Efficient and Secure Ranked Multi-Keyword Search on Encrypted Cloud Data

Cengiz Örencik
Faculty of Engineering & Natural Sciences
Sabanci University, Istanbul, 34956, Turkey
cengizo@sabanciuniv.edu

Erkay Savaş
Faculty of Engineering & Natural Sciences
Sabanci University, Istanbul, 34956, Turkey
erkays@sabanciuniv.edu

ABSTRACT

Information search and document retrieval from a remote database (e.g. cloud server) requires submitting the search terms to the database holder. However, the search terms may contain sensitive information that must be kept secret from the database holder. Moreover, the privacy concerns apply to the relevant documents retrieved by the user in the later stage since they may also contain sensitive data and reveal information about sensitive search terms. A related protocol, Private Information Retrieval (PIR), provides useful cryptographic tools to hide the queried search terms and the data retrieved from the database while returning most relevant documents to the user. In this paper, we propose a practical privacy-preserving ranked keyword search scheme based on PIR that allows multi-keyword queries with ranking capability. The proposed scheme increases the security of the keyword search scheme while still satisfying efficient computation and communication requirements. To the best of our knowledge the majority of previous works are not efficient for assumed scenario where documents are large files. Our scheme outperforms the most efficient proposals in literature in terms of time complexity by several orders of magnitude.

1. INTRODUCTION

We are living in a highly networked environment, where huge amounts of data are stored in remote, but not necessarily trusted servers. There are several privacy issues regarding to accessing data on such servers; two of them can easily be identified: sensitivity of i) keywords sent in queries and ii) the data retrieved; both need to be hidden. A related protocol, Private Information Retrieval (PIR) [6] enables the user to access public or private databases without revealing which data he is extracting. Since privacy is of a great concern, PIR protocols have been extensively studied in the past [2, 6, 9, 11].

In today's information technology landscape, customers that need high storage and computation power tend to outsource their data and services to clouds. Clouds enable customers to remotely store and access their data by lowering the cost of hardware ownership while providing robust and fast services [12]. The importance and necessity of privacy preserving search techniques are even more pronounced in the cloud applications. Due to the fact that large companies that operate the public clouds like Google or Amazon may access the sensitive data and search patterns, hiding the query and the retrieved data has great importance in ensuring the privacy and security of those using cloud services.

We aim to achieve an efficient system where any authorized user can perform a search on a remote database with multiple keywords, without revealing neither the keywords he searches for, nor the contents of the documents he retrieves. Our proposed system differs from the previous works which assume that only the data owner queries the database [2, 4]. In contrast to previous works, our proposal facilitate that a group of users can query the database provided that they possess trapdoors for search terms that authorize the users to include them in their queries. Moreover, our proposed system is able to perform multiple keyword search in a single query and ranks the results so the user can retrieve only the top matches.

The contributions of this paper can be summarized as follows. Firstly, we provide formal definitions for the security and privacy requirements of keyword search on encrypted cloud data. Secondly, we propose an efficient ranked multi-keyword search scheme and formally prove that it is secure in accordance with the defined requirements. Thirdly, we propose a ranking method that proves to be efficient to implement and effective in returning documents highly relevant to submitted search terms. Lastly, we implement the proposed scheme and demonstrate that it is much more efficient than existing methods in literature.

The rest of this paper is organized as follows. In Section 2, we discuss the related previous works. Section 3 gives the system model and the basics of the scheme. The privacy requirements are defined in Section 3.1. Then we provide detailed description of the proposed scheme in Section 4. Section 5 explains the ranking method. Query randomization, along with its formal analysis is presented in Section 6. In Section 7, we formally prove that the proposed method satisfies the privacy requirements. An extensive cost analysis of the proposed technique (in terms of both communication and computation) and implementation details are presented in Section 8. Finally, Section 9 gives the concluding remarks of the paper.

PAIS 2012, March 30, 2012, Berlin, Germany.

2. RELATED WORK

The problem of Private Information Retrieval was first introduced by Chor et al. [6]. Recently Groth et al. [9] propose a multi-query PIR method with constant communication rate. However, any PIR-based technique requires highly costly cryptographic operations in order to hide the access pattern. This is inefficient in the large scale cloud system and as an alternative approach, privacy preserving search is employed which aims to hide the content of the retrieved data instead of which data is retrieved.

