A Fast Privacy-Preserving Multi-keyword Search Scheme on Cloud Data

Ce Yang, Weiming Zhang, Jun Xu, Jiajia Xu, Nenghai Yu
Electronic Engineering and Information Science University of Science and Technology of China Anhui, China
Email: {cn.yang.ce, zwmshu, junxlcustc}@gmail.com, xujiajia@mail.ustc.edu.cn, ynh@ustc.edu.cn

***Abstract*—Nowadays, more and more people outsource their data to cloud servers for great flexibility and economic savings. Due to considerations on security, private data is usually protected by encryption before sending to cloud. How to utilize data efficiently while preserving user's privacy is a new challenge. In this paper, we focus on a efficient multi-keyword search scheme meeting a strict privacy requirement. First, we make a short review of two existing schemes supporting multi-keyword search, the kNN-based MRSE scheme and scheme based on bloom filter. Based on the kNN-based scheme, we propose an improved scheme. Our scheme adopt a product of three sparse matrix pairs instead of the original dense matrix pair to encrypt index, and thus get a significant improvement in efficiency. Then, we combine our improved scheme with bloom filter, and thus gain the ability for index updating. Simulation Experiments show proposed scheme indeed introduces low overhead on computation and storage.**

I. Introduction

Because of the rapid growth of data, personal hard drive and company's data center couldn't satisfy people's demands. At this time, cloud storage appears. Cloud storage provides on-demand, scalable and QoS guaranteed storage resource, and users can operate their data anytime and anywhere.

Facing the powerful and appealing advantages of cloud storage, however, a lot of people and companies are hesitant to put their data in cloud. The main reason is that people and companies are afraid of losing control on their data. Examples like Salesforce.com falling for phishing attack [1] and Amazon Cloud Services' Interruption recently [2] verify people's fears. Besides external attacks and server failure, the cloud platform itself is considered suspicious. As a fact, none of Amazon EC2's top-20 client's main business is e-commerce. Therefore, to be sustainable, indepth development, cloud storage must address the privacy concern. In order to protect people's privacy, encryption is a commonly used method.

However, data encryption obsoletes the traditional data utilization service, including the most widely-used keyword-based information retrieval. Storing data into the cloud serves no purpose, unless they can be easily searched and utilized. Thus, to take advantage of the convenience brought by cloud service, developing privacy-preserving and effective search scheme over encrypted cloud data is a priority. As proved in traditional information retrieval, to meet user's demands, and to improve the result accuracy, the retrieval system should provide support to ranked search and multi-keyword search. It is a challenging problem to meet the requirements of performance, system usability, while keeping data and search privacy.

To utilize data effectively, Song et al. [3] propose a solution for searching with sequential scan. This scheme is provable secrecy. Similar scheme is proposed as new cryptographic primitive, such as searchable encryption [4]–[7], or more general, functional encryption [8]–[10]. These schemes operate directly on the ciphertext, and hence don't need to build a separate index. However, direct operation on ciphertext also bring the problem of efficiency and security. The server needs a sequential scan, and will even know the keyword's location. Homomorphic encryption [11] proposed recent years implies the ability to support search on the ciphertext, but it is still underdeveloped, and also suffers from efficiency problems. To solve the efficiency problem, building an encrypted index particularly for search are suggested. Bloom filter [12], [13] is introduced as a secure index scheme supporting $O(1)$ search on each document. C. Wang et al. [14] propose an efficient scheme, using the order preserving encryption [15] (OPE) on the traditional inverted index to protect term frequency, and traditional symmetric encryption on keywords to protect keyword privacy. The two kinds of scheme work well in the known ciphertext model, but are not secure under the more strict known background model. Cao et al. propose a kNN-based multi-keyword ranked search over encrypted cloud data scheme (kNN-based MRSE) [16], which meets the strict privacy requirement in most realistic situation. However, its space and time cost have to be improved before put in practice.

In this paper, we make a short overview of two state-of-the-art schemes supporting multi-keyword search, the kNN-based MRSE scheme [16] and secure index based on bloom filter [13]. We propose our solution, which is a novel scheme that realizes efficient multi-keyword search supporting dynamic updating, meeting strict privacy requirements. Based on the kNN-based MRSE scheme, firstly we propose a improved scheme, which significantly improves the time and space efficiency. Then we combine bloom filter with it to support dynamic update. Experiments on the real-world dataset show that the proposed scheme is significantly better than the kNN-based MRSE scheme.

