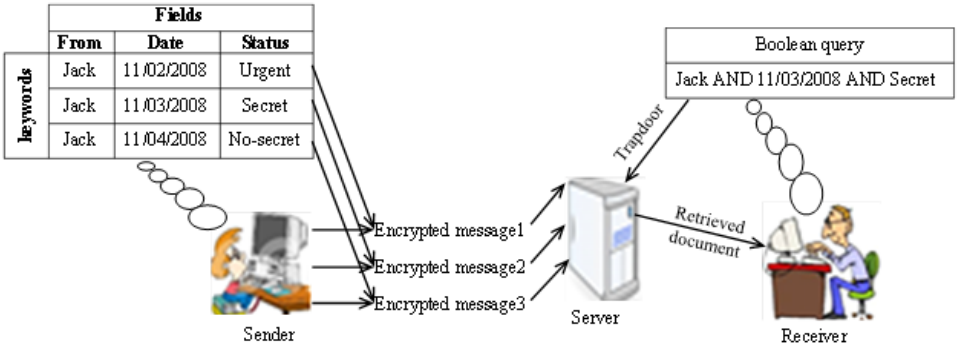

Fig 1: An example of searchable email system

To support multiple Boolean encrypted keywords, such as conjunction operation, we consider a mail server, shown in Figure 1, which retrieves a stream of email encrypted messages, each email will be defined some keyword fields, like "From", "Date" and "Status". Before sending the message, the sender, for example Jack, should encrypt the message content by using a public key encryption algorithm with the recipient's public key, and then adds some additional encrypted keywords of the above keyword fields, like "Jack", "11/02/2008" and "secret". When the recipient wants to retrieve the encrypted messages which are sent by "Jack" at "11/02/2008" and having "secret" status, rather than retrieving all messages from "Jack", he sends a "trapdoor" with multi keywords "Jack" AND "11/02/2008" AND "secret" to the mail server which in turn routes the corresponding encrypted emails to receiver without learning any information.

Existent schemes for conjunctive keywords search ([11] and subsequent works) were supporting keyword fields in the index. This setting is not useful and much more difficult to search in most systems, such as the database text and the body of e-mail.

Despite the efficiency of Public-key Encryption with Keyword Search scheme PEKS[4], there are some important cases relating the use of PEKS, which were studied in[18]. One of these cases is that the scheme did not support the notion of the multiple keywords search.

Our proposed solution to solve the above problem is to define a secure scheme of public key encryption with keyword field free conjunctive keyword searches (PKE-KFF-CKS) that allows conjunctive keyword search queries on encrypted data without needing to specify the positions of the keywords (hide the keywords positions from the querier) where the keywords can be in any arbitrary sequence. Furthermore, instead of giving the server one trapdoor for each keyword in the conjunction, we combine individual keywords to make them regarded as one keyword, this can be done using the template concatenation function $w_1||w_2||\ldots||w_m$ without needing for conjunctive search mark $\wedge$, the cloud server cannot know the number of keywords, in other meaning if the users want to retrieve the documents that contain a set of keyword, they have not to repeatedly perform the search protocol for m keywords times. Also, we show that our scheme is secure against adaptive chosen-keyword attacks in the random oracle model ROM under the Bilinear Diffie Hellman assumption.

1.1. Main Contributions

Our main contributions can be summarized as:

(1) Our scheme dealing with keyword field-free conjunctive keyword searches, we design a novel algorithm that converts the conjunctive keywords search to a single keyword search and consequently the model cannot support the posting list intersection protocol. With this new scheme, we can greatly reduce the search time and the storage cost of the searchable index.

(2) Creating Indistinguishability-Chosen Keyword Attack (IND-CKA) secure index using a bloom filter for each file in a collection of files.

(3) Security of our scheme based on the Bilinear Diffie-Hellman assumption.

1.2. Previous Work

Song, Wagner, and Perrig [1] first proposed the notion of searchable encryption for a single-user. They introduced a scheme in the symmetric key setting, which encrypts each word of a document separately. Goh [7] proposed a method for secure index using the Bloom filters. Each keyword is processed using the keyed hash function f as the pseudo-random function and then inserted into a Bloom filter. The trapdoor consists of an indicator of that which bits in the Bloom filter should be tested. In the public key setting, Boneh et al.[4] first proposed public key module for keyword search, where anyone can use public key and write to the data stored on remote server, but only authorized users with the secret key can search. Furthermore, the keyword security could not be protected in the public key setting since remote server could encrypt any keyword with public key and then use the received trapdoor to evaluate this ciphertext. However, these above approaches focus only on single keyword search. To improve search functionalities, many boolean keyword search schemes over encrypted data have been proposed. Obviously, there are two naive solutions to achieve conjunctive keyword search: the first is to get the intersection of all sets of documents where each set is the searching result for every keyword in the conjunctive; the second is to define a meta-keyword for every possible keywords conjunction. The first

work for conjunctive keyword-searchable encryption was proposed by Golle et al.[11], their works consisted of two schemes: the first scheme compares two hash codes of the keywords to find the required documents, the transmission cost of the trapdoors is very high. The second scheme tests two outputs of bilinear pairing constructed from input keywords and checks if the keywords are included in the document. Boneh and Waters [19] introduced a public key CKS scheme from a generalization of anonymous identity based encryption. Their paper supports comparison queries and general subset queries. Byun et al. [20] presented an efficient scheme using bilinear pairings, which has a constant size of trapdoors and requires two pairing operations per document for searching. The scheme is more efficient than both schemes by Golle et al.[11] in terms of communication overhead, but it has higher computational overhead for the encryption process of each document by requiring as many pairing operations as the number of the associated total keywords. Ryu and Takagi [21] introduced an efficient scheme for conjunctive keyword searches where the size of the trapdoors for several keywords is nearly the same as for a single keyword. They use asymmetric pairings in groups of prime order. The encryption process requires one pairing per document and the server has to perform two pairings per document to search. Hwang and Lee [12] introduced a public key encryption scheme with conjunctive keyword search (PECK) and gave a new concept called multiuser PECKS. The notion of their scheme is to minimize the communication and storage overhead for the remote server and also for the user.

Recently Wang et al. [22] proposed the first keyword-field-free conjunctive keyword search scheme KFF-CKS for dynamic groups that is proven secure in the ST model. The notion is to remove the keyword fields by using a bilinear map per keyword per document index.

1.3. Security Requirements

1. Data security [23]: when the data owners encrypt the keywords and the message using the authorized user's public key, only the corresponding secret key can decrypt the content, that mean no one could derive the embedded keywords from the cipher-text.
2. User authentication: After encrypting, no information can be extracted from the trapdoor and the ciphertexts, but the remote server still has to check whether the users who send the trapdoor are the authorized users. [24,25,26].
3. Trapdoor security [23]: Whenever the receiver wants to search the encrypted data, he sends the trapdoor containing the corresponding keywords to the remote server; other users can get nothing from the trapdoor even if the trapdoors are obtained by the adversaries.
4. Against off-line keyword-guessing attack: any proposed security scheme should stand against outside adversaries and inside attackers (malicious servers) [10,12].