Ogata and Kurosawa [10] show privacy preserving keyword search protocol based on RSA blind signatures. The scheme requires a public-key operation per item in the database for every query and this operation must be performed on the user side. Freedman et al. [8], proposed an alternative implementation for private keyword search that uses homomorphic encryption and oblivious polynomial evaluation methods. The computation and communication costs of this method are quite large since every search term in a query requires several homomorphic encryption operations both on the server and the user side. A recent work proposed by Wang et al. [13] allows ranked search over an encrypted database by using inner product similarity. However, this work is only limited to single keyword search queries.

One of the closest methods to our solution is proposed by Cao et al. [3]. Similar to our approach presented here, it proposes a method that allows multi-keyword ranked search over encrypted database. In this method, the data owner needs to distribute a symmetric-key which is used in trapdoor generation to all authorized users. Additionally, this work requires keyword fields in the index. This means that the user must know a list of all valid keywords and their positions as a compulsory information to generate a query. This assumption may not be applicable in several cases. Moreover, it is not efficient due to matrix multiplication operations of square matrices where the number of rows are in the order of several thousands.

Wang et al. [14] propose a trapdoorless private keyword search scheme, where their model requires a trusted third party which they named as the Group Manager. We adapt their indexing method to our scheme, but we use a totally different encryption methodology to increase the security and efficiency of the scheme.

3. FRAMEWORK OF THE PROPOSED METHOD

The problem that we consider is privacy-preserving keyword search on private database model, where the documents are simply encrypted with the secret keys unknown to the actual holder of the database (i.e. Cloud Server). We consider three roles coherent with previous works [3, 14]:

- *Data owner*, who is the actual owner of the database. The data owner collects and/or generates the information in the database and lacks the means (or is unwilling) to maintain/operate the database,
- *Users* are the members in a group who are entitled to access (part of) the information of the database,
- *Server*, is a professional entity (e.g. cloud) to offer information services to authorized users. It is often required that the server is oblivious to content of the database it maintains, the search terms in queries and documents retrieved.

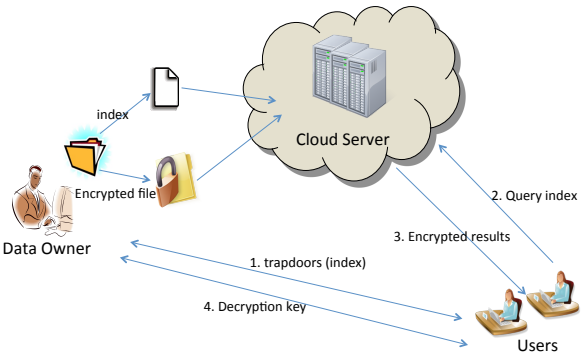


Figure 1: Architecture of the search method

The overview of the proposed system is illustrated in Figure 1. We assume that the parties are semi-honest and do not collude with each other to bypass the security measures.

In Figure 1, steps and typical interactions between the participants of the system are illustrated. In an offline stage, the data owner creates a search index for each document. The search index file is created using a secret key based trapdoor generation function where the secret keys[1] are only known by the data owner. Then, the data owner uploads these search index files to the server together with the encrypted documents. We use symmetric-key encryption as the encryption method since it can handle large document sizes efficiently. This process is referred as the *index generation* henceforth and the *trapdoor generation* is considered as its one of the steps.

When a user wants to perform a keyword search, he first connects to the data owner. He learns the trapdoors (cf. Step 1 in Figure 1) for the keywords he wants to search for, without revealing the keyword information to the data owner. Since the user can use the same trapdoor for many queries containing the corresponding search term, this operation does not need to be performed every time the user performs a query. After learning the trapdoor information, the user generates the query (referred as *query generation* henceforth) and submits it to the server (cf. step 2 in Figure 1). In return, he receives metadata[2] for the matched documents in a rank ordered manner as will be explained in subsequent sections. Then the user retrieves the encrypted documents (cf. Step 3 in Figure 1) he chooses after analyzing the metadata that basically conveys *a* relevancy level of the each matched document, where the number of documents returned is specified by the user. Finally, the user interacts with the data owner in order to decrypt the documents and get the corresponding plaintext (cf. Step 4 in Figure 1); and in the process the data owner does not learn the documents

[1]More than one key can be used in trapdoors for the search terms.