The reminder of this paper is organized as follows. In Section II, we declare the system model, privacy requirements, and notations. Section III describes two scheme supporting multi-keyword search and their drawbacks, followed by Section IV, which describes our solution. Section V presents simulation

978-0-7695-4910-1/12 $26.00 © 2012 IEEE
DOI 10.1109/CSC.2012.23

results. We conclude the paper in Section VI.

II. Problem Formulation

In this section, we describe the searching model and privacy requirements, based on the analysis and definition in [16]. We add the update module to meet our demands.

A. System Model

A cloud service can be described as three entities, the data owner, the data user, and the cloud server. The data owner has a collection of m document $\mathcal{D} = \{D_1, D_2, \cdots, D_m\}$, that to be outsourced to the cloud server, usually in the encrypted form $\mathcal{E} = \{E_1, E_2, \cdots, E_m\}$. To enable searching, the data owner builds an index $\mathcal{I} = \{I_1, I_2, \cdots, I_m\}$ from $\mathcal{D}$, where I_i is built for D_i, and then outsources it. The terms, or keywords, appeared in the document collections can be denoted as dictionary $\mathcal{W} = \{W_1, W_2, \cdots, W_n\}$.

To search over $\mathcal{D}$, the data user sends a query $\mathcal{Q}$, which consists of a subset of $\mathcal{W}$, to the data owner, and the data owner generates a trapdoor $\mathcal{T}$ from $\mathcal{Q}$. Receiving $\mathcal{T}$, the cloud server searches $\mathcal{I}$ and returns the top-k result $\mathcal{R}$ to user.

To update the index, the data owner adds a new entry I_i to index for a new document D_i, and new terms W_j to dictionary for words not in the dictionary yet. In some cases, the existed entry in the index $\mathcal{D}$ may be modified, and even the secret key K may need to regenerate

The whole system can be impled by five algorithms, as below:

- **KeyGen**
  The data owner uses a parameter s to generate a key, K.
- **BuildIndex**
  The data owner builds a privacy-preserving index $\mathcal{I}$ from a dataset $\mathcal{D}$ based on K. After the index construction, the document can be independently encrypted and outsourced.
- **TrapGen**
  The data owner generates a trapdoor $\mathcal{T}$ according to a query $\mathcal{Q}$ including several interested terms.
- **Query**
  The cloud server performs a ranked search using the trapdoor $\mathcal{T}$ on the index $\mathcal{I}$, then returns the result $\mathcal{R}$ to user.
- **Update**
  The data owner regenerates key S, adds new entries to the index and dictionary, and informs server the modification.

The system framework is shown in Fig. 1.

B. Privacy Requirements

The cloud server is considered as "honest-but-curious" in our model. Specifically, the cloud server acts according to the designated protocol correctly, and will not distort or counterfeit user data or query results. However, the server may use all information (including outsourced data, index, and trapdoor) to analyze so as to learn additional information. We describe the threat model and privacy requirement based on [16].

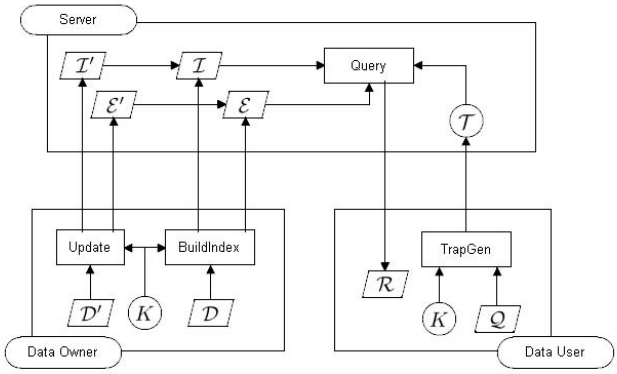

Fig. 1. System Framework

The threat model is considered as **Known Background Model**. That is to say, the cloud server knows not only data stored on it, $\mathcal{E}$ and $\mathcal{I}$, but also some background knowledge, such as statistical information of the dataset. In this model, document/keyword frequency may be used by cloud server to guess keyword [17].

In our scheme, data privacy can be protected by traditional symmetric encryption, which can be considered secure. Then, we hope the index and trapdoor should leak as less information as possible. Index and search privacy can be described as follow.

Keyword Privacy Trapdoor should not leak information about keywords. Although trapdoor can be generated as encrypted, such as hash values or encrypted values, to protect the query keywords, the cloud server could do some statistical analysis over the search result to make an estimate. As a kind of statistical information, document frequency (i.e., the number of documents containing the keyword) is sufficient to identify the keyword with high probability.