1.4. Outline

The rest of the paper is organized as follows. Section 2 introduces the preliminaries. Then we provide the outline of the proposed work, notations, semantic security of the PKE-KFF-CKS scheme and construction of PKE-KFF-CKS in Section 3. Section 4 gives the security analysis, performance and comparisons. Finally, Section 5 introduces the brief conclusions.

2. PRELIMINARIES

2.1. The Bilinear Pairings and Complexity Assumptions

We briefly show theoretical background and complexity assumptions that used throughout our paper.

(1) Bilinear maps: Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of prime order q. $\hat{e}: \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ be a map which satisfies the following properties:

- Bilinear: for all $h \in \mathbb{G}_1$and , $\beta \in Z_q$, $\hat{e}(h^{\alpha}, h^{\beta}) = \hat{e}(h,h)^{\alpha\beta}$
- Non-degenerate: there exist $h \in \mathbb{G}_1$ such that $\hat{e}(h,h) \neq 1$, where 1 is the identity of $\mathbb{G}_2$.
- Computable: for all $g \in \mathbb{G}_1, (g,g)$ is computable in polynomial time.

(2) Bilinear Diffie-Hellman (BDH) problem: given $h, h^{\alpha}, h^{\beta}, h^{\gamma} \in \mathbb{G}_1$, output $\hat{e}(h,h)^{\alpha\beta\gamma} \in \mathbb{G}_2$.

An algorithm $\mathcal{A}$ solves BDH problem with the probability ϵ if Pr[$\mathcal{A}$ $(h, h^{\alpha}, h^{\beta}, h^{\gamma})$ =$\hat{e}(h,h)^{\alpha\beta\gamma}$]$\geq \epsilon$, where the probability is over the random choice of generator $h \in \mathbb{G}_1^*$, the random choice of $\alpha, \beta, \gamma \in Z_q^*$ and random coins consumed by $\mathcal{A}$.

2.2. Outline of the Conjunctive Keyword Searchable Encryption [11].

A conjunctive keyword searchable encryption (CKSE) consists of the following four algorithms:

- KeyGen(*k*): It is run by the data owner to initiate the scheme. It takes a security parameter *k*, and returns a secret key *SK*.
- Enc(*SK*,D_i): It is run by the data owner to create searchable ciphertexts. It takes a secret key *SK* and a document D_i= {$W_{i,1}$, ...,$W_{i,m}$} as inputs, and returns a ciphertext C_i which is a conjunctive keyword searchable encryption of D_i.

- Trapdoor(SK,{$j_1,\ldots, j_l$},{$W_{j1},\ldots,W_{jl}$}): It is run by the data owner to create a trapdoor for given keywords. It takes a secret key SK, keyword field indices $j_1,\ldots, j_l$ ($1 \le l \le m$) and l keywords $W_{j1}, \ldots, W_{jl}$ as inputs, and returns a trapdoor value T.
- Test(T,C_i): It is run by the remote server in order to search for the documents containing some specific keywords. It takes a trapdoor T=Trapdoor(SK,{$j_{1,\ldots,} j_l$}, {$W_{j_1},\ldots,W_{j_l}$}) and a ciphertext C_i = Enc(SK, D_i) as inputs, and returns true if the condition (($W_{i,j1}$= W_{j1})∧($W_{i,j2}$= W_{j2}) ∧…∧ ($W_{i,jl}$= W_{jl})) holds and false otherwise.

2.3. Outline of the PEKS scheme [4].

A public key encryption with keyword search (PEKS) scheme consists of the following algorithms:

(1) KeyGen(λ): Takes a security item λ as input, and creates a public/private key (R_{pub}, R_{priv}) for the receiver.

(2) PEKS(R_{pub},W) : Given Receiver's public key R_{pub} and a word W, computes a searchable encryption S for W.

(3) Trapdoor(R_{priv},W): Given Receiver's private key R_{priv} and a keyword W, computes a trapdoor T_W for W.

(4) Test(R_{pub},S,T_W): Given Receiver's public key R_{pub}, a searchable encryption S=PEKS(R_{pub},W'), and a trapdoor T_W Trapdoor (R_{priv},W), outputs 'yes' if $W = W'$ and 'no' otherwise.

IND-CKA game:

- *KeyGen*: The challenger $\mathcal{C}$ runs the KeyGen(λ) algorithm to create the public key pk and the secret key sk. He gives pk to the attacker, while sk is kept secret from him.
- *Phase 1*: $\mathcal{A}$ can adaptively ask $\mathcal{C}$ for the trapdoor T_W for any keyword $W \in \{0,1\}^*$ of his choice.
- *Challenge*: At some point, $\mathcal{A}$ sends $\mathcal{C}$ two words W_0,W_1 on which it wishes to be challenged. The only restriction is that $\mathcal{A}$ did not previously ask for the trapdoors T_{W0} or T_{W1}. $\mathcal{C}$ picks a random $b \in \{0,1\}$ and gives the attacker $\mathbb{C}$=PEKS(pk,W_b) as the challenge PEKS ciphertext.
- *Phase 2*: The attacker continues to ask for trapdoors T_W for any keyword W of his choice as long as W , W_0,W_1.
- *Response*: Finally, $\mathcal{A}$ outputs $b' \in \{0,1\}$ and wins the game if $b = b'$.

Such an adversary $\mathcal{A}$ is called an IND-CKA adversary. $\mathcal{A}$'s advantage in attacking the scheme ϵ is defined as the following function of the security parameterλ:

$$|Adv_{\epsilon,\mathcal{A}}(\lambda) = |\Pr[b = b'] - 1/2|$$

The probability is over the random bits used by the challenger and the adversary. A PKES scheme ϵ is IND-CKA secure if for any polynomially time adversary, $Adv_{\epsilon,\mathcal{A}}(\lambda)$ is negligible.

2.4. Bloom Filter *BF*

Bloom filter is a space-efficient data structure which is used to check whether an element is a member of a set. Burton H. Bloom [27] introduced this data structure in 1970. *BF* is used to test whether an element s is a member of a set F = ($w_1,\ldots,w_n$). The set F is coded as an array *BF* of x bits, where all bits are initially set to 0. The filter uses r independent hash functions $h_{1,\ldots,}h_r$, to map items into a domain between 0 and x-1. For each element $w_i \in F$ where $1 \le i \le n$, the array bits at the positions $h_1(w_i),\ldots,h_r(w_i)$ are set to 1. Note that, a location may be set to 1 multiple times. The elements themselves are not stored in *BF*, only their membership may be queried by an application. To determine if a word $s \in F$, we check whether the bits at positions $h_1(s),\ldots, h_r(s)$ in *BF* are all 1. If any bit is 0, then s ∉ F. Otherwise, we say ∈ F with high probability.