[2]Metadata does not contain useful information about the content of the matched documents.

that it is assisting to decrypt.

3.1 Privacy Requirements

The privacy definition for search methods in the related literature is that the server should learn nothing but the search results [3]. We further tighten the privacy over this general privacy definition and establish a set of privacy requirements for privacy-preserving search protocols. A multi-keyword search method must provide the following user and data privacy properties (first intuitions and then formal definitions are given):

1. (Data Privacy) No one but the user can learn the actual retrieved data.
2. (Index Privacy) The search index or the query index do not leak any information about the corresponding keywords.
3. (Trapdoor Privacy) Given one trapdoor for a set of keywords, the server cannot generate another valid trapdoor.
4. (Non-Impersonation) No one can impersonate a legitimate user.

DEFINITION 1. *(Data Privacy) A multi-keyword search protocol has data privacy, if there is no polynomial time adversary A that, given the retrieved encrypted data and the corresponding encrypted secret key, learns any information about the data.*

DEFINITION 2. *(Index Privacy) A multi-keyword search protocol has index privacy, if for all polynomial time adversaries A that, given two different keyword lists L_1 and L_2 and an index $\mathcal{I}_b$ generated from the keyword list L_b where $b \in_R \{0,1\}$, the advantage of A in finding b is negligible. The advantage of A is the absolute value of the difference between its success probability and $1/2$.*

DEFINITION 3. *(Trapdoor Privacy) A multi-keyword search protocol has trapdoor privacy, if for all polynomial time adversaries A that, given a valid trapdoor for a set of keywords, A cannot generate a valid trapdoor for its subset.*

DEFINITION 4. *(Non-Impersonation) A multi-keyword search protocol has non-impersonation property, if there is no adversary A that can impersonate a legitimate user U with probability greater than ϵ where ϵ is the probability of breaking the underlying signature scheme.*

Section 7 shows that the proposed method satisfies these privacy requirements.

4. THE PRIVACY-PRESERVING RANKED MULTI-KEYWORD SEARCH

In this section, we provide the details for the crucial steps in our proposal, namely index generation, trapdoor generation, and query generation.

4.1 Index Generation

Recently Wang et al. [14] proposed a conjunctive keyword search scheme that allows multiple-keyword search in a single query. We use this scheme as the base of our index creation scheme.

The original scheme uses forward indexing, which means that a searchable index file for each document is maintained to indicate the search terms existing in the document. In the scheme of Wang et al. [14], a secret cryptographic hash function that is secretly shared between all authorized users is used to map every search term to a sequence of l bits where $l = rd$ for some integers r and d. Using a single secret hash function shared by several users forms a security risk since it can easily leak to the server. Once the server learns the hash function, he can break the system if the input set is small. There are approximately 25000 commonly used keywords in English [1] and users usually search for a single or two keywords. For such small input sets, given the hashed index for a query, it will be easy for the server to identify the queried keywords by performing a brute-force attack. For instance, assuming that there are approximately 25000 possible keywords in a database and a query submitted by a user involves two keywords, there will be $25000^2 < 2^{28}$ possible keyword pairs. Therefore, approximately 2^{27} trials will be sufficient to break the system and learn the queried keywords.

We propose using a trapdoor based system where the trapdoor can only be generated by the data owner through the utilization of secret keys. The usage of secret keys eliminates the feasibility of a brute force attack. The rest of the index creation is very similar to [14] as explained in the following.

While generating the search index for a document R that has keywords $\{w_1, \ldots, w_m\}$, we take HMAC (Hash-based Message Authentication Code) of each keyword which gives $l = rd$ bits output (HMAC: $\{0,1\}^* \rightarrow \{0,1\}^l$). Let x_i be the output of HMAC for input w_i and index of a keyword w_i be denoted as I_i where I_i^j represents the j^{th} bit of I_i, i.e. $I_i^j \in GF(2)$ where GF stands for Galois field [7]. The index of a keyword w_i, $I_i = (I_i^{r-1}, \ldots, I_i^j, \ldots, I_i^1, I_i^0)$ is calculated as follows.

