2012 IEEE 18th International Conference on Parallel and Distributed Systems

# Efficient Multi-Keyword Ranked Query on Encrypted Data in the Cloud

Zhiyong Xu[1,3], Wansheng Kang[1,2], Ruixuan Li[2], KinChoong Yow[1], and Cheng-Zhong Xu[1,4]

[1]Shenzhen Institute of Advanced Technology, Chinese Academy of Science, China
[2]School of Computer Science and Technology, Huazhong University of Science and Technology, China
[3]Math and Computer Science Department, Suffolk University, USA
[4]Electrical and Computer Engineering, Wayne State University, USA
Email: zxu@mcs.suffolk.edu, kwshinsdu@smail.hust.edu.cn, rxli@hust.edu.cn, {kc.yow, cz.xu}@siat.ac.cn

*Abstract*—**Cloud computing is becoming increasingly prevalent in recent years. It introduces an efficient way to achieve management flexibility and economic savings for distributed applications. To take advantage of computing and storage resources offered by cloud service providers, data owners must outsource their data onto public cloud servers which are not within their trusted domains. Therefore, the data security and privacy become a big concern. To prevent information disclosure, sensitive data has to be encrypted before uploading onto the cloud servers. This makes plain text keyword queries impossible. As the total amount of data stored in public clouds accumulates exponentially, it is very challenging to support efficient keyword based queries and rank the matching results on encrypted data. Most current works only consider single keyword queries without appropriate ranking schemes.**

**The multi-keyword query problem was being considered only recently. MRSE [1] is one of the first research works to define and address the problem of effective yet secure ranked multi-keyword search over encrypted cloud data. However, the keyword dictionary used in MRSE is static and must be rebuilt when the number of keywords in the dictionary increases. It also has severe out-of-order problems in the matching results and does not take the keyword access frequencies into account, which greatly affects its usability. In this paper, we propose a novel approach, called MKQE, to address these issues. Only minor changes in the dictionary structure have to be done when extra keywords are introduced. We also introduce new trapdoor generation and scoring algorithms to make in-order query results. Furthermore, the keyword access frequency is considered so as to select an adequate matching file set. We conduct extensive simulations and the results prove that our approach performs much better than previous solutions.**

**Keywords:** multi-keyword query, ranked query, encrypted data, cloud computing, heavy tail

## I. Introduction

Cloud computing is getting more and more attention from both academic and industry communities as it becomes a major deployment platform of distributed applications, especially for large-scale data management systems. End users can outsource their personal data onto public clouds, and then access those data at anytime and anywhere. In the cloud environment, resources allocated for each application can be scaled up and down according to the fluctuating demand. It adopts a pay-per-use resource sharing model which allows a user to pay only for the number of service units it consumes. Cloud computing provides a flexible and economic strategy for resource sharing. It can reduce hardware, software and data maintenance overheads. It can also offer a convenient communication channel to share resources among data owners and data consumers. With the popularity of cloud services such as Amazon Web Services [2], Microsoft Azure [3], Apple iCloud [4], Google AppEngine [5], more and more companies are planning to move their data onto the cloud.

Despite of its advantages, the cloud computing infrastructure faces very challenging tasks, especially on data privacy, security, and reliability issues. As data is placed on public clouds which are out of their trusted domains, data owners do not have direct control over their sensitive data and worry about possible data loss and/or illegal use of their private data. Usually, cloud servers are considered as curious and untrusted. Data owners will hesitate to adopt cloud technologies if there are risks of data exposure to the cloud service provider or a third party. Thus, providing sufficient security and privacy protections is very important, especially for applications processing health information, financial system and government data, etc. To prevent information disclosure, the mainstream solution is to encrypt private data before uploading onto the cloud. On one hand, it ensures that data is not visible to external users and cloud administrators. On the other hand, there are severe processing limitations on encrypted data. For example, standard plain text based searching algorithms are not applicable any more. To perform a keyword based query, the entire data set has to be decrypted even if the matching result set is very small. It poses unbearable query latency and incurs unacceptable computational overhead.

Current solutions use the following strategy to provide keyword based searching capabilities on encrypted data. First, a set of keywords are defined. An index vector is calculated for each individual file maintaining the information of which keywords this file contains. Then, an index file which combines all index vectors is generated. The index file is also encrypted. Second, both the encrypted data and the encrypted index file are uploaded onto data center servers in the cloud. The cloud servers can then support cipher text based queries as follows, a data consumer submits a keyword based query, and the encrypted keywords are sent to the cloud server. The cloud server conducts a search on the encrypted index and returns a list of most relevant files. The user makes the decision which files are needed and retrieve them from the server. After receiving encrypted files, the user decrypts the files with the associated key. This approach can guarantee the data security and preserve data privacy. During the whole process, no plain text data or keyword is visible to the cloud servers.

Although substantial research works [12-30] have been done to study keyword based queries on encrypted data, many of them only address single keyword queries. Others use disjunctive or conjunctive searches for multi-keyword queries which have great limitations in flexibility and performance. Furthermore, few include result ranking algorithm. MRSE [1] is one of the first works to define such a multi-keyword ranked query problem, and proposed a viable solution to address it. In MRSE, all keywords are defined in a dictionary and a keyword is identified by its location in the dictionary. Two randomly generated invertible matrices are used for data and file index encryptions. It uses the inner product of two vectors to build the trapdoor for secure keyword queries. It applies an internal ranking algorithm to determine the top k files to be returned to the data consumer. However,this approach suffers from three major drawbacks. First, it uses a static dictionary. If new keywords are added, the dictionary has to be rebuilt completely. Second, using its trapdoor generation algorithm, an out-of-order problem occurs. Such a problem results in that files with more matching keywords are likely excluded from the top k positions in the matching set. This means the data consumer may not be able to find the most relevant files. Lastly, MRSE does not consider the effects of keyword access frequencies, thus the files which contain frequent keywords might not be included in the top k return result.

In this paper, we design a new strategy called MKQE to address the aforementioned issues. In MKQE, we assume that the amount of data continues to increase from time to time. Accordingly, the keyword dictionary has to be expanded periodically. We propose a new dictionary construction paradigm, introduce a new trapdoor generation algorithm and take the keyword access frequencies into consideration to reduce the query latencies and generate better matching result sets.