Trapdoor Unlinkability The same query should generate different trapdoors when queried twice. The server should not be able to deduce relationship between trapdoors. Otherwise, the server can accumulate trapdoors to guess the query keywords.

Access Pattern Access pattern is the rank order of the results. As the computation is finished in the server, the protection of access pattern usually resort to secure hardware. Our schemes are not designed to protect it.

III. Previous Work

To achieve effective retrieval, we mainly focus on multi-keyword search. In this section, we will briefly present two previous scheme in this field, and discuss whether they complete the retrieval task efficiently while meeting our security requirements.

A. kNN-based MRSE

The kNN-based MRSE [16] uses inner product similarity to evaluate the relative score between document and query. Inner

product similarity is described as follow: If the index I_i and query Q are represented by boolean vectors p_i and q, where the k-th dimension $p_i[k]$ and $q[k]$ indicate if D_i and Q contain term W_k, then relative score can be calculated by multiplying p_i with q. In such a way, document containing more keywords get a higher score than document with few keywords. The kNN-based MRSE takes several steps to enhance security. Its secret key includes four matrices and a vector. Two matrices, M_1 and M_2, and the vector S are used in index construction. Query generation needs the participation of query key, which consists of two matrices M_1^{-1}, M_2^{-1} and the vector S. We give a detailed description of the kNN-based MRSE scheme below.

Assume the dictionary is composed of n different terms. First, n-dimensional vectors p_i and q are generated from index I_i and query Q. U dummy terms are added into dictionary, hence the vector p_i and q are extended to $(n+U)$-dimensional vectors as $\vec{p_i}$ and $\vec{q}$. The values of dummy keywords in $\vec{p_i}$ and $\vec{q}$ are independent random variables. U is a system parameter can be adjusted according to user's demand.

Then, $\vec{p_i}$ and $\vec{q}$ are both split into two random vectors as $\{\vec{p_i}', \vec{p_i}''\}$ and $\{\vec{q}\,', \vec{q}\,''\}$. Here the vector S functions as a splitting indicator. If the j-th bit of S is 0, $\vec{p_i}'[j]$ and $\vec{p_i}''[j]$ are set as the same as $\vec{p_i}[j]$, while $\vec{q}\,'[j]$ and $\vec{q}\,''[j]$ are set to two random numbers so that their sum is equal to $\vec{q}[j]$; if the j-th bit of S is 1, the splitting process is similar except that $\vec{p}$ and $\vec{q}$ are switched.

The split data vector pair $\{\vec{p_i}', \vec{p_i}''\}$ is encrypted as $\{\vec{p_i}' \cdot M_1, \vec{p_i}'' \cdot M_2\}$, and the split query vector pair $\{\vec{q}\,', \vec{q}\,''\}$ is encrypted as $\{\vec{q}\,' \cdot (M_1^{-1})^T, \vec{q}\,'' \cdot (M_2^{-1})^T\}$.

In the query step, the product of index vector pair and query vector pair, i.e., $(\vec{p_i} \cdot \vec{q}^{\,T})$, is serving as the indicator of inner product similarity $(p_i \cdot q^T)$ to select top-k results.

To show the procedure clearly, we make a simple demonstration here. Assume the dictionary size $n = 4$, and there is no dummy keyword. We generate S and matrices M as follow:

$$S = 1001,$$

$$M_1 = \begin{pmatrix} 0.92 & 0.15 & 0.7 & 0.57 \\ 0.33 & 1 & 0.3 & 0.48 \\ 0.24 & 0.39 & 0.03 & 0.74 \\ 0.97 & 0.54 & 0.48 & 0.19 \end{pmatrix},$$

$$M_2 = \begin{pmatrix} 0.57 & 0.7 & 0.87 & 0.13 \\ 0.77 & 0.51 & 0.83 & 0.55 \\ 0.27 & 0 & 0.87 & 0.75 \\ 0.23 & 0.22 & 0.03 & 0.9 \end{pmatrix}.$$

M_1^{-1} and M_2^{-1} are easy to calculate and therefore omitted.