A false positive is possible which can be controlled by changing the filter length x as follows:

$$x = \frac{-n \ln(FPR)}{(ln\ 2)^2} \qquad (1)$$

where n is the number of elements, *FPR* is the user defined False Positive Rate, (*FPR*) can be approximated as:

$$FPR = (1 - e^{\frac{-rn}{x}})^r$$

False positive matches are possibly occurred, but false negatives are not, thus a Bloom filter has a 100% recall rate.

The amount of space required to store bloom file is significant less compared to data structures, such as self-balancing binary search trees, hash tables, or simple arrays or linked lists, etc.

The time required either to add elements or to check whether an element is in the set or not is a completely independent of the number of elements already in the set. We just need to find the r indexes using r hash functions. In a hardware implementation, the Bloom filter regards as a perfect scheme because its r lookups are independent and can be parallelized.

3. OUTLINE OF THE PROPOSED SCHEME

3.1. Notations

- *D*: the collection of *n* plaintext document to be outsourced, denoted as $D = \{D_1, D_2, \ldots, D_n\}$.
- *ID*: the collection of *n* documents identifiers, denoted as $ID = \{ID_1, ID_2, \ldots, ID_n\}$.
- *EncDoc*: the collection of *n* encrypted data documents stored in the remote server, denoted as $EncDoc = \{Enc_{D_1}, Enc_{D_2}, \ldots, Enc_{D_n}\}$.
- *DecDoc*: the collection of *n* decrypted data documents stored in the remote server, denoted as $DecDoc = \{Dec_{D_1}, Dec_{D_2}, \ldots, Dec_{D_n}\}$.
- Enc_D: the collection of *k* retrieved documents from the remote server contained the conjunctive keyword, denoted as $Enc_D = \{Enc_{D_1}, Enc_{D_2}, \ldots, Enc_{D_k}\}$.
- W_D: the collection of *s* distinct keywords extracted from each document D_i in collection *D*, denoted as $W_D = \{w_1, w_2, \ldots, w_s\}$.
- W_{Di}: the collection of *m* distinct keywords per trapdoor extracted from each document D_i in collection *D*, denoted as $W_{Di} = \{w_1, w_2, \ldots, w_m\}$.
- *P*: the collection of possible permutation extracted from keywords sequence W_{Di}, denoted as $P = \{per_1, per_2, \ldots, per_{m!}\}$.
- Per_j: the collection of *m* keywords regards as one keyword using concatenation operation, denoted as $per_j = fw_1||w_2||\ldots||w_m\}$, $j = 1 \ldots m!$.
- *Q*: the collection of *l* keywords in a search request, denoted as $Q = \{q_1, q_2, \ldots, q_l\}$.
- *Tq*: the trapdoor for *l* conjunctive queried keywords denoted as $Tq = \{q_1||q_{2||\ldots}||q_l\}$.
- I_D: the collection of *n* indexes I_{D_i}, denoted as $I_D = \{I_{D_1}, I_{D_2}, \ldots, I_{D_k}\}$.

3.2. Semantic Security of the PKE-KFF-CKS 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. PKE-KFF-CKS consists of two public key encryption algorithms, i.e., algorithms *BuildIndex* and *DocEncrypt*, where *BuildIndex* algorithm closely follows the PEKS algorithm. Therefore, we define security for the PKE-KFF-CKS scheme in the sense of semantic security of [4] as follows:

Given the security parameter(λ), the challenger $\mathcal{C}$ calls the key generation algorithm *KeyGenerator*(λ) to generate secret key U_{sk} and public key U_{pub}, then he sends U_{pub} to $\mathcal{A}$ and keeps U_{sk} to itself. Let $\mathcal{A}$ be an adversary that can adaptively ask the challenger for the trapdoor T_W for any keyword $W \in \{0,1\}^*$ of it's choice, where $W = \{w_1||w_2||\ldots||w_s\}$. Firstly, $\mathcal{A}$ chooses two sets of conjunctive words $W_0 = \{w_{01}||w_{02}||\ldots||w_{0s}\}$ and $W_1 = \{w_{11}||w_{12}||\ldots||w_{1s}\}$, which are not to be asked for the trapdoors T_{W_0} or T_{W_1} previously, and sends them to the challenger. Then $\mathcal{C}$ picks a random $\mu \in \{0,1\}$ and creates the secure index I_{w_μ} using the *BuildIndex* algorithm and gives the attacker $\mathbb{C}_{w_\mu} = \{U_{pub}, I_{w_\mu}\}$. $\mathcal{A}$ can continue to ask for trapdoors T_W for any keyword $W = \{w_1||w_2||\ldots||w_s\}$ of his choice as long as $W \neq W_0, W_1$. Finally, $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$ and wins the game if $= \mu'$.

We define $\mathcal{A}$'s advantage in breaking the PKE-KFF-CKS scheme as:

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

3.3. Construction of PKE-KFF-CKS

Contrary to Golle et al. scheme[11] we do not target the fixed field keyword; we rather consider an enhanced query model consisting of Boolean expression on keywords expressed in the conjunctive form without needing to specify the positions of the keywords where the keywords can be in any arbitrary order.

In our model, we have the data owner *O*, the data user *U* and the cloud server *S*. Let *D* be a document collection consisting of n documents, where ID_i is a unique document identifier. *O* extracts *m* keywords from each document D_i as $W = \{w_1, w_2, \ldots, w_m\}$ and combines them as one keyword with different *m*! possible permutations $P = \{per_1, per_2, \ldots, per_{m!}\}$, where each permutation set per_j has *m* combined keywords, $Per_j = \{w_1||w_2||\ldots||w_m\}$ where $j \in [1, m!]$. For example, if $m = 3$ keywords, and the keywords are *A,B,C*. The Owner creates 6 different permutations of keywords sequence and each permutation, consists of three keywords, regards as one keyword using concatenation operation $P_{ABC} = \{(A||B||C); (A||C||B); (B||A||C); (B||C||A); (C||A||B); (C||B||A)\}$. Based on Goh et al. [7], we use a Bloom Filter as a per document index to track the conjunctive keyword in each document, each bloom filter represents a set of *m*! possible permutations *P* of *m* keywords sequence. This is best shown by the toy example illustrated in Figure 2, where 6 permutations are actually stored in a Bloom filter with 3 hash functions. After that *O* encrypts each bloom filter and sends it with the encrypted documents to *S*.