In summary, we make the following contributions:

- We introduce partitioned matrices in the system design. The keyword dictionary can be expanded dynamically without touching the contents in the original dictionary. With this strategy, we can greatly reduce the overhead of the dictionary reconstruction as new keywords are added.
- We design a novel trapdoor generation algorithm. It can effectively reduce the impacts of dummy keywords on the ranking scores. With this new strategy, the out-of-order problem in the matching result set is solved.
- We take into account the keyword access frequencies are taken into account when we build the ranked list of the return results. The files which contain more frequently accessed keywords have higher chances to appear at the first k locations of the returned results than the files with less frequent accessed keywords. Thus, the data consumers have higher probabilities to retrieve the desired files.

The rest of the paper is organized as follows: Section II defines the problem. Section III introduces MRSE and its drawbacks. Section IV presents the system overview and the technical details of the proposed MKQE solution. Section VI discusses the experimental configurations and performance evaluations. Section VII describes the related works. Finally, Section VIII concludes the paper and gives the future work.

## II. Problem Definition

We aim to design a new approach to improve the performance for multi-keyword ranked queries on encrypted data in publc cloud servers. In this section, we define the problem.

### A. Notations

- F: the set of original files, assume there are m files. F is denoted as $F = (F_1, F_2, F_3 ... F_m)$
- C: the set of encrypted files, corresponding to the files in F. Denoted as $C = (C_1, C_2, C_3 ... C_m)$
- W: keyword dictionary, assume we have n keywords. W is denoted as $W = (W_1, W_2, W_3 ... W_n)$
- $F_{idx}$: the keyword set of each file, it is denoted as $F_{idx} = (F_{idx1}, F_{idx2}, F_{idx3} ... F_{idxn})$
- p: the index vectors for $F_{idx}$, p is denoted as $p = (p_1, p_2, p_3 ... p_n)$
- I: the encrypted index vectors for p. I is denoted as $I = (I_1, I_2, I_3 ... I_n)$
- $W_q$: a plain text query, assume it contains k keywords, and can be represented as $W_{kw1,kw2,...kwk}$
- q: for a query $W_q$, the corresponding query vector.
- $T$, the trapdoor for a query $W_q$, which is based on q.
- $\bar{R}$ the list of files in the returned matching result set. It is a sorted list, the order of the files is determined by the scores.

### B. Problem Description

Figure 1 depicts the problem we are trying to solve. There are three kinds of users, Data Owner (DO), Data Consumer (DC) and Cloud Service Provider (CSP). The DO identifies the files containing sensitive data. Those files are then encrypted using a standard symmetric algorithm such as DES [6] or AES [7]. It also specifies the set of keywords to form a keyword dictionary to be used for queries. In our discussions, we assume the keyword dictionary is dynamic, the DO may add keywords later depending on the changes in the set of sensitive files. For each file, an index vector is generated based on which keywords are contained in it. Those index vectors are also encrypted and combined together to generate an encrypted index file. Both the encrypted files and the encrypted index file must be uploaded onto the cloud servers. To facilitate secure queries, the DO needs to define a secret key. The secret key contains two invertible matrices and a bit vector. All the elements in the secret key are randomly generated (Details in the following section). The secret key is kept on the DO. After the above processes finish, the CSP has all the sensitive files stored in the encrypted formats, no information leakage occurs. Now, the DCs are able to conduct secure multi-keyword ranked queries on those encrypted files.

A query is executed as follows. First, a DC sends the set of keywords it is searching for to the DO. Next, the DO builds the trapdoor **T** based on this set of keywords using the secret key. T is returned to the DC who submits the request. Finally, the DC sends T to the CSP who stores the encrypted files. A matching process based on T is conducted and a set of encrypted files is identified. All these operations on the CSP are executed on the encrypted data only, there's no plain text information exposure to the CSP. Because the number of files which contains one or more keywords specified in the T could be very large, it results in considerable overhead to return all the results to the DC. In order to select an adequate set of files, a ranking algorithm is applied on these files based on the relevance scoring. Typically, the DC only has to retrieve the top k most relevant files. It sends the requests to the DO for the decryption keys and then decrypts these files.

### C. Threat Model

Normally, the CSP is considered as "honest but curious" [8]. In other words, the CSP is expected to execute the procedure or algorithm faithfully (without any changes), but it is eager to know the contents in the data files stored on its servers. We adopt two threat models defined in [3]. The first model is "Known Ciphertext Model". This model assumes that the CSP can only see the encrypted files and indexes. The second one is "Known Background Model". Here, the CSP may intentionally collect the statistical information of queries (trapdoors). Based on that, the CSP may be able to calculate which file contains a certain keyword.

The privacy preserving requirement is defined as follows: the CSP cannot obtain the information of which file contains which keywords directly from the indexes. The CSP does not know how many keywords are searched as well. For a certain query, the CSP can only calculate the scores of the files in the result set using a given ranking algorithm, but knows nothing about the matching keywords.

Furthermore, for two different queries which have the same set of keywords, the generated trapdoors must be different, and the scores must not be the same as well. In our paper, we consider the keyword dictionary can expand dynamically. Thus, we assume the CSP might

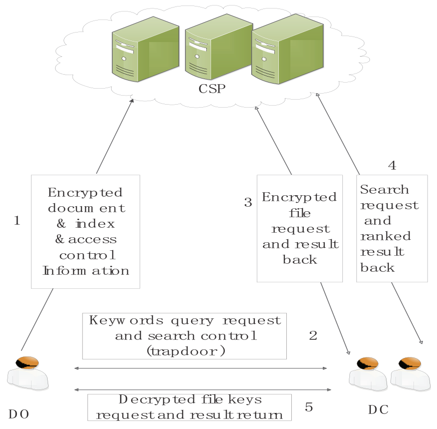
CSP
1
Encrypted document & index & access control Information
3
Encrypted file request and result back
4
Search request and ranked result back
Keywords query request and search control (trapdoor)
2
Decrypted file keys request and result return
5
DO
DC

Fig. 1. System Model

know how many new keywords are added. However, it does not pose a security threat since the CSP has no ideas of how many original keywords we have and what are the keywords newly added.

## D. Our Goals

In this paper, we aim to design a new algorithm to achieve the following goals.

**Multi-keyword Query** It can support efficient multi-keyword based query with low keyword, trapdoor and encryption overheads. It should be able to rank the query result as well.

**Dynamic Keyword Dictionary Size** It supports dynamic keyword dictionary

size. An efficient algorithm should be designed. Only minor changes are necessary when the keyword dictionary expands.

**In-order Ranking** The ranking algorithm should be effective. The system makes the in-order result set which contains the most relevant files in the top k locations with a high probability.