The l-bit output of HMAC, x_i can be seen as an r-digit number in base-d, where each digit is d bits. Also let $x_i^j \in GF(2^d)$ denotes the j^{th} digit of x_i and we can write

$$x_i = x_i^{r-1}, \ldots x_i^1, x_i^0.$$

After this, each r-digit output is reduced to r-bit output with the mapping from $GF(2^d)$ to $GF(2)$ as shown in Equation 1.

$$I_i^j = \begin{cases} 0 & \text{if } x_i^j = 0 \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

As a last step in index generation, the bitwise product of indices of all keywords (I_i for $i = 1 \ldots m$) in the document R is used to obtain the final searchable index $\mathcal{I}_R$ for document R as shown in Equation 2.

$$\mathcal{I}_R = \odot_{i=1}^{m} I_i \quad (2)$$

where $\odot$ is bitwise product operation. The resulting index $\mathcal{I}_R$ is an r-bit binary sequence and its j^{th} bit is 1 if for all i, j^{th} bit of I_i is 1 and 0 otherwise.

In the next subsection, we explain the technique used to generate trapdoors from the index files of each document.

4.2 Trapdoor & Query Generation

Search index files for the documents in the database are generated by the data owner using secret keys. A user who wants to include a search term in his query, needs the corresponding trapdoor from the data owner since he does not

know the secret keys used in the index generation. Asking for the trapdoor openly would violate the privacy of the user against the data owner, therefore a technique is needed to hide the trapdoor asked by the user from the data owner.

We adopt the idea from [8] to distribute keywords into a fixed number of bins depending on their hash values. More precisely, every keyword is hashed by a public hash function, and certain number of bits in the hash value is used to assign the keywords to these bins. The number of bins and the number of keywords in each bin can be adjusted according to security and efficiency requirements of the system.

In our proposal for obtaining trapdoors, we utilize a public hash function with uniform distribution, named `GetBin` that takes a keyword and returns a value in $\{0, \ldots, (\delta - 1)\}$ where δ is the number of bins. All the keywords that exist in a document are mapped by the data owner to one of those bins using the `GetBin` function. Note that δ is smaller than the number of indexed keywords so that each bin has several elements in it to provide obfuscation. The `GetBin` function has uniform distribution, therefore each bin will have approximately equal number of items in it. Moreover, δ must be chosen deliberately such that there are at least ϖ items in each bin where ϖ is a security parameter. The same key can be used in the index generation phase for all the keywords in the same bin.

When the user connects to the data owner to obtain the trapdoor for a keyword, he first calculates the bin IDs of keywords and sends these values to the data owner. The data owner then returns the secret keys of the bins requested for, which can be used by the user to generate the trapdoors[3] for all keywords in these bins. Alternatively, the data owner can send trapdoor of each keywords in corresponding bins resulting in an increase in the communication overhead. However, the latter method relieves the user of computing the trapdoors. After obtaining the trapdoors, the user can calculate the query index in a similar manner to the method used by the data owner to compute the search index files. More precisely, if there are n search terms in a user query, the following formula is used to calculate the privacy-preserving query given that the corresponding trapdoors (i.e. $I_{j_1}, \ldots, I_{j_n}$) are available to the user:

$$Q = \odot_{i=1}^{n} I_{j_i}.$$

Finally the user sends this r-bit query index value to the server.

The server receives the query of the user that hides the users' keywords. The users' keywords are protected against disclosure since the secret keys used in trapdoor generation is chosen by the data owner and never revealed to the server. In order to avoid impersonation, the user signs his messages.

4.3 Oblivious Search on the Database

A user's query index, in fact, is just an r-bit binary sequence (independent of the number of search terms in it) and therefore, index searching consists of as simple operations as binary comparison only. If the search index of the document ($\mathcal{I}_R$) has 0 for all the bits, for which the query index (Q) has also 0, then the query matches to that document as shown in Equation 3.

$$\text{result}(Q, \mathcal{I}_R) = \begin{cases} \text{match} & \text{if } \forall j \; Q^j = 0 \; \Rightarrow \mathcal{I}_R^j = 0 \\ \text{not match} & \text{otherwise} \end{cases} \tag{3}$$

Note that given a query index, it should be compared with search index of each document in the database.