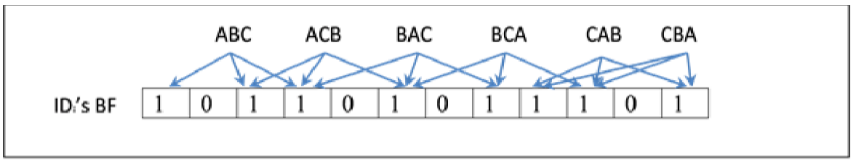

Fig 2: A Toy Example of Bloom Filter

When the data user wants to retrieve the document ID_i that has the following keywords (*A* and *B* and *C*), he can create a trapdoor in arbitrary order as one search token, that's mean he can send one of the following combined keywords (*A*||*B*||*C*), (*A*||*C*||*B*), (*B*||*A*||*C*), (*B*||*C*||*A*), (*C*||*A*||*B*) or (*C*||*B*||*A*) as a query to the remote server. Then the server tests the Bloom filter against the trapdoor and retrieves the associated matched document to the *U* without needing for the posting list intersection protocol.

Our scheme consists of six algorithms *KeyGenerator, BuildIndex, DocEncrypt, TrapdoorGen, SearchIndex* and *DocDecrypt* which are scattered between two phases, Sender Phase and Retrieval Phase.

3.3.1. Sender Phase

This phase includes three algorithms as detailed below:

1: **Key generator**: The data owner *O* initiates the scheme by using KeyGenerator(λ) algorithm. This algorithm takes the security parameter λ as input to obtain the public parameters $PP=\{U_{pub},O_{pub},V,q,g,\hat{e},H,H_1,H_2,H_3\}$ and the private keys $PR=\{U_{pr},O_{pr}\}$.

2: **Index construction**: For each document $D_i \in D$, *O* dedicates a secure index I_{D_i}, which is stored at the service provider that will help *O* perform a keyword search by calling $BuildIndex(D_i,W_{D_i},PP,O_{pr})$. Each document D_i comprising of an unique identifier $ID_i \in \{0,1\}^n$. Firstly, to protect the document Identifiers ID_i, *O* encrypts this ID_i with El Gamal encryption technique, such technique assurances that if the same document identifier is encrypted multiple times, it will create different ciphertexts but all decrypted to the same value. Then *O* creates one Bloom filter *BF* for each document, this filter consists of an array of *x*-bits, and uses *r* independent hash functions $h_1,\ldots,h_r$. The filter allows the data owner to perform keyword searches efficiently, but could result in some false positive retrieval. A classical Bloom Filter may reveal information about the contents of the document since the hash functions are publicly known. So, in our work a suitable solution to create a searchable index using Bloom Filter is to instead index each conjunctive keyword by its encrypted image. To do so, we apply bilinear maps on elliptic curves, we use two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order *q* and a bilinear map $\hat{e}: \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$, also we need three hash functions $H_1: \{0,1\}^* \to \mathbb{G}_1$, $H_2: \mathbb{G}_2 \to \{0,1\}^{\log q}$ and $H_3: \mathbb{G}_2 \to \{0,1\}^n$. The owner creates *m*! possible permutations of these keywords sequence $P = \{per_1, per_2,\ldots, per_{m!}\}$ and makes each permutation per_j looks like one keyword using concatenation operation as $per_j = \{w_1||w_2||\ldots||w_m\}$ where $j \in [1,m!]$. We use such a bilinear map with each permutation per_j as $Enc_{per_j} = H_2(\hat{e}(U_{pub}, H_1(per_j)^a))$. Then the BF_{ID_i} will be constructed using the hash values on the conjunctive string $h_z(Enc_{per_j})$, $z = 1,\ldots, r$, instead of applying the hash values on per_j directly. After that the array bits at the positions $h_1(Enc_{per_j}),\ldots, h_r(Enc_{per_j})$ are set to 1. Finally, *O* stores the encrypted *ID* Enc_{ID_i} and associated bloom filter BF_{ID_i} in I_{D_i}.

In Bloom Filter, the number of 1s is reliant on the number of *BF* entries, in this case, the number of different permutations. As a consequence, the scheme reveals the number of keywords in each document. To avoid this problem, padding number of dummy keywords may be used to make sure that the number of 1s in the Bloom Filter is nearly the same for various documents. Padding process is costly compared to the scheme without it because the higher rate of false-positive.

3: **Document collection encryption**: To protect data privacy and undesired accesses, the document collection should be encrypted before outsourcing them onto remote servers which are not within their trusted domains. To do so, *O* calls $DocEncrypt(D,PP,O_{pr})$ algorithm to encrypt each file $D_i \in D$ using El Gamal encryption technique.

The final step in the sender phase algorithm is sending the I_D and encrypted documents *EncDoc* set to remote server.

The algorithms 1,2 and 3 below show the key generator algorithm, build index algorithm and document set encryption algorithm respectively.

Algorithm 1- KeyGenerator(λ)

Given a security parameter $\lambda \in Z^+$ which determines the size of $\mathbb{G}_1$ and $\mathbb{G}_2$, the algorithm works as follows:

1: generate a prime *q*, two groups $\mathbb{G}_1$, $\mathbb{G}_2$ of order *q* and a bilinear map $\hat{e}: \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$.

2: select a random generator g of $\mathbb{G}_1$.

3: pick a random $b \in Z_q^*$ as a private key, $U_{pr} = b$, for user and calculate the corresponding public $U_{pub} = g^b$.

4: pick a random $a \in Z_q^*$ as a private key, $O_{pr} = a$, for owner and calculate the corresponding public $O_{pub} = g^a$.

5: pick a random $s \in Z_q^*$ as an escrow key, and calculate $V = g^s$.

6: select three hash functions $H_1: \{0,1\}^* \rightarrow \mathbb{G}_1$, $H_2: \mathbb{G}_2 \rightarrow \{0,1\}^{\log q}$ and $H_3: \mathbb{G}_2 \rightarrow \{0,1\}^n$ where H_1, H_2 and H_3 are random oracles.

7: select r hash functions for the bloom filter $H = \{h_1,\ldots,h_r\}$.

8: output(public parameter $PP=\{U_{pub},O_{pub},V,q,g,\hat{e},H,H_1,H_2,H_3\}$, private key $PR=\{U_{pr},O_{pr}\}$).

Algorithm 2: BuildIndex(D_i,W_{Di},PP,O_{pr})

The algorithm is executed by the owner *O* to encrypt the conjunctive keyword W_{Di} and produce a searchable encrypted index I_{Di} as follows:

1: encrypt the ID_i using El Gamal cipher under *O*'s private key a and *U*'s public key U_{pub} as $Enc_{IDi}=ID_i \oplus H_3(G^a)$, where $G=\hat{e}(U_{pub},V)$.

2: compute the length *x* of the Bloom filter BF_{IDi} as in equation (1).