**Keyword Access Frequency Consideration** It must take the keyword access frequency into consideration. The ranking algorithm should rank the files with more popular keywords with higher scores.

## III. Existing Solution and Drawback

MRSE [1] is one of the first works to address multi-keyword ranked queries on encrypted data in the cloud. In this section, we briefly introduce their solution.

## A. MRSE Solution

In MRSE, the DO first defines a set of keywords and builds a dictionary containing them. Assume **n** keywords are identified, MRSE sort these keywords. Thus, the positions of the keywords in the dictionary are fixed. In order to provide the privacy preserving property, **u** dummy keywords are added. An extra random bit is placed at the end. After that, the DO generates two invertible matrices $M_1$ and $M_2$ with the sizes (n+u+1) × (n+u+1) and a vector *S* of the size (n+u+1).The DO uses then as secret key.

For each file, an index vector *p* with the same size as S is created. This vector is built as follows. If the file has the keyword $W_i$ ($1 \leq i \leq n$), the ith element p[i] is set to 1. Otherwise, p[i] is set to 0. Dummy keywords are represented by the elements located between positions n+1 and n+u. The values in these locations are set as p[i] = $\epsilon$ (n+1 $\leq$ i $\leq$ n+u). $\epsilon$ follows the uniform distribution M$(u' - c, u' + c)$ ($u'$ and c are the two parameters, and are described below). The (n+u+1)th dimension is always set to the constant 1.

For a multi-keyword query *q*, a query vector with the size (n+u+1) is constructed. The values of the first n locations are determined using the same method for the file index generation. For the locations between n+1 and n+u, MRSE uses the following strategy to set the values. First, v locations are chosen randomly, and the values of these locations are set to 1. Then, the values of all the remaining locations are set to 0. Finally, the system picks two numbers *t* and *r*. The value of the last dimension (n+u+1) is set to t, the values of all the other locations are multiplied by r. After that, a query vector is created. Normally, v is set to $u/2$, The sum of all the dummy keywords follow the normal distribution N($\mu$, $\sigma^2$), $u^{'}$ = $\mu$/v and c = $\sqrt{\frac{3}{v}}\sigma$.

The DO applies the same strategy to encrypt the file index p and the query vector q. Two vectors $\overrightarrow{p_1}$ and $\overrightarrow{p_2}$ are constructed for p, and two vectors $\overrightarrow{q_1}$ and $\overrightarrow{q_2}$ are constructed for q. Each vector has (n+u+1) locations. They follow the rule that,

for any number *j*, if S[j] = 1,

$$p[j] = \overrightarrow{p_1}[j] + \overrightarrow{p_2}[j], q[j] = \overrightarrow{q_1}[j] = \overrightarrow{q_2}[j] \tag{1}$$

Otherwise, if S[j]=0,

$$p[j] = \overrightarrow{p_1}[j] = \overrightarrow{p_2}[j], q[j] = \overrightarrow{q_1}[j] + \overrightarrow{q_2}[j] \tag{2}$$

The system then uses the invertible matrices $M_1$ and $M_2$ to compute the encrypted index for the file and the trapdoor for the query. The formulas are,

$$encrypt(p) = \{M_1^T\overrightarrow{p_1}, M_2^T\overrightarrow{p_2}\} \tag{3}$$

$$encrypt(q) = \{M_1^{-1}\overrightarrow{q_1}, M_2^{-1}\overrightarrow{q_2}\} \tag{4}$$

During the query q execution, the DC sends the trapdoor to the CSP. The CSP uses "inner product similarity" [9] to compute the scores. The scores measure the coordinate matching between the indexes and the query. Finally, the first k results with the highest scores are returned to the DC. The formula to calculate the score is,

$$\begin{aligned} score &= encrypt(p) \times encrypt(q) \\ &= \{M_1^T\overrightarrow{p_1}, M_2^T\overrightarrow{p_2}\} \times \{M_1^{-1}\overrightarrow{q_1}, M_2^{-1}\overrightarrow{q_2}\} \\ &= M_1^T\overrightarrow{p_1} \times M_1^{-1}\overrightarrow{q_1} + M_2^T\overrightarrow{p_2} \times M_2^{-1}\overrightarrow{q_2} \\ &= \overrightarrow{p_1}^T M_1 \times M_1^{-1}\overrightarrow{q_1} + \overrightarrow{p_2}^T M_2 \times M_2^{-1}\overrightarrow{q_2} \\ &= \overrightarrow{p_1}^T\overrightarrow{q_1} + \overrightarrow{p_2}^T\overrightarrow{q_2} \\ &= p^T q \end{aligned} \tag{5}$$

Clearly, the more common keywords in the query vector and the file index, the higher the score a file will get.

## B. Drawbacks

MRSE is an effective mechanism which can partially solve the multi-keyword ranked query problem. However, it has the following drawbacks which affect its efficiency.

First, the keyword dictionary is static and is created at the beginning. MRSE does not provide a viable solution for dynamically expanding the set of keywords. In case new keywords are added, the keyword dictionary has to be reconstructed. The original set of index vectors cannot be used for the queries using the newly constructed trapdoors. Thus, the index vectors of all the files have to be recalculated from scratch. Such an approach incurs unacceptable overhead.

Second, in MRSE, the values of dummy keywords follow the normal distribution N($\mu$, $\sigma^2$). In our experiments, we found that such an approach has great impact on the file score. For any query, the scores that the matching files receive would vary widely. A file with more matching keywords may have a much lower score than another file which has fewer matching keywords. It results in severe out-of-order placement issue, which means that in MRSE, it has a high probability that highly relevant files may be excluded from the top k locations in the result set, especially when the number of matching keywords is small and the variance $\sigma$ is relatively big.

Third, MRSE does not take the access frequencies of the keywords into account. The ranking algorithm assumes that all the keywords are the same when calculating the scores. In other words, the system considers a file f1 to have a higher score than another file f2 if it has more matching keywords even if the set of keywords matching in f2 are more popular than the keywords matching in f1. However, in reality, keywords have vastly different popularity and popular keywords are always more preferable than less-popular keywords from the data consumer's point of view. Thus, the keyword access frequencies should be considered to better serve the DCs.

## IV. System Design

We propose a set of novel strategie for multi-keyword ranked query on encrypged data(shorted for *MKQE*) to address the above issues. In this section, we present the details of our solution.