Then the server sends metadata of the matching documents to the user. The metadata is the search index of that document where the user can perform further analysis to learn more about the relevancy of the document. After analyzing the indices, the user retrieves ciphertexts of the matching documents of his choice from the server. The user obtains the key from the data owner and decrypts the results. The method employed by the user to learn decryption keys is explained in Section 4.4.

For improving the security, the data owner can change the HMAC keys periodically. Each trapdoor will have an expiration time. After this time, the user needs to get a new trapdoor for the keyword he previously used in his queries. This will alleviate the risk when the HMAC keys are compromised.

4.4 Document Retrieval

The problem in retrieving the relevant documents is that users may not want documents, which they request, to be exposed, since their contents may be sensitive and they are usually directly related to sensitive search terms in their queries. In our scheme, the server can return the requested documents to the user.

In the proposed protocol, we need the data owner or its delegate that does not collude with the server, to be active. The usage of an active delegate for the data owner is a common approach that is coherent with previous works [3, 13]. As explained in Section 3, the data owner encrypts documents with a symmetric-key encryption method using a different secret key for each document. The server should not be able to decrypt those ciphers since this would imply that the server learns the content of the document the user requests for. Therefore, the data owner encrypts the symmetric-keys with a public-key encryption method, which has blinding capability, and stores the encrypted symmetric-keys in the server. In cryptography, blinding is a technique, whereby an agent can compute a cryptographic function (e.g. signing and decryption), without knowing either the real input or the real output of the function [5]. We choose the RSA as the public-key encryption, which supports blinding.

Assume that the user requests the document R. He receives the RSA encryption of the symmetric-key (sk), namely $RSA_e(sk)$, where e denotes the public-key of the data owner. The user does not know the private key of the RSA (i.e. d), therefore he needs the data owner to perform the decryption of sk without showing $y = RSA_e(sk)$, which would reveal the document he retrieves. The user employs the blinding technique and interacts with the data owner for decrypting the RSA encryption without learning the private key d. Firstly, y is blinded by a random blinding factor c chosen by the user as $z = c^e y \mod N$, where N is the RSA modulus. Then, the user sends the blinded result z to the data owner, who decrypts it using his private key and returns the result ($\bar{z} = z^d \mod N$) back to the user. Finally the user un-blinds the result using the blinding factor as $sk = \bar{z}c^{-1} \mod N$. The data owner cannot determine which secret key

[3]In fact, I_i, which is calculated for the search term w_i as explained in Section 4.1 is the trapdoor for the keyword w_i.

it is decrypting since the ciphertext is blinded, hence random looking.

5. RANKED SEARCH

The multi-keyword search method explained in Section 3 checks whether queried keywords exist in a document or not. If the user searches for a single or more keywords, there will possibly be many correct matches where some of them may not be useful for the user at all. Therefore, it is difficult to decide as to which documents are the most relevant. We add ranking capability to the system by adding extra index information for frequently occurring keywords in a file. With ranking, the user can retrieve only the top τ matches where τ is chosen by the user.

In order to rank the documents, a ranking function is required, which assigns relevancy scores to each document matching to a given search query. One of the most widely used metrics in information retrieval is the term frequency [15]. Term frequency is defined as the number of times a keyword appears in a document. Instead of using term frequency itself, we assign relevancy levels based on the term frequencies of keywords.

We assume that there are η levels of ranking in our proposed method for some integer $\eta \geq 1$. For each document, each level stores an index for frequent keywords of that document in a cumulative way in descending order. This basically means that i^{th} level index includes all keywords in the $(i+1)^{th}$ level and the keywords that have term frequency for the i^{th} level. The higher the level, the higher the term frequency of the keywords is. For instance, if $\eta = 3$, level 1 index includes keywords that occur at least once in the document while levels 2 and 3 include keywords that occur at least, say 5 times and 10 times[4], respectively. There are several variations for relevancy score calculations [15] and we use a very basic method. The relevancy score of a document is calculated as the number representing the highest level search index that the query index matches.