3: initialize the Bloom filter BF_{IDi} of *x* zero-bits.

4: generate *m*! permutations of conjunctive keyword from *m* keywords $P = \{per_1, per_2,\ldots, per_{m!}\}$.

5: for each per_j for $j \in [1,m!]$ do

6: encrypt per_j, set the encrypted keyword as $\omega = \hat{e}(U_{pub}, H_1(per_j)^a)$.

7: $Enc_W = H_2(\omega)$.

8: for $z = 1$ to r do

9: calculate independent hash functions: $b_z = h_z(Enc_W)$

10: set $BF_{IDi}[b_z] = 1$.

11: end for

12: end for

13: pad number of dummy keywords.

14: store $\{Enc_{IDi}, BF_{IDi}\}$ in I_{Di}.

15: return the index I_{Di} as the index for D_i.

Algorithm 3: DocEncrypt(D,PP,O_{pr})

The algorithm is executed by the owner to encrypt the plaintext of *D* as follows:

1: for each document $D_i \in D$ for $i \in [1,n]$ do

2: encrypt the plaintext of D_i using also El Gamal cipher under *O*'s private key a and *U*'s public key U_{pub} as $Enc_{Di} = D_i \oplus H_3(G^a)$, where $G=\hat{e}(U_{pub},V)$.

3: end for.

4: return *EncDoc.*

3.3.2. Retrieval Phase

This phase includes three algorithms as detailed below:

1: **Trapdoor Generator**: To retrieve only the documents containing keywords *Q*, the data user *U* has to ask the *O* for public key O_{pub} to generate trapdoors; If *O* is offline these owners' data can't be retrieved in time. If not, *U* will get the public key O_{pub} and create one trapdoor for a conjunctive keyword set $Q = \{q_1, q_2,\ldots, q_l\}$, using $TrapdoorGen(Q,PP,U_{pr})$ algorithm. Firstly, the data user combines the conjunctive query to make them look like one query, $Tq = \{q_1||q_2||\ldots||q_l\}$, then *U* will compute the trapdoor of the search request of concatenated conjunctive keyword *Tq* under his private key *b*, $Tw = H_1(Tq)^b \in \mathbb{G}_1$. Finally, *U* submits *Tw* to the cloud server.

2: **Search Index**: Upon receiving the trapdoor *Tw*, server will call the $SearchIndex(I_{Di},Tw,PP)$ algorithm on each searchable index and return the associated Bloom filter BF_{IDi}, then compute $T=H_2(O_{pub},\text{Tw}) \in \mathbb{G}_2$ and independent hash functions $h_i(T)$ where $i = 1\ldots r$. Then *S* test *BF* in all *r* locations, if all *r* locations of all independent hash functions in *BF* are 1, the remote server returns the relevant encrypted file corresponding the ID_i to *U*. In other words, searchable index I_D can be used to check set membership without leaking the set items, and for accumulated hashing.

3: **Document collection decryption**: Once *U* receives the encrypted files from cloud server, he calls $DocDecrypt(Enc_D,PP,U_{pr})$ algorithm to decrypt each retrieved document $Enc_{Di} \in Enc_D$ using El Gamal encryption technique.

The algorithms 4,5 and 6 below show the trapdoor generator algorithm, search index algorithm and document set decryption algorithm respectively.

Algorithm 4: TrapdoorGen(Q,PP,U_{pr})

The algorithm is executed by the user to generate a trapdoor as follows:

1: create the combined keywords set as $Tq = \{q_1||q_2||\ldots||q_l\}$.

2: compute trapdoor under U's private key b as: $Tw=H_1(Tq)^b \in \mathbb{G}_1$.

3: send the generated trapdoor Tw to the server.

Algorithm 5: SearchIndex(I_{Di},Tw,PP)

The algorithm is executed by the server S to determine whether a given Index I_{Di} contains a conjunctive keyword Tq as follows:

1: compute $T = H_2(\hat{e}(O_{pub},Tw))$.

2: calculate independent hash functions: $h_1(T)$, $h_2(T)$,…, $h_r(T)$.

3: for each I_{Di}

4: if all r locations of all independent hash functions in BF_{IDi} are 1, then return the relevant encrypted document Enc_{Di} to U.

5: end for

Algorithm 6: DocDecrypt(Enc_D,PP,U_{pr})

The algorithm is executed by the user to decrypt the encrypted documents

1: for each retrieved document $Enc_{Di} \in Enc_D$ for $i \in [1,k]$ do

2: decrypt the ciphertext of Enc_{Di} using El Gamal cipher under U's private key b and O's public key O_{pub} as $Dec_{Di} = Enc_{Di} \oplus H_3(\hat{e}(O_{pub},V)^b)$.

3: end for

4: return Dec_D.

4. ANALYSIS

4.1. Security Analysis

Theorem 4.1. The proposed PKE-KFF-CKS scheme is semantically secure against chosen keyword attacks under the BDH assumption.

Proof. Suppose there is an attack algorithm $\mathcal{A}$ that has advantage ϵ in breaking our scheme. Suppose $\mathcal{A}$ makes q_{H_2} hash queries to H_2 and q_T trapdoor queries. Then we built an algorithm $\mathcal{C}$ that solves the BDH problem with the advantage at least $\epsilon' = \epsilon/e((m!)\ q_T+1)$ where $m!$ is the number of conjunctive keyword sets. Algorithm $\mathcal{C}$ is given $(q,\mathbb{G}_1,\mathbb{G}_2,\hat{e},g,g^a,g^b,g^c)$. It's goal is to compute $\hat{e}\ (g,g)^{abc} \in \mathbb{G}_2$. $\mathcal{C}$ sets $X=g^a$, $Y=g^b$, $Z=g^c$, where a, b, c are random elements in Z_q .

- KeyGen: $\mathcal{C}$ sends $[g,X]$ as the public key to $\mathcal{A}$.

H_1, H_2-Hash queries. To respond to H_1 queries, the challenger $\mathcal{C}$ maintains a list of tuples $<W_i,h_j>$ called the H_1-list. The list is initially empty. When the attacker $\mathcal{A}$ issues a hash query for a conjunctive keyword $W_i = \{w_1||w_2||\ldots||w_s\}$, algorithm $\mathcal{C}$ checks whether $W_i = W_l$, if so, algorithm $\mathcal{C}$ answers consistently with the previous queries by responding with $H_1(W_l)=h_l$. And the corresponding pair (W_i,h_l) has to be saved in memory for future use. Otherwise, $\mathcal{C}$ generates a random coin $\delta_i \in \{0,1\}$ so that $\Pr[\delta_i = 0] = 1/(q_T+1)$, then $\mathcal{C}$ selects a random element $\gamma_i \in Z_q$, if $\delta_i=0$, $\mathcal{C}$ computes $h_i = Y.\ g^{\gamma i} = g^b.\ g^{\gamma i} = g^{(b+\gamma i)}$, otherwise, $\mathcal{C}$ computes $h_i = g^{\gamma i}$, $\mathcal{C}$ adds the tuple $<W_i,h_i>$ to H_1-list, and responds to $\mathcal{A}$ with $H_1(W_l)=h_l$. $\mathcal{C}$ can construct the searchable index I_D, using g as a part of BDH challenge, by executing the algorithm $BuildIndex(D,W_i,PP,PR)$. Then it returns the index I_D to $\mathcal{A}$.