### A. Overview

In MKQE, we adopt "inner product similarity" to quantitatively evaluate the coordinate matching like MRSE. MKQE also defines an index vector for each file based on the keywords it contains. Two invertible matrices and a bit vector are also used for the index vector encryption and the trapdoor generation. In MKQE, when a multi-keyword query comes, a query vector based on the set of requesting keywords is constructed. However, our approach differs from MRSE in the way new keywords are added into the dictionary. In MKQE, only minimal overhead is introduced in this scenario. We also modify the trapdoor generation mechanism to improve the in-oder ranking in the matching file set. Furthermore, we take the keyword access frequency distribution into consideration when creating the ranked list. Such a strategy can better reflect the real world situations.

### B. MKQE Framework

The MKQE system consists of the following components:

- **Setup**: based on the sensitive data, the DO determines the keyword dictionary size **n**, the number of dummy keywords **u**, and then sets the parameter **d** = 1+u+n.
- **Keygen(d)**: the DO generates a secret key **SK** k1, two invertible matrices $M_1$ and $M_2$ with the dimension d $\times$ d, and a d-bit vector S.
- **Extended-Keygen(k1, z)**: if $z$ new keywords are added in the dictionary, the DO generates a new **SK** k2, two invertible matrices $M_1^{'}$ and $M_2^{'}$ with the dimension d+z $\times$ d+z, and a new (d+z)-bit vector $S^{'}$.
- **Build-Index(F, SK)**: for each file, the DO determines the set of keywords $F_{idx}$, and builds p for it. Then it encrypts the index vector with an SK (either k1 or k2). After that, all the encrypted indexes are added to I. All the files are encrypted with DES or AES, and added to C. Finally, upload I and C onto the CSP.
- **Trapdoor($\mathbf{W_q}$, SK)**: The DC sends a multi-keyword ranked query $W_q$ to the DO. The DO generates an index vector q and calculates the trapdoor T using an SK and sends it back to the DC.
- **Query(T, k, I)**: The query is sent to the CSP. The CSP runs the query on I and returns the most relevant top k scored files back to the DC.

### C. Detailed Design

*1) Vector Structure:* As we described in Section III, in MRSE, a file index vector contains three parts, and the order of these parts is fixed. The first n locations are used for the real keywords, followed by u locations for the dummy keywords, and the last dimension is the constant 1. This structure is not suitable in MKQE because the number of keywords changes from time to time. When there are new keywords introduced, we have to increase the number of locations for index vectors as well. If we add them in the end, the locations representing new keywords are not adjacent to the locations representing the original keywords. When executing a query, such a structure causes difficulty in checking the results of matrix operations.

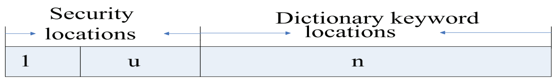

Fig. 2. The index and trapdoor structure in MKQE

To solve this issue, in MKQE, we reorganize the structure of the file index vectors. As shown in Figure 2, the secure locations are now placed at the beginning of the vector, followed by the n locations used for the real keywords. With this approach, for any position j in the vector, we have the following relation: if a file has the real keyword $W_j$, then in the corresponding index vector, p[1+u+j] = 1. Otherwise, it is 0. For query vectors, the same structure is applied.

*2) Extended-Keygen:* As we discussed in previous sections, a secret key can be used to encrypt file index vectors and query vectors. However, in MRSE, the size of the matrices and the vector in the secret key is determined by the keyword dictionary size n. If the dictionary is expanded, the value of n has to change. Thus, the current secret key cannot be used for file and index encryptions any more. In other words, the DO has to generate a new secret key, and the system has to encrypt all the file index vectors with the newly generated key again. As more and more data are accumulated in the cloud, we suspect that adding more query keywords is not a rare operation. Clearly, MRSE is not suitable for such a scenario, it incurs severe computational overhead. To resolve this issue, in MKQE, we introduce a new approach using partitioned matrix operations to reduce the computational overhead.

In MKQE, when there are new keywords added in the dictionary, we do not change the original secret key. Instead, we only add a new secret key to support queries for the newly added keywords. With this approach, we can keep the original secret key untouched, and avoid the expensive reconstruction process. The new algorithm is described as follows: Assume there are z new keywords added,

- Generates two invertible matrices $M_z$ and $M_z^{'}$ with the dimension $z \times z$. The system also generates a $z$-bit vector $S_z$.
- The new matrices $M_1^{'}$, $M_2^{'}$ and the secret key $S^{'}$ are generated with the formula:

$$M_1^{'} = \begin{pmatrix} M_1 & 0 \\ 0 & M_z \end{pmatrix} \qquad M_2^{'} = \begin{pmatrix} M_2 & 0 \\ 0 & M_z^{'} \end{pmatrix} \tag{6}$$

$$S^{'} = (S, S_z) \tag{7}$$

In MKQE, we use the matrix transpose and inverse operations to encryte file index vectors and query vectors. According to the block matrix theorem, we have the following formulas:

$$M_1^{'T} = \begin{pmatrix} M_1^T & 0 \\ 0 & M_z^T \end{pmatrix} \qquad M_2^{'T} = \begin{pmatrix} M_2^T & 0 \\ 0 & M_z^{'T} \end{pmatrix} \tag{8}$$

$$M_1^{'-1} = \begin{pmatrix} M_1^{-1} & 0 \\ 0 & M_z^{-1} \end{pmatrix} \quad M_2^{'-1} = \begin{pmatrix} M_2^{-1} & 0 \\ 0 & M_z^{'-1} \end{pmatrix} \tag{9}$$

Figure 3 shows the encryption procedure as new keywords are added. As we can observe, the first 1+u+n locations are untouched.

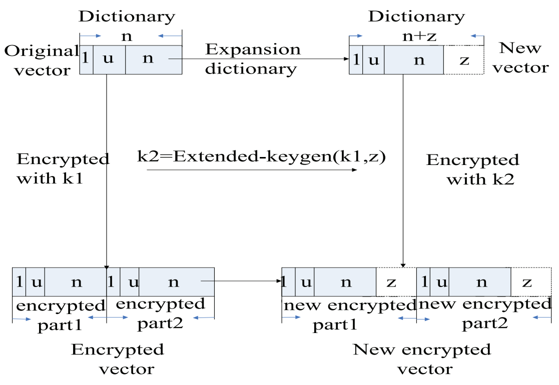

Fig. 3. Keyword Dictionary Expansion Operations

*3) Build-Index:* Compared to the MRSE algorithm, MKQE can greatly reduce the amount of computations when the keyword dictionary is expanded. For original files, the file index vectors only need minor changes by expanding the size by s. If a file has a newly added keyword j ($1 \leq j \leq z$), then in its corresponding index vector, p[1+u+n+j] is set to 1, otherwise it is 0. The first 1+u+n locations are untouched. For a new coming sensitive file, an index vector with the dimension 1+u+n+z has to be built. The values of the first 1+u security dimension are determined using the same strategy described in section 3. For each of the last n+z locations of the real keywords, its value is also decided based on the fact if the file has the corresponding keyword or not.