All the keywords that exist in a document are included in the first level search index of that document as explained in Section 4.1. The other higher level indices include the frequent keywords that also occur in its previous level, but this time they have to occur the number of times, which are at least the term frequency of the corresponding level. The highest level includes only the keywords that have the highest term frequency. In the oblivious search phase, the server starts comparing the user query against the first level indices of each document. The matching documents found as a result of the comparison in the first level are then compared with the search indices in the other levels according to the Algorithm 1.

In this method, some information may be lost due to the ranking method employed here. Rank of two documents will be the same if one involves all the queried keywords infrequently and the other involves all the queried keywords frequently except one infrequent one. The rank of the document is identified with the least frequent keyword of the query. We tested the correctness of our ranking method by comparing with a commonly used formula for relevance score calculation [13], which is given in the following:

[4]The number of levels and the term frequency of each level can be chosen in any convenient way.

Algorithm 1 Ranked Search

```
for all documents R_i do
  Compare(level_1 index of R_i , query index)
  j = 1
  while match do
    increment j
    Compare (level_j indices of R_i, query index)
  end while
  rank of R_i = highest level that match with query index
end for
```

$$Score(W,R) = \sum_{t \in W} \frac{1}{|R|} . (1 + \ln f_{R,t}) . \ln \left(1 + \frac{M}{f_t}\right). \quad (4)$$

Here W, $f_{R,t}$, f_t, and M denote the set of searched keywords, the term frequency of term t in file R, the number of files that contain term t, and the number of files in the database, respectively. $|R|$ is the length of the file R.

We use a synthetic database to compare the two ranking methods. We assume that there are 1000 files of equal lengths in the database and 3 keywords are searched for. We further assume that the number of files containing the queried keywords (f_t) is equal to 200 and only 20 of those contain all 3 keywords. Term frequencies of the keywords in the 20 matching files are chosen uniformly randomly between 1 and 15 and $\eta = 5$ in our proposed ranking method.

In 40% of the time, the top match for a given relevance score in Equation 4, is also the top match for our proposed ranking method, and 100% of the time in the top 3 matches of our ranking method. Additionally in 80% of the time, at least 4 of the top 5 matches for the given relevance score is also in the top 5 of our proposed ranking method. Note that since we assume f_t is the same for all $t \in W$, changing f_t has no effect on the performance of both methods. As can be observed from these experimental figures, while the performance of the proposed method is in acceptable levels, the choice of the method parameters (especially η and term frequency of each level) depends very much on the characteristics of the database and the documents.

While this new method necessitates an additional r-bit storage per level for a document, it reduces the communication overhead of the user since matches with low rank documents will not be retrieved unless the user requests. Considering η search indices are stored instead of a single search index per document, storage overhead for indexing mechanism increases η times due to ranking. This additional cost is not a burden for the server since the index sizes are usually negligibly small compared to actual document sizes.

6. QUERY RANDOMIZATION

Search pattern is the information that can be inferred from the queries of a user by linking one of his queries to another where both contain the identical keywords. The proposed scheme fails to hide the search pattern since the search indices are generated in a deterministic way. Any two search indices created from the identical keywords will be exactly the same. In order to hide the search pattern of a user we introduce randomness into the index generation phase. This process is known as *query randomization*, which should be carefully implemented so that the indices do not leak infor-

mation about the search patterns. In this section we analytically demonstrate the efficiency of the proposed query randomization method.

For introducing non-determinacy into the search index generation, we choose U random keywords that do not exist in the dictionary (i.e. they are simply random strings). We add these U random keywords in every document index along with the genuine keywords. When generating the query (i.e. query index), the user adds randomly selected V keywords ($V \leq U$) together with the genuine search terms. The number of different choices of V keywords out of a total of U keywords is calculated as $\binom{U}{V}$, which is maximized when $U = 2V$.

We can formalize the problem as follows. Let

$$\mathcal{Q}_i = \{Q_{i1}, Q_{i2}, \ \ldots, Q_{i\mu}\}$$

be the set of query indices that are generated from the same search terms using different random keywords. Furthermore, let $\mathcal{Q}_x$ be the set of all possible other query indices. Given two query indices $Q_i \in \mathcal{Q}_i$ and Q_j, identifying whether $Q_j \in \mathcal{Q}_i$ or $Q_j \in \mathcal{Q}_x$ should be hard.