For document D_1, if it contains the first term and the second term, then there is:

$$p_1 = 1100.$$

Split it with S, we get:

$$\begin{pmatrix} \vec{p_1}' \\ \vec{p_1}'' \end{pmatrix} = \begin{pmatrix} 0.04 & 1 & 0 & 0.47 \\ 0.96 & 1 & 0 & -0.47 \end{pmatrix}.$$

Multiply $\vec{p_1}'$ and $\vec{p_1}''$ with M_1 and M_2:

$$\begin{aligned} I_1 &= \left(\vec{p_1}' \cdot M_1, \quad \vec{p_1}'' \cdot M_2\right) \\ &= \left(0.82 \quad 1.26 \quad 0.55 \quad 0.59 \quad 1.21 \quad 1.08 \quad 1.65 \quad 0.25\right). \end{aligned}$$

For a query with the second and third term, there is

$$q = 0110.$$

With a similar procedure, we generate $\vec{q}$ from q, split $\vec{q}$ to $\vec{q}\,'$ and $\vec{q}\,''$, multiply with $(M_1^{-1})^T$ and $(M_2^{-1})^T$, then we get the trapdoor:

$$\begin{aligned} \mathcal{T} &= (\vec{(q)}' \cdot (M_1^{-1})^T, \vec{q}\,'' \cdot (M_2^{-1})^T) \\ &= \left(0.06 \;\; 0.21 \;\; -0.59 \;\; 0.57 \;\; 2.01 \;\; -1.68 \;\; 0.06 \;\; -0.11\right). \end{aligned}$$

Calculating the inner product of I_1 and $\mathcal{T}$, we get the result is 1.01.

Without prior knowledge of secret key, neither data vector nor query vector, after such a series of processes, can be recovered by analyzing their corresponding ciphertext.

However, this scheme still have some drawbacks which make it hardly a practical method. The most severe one is efficiency. Consider applying this scheme to a collection of m documents. Assume its text size is N, and it consists of n unique terms. The time of building index is $O(mn^2)$, generating trapdoor is $O(n^2)$, querying is $O(mn)$. The space of index is $O(mn)$, query key is $O(n^2)$. Procedure of generating two matrix pairs is also time-consuming, but in this paper, we just assume that the kNN-based scheme adopts a fast algorithm so that its time cost is tolerable.

Large collections comply with Heap's Law [18]: $n = \alpha N^\beta$, where α and β are constants depend on document collections. Typical values for α and β are: $10 \leq \alpha \leq 100, 0.4 \leq \beta \leq 0.6$. Then $n^2 \approx N$. It means that the calculation of index will be nonlinear and therefore inefficient, and the storage space of key will be the same or even greater to the document collection.

Without considering index rebuilding caused by adding new document, the time cost of index construction and the space cost of M_1 and M_2 are relatively unimportant, because it is a one-time operation, and the two matrices M_1 and M_2 can be abandoned afterwards. Index is stored in the server and hence its space cost can be considered posterior to other factors, unless it becomes too large. However, the time cost of query generation and space cost of query key must be taken into consideration. The vast overhead will neutralize the convenience brought by cloud service, make it even no advantages compared to the trivial solution: data owner can keep a plaintext index and use it to respond to other users' request, then data users can retrieve related documents from server according to the result list returned by the data owner. Simulation result on a real world dataset is shown and discussed later.

Another problem involves the updating of index. Due to efficiency consideration mentioned above, dictionary size should be set as little as possible. Consider a dataset of 1000 unique

terms. If we set the dictionary size as 1200, then when new terms gained from new documents exceed 200, the whole index should be recalculated to add support for new keywords. Thus to say, supporting index update will lead to either vast computation and storage waste, or unendurable expense of index reconstruction.

B. Bloom Filter Based Scheme

A Bloom filter is a data structure which is used to answer set membership queries. It is represented as an array of b bits which are initially set to 0. In general the filter uses r independent hash functions h_t, where $h_t : \{0,1\}^* \rightarrow [1,b]$ for $t \in [1,r]$, each of which maps a set element to one of the b array positions. For each element e in the set $S = \{e_1, \cdots, e_m\}$ the bits at positions $h_1(e), \cdots, h_r(e)$ are set to 1. To check whether an element x belongs to the set S, we check if the bits at positions $h_1(x), \cdots, h_r(x)$ are set to 1. If so, x is considered a member of set S. Bloom filters have a possibility of false positives, because the positions of an element may have been set by one or more other elements. With appropriate parameters the false positive probability can be reduced to a desired error rate.