*4) Trapdoor Generation:* When a multi-keyword query q comes, a query vector is created using the strategy discussed in the previous section. Again, v number of these dummy locations are set to 1 and all the remaining locations are set to 0.

We use a score to determine the location of a file in the matching result set. For a file with an index $p_i$, its score is calculated as follows:

$$p_i \bullet q = r(x_i + \sum \epsilon_i^{(v)}) + t_i \tag{10}$$

$x_i$ represents the number of keywords appearing in both $p_i$ and $q$. $\epsilon_i^{(v)}$ represents the v dummy keywords. Typically, v is set to $u/2$. $t_i$ is the constant dimension. According to the previous description, the sum of the selected dummy keywords follows the normal distribution N($\mu$, $\sigma^2$). Thus, the value of the sum is in the range $[v * (u' - c), v * (u' + c)]$, where $u'$ = $\mu$ /v and c = $\sqrt{\frac{3}{v}}\sigma$. The difference between the maximal and minimal values is $2 \times v \times \sqrt{\frac{3}{v}} \times \sigma$. In our experiments, we found that such a large variance can significantly affect the ranking of the results,and files which contain popular keywords could have very low scores and are not placed in the top k locations in the result set. We define such a problem as an out-of-order problem.

To improve the in-order ranking result and maintain the privacy preserving property, we change the structure of query vectors. For all the real keywords in the dictionary, the values in the corresponding locations are multiplied by a random number *r*. For all the locations representing dummy keywords, they are multiplied by another number *r2*. r2 is calculated as follows:

$$r2 = \frac{Random(0, r)}{(v * MAX(abs(u - c), abs(u + c)))} \tag{11}$$

After $q$ is created, we use the formulas 1, 2 and 4 to split and encrypt q. Finally, the trapdoor T is generated. Now, for an index $p_i$ and the query q, the matching score is computed as follows:

$$p_i \bullet q = I_i \bullet T = r \times x_i + r2 \times \sum \epsilon_i^v + t_i \tag{12}$$

The values of $r2 \times \sum \epsilon_i^v$ is within [-r, r]. Here, $x_i$ is a common keyword in $p_i$ and q. Since the sum of the chosen dummy keywords is within [-r, r], it has much smaller impact on the score calculation. The out-of-order problem is resolved. We analysis the in order ranking in following. We define a variable $b$ to denote the sum of the dummy keyword values in p, it is within [-r, r]. Now, let's assume there are two files i and j. $p_i$ matches n1 keywords with score1, $p_j$ matches n1+1 keywords with score2. If score1 $\geq$ score2, we define it as an out-of-order error. It is caused by the inequation $b_i$+n1*r $\geq$ $b_j$+(n1+1)*r. This could happen if $u'$ -c $<$ 0 and $u'$+c $>$ 0. However, we can avoid this situation if $u'$ -c $>$ 0 (by changing the mean and variance values in the normal distribution). Because $b_i$ and $b_j$ are both within (0, r], it is impossible that $b_i \geq b_j$+r. When the variable is set less than r/2, then the the value of $b$ is within (-r/2,r/2). In our experiments, we found that even if the variable is set to r, the probability of out-of-order errors is extremely low in most situations. For example, if we set $b_i$ is within [0.5r, r] and $b_j$ is within [-r, -0.5r], according to the formula 11 and the normal distribution function, the probability of out-of-order errors is (1-F(v*c/2))*F(-v*c/2). If we set v to 7 and use the normal distribution N(0,1), the probability is only 0.0121%.

*5) Keyword Weights:* Our framework also takes the keyword access frequencies (weights) into consideration which is not addressed in other works linke MRSE. Substantial works [10], [11] have been conducted to analyze keyword based queries and found out that keywords have different access frequencies and this feature needs to be considered when generating the result rankings. In multi-keyword based query operations, if no keyword weight is used for result ranking, a file containing a smaller number of popular keywords may get a much lower score compare to another file which has more unpopular keywords, and it might be excluded from the top k result.

To resolve this problem, we make the following changes in MKQE. Each keyword is assigned with a weight. The weight is determined by the historical statistical information depending on a certain period of the past history. This can introduce greater flexibility and reflects the real world situations. Thus, for a multi-keyword query q, which contains the keyword set ($W_{kw1}$,$W_{kw2}$,...$W_{kwk}$), MKQE sets the corresponding weight ($w_1$,$w_2$,...$w_k$)for each keyword. Now, when we calculate the score, the weight of each keyword is multiplied.

The new formula to calculate the score is,

$$p_i \bullet q = r \times \sum w_j + r2 \times \sum \epsilon_i^v + t_i \tag{13}$$

where $w_j$ is the weight of any keyword $W_{kwj}$ which appears in both $p_i$ and q.

*6) Query:* With the trapdoor T, the CSP computes the score of each file in the encrypted index set I. After that,it sorts the results based on the scores, and the CSP only returns the top k files in the resulting set back to the DC. In our trapdoor algorithm, the score is calculated using the formula 12. When the keyword weight is considered, the values of the locations in the query vector are set according to their corresponding weights. Then, the scores is calculated using the formula 13.