To answer H_2 queries from $\mathcal{A}$, the algorithm $\mathcal{C}$ maintains a list of tuples $<\varpi_j,\theta_j>$ called the H_2- list. The list is initially empty. When $\mathcal{A}$ queries H_2 at a point of Q'_j, $\mathcal{C}$ checks weather $\varpi_i = \varpi_j$. If so, $\mathcal{C}$ responds to $\mathcal{A}$ with $H_2(\varpi) = \theta_j$, where $\theta \in \{0,1\}^{logq}$. Otherwise $\mathcal{C}$ chooses a random element θ_i, adds the tuple $<\varpi_i,\theta_i>$ to H_2-list, and answers $\mathcal{A}$ with $H_2(\varpi_i)=\theta_i$.

- Trapdoor queries. $\mathcal{C}$ can use the algorithm trapdoor(Q,PP,PR) to issue the trapdoor Tw corresponding the conjunctive keyword query $Tq=\{q_1||q_2||\ldots||q_l\}$. Tw is a valid trapdoor for Q, because $\mathcal{C}$ uses the same value for the conjunctive keyword Q under the public key $[g,X]$.

- Challenge. Algorithm $\mathcal{A}$ picks and sends a pair of conjunctive keyword $W_0=\{w_{01}||w_{02}||\ldots||w_{0s}\}$ and $W_1=\{w_{11}||w_{12}||\ldots||w_{1s}\}$ to $\mathcal{C}$ on which it wishes to be challenged, and $\mathcal{A}$ must not have asked previously for the trapdoors of any conjunctive word W_0 or W_1. For each conjunctive keyword W_i where $i \in \{0,1\}$, algorithm $\mathcal{C}$ calls the above random oracle algorithm for responding to H_1-queries to obtain $h_i \in \mathbb{G}_1$, where $H_1(W_l)=h_l$. Let $<W_i,h_i>$ be the corresponding tuple on the H_1-list, if both δ_0 and δ_1 are not 0 then $\mathcal{C}$ reports failure and terminates. Otherwise, both δ_0 and δ_1 are equal to 0, $\mathcal{C}$ creates a random coin $\mu \in \{0,1\}$ If only one δ_μ is equal to 0 then no randomness is needed.

Since $\mathcal{C}$ is given g, g^a, g^b, g^c as part of BDH challenge, $\mathcal{C}$ creates the secure index I_{D_μ} by executing the algorithm *BuildIndex*(*D*,W_μ,*PP*,*PR*) and sends the challenge $\mathbb{C}=[Z, I_{D_\mu}]$ to $\mathcal{A}$. The index includes *s*! encrypted permutations of W_μ, $T_\mu = H_2(\hat{e}(H_1(W_\mu), X^c))$, that means $T_\mu = H_2(\hat{e}(g^{(b+\gamma_\mu)}, g^{ac})) = H_2(\hat{e}(g,g)^{(b+\gamma_\mu)ac}) \in \{0,1\}^{logq}$. *BF* =$h_1(T_\mu)$, $h_2(T_\mu)$,…, $h_r(T_\mu)$ if all *r* locations of all independent hash functions in *BF* are 1, then $\mathbb{C}$ is a valid encryption for W_μ as required.

- More queries. After the above challenge query, $\mathcal{A}$ is allowed again to query $\mathcal{C}$ with the same restriction that $W_i \neq W_0, W_1$, $\mathcal{C}$ responds to these queries in the same way as before. Upon receiving the challenge, $\mathcal{A}$ can call the algorithm *SearchIndex*(I_D,*Tw*,*PP*) on the secure index I_{D_μ} to determine if the conjunctive keyword in the I_{D_μ} is the same of W_μ or not. $\mathcal{A}$ does not know the private key that is chosen independently of any conjunctive keyword, that's mean, at this stage, $\mathcal{A}$ has no solution to distinguish the μ from 0 or 1.

- Output. Finally, $\mathcal{A}$ returns its guess $\mu' \in \{0,1\}$ indicating whether the challenge $\mathcal{C}$ is the result of encryption process for W_0 or W_1. As this point, algorithm $\mathcal{C}$ selects a random pair (ϖ, θ) from the H_2-list and returns $\varpi / e(X, Z)^{\gamma_\mu}$ as its guess for $\hat{e}(g,g)^{abc}$, A must have deliver his query for either $H_2(\hat{e}(W_0, X^c))$ or $H_2(\hat{e}(W_1, X^c))$. Hence, with probability 1/2 H_2-list includes a pair whose left hand is $\varpi = \hat{e}(W_\mu, X^c) = \hat{e}(g,g)^{ac(b+\gamma_\mu)}$. If $\mathcal{C}$ selects this pair (ϖ, θ) from the H_2-list then $\varpi / e(X,Z)^{\gamma_\mu} = \hat{e}(g,g)^{abc}$ as required.

To complete the proof of theorem (4.1), we now use the same approach as in [4] to analyze the probability that $\mathcal{C}$ does not abort during the above experiment. We define the following three events:

- Ev_1: $\mathcal{C}$ does not abort during the Trapdoor queries.

- Ev_2: $\mathcal{C}$ does not abort during the Challenge queries.

- Ev_3: $\mathcal{A}$ does not issue a query for either $H_2(\hat{e}(g^b, H_1(W_0)))$ or $H_2(\hat{e}(g^b, H_1(W_1)))$.

We suppose that both events Ev_1 and Ev_2 occur with sufficiently high probability. Let we consider the first event Ev_1, the probability of Ev_1 is $(1-1/(q_T+1))^{q_T} \geq 1/e$, where $1/(q_T+1)$ is the probability that a trapdoor query makes $\mathcal{C}$ to abort.

For the second event Ev_2, the algorithm $\mathcal{C}$ does not abort during the challenge phase if one of δ_0 and δ_1 is 0. By the definition of H_1-list $\Pr[\delta_i] = 1/(q_T+1)$ where $i \in \{0,1\}$ and the two values are independent of one another, we have that both $\Pr[\delta_0 = \delta_1 = 1] = 1-1/q_T \geq (1-1/(q_T+1))^2$. Hence, the $\Pr[Ev_2]$ is at least $1/q_T$. Since $\mathcal{A}$ never issues trapdoor queries for target keyword vectors, Ev_1 and Ev_2 are independent. Hence, the probability that $\mathcal{C}$ does not abort during the entire simulation, that is $\Pr[Ev_1 \wedge Ev_2] \geq 1/(eq_T)$.