Bloom filter can be used to construct secure index. For each document D_i, a bloom filter I_i is generated. Bloom filter's length should be set no less than a value depends on the quantity of terms in the document, to restrict the false positive probability. The trapdoor T for query $\{x_1, \cdots, x_q\}$ consists of the hash values $h_j(x_k)$, $1 \leq k \leq q$. Different from the kNN-based MRSE scheme, index I_i doesn't need to contain a mark explicitly if D_i doesn't contain term W_j. Then new terms don't affect existed index, and adding a document merely needs to add a new bloom filter to the index set.

However, bloom filter doesn't satisfy our security demands. For the same query, bloom filter will generate the same trapdoor, thus the trapdoor unlinkability is not satisfied. Besides, the server can subtract two bloom filters to deduce the relationship of two documents. As a example, we make an experiment on our dataset, and the result is shown in fig. 2. Assume the difference of two bloom filters is 151. Because one term corresponds to five hash values in our experiment, firstly we integer divide the difference by 5 and get 30. Then we can infer from the figure that the two documents have about 33 different terms.

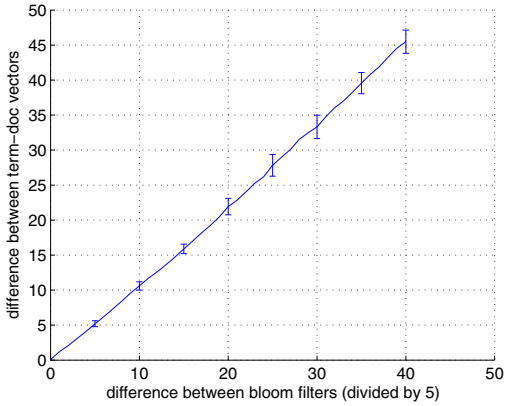

Fig. 2. The difference between term-document vectors as a function of the difference between bloom filters. Bloom filters are divided by 5. The error bar on the fig represents the largest error of 5 continues points.

IV. Proposed Scheme

In this section, we propose our scheme. Even under strict privacy requirements, our scheme can achieve ranked multi-keyword search. We first propose a basic scheme, which is adapted from kNN-based MRSE scheme [16] and much more efficient than the original one, and then modify it support updating.

A. Improving kNN-based Scheme

The essential reason of inefficiency in MRSE scheme is the large scale of the matrices used in index construction and query generation. In our more advanced scheme, local matrix is adopted rather than global matrix. We can divide the terms into different subsets, and use a small-scale matrix to mix them. That's to say, we can set the matrix M_1 as the product of a block diagonal matrix M_1' and a permutation matrix P.

$$M_1 = M_1' \cdot P,$$

where

$$M_1' = diag(B_1, B_2, \cdots, B_h).$$

Every B_k is a n' by n' random matrix, and P is a n by n random permutation matrix. In the original kNN-based MRSE scheme, each document requires n^2 floating-point operations, and this change reduces it to $n \cdot n'$ floating-point operations. For example, when building index for a dataset containing 10,000 unique terms(the pocket oxford dictionary includes 65,000 entries), the proposed scheme will be 100 times faster than kNN-based MRSE scheme, and the space cost of query key will be reduced from 1526 MB to 1.9 MB, if we set n' to 100.

However, the division will lower security. In the original scheme, all the n dimensions in a vector $\vec{p_i}$ is mixed, while this scheme only mixes n' keywords. The server now can pick random 100 subscript, j_1 to j_{100}, add corresponding numbers up to see if the sum $s = I_i[j_1] \cdot T[j_1] + \cdots + I_i[j_{100}] \cdot T[j_{100}]$ equals to 0, to analysis the division of keywords and to judge if the query relates with the 100 keywords. This will weaken the trapdoor unlinkability severely.

To enhance security, we can multiply another block matrix:

$$M_1 = M_1' \cdot P \cdot M_1'',$$

where M_1'' is another block diagonal matrix like M_1'. After this step, different keywords have been mixed, and it is very difficult for the server to learn about query. M_2 is generated in the same manner.