## V. Security and Privacy Analysis

For two separate queries, we choose different set of dummy keyword locations. The probability of two $\sum \epsilon^v$ having the same value is less than $1/2^{(v)}$. Since r2 is randomly chosen for each individual query, the CSP cannot reproduce the same queries and extract useful information. We might have a problem when the value of random(0,r) is set too small in two queries, the CSP can construct a pseudo-query if it has some background information about the two trapdoors. This is because, in this scenario, the variance effects of r2 become too small. Thus, in order to avoid this, each time when

TABLE I
KEY GENERATION OVERHEAD (S), STARTING FROM 1000 KEYWORDS

| | 1000 | 2000 | 4000 | 6000 | 8000 |
|---|---|---|---|---|---|
| MRSE | 3.8 | 33.9 | 320 | 1013 | 2540 |
| MKQE | 3.8 | 4.1 | 36.1 | 38.1 | 40.7 |

TABLE II
COMPARISON OF TIME CONSUMPTION ON TRAPDOOR GENERATION (MS)

| | 1000 | 2000 | 4000 | 6000 | 8000 |
|---|---|---|---|---|---|
| MRSE | 27 | 45 | 97 | 192 | 313 |
| MKQE | 27 | 42 | 90 | 153 | 238 |

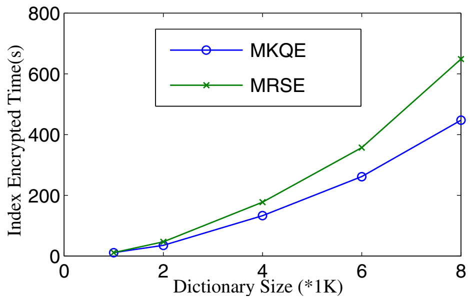

Fig. 4. Index Encryption Time Comparison with 2000 file indexes encrypted

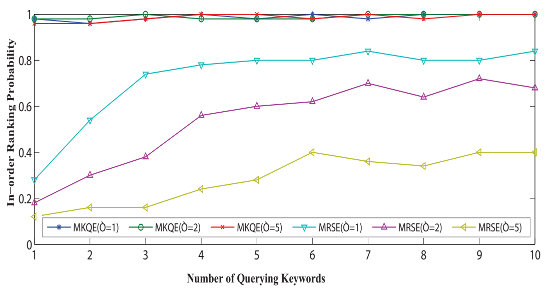

Fig. 5. The in-order ranking probability with different $\sigma$

we choose the variable random(0,r), a large value should be used. Although how many keywords are increased in the newly generated dictionary may be known by the CSP, it does not result in a real data privacy problem since the CSP does not know what are these keywords.

## VI. PERFORMANCE EVALUATION

We conduct extensive simulation experiments to evaluate MKQE. Our testbed consists of a server with a Xeon Processor 2.40GHz *16 core. The OS is Ubuntu 12.04, with Java JRE 7.0 and J2EE 7.0 SDK are installed. The total number of simulation code is 5000 lines written in java language. The data type of the elements in invertible matrics is double, and these elements are randomly generated using jema.jar package. In our experiments, we chose a real-world dataset, Enron Email Dataset [12] as the underlying dataset, and randomly select various numbers of emails from it.

### *A. The Overhead of Keyword Dictionary Expansion*

A superior feature of MKQE over previous solutions is that it can naturally extend the keyword dictionary set at the minimal cost. In this set of experiments, we first compare the time consumption of the proposed Extended-Keygen algorithm with the MRSE algorithm when new keywords are added to the dictionary. Then, we compare the performance of the index construction and the query execution delay using MKQE with the original MRSE.

The time consumption includes two parts: the time of generating a bit vector and two inverse random matrices, the time of the transport and inverse of two matrices computations. As we can observe from Table I, MKQE is much more efficient than MRSE. As the number of keywords in the dictionary increases from 1000 to 2000, the overhead to generate the new secret keys in MRSE is increased almost 9 times, while in MKQE, the difference is minimal. The time consumption in MRSE is about 825% higher than MKQE. Furthermore, the performance gap becomes even wider as more and more keywords are added. When the number of keywords increases from 6000 to 8000, the overhead in MRSE is 2540s, which is about 62 times higher than MKQE. Apparently, MKQE achieves much better performance than MRSE because it reuses the original set of indexes during the keyword expansion. The number of elements it needs to produce in the matrices is much smaller than in MRSE. In MRSE, all the elements have to be regenerated.

Figure 4 compares the time to generate the file indexes with different sizes of the keyword dictionary. In this experiment, we assume there are 2000 file indexes need to be encrypted. MKQE takes less time than MRSE in all scenarios. When the number of keywords in the dictionary is 2000, MKQE spends 35.3 seconds to generate the encrypted indexes, while MRSE takes 46.9 seconds to do so. MKQE only spends 75.3% of time MRSE takes. As the dictionary becomes larger, MKQE still outperforms MSRE. When the dictionary has 8000 keywords, MRSE takes 648.8 seconds to encrypt indexes, and MKQE only spends 447.4 seconds which is 68.9% of time MRSE spends. This performance gain comes from the fact that in MKQE, the invertible matrices contain more elements with 0 than matrices used in MRSE. This property makes the matrix inner product operations more time efficient than in MRSE. The larger the size of the keyword dictionary, the more performance gain we can achieve.

Each time a multi-keyword query is executed, a trapdoor has to be constructed on the DO. Thus, the trapdoor generation is a critical operation, and its performance has great influence on the system overall performance. Table II compares the trapdoor generation time consumption of MRSE and MKQE. From the result, we can see that it shows similar behaviors as the index encryption operation. In all situations, MKQE outperforms MRSE, and the performance gap becomes larger as the size of keyword dictionary increases. We believe this is because of the same reason as we mentioned above.

Finally, we also conduct the experiments to compare the query execution time between MRSE and MKQE, and we found that these two algorithms achieve comparable performance. Thus, we do not present the results here. However, from all the discussions above, we can draw the conclusions that MKQE has less time consumption than MRSE in all major encryption operations.

One advantage of our seamless expanding strategy is that we can use the newly generated secret key to search the original set of files. There is no need to restart the encrypt index generation operations for the original set of files. This feature provides great flexibility, especially for the distributed applications which cannot have service interruptions at any time.

### *B. In-order Result Evaluation*

We also compare the in-order ranking in the returned result set of MRSE with MKQE. In this experiment, we set the number of file indexes is 1000, the number of keywords in the dictionary is 1000, and the number of keywords in a query varies from 1 to 10. The sum of the dummy keywords follows the normal distribution with the mean u=0. We choose three different variance $\sigma$: 1, 2 and 5. For each query, the top 50 ranked files are returned. As we mentioned

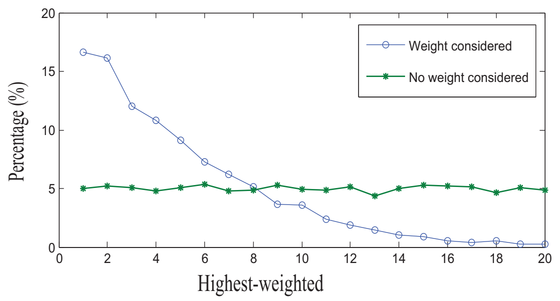

Fig. 6. Percentage of files which contaning highest-weighted keywords in the top 10 locations

above, for two files i and j, if $I_i$ matches less keywords and gets a lower score than $I_j$, we call it is an in-order ranking result. Otherwise, an out-of-order error occurs. Figure 5 shows the result.