For the last event, when Ev_3 occurs, the bit $\mu \in \{0,1\}$ indicating whether the challenge, an encryption of W_0 or W_1, is independent of $\mathcal{A}$'s view

$$\Pr[\mu = \mu'] = \Pr[\mu = \mu' | Ev_3] \Pr[Ev_3] + \Pr[\mu = \mu' | \neg Ev_3] \Pr[\neg Ev_3]$$
$$\leq \Pr[\mu = \mu' | Ev_3] \Pr[Ev_3] + \Pr[\neg Ev_3]$$
$$= \frac{1}{2} \Pr[Ev_3] + \Pr[\neg Ev_3]$$
$$= \frac{1}{2} + \frac{1}{2} \Pr[\neg Ev_3]$$

It follows that $\epsilon \leq |\Pr[\mu = \mu'] - 1/2| \leq 1/2 \Pr[\neg Ev_3]$.Hence, $\Pr[\neg Ev_3] \geq 2\epsilon$ By Ev_3, if $\mathcal{C}$ does not abort during the simulation, it will choose a correct tuple in H_2-list with probability at least $1/q_{H_2}$, and will produce the correct answer with probability at least ϵ/q_{H_2}. Overall by combining Ev_1, Ev_2 and Ev_3, we have $\mathcal{C}$'s success probability is at least $\epsilon/eq_T q_{H_2}$.

4.2. Performance

1. **Time Efficient**: In the former works, when the user *U* wants to retrieve documents containing each of several keywords, he must give the server trapdoors for each of the keywords individually and rely on an intersection operation. This solution is not desirable, it requires $O(n) \times m$ search time, where *n* is the number of documents and *m* is the number of keywords in conjunctive string. In other words, the sever needs $O(n)$ search time for each keyword in conjunctive string. While in our work, the user computes one trapdoor for all conjunctive keyword and sends it to the remote server and with one round over all conjunctive keywords, a server calls *SearchIndex* algorithm once on each trapdoor. Documents whose indexes match trapdoors are returned to *U*. The overhead of such Boolean queries is linear with the number of keywords in the Boolean expression, but can be completed with a one round over the document without needing intersection operation, whereas the naive solution of performing such boolean queries involves multiple rounds over the document. Hence the proposed scheme, based on Bloom filters [7], requires $O(n)$ search time for all keywords in conjunctive string, and $O(1)$ for communication cost. The scheme tests each *BF* only once per search. in other words, the time required to check whether a conjunctive keyword is present or not is independent of the number of keywords present in the set, we just need $O(n)$ to find the *r* indexes using *r* hash functions.

   The overall performance of our scheme includes the cost of index construction and the time necessary for searches. Updating or adding the documents require calling *BuildIndex* algorithm, which has cost linear in the size of the documents, while deleting the documents requires a constant time computation.

2. **Space Efficient**: Using Bloom Filter makes the required space to represent PKE-KFFCKS is sufficiently less as compared to other data structures like hash tables, linked lists, arrays, etc. a Bloom filter BF requires a fixed number of

bits to store all the items. Hence the server storage cost that required to store the Index I_D is $O(n)$ which is very small compared to the amount of data.

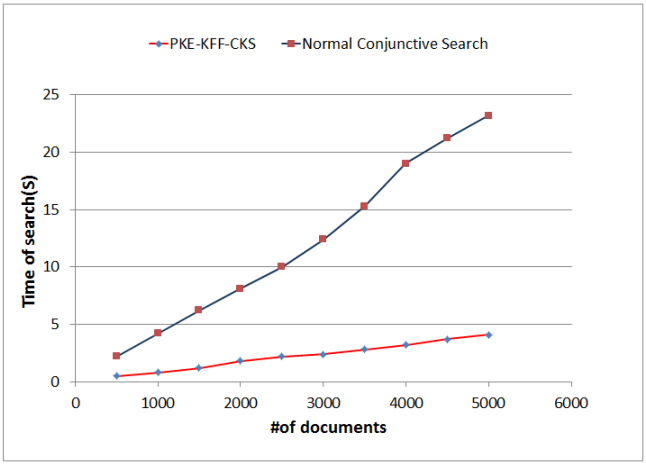

Fig 3: Time cost of query for different size of the file collection with the fixed keywords number *m*=5 in the query

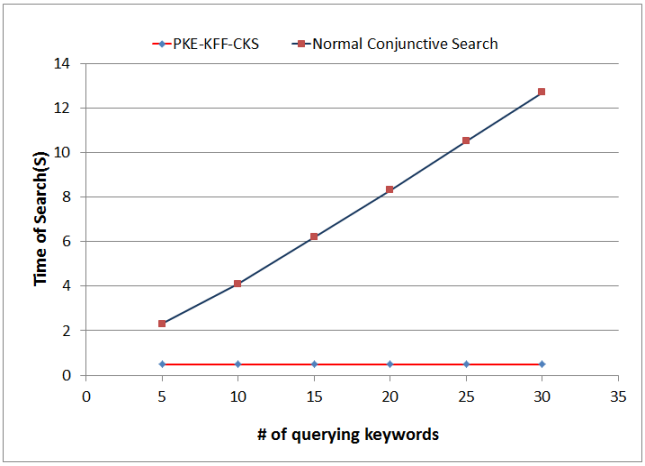

Fig 4: Time cost of query for different size of the query collection in the same dataset, *n*= 500.

4.3. Experimental Evaluation

Figure 3 demonstrates the time consumption of a remote server for performing a search query with both normal conjunctive search and PKE-KFF-CKS on the encrypted Bloom filter. Obviously, with the increasing of the number of files *n*, the time spend using the proposed scheme is much less than the time spend using normal conjunctive search, in other words, the efficiency of PKE-KFF-CKS is far higher than normal conjunctive search method. Figure 4 shows that the time consumption using the normal conjunctive search grows linearly with the size of the query collection *m*, while the time consumption using the proposed scheme has little impact with the size of the query collection *m*, where the number of the keywords in the query *m* increases from 5 to 30. Furthermore, while the search cost is linear with the number of query keywords in other conjunctive keyword search schemes [11,19], PKE-KFF-CKS introduce nearly constant overhead while increasing the number of the keywords in the query.