To make the procedure clear, we make a demonstration using dataset the same as the example in section III. If we set $n' = 2$, then the keywords are divided into $h = 4/2 = 2$

groups. The matrices used for encryption can be generated as follow:

$$M_1' = \begin{pmatrix} 0 & 0.63 & 0 & 0 \\ 0.92 & 0.15 & 0 & 0 \\ 0 & 0 & 0.03 & 0.74 \\ 0 & 0 & 0.97 & 0.54 \end{pmatrix}$$

$$M_2' = \begin{pmatrix} 0.03 & 0.9 & 0 & 0 \\ 0.93 & 0.74 & 0 & 0 \\ 0 & 0 & 0.73 & 0.97 \\ 0 & 0 & 0.83 & 0.01 \end{pmatrix}$$

$$P_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, P_2 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

$$M_1'' = \begin{pmatrix} 0.42 & 0.57 & 0 & 0 \\ 1 & 0.24 & 0 & 0 \\ 0 & 0 & 0.14 & 0.34 \\ 0 & 0 & 0.74 & 0.01 \end{pmatrix}$$

$$M_2'' = \begin{pmatrix} 0.83 & 0.45 & 0 & 0 \\ 0.25 & 0.81 & 0 & 0 \\ 0 & 0 & 0.13 & 0.58 \\ 0 & 0 & 0.43 & 0.12 \end{pmatrix}$$

Denote $M_1 = M_1' \cdot P_1 \cdot M_1''$ and $M_2 = M_2' \cdot P_2 \cdot M_2''$. Encryption of p_1 is realized by multiplying M_1', P_1 and M_1'' sequentially. p_2 is encrypted by multiplying M_2', P_2 and M_2''. The encrypted index is:

$$\begin{aligned} I_1 &= \left(\vec{p_1}' \cdot M_1,\ \vec{p_1}'' \cdot M_2\right) \\ &= \left(0.84\ 0.63\ 0.17\ 0.09\ 0.4\ 1.3\ 0.36\ -0.11\right). \end{aligned}$$

The trapdoor can be generated in similar ways. We generate $\vec{q}$ from q, and split $\vec{q}$ to $\vec{q}\,'$ and $\vec{q}\,''$. Then we multiply $\vec{q}\,'$ with $(M_1'^{-1})^T$, $(P_1^{-1})^T$, $(M_1''^{-1})^T$, and multiply $\vec{q}\,''$ with $(M_2'^{-1})^T$, $(P_2^{-1})^T$, $(M_2''^{-1})^T$. We get:

$$\mathcal{T} = \left(-0.65\ 1.1\ -0.05\ 2.03\ 0.77\ -0.27\ 1.84\ -0.43\right).$$

The inner product of I_1 and $\mathcal{T}$ is 0.99.

However, this scheme doesn't solve the problem of index updating. So we adopt the bloom filter to support index updating.

B. Combining kNN-based Scheme with Bloom Filter

We can make a combination of bloom filter and kNN-based MRSE scheme to gain the ability of updating.

In this Scheme, we use bloom filter bf_i, instead of the term-document boolean vector p_i to represent a document D_i. Then we can use matrix described previously to encrypt bf_i and get I_i. To improve efficiency, the length of bloom filters should be chosen according to the number of terms included in the document, rather than a fixed length.

However, the same query vector can't suit index vectors of different length at the same time. Consider a collection containing 3 documents. Assume that to guarantee that the false positive is below 5%, I_1 and I_2 should be longer than 100, I_3 should be longer than 120. If we set $n' = 10$ and bloom filters as their minimum length, then in the query step, server will need 2 query vectors, the one's length is 100, the other's length is 120. This setting result in a calculation of 2520 floating operations, 2200 operations in generating two query vectors, and 320 in calculating the inner product. If we set all bloom filters' length to 120, then the calculation of inner product will increased by 40. However, because there is no need to generate the query vector of length 100 now, the calculation of trapdoor generation will reduced by 1000, and the whole procedure will be faster than former. So the length of bloom filters should be designed carefully if we want to minimize the amount of calculation.

In practice, we take an approximate solution instead of the optimist solution. To guarantee the false-positive, we assume there are b_i documents whose bloom filter's length should be no less than l_i. We sort the l_i as $l_i > l_{i+1}$. The algorithm runs as follow:

Require:
set of length: $\mathcal{L} = l_i$, where $i = 1, 2, \cdots, r$ and $l_i > l_{i+1}$;
corresponding document number: b_i;
the scale of submatrix n';
Ensure:
set of chosen length: $\mathcal{C} = \{c_j\} \subseteq \mathcal{L}$, where $c_j > c_{j+1}$;
1: $\mathcal{C} = \{c_1\}$, $c_1 = l_1$
2: $i = 1$, $k = 1$
3: **for** $j = 1$ to r **do**
4: **if** $((l_i - l_j) \cdot b_j > l_j \cdot c_k)$ **then**
5: $i = j$, $k = k + 1$
6: add $c_k = l_i$ to $\mathcal{C}$
7: **end if**
8: **end for**
9: **return** $\mathcal{C}$;

We can generate bloom filter with $\mathcal{C} = \{c_i\}$. If $c_j >= l_i > c_{j+1}$, then a bloom filter of length l_i should be extended to length c_j.