As we can see from the results, for MRSE, the larger the variance $\sigma$, the lower the probability of in-order ranking. Furthermore, when the number of keywords in the query becomes smaller, the probability turns to be even lower. For example, when there is only 1 keyword and the variance is 5, the probability is less than 5% in MRSE algorithm. Even for the queries with 10 keywords, the in-order probability are only 40%, 60% and 82% for the variance 1, 2 and 5, respectively. Such a result is not very satisfactory, the DC has a pretty high chance to miss the files they really needs. While in MKQE algorithm, no matter how many keywords are included in a query, and no matter what is the variance $\sigma$, the in-order probability is way above 95%. Especially, when the number of keywords is large ($\geq$ 6), MRSE can correctly identify the matching files with the in-order probability at almost 100%. In another word, the DC has very high probability to retrieve the files it really needs.

As we discussed in Section IV, The reason MKQE achieves such a high in-order characteristic is because in MRSE, $\sigma$ has great impacts on the range the sum of the dummy keyword values locates. A large variance always results in a lot of out-of order errors. In MKQE, we carefully choose two parameters r and r2 to narrow the range, thus achieve better in-order performance.

### C. *Keyword Access Frequency Analysis*

Many research works [10], [11] have studied the multi-keyword query problem and found out that information retrieval applications often have convenient power-law constraints (also known as Zipf's Law and long tails), thus the access frequency (weight) always follows the heavy tail distributions. Typically, a DC prefers to see the files which have high weight keywords. However, the current solutions on the multi-keyword queries over encrypted data do not take this into account and the returned file set might contain biased result. In another word, the keywords are considered with the equal weights. Such an assumption violates the rules in the real world applications. To solve this issue, in MKQE, we add this factor to determine the top k locations in the query result.

In this set of experiments, we compare the query results in MRSE and MKQE when taking keyword weights into accounts. We submit 1000 queries, and each query has 4 keywords. The total number of keywords in the system is 1000. The weights of keywords follows the heavy tail distributions, and we count the percentage of the 20 highest-weighted keywords appeared in the matching result set. Here, we set k = 10, which means the first 10 files with the highest scores will be returned for each query. Figure 6 shows the result. X axis represents the highest-weighted keywords, the first one has the highest weight overall. Y axis shows the percentage of the returned files in which those keywords appears. As we can see from the figure, in MRSE, no matter how popular a keyword is, only 5% of the returned files have one of these keywords. While in MKQE, a keyword with a larger weight has much higher probability to appear in the result set. Clearly, due to the lack of the weight consideration, the data consumers using MRSE has less chances to get the files they really need. MKQE solve this problem by giving more preference on the highest-weighted keywords. We found that the probability of a file contains certain keywords also follows the heavy-tail distribution.

Although we use the heavy tail distribution for keyword weights, as we can see from the formulas 12 and 13, MKQE works for any distributions where the weights of the keywords are uneven.

## VII. Related Works

Cloud computing provides an efficient way for end users to access data anywhere and anytime. However, security and privacy concerns force data owners to encrypt sensitive data before uploading onto the cloud. Thus, supporting keyword-based searching capability on encrypted data is critical. The standard mechanism is to encrypt keyword indexes of stored files and upload them onto the cloud servers as well. The search is conducted with a secure key generated trapdoor on the data owner. S. Kamara et al [13] propose a possible architecture for a cryptographic storage service for cloud storage. When preparing data to be stored in the cloud, the data owner creates indexes and encrypts the data with a symmetric encryption scheme (e.g., AES) under a unique key. It then encrypts the indexes using a searchable encryption scheme and encrypts the unique key with an attribute-based encryption scheme under an appropriate policy. Finally, it encodes the encrypted data and indexes in such a way that the data verifier can later verify their integrity using a proof of storage. The similar strategy used in many other research projects as well. In [14], Song et al apply a deterministic encryption algorithm to encrypt the keywords, and use stream ciphers to post-encrypt keywords for security. Goh et al [15] adopt the bloom-filter technology to test if a keyword is included in an index, which can tolerate low probability of mistaken identification to speed up the queries. Other researches [16], [17] have made improvements on this issue by using xor operations between encrypted index vectors and query vectors. In [18], Wang defines a novel infrastructure on secure ranked query, which is based on the file length and term frequency. Liu et al [19] employ an asymmetric setting to construct searchable encryptions, in which users encrypt their indexes with a public key, and authorized users have a private key to conduct searches. Similar ideas are proposed in [20], [21], [22] as well. However, all these solutions focus on the single keyword query problem only.

Many research works have been conducted on multi-keyword queries to enrich search functionalities. Park et al[23] proposes a multi-keyword search scheme on encrypted data. It is based on the bilinear map, which was used in the single keyword query [19]. This paper defines two schemes for multi-keyword queries. Every index contains the same number of keyword fields, and each field is filled with a keyword. The keyword is encrypted with the public key. The trapdoor constructed with the privacy key must have the same number of keyword fields under a specified order. Research works in [24], [25], [26], [27], [28] adopt conjunctive searches over encrypted keywords. They also require that the indexes of the returned files must match all the query keywords. Recent works [29], [30], [31] propose a more general approach. They employ bilinear maps to encrypt the keywords and uses Lagrangian Coefficients to find the matching keywords. If one or more keywords are matched in an index, the corresponding file is returned. It can support both conjunctive and disjunctive searches. However, a disjunctive keyword search could result in undifferentiated results, and it is hard to design an effective ranked algorithm on the result. Thus, for a multi-keyword query, all the files which contain a subset of the searching keywords are returned. The large number of files makes it hard to find the ones the DC really needs.

None of the above algorithms can support multi-keyword ranked search. This question was first raised in [1]. As more enterprises and private data owners are migrating their data onto the cloud environment, it is increasingly important to provide an efficient solution. MKQE addresses this issue with a novel algorithm, it can support effective multi-keyword ranked queries without the drawbacks in [1].