Table 1. Comparison of security assumption and other attributes

Scheme	Security assumption	Keyword filed free	Using in unstructured data	Index generation	User
Golle et al.-I [11]	IND1-CKA under DDH in the ROM	×	×	-	single user
Golle et al. -II [11]	IND1-CKA under new nonstandard hardness assumption	×	×	-	single user
Ballard et al. [9]	IND1-CKA based on the security of SSS in the ST	×	×	uses a pseudorandom function per keyword	single user
Byun et al. [20]	IND1-CKA under BDH in ROM	×	×	-	single user
Wang et al. [22]	IND1-CKAt under DL, 1-DDHI in ST	√	√	uses l-degree polynomial per document and compute l hash functions	multi user
Ryu et al. [21]	IND1-CKA under coXDH in ROM	×	×	-	single user
Cash et al. [14]	IND2-CKA under DDH	×	√	uses pseudo – random function and hash function	multi user
Our scheme	IND1-CKA under BDH in ROM	√	√	uses Bloom filter and hash function	single user

4.4. Comparisons

We compare our scheme to the previous conjunctive keyword schemes in terms of security assumption with other attributes in Table 1. The table is arranged by the query expressiveness. The first column shows the paper and reference. The ”security assumption” column shows the security definitions, assumptions, and if ROM is used to prove the secure of the scheme. The ”keyword field free” column shows whether the scheme uses the fixed-position keyword field keyword search or the keyword field free keyword search. The ”using in unstructured data” column shows whether the scheme is practical for using in unstructured data or not. The ”Index generation” column shows whether the construction of each scheme is based on the index generation or not. The last column shows whether the schemes can be used with a single or multi user. Compared with previous conjunctive keyword search schemes we show that, in the last row, just our scheme uses the bloom filter with such a search. Finally, the last difference between these schemes and PKE-KFF-CKS is that the formers use the different public and symmetric key encryption algorithms to encrypt the data set without specifying their specific implementation, whereas our scheme specifies the encryption and decryption operations by using El Gamal algorithm to encrypt *D* under *O*'s private key and *U*'s public key, and decrypt the encrypted document under *U*'s private key and *O*'s public key.

ACKNOWLEDGMENTS

The author would like to thank professor Song Feng Lu (School of Computer Science and Technology, Huazhong University of Science and Technology) for giving advice on this work and improving this paper.

REFERENCES

1. Song D., Wagner D. and Perrig A. 2000. Practical techniques for searches on encrypted data. Proc. of IEEE Symposium on Security and Privacy.
2. Bao F., Deng R., Ding X. and Yang Y. 2008. Private query on encrypted data in multi-user settings. Proc. Of ISPEC.
3. Bellare M., Boldyreva A. and ONeill A. 2007. Deterministic and efficiently searchable encryption. Proceedings of Crypto, 4622 of LNCS. Springer-Verlag.
4. Boneh D., Crescenzo GD., Ostrovsky R. and Persiano G. 2004 Public key encryption with keyword search. Proc. of EUROCRYP.
5. Chang YC. and Mitzenmacher M. 2005. Privacy preserving keyword searches on remote encrypted data. Proc. of ACNS.
6. Curtmola R., Garay JA., Kamara S. and Ostrovsky R. 2006. Searchable symmetric encryption: improved definitions and efficient constructions. Proc. of ACM CCS.
7. Goh EJ. 2003. Secure indexes, Cryptology ePrint Archive. http://eprint.iacr. org/.2003/216.
8. Waters B., Balfanz D., Durfee G. and Smetters D. 2004. Building an encrypted and searchable audit log. Proc. Of 1th Annual Network and Distributed System.
9. Ballard L., Kamara S. and Monrose F. 2005. Achieving efficient conjunctive keyword searches over encrypted data. In: Qing et al. (eds.) ICICS 2005. LCS, Springer, Heidelberg vol. 3783, pp. 414-426.
10. Chen Z.,Wu C.,Wang D. and Li S. 2012. Conjunctive keywords searchable encryption with efficient pairing, constant ciphertext and short trapdoor. In: Chau et al. (eds.) PAISI 2012. LNCS, vol. 7299, pp. 176-189. Springer, Heidelberg.
11. Golle P., Staddon J. and Waters B. 2004. Secure conjunctive keyword search over encrypted data. In: Jakobsson, M., Yung, M. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 31-45. Springer, Heidelberg.
12. Hwang Y. H. and Lee P. J. 2007. Public key encryption with conjunctive keyword search and its extension to a multiuser system. In: Takagi et al. (eds.) Pairing 2007. LNCS, vol. 4575, pp. 2-22. Springer, Heidelberg.
13. Park D. J., Kim K. and Lee P. J. 2004. Public key encryption with conjunctive field keyword search. In: Lim, C.H., Yung, M. (eds.) WISA 2004. LNCS, vol. 3325, pp.73-86. Springer, Heidelberg.
14. Cash D., Jarecki S., Jutla C. S., Krawczyk H., Rosu M. and Steiner M. 2013. Highly- Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In Canetti, R., Garay, J. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 353-373. Springer, Heidelberg.
15. Moataz T. and Shikfa, A. 2013. Boolean symmetric searchable encryption. In: ACM ASIACCS 2013, pp. 265-276.
16. Katz J., Sahai A. and Waters B. 2008. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 146-162. Springer, Heidelberg.
17. Lai J., Zhou X., Deng R. H., Li Y. and Chen K. 2013. Expressive search on encrypted data. In: ACM ASIACCS 2013, pp. 243-252.
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 Waters B. 2007. Conjunctive, subset, and range queries on encrypted data. Proc. of TCC.pp.535-554.

20. Byun, J., Lee, D. and Lim J. 2006. Efficient Conjunctive Keyword Search on Encrypted Data Storage System, In Proceedings of EuroPKI 2006, LNCS 4043, Springer-Verlag, pp.184-196.

21. Ryu, E. K. and Takagi, T. 2007. Efficient conjunctive keyword-searchable encryption. In AINAW. IEEE Computer Society, Washington, DC, 409414. DOI:http://dx.doi.org/10.1109/AINAW.2007.166.

22. Wang, P., Wang, H. and Pieprzyk, j. 2008. Keyword field-free conjunctive keyword searches on encrypted data and extension for dynamic groups. In CANS (LNCS), Vol. 5339 pp.178-195, Springer.

23. Zhang, BO. and Zhang, F. 2011. An efficient public key encryption with conjunctive-subset keywords search," Journal of Network and Computer Application, vol. 34, no. 1, pp. 262-267.

24. Kumar, M. 2010. A new secure remote user authentication scheme with smart cards, International Journal of Network Security, vol. 11, no. 2, pp. 88-93, 2010.

25. Lee, C. C. 2009. On security of an efficient nonce-based authentication scheme for SIP, International Journal of Network Security, vol. 9, no. 3, pp. 201-203.

26. Tsai, C. S. and Lee, C. C., 2006. Hwang M. S.: Pass-word authentication schemes: current status and key issues, International Journal of Network Security,vol. 3, no. 2, pp.101-115.

27. 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,Volume 13, Issue 7.