The whole system is described as follow.

- **KeyGen**
  Generate a vector S and six pairs of matrices M_1', $(M_1')^{-1}$, M_1'', $(M_1'')^{-1}$, M_2', $(M_2')^{-1}$, M_2'', $(M_2'')^{-1}$, P_1, P_1^{-1}, P_2, P_2^{-1}.
- **BuildIndex**
  Generate bloom filter $\vec{bf_i}$ from the document D_i, $i = 1, 2, \cdots, m$.
  Split bloom filter $\vec{bf_i}$ to $\{\vec{bf_i}', \vec{bf_i}''\}$ with secret vector S.
  Get I_i by multiplying $\vec{bf_i}'$ and $\vec{bf_i}''$ with matrices M_1', P_1, M_1'' and M_2', P_2, M_2''.
  Upload I_i to the server.
- **GenTrapdoor**
  Generate bloom filter $\vec{qf}$ from the query Q.
  Split bloom filter $\vec{qf_i}$ to $\{\vec{qf_i}', \vec{qf_i}''\}$ with secret vector S.
  Get T by multiplying $\vec{qf_i}'$ and $\vec{qf_i}''$ with matrices $(M_1')^{-1}$, P_1^{-1}, $(M_1'')^{-1}$ and $(M_2')^{-1}$, P_2^{-1}, $(M_2'')^{-1}$.

TABLE I
TIME COST (S)

Scheme	KeyGen	BuildIndex	GenTrapdoor	Query
kNN-based MRSE		1.64×10^5	662	360
Proposed Scheme	18	703	15.1	54.5

TABLE II
SPACE COST (MB)

Scheme	QueryKey	Index	Trapdoor
kNN-based MRSE	4880	2121	0.27
Proposed Scheme	54.8	536	1.88

Request the server with T.

- **Query**
  Compute the inner product similarity between D_i and trapdoor T.
  Sort and return the result list R.
- **Update**
  Generate bloom filters $\vec{bf}_j$ for new document D_j, $j = m+1, m+2, \cdots, m+t$.
  Split and encrypt $\vec{bf}_j$ to D_j as building index with secret key.
  Upload E_j to the server.

V. RESULT AND COMPARISON

In this section, we demonstrate a thorough experimental evaluation of the proposed technique on the TREC data [19]. We pick a subset randomly, consists of 7594 documents, 16864 different terms, total size is 37.5 MB. The experiment is implemented by C++ language on a Computer with Pentium 4 3.00 GHz Processor, on Windows XP system. For proposed scheme, we set $n' = 100$. The performance of our method is compared with the original MRSE scheme. Bloom filter doesn't satisfy our security demands and is hence not in comparison.

A. Efficiency

The proposed scheme and the kNN-based scheme is described in detail as in previous sections, except the KeyGen algorithm. In our scheme, We adopt Gauss-Jordan elimination to calculate the matrices' inverse. The time of generating key is tolerable because of the small scale of the matrix. The secret key of kNN-based MRSE scheme includes two large scale matrix pairs, and its calculation is time-consuming, so we do not simulate it.

Time cost and space cost are listed as table I and table II. As shown in the table, our scheme is 200 times as fast as original scheme when building index, 40 times for generating trapdoor, and 6 times for querying. The kNN-based MRSE scheme needs a storage space 80 times as large as proposed scheme for the query key, and 4 times for the index. The efficiency of index construction step of proposed scheme is even comparable with plaintext index, which cost 470s.

Because the proposed scheme need to generate different trapdoor vectors for bloom filters of different lengths, the size of trapdoor is larger than the kNN-based MRSE scheme.

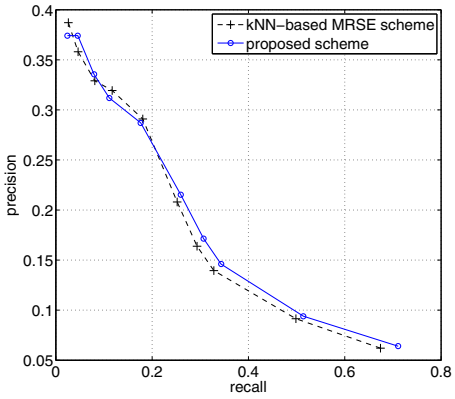

Fig. 3. Retrieval results shown as recall-precision curve.