## VIII. Conclusions and the Future Work

In this paper, we aim to provide a viable solution for multi-keyword ranked query problems over encrypted data in the cloud environment. We first define the problem, analyze the existing solutions and design a novel algorithm called MKQE to address the issues. MKQE uses a partitioned matrices approach. When the amount of encrypted data increases and more keywords need to be introduced, the searching infrastructure can be naturally expanded with the minimal overhead. We also design a new trapdoor generation algorithm, which can solve the out-of-order problem in the returned result set without losing the data security and privacy property. Furthermore, the weights of the keywords are taken into consideration in the ranking algorithm when generating the query result. The DC has high probability to retrieve the files they really need. The simulation experiments confirm that our approach can achieve better performance with a satisfactory security level.

In the future, we will explore new approaches to further enhance multi-keyword query capabilities. We are designing new algorithms to provide extra functionalities such as semantic query and fuzzy keyword query. We are also working on applying new storage techniques such as SSD to boost the query performance.

## Acknowledgments

This research is partially supported by National Natural Science Foundation of China under grants 61173170 and 60873225, Innovation Fund of Huazhong University of Science and Technology under grants 2012TS052 and 2011TS135.

## References

[1] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in *INFOCOM*, pp. 829–837, IEEE, 2011.

[2] "Amazon web services." http://aws.amazon.com.

[3] "Microsoft azure." http://www.windowsazure.com.

[4] "Apple icloud." https://www.icloud.com/.

[5] "Google appengine." https://appengine.google.com/.

[6] National Bureau of Standards, *Data Encryption Standard*. U. S. Department of Commerce, Washington, DC, USA, Jan. 1977.

[7] H. Dobbertin, V. Rijmen, and A. Sowa, *Advanced Encryption Standard – AES: 4th International Conference, AES 2004, Bonn, Germany, May 10-12, 2004 : Revised Selected and Invited Papers*. Lecture Notes in Computer Science, Springer, 2005.

[8] L. Kissner, *Privacy-Preserving Distributed Information Sharing*. PhD thesis, University of Carnegie Mellon, 2006.

[9] I. H. Witten, A. Moffat, and T. C. Bell, *Managing Gigabytes : Compressing and Indexing Documents and Images*. San Francisco, CA: Morgan Kaufmann, 2. ed., 1999.

[10] X. Z. Jie, J. Yu, and J. Doyle, "Heavy tails, generalized coding, and optimal web layout," in *In Proceedings of IEEE INFOCOM*, pp. 1617–1626, IEEE Press, 2001.

[11] S. Chaudhuri, K. Church, A. C. König, and L. Sui, "Heavy-tailed distributions and multi-keyword queries," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, (New York, NY, USA), pp. 663–670, ACM, 2007.

[12] W. W. Cohen, "Enron email dataset." http://www.cs.cmu.edu/enron/.

[13] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *Financial Cryptography and Data Security* (R. Sion, R. Curtmola, S. Dietrich, A. Kiayias, J. Miret, K. Sako, and F. Seb, eds.), vol. 6054 of *Lecture Notes in Computer Science*, pp. 136–149, Springer Berlin, Heidelberg, 2010.

[14] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Security and Privacy, 2000. S P 2000. Proceedings. 2000 IEEE Symposium on*, pp. 44 –55, 2000.

[15] E.-J. Goh, "Secure indexes." Cryptology ePrint Archive, Report 2003/216, 2003.

[16] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Applied Cryptography and Network Security* (J. Ioannidis, A. Keromytis, and M. Yung, eds.), vol. 3531 of *Lecture Notes in Computer Science*, pp. 391–421, Springer Berlin / Heidelberg, 2005.

[17] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, (New York, NY, USA), pp. 79–88, ACM, 2006.

[18] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, pp. 253 –262, june 2010.

[19] Q. Liu, G. Wang, and J. Wu, "An efficient privacy preserving keyword search scheme in cloud computing," in *Computational Science and Engineering, 2009. CSE '09. International Conference on*, vol. 2, pp. 715 –720, aug. 2009.

[20] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology - EUROCRYPT 2004* (C. Cachin and J. Camenisch, eds.), vol. 3027 of *Lecture Notes in Computer Science*, pp. 506–522, Springer Berlin / Heidelberg, 2004.

[21] 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," in *Advances in Cryptology CRYPTO 2005* (V. Shoup, ed.), vol. 3621 of *Lecture Notes in Computer Science*, pp. 205–222, Springer Berlin / Heidelberg, 2005.

[22] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *INFOCOM, 2010 Proceedings IEEE*, pp. 1 –5, march 2010.

[23] D. Park, K. Kim, and P. Lee, "Public key encryption with conjunctive field keyword search," in *Information Security Applications* (C. Lim and M. Yung, eds.), vol. 3325 of *Lecture Notes in Computer Science*, pp. 73–86, Springer Berlin / Heidelberg, 2005.

[24] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Applied Cryptography and Network Security* (M. Jakobsson, M. Yung, and J. Zhou, eds.), vol. 3089 of *Lecture Notes in Computer Science*, pp. 31–45, Springer Berlin / Heidelberg, 2004.

[25] L. Ballard, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data," in *Information and Communications Security* (S. Qing, W. Mao, J. Lpez, and G. Wang, eds.), vol. 3783 of *Lecture Notes in Computer Science*, pp. 414–426, Springer Berlin / Heidelberg, 2005.

[26] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Theory of Cryptography* (S. Vadhan, ed.), vol. 4392 of *Lecture Notes in Computer Science*, pp. 535–554, Springer Berlin / Heidelberg, 2007.

[27] R. Brinkman, *Searching in encrypted data*. PhD thesis, University of Twente, 2007.

[28] Y. Hwang and P. Lee, "Public key encryption with conjunctive keyword search and its extension to a multi-user system," in *Pairing-Based Cryptography Pairing 2007* (T. Takagi, T. Okamoto, E. Okamoto, and T. Okamoto, eds.), vol. 4575 of *Lecture Notes in Computer Science*, pp. 2–22, Springer Berlin / Heidelberg, 2007.

[29] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *Proceedings of the theory and applications of cryptographic techniques 27th annual international conference on Advances in cryptology*, EUROCRYPT'08, (Berlin, Heidelberg), pp. 146–162, Springer-Verlag, 2008.

[30] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, "Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption," in *Advances in Cryptology EUROCRYPT 2010* (H. Gilbert, ed.), vol. 6110 of *Lecture Notes in Computer Science*, pp. 62–91, Springer Berlin / Heidelberg, 2010.

[31] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," in *Theory of Cryptography* (O. Reingold, ed.), vol. 5444 of *Lecture Notes in Computer Science*, pp. 457–473, Springer Berlin / Heidelberg, 2009.