Larger trapdoor needs a longer time for transmission, however, this disadvantage can be redeemed by the advantages gained in trapdoor generation and query step.

B. Precision and Recall

Bloom filter will cause Pseudo-positive in the result. To evaluate the retrieval result, we adopt recall-precision curve of the standard results, which is widely used in the traditional information retrieval field. The result is shown as fig. 3.

For a clear comparison, we don't add dummy terms in query. As shown in fig. 3, though bloom filter has false-positive, its retrieval result is almost the same as kNN.

VI. CONCLUSION

In this paper, we make a overview of the system model and security demands for cloud service. For meeting the challenge of supporting multi-keyword semantic without privacy breaches while supporting index updating, we make a combination of kNN-based MRSE scheme and bloom filter, propose a new multi-keyword search scheme based on inner product similarity. Experiment results show that our scheme is more efficient than the kNN-based MRSE scheme.

As our future work, we will try to get access to more accurate retrieval result on the same security level.

ACKNOWLEDGEMENT

This work was supported in part by the Natural Science Foundation of China under Grant 61170234 and Grant 60803155, by the Strategic and Piloted Project of CAS under Grant XDA06030601, and by the National Science and Technology Major Project of China under Grant 2010ZX03004-003.

REFERENCES

[1] T. Espiner, *Salesforce tight-lipped after phishing attack,* http://www.zdnet.com/salesforce-tight-lipped-after-phishing-attack-3039290616/, 11, 2007

[2] Q. Hardy, *A Summer Storms Disruption Is Felt in the Technology Cloud,* http://www.nytimes.com/2012/07/02/technology/ amazons-cloud-service-is-disrupted-by-a-summer-storm.html, 7, 2012

[3] D. Song, D. Wagner, and A. Perrig, *Practical techniques for searches on encrypted data,* in Proc. of S& P, 2000
[4] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, *Public key encryption with keyword search,* in Proc. of EUROCRYPT, 2004.
[5] Y.-C. Chang and M. Mitzenmacher, *Privacy preserving keyword searches on remote encrypted data,* in Proc. of ACNS, 2005.
[6] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, *Searchable symmetric encryption: improved definitions and efficient constructions,* in Proc. of ACM CCS, 2006.
[7] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi, *Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions,* J. Cryptol., vol. 21, no. 3, pp. 350C391, 2008.
[8] A. Shamir, *Identity-based cryptosystems and signature schemes,* CRYPTO 1984. LNCS, vol. 196, pp. 47C53, 1984.
[9] J. Katz, A. Sahai and B. Waters, *Predicate encryption supporting disjunctions, polynomial equations, and inner products,* Proc. Int'l Conf. Advances in Cryptology (EUROCRYP'08), 2008.
[10] D. Boneh, A. Sahai and B. Waters, *Functional Encryption: Definitions and Challenges,* Theory of Cryptography, 2011.
[11] C. Gentry, *Fully Homomorphic Encryption Using Ideal Lattices,* In the 41st ACM Symposium on Theory of Computing (STOC), 2009.
[12] E.-J. Goh, *Secure Indexs,* Technical Report 2003/216,Cryptology ePrint Archive, http://eprint.iacr.org/, 2003.
[13] C. Bosch, R. Brinkman, P. Hartel, and W. Jonker, *Conjunctive Wildcard Search over Encrypted Data,* Secure Data Management, 2011.
[14] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, *Secure ranked keyword search over encrypted cloud data,* in Proc. of ICDCS10, 2010.
[15] R. Agrawal, J. Kiernan, R. Srikant and Y. Xu, *Order preserving encryption for numeric data,* SIGMOD '04, 2004.
[16] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, *Privacy-Preserving Multi-keyword Ranked Search over Encrypted Cloud Data,* INFOCOM '11, 2011.
[17] S. Zerr, E. Demidova, D. Olmedilla, W. Nejdl, M. Winslett, and S. Mitra, *Zerber: r-confidential indexing for distributed documents,* in Proc. of EDBT, pp. 287C298, 2008.
[18] H. Williams and J. Zobel, *Searchable words on the Web*, International Journal on Digital Libraries, pp. 99-105, 2005.
[19] P. Bailey, N. Craswell, I. Soboroff, and A. P. Vries, *The CSIRO Enterprise Search Test Collection*, SIGIR Forum 41(2), pp. 42-45, 2007.