We use the Hamming distance metric for evaluating the similarity of two query indices, which is defined as the number of different bits in the corresponding positions in the queries. We define two new functions to calculate analytically the Hamming distance.

- $F(x)$ is the expected number of 0's in an index with x keywords.

- $C(x)$ is the expected number of 0's that overlap in a x keyword query index (Q_a) compared with a single keyword query index (Q_b).

Note that r is the size of an index, d is the reduction value (cf. Section 4.1) and $Q[i]$ is the i^{th} bit of Q.
The functions are calculated as follows:

$$\begin{aligned} F(1) &= \frac{r}{2^d} \\ F(x) &= F(x-1) + F(1) - C(x-1) \\ C(x) &= \sum_{i=0}^{r-1} P(Q_a[i] = 0 \ \&\& \ Q_b[i] = 0) \\ &= r\frac{F(x)}{r}\frac{F(1)}{r} = \frac{F(x)}{2^d} \end{aligned}$$

The expected Hamming distance between two query indices (i.e. Q_1 and Q_2) with x keywords each, where they have $\bar{x}$ common keywords and $x - \bar{x}$ different keywords is calculated as

$$\begin{aligned} \Delta(Q_1, Q_2) = & \sum_{i=0}^{r-1} P(Q_1[i] \neq Q_2[i]) \\ = & \ rP(Q_1[1] \neq Q_2[1]) \\ = & \ r\,[P(Q_1[1] = 0)P(Q_2[1] = 1|Q_1[1] = 0) \\ & + P(Q_1[1] = 1)P(Q_2[1] = 0|Q_1[1] = 1)] \\ = & \ r\left[\frac{F(x)}{r}\left(\frac{F(\bar{x})}{F(x)}.0 + \frac{F(x) - F(\bar{x})}{F(x)}.\frac{r - F(x)}{r}\right)\right. \\ & \left. + \frac{r - F(x)}{r}\left(\frac{F(x)}{r}\right)\right] \\ = & \ \frac{(F(x) - F(\bar{x}))\,(r - F(x))}{r} + \frac{F(x)(r - F(x))}{r} \end{aligned} \tag{5}$$

where $P(A|B)$ is the conditional probability of A given B.

Each query index chooses V keywords out of $U = 2V$ random keywords. While comparing two arbitrary queries, the expected number of keywords that exist in both of them (E_O) is calculated as

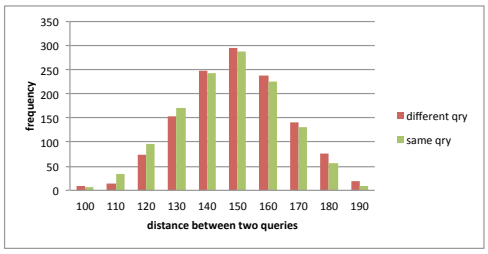
$$E_O = \sum_{i=0}^{V} \frac{\binom{V}{i}\binom{V}{V-i}}{\binom{2V}{V}} i = \frac{V}{2}. \tag{6}$$

The first query chooses V keywords and the probability that i ($i \leq V$) keywords chosen by the second query also exist in the first query is calculated as follows: i keywords are chosen from the set of keywords that is selected also by the first query and $V - i$ are chosen from the set of unselected keywords. Then we use summation to calculate the expected value in (6).

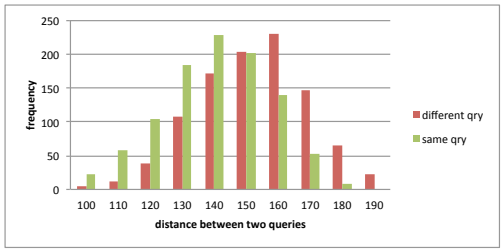
Using the formulae in (5) and (6), one can demonstrate that the distance between two query indices $Q_i \in \mathcal{Q}_i$ and $Q_j \notin \mathcal{Q}_i$ is sufficiently close to the distance between $Q_{i\alpha} \in \mathcal{Q}_i$ and $Q_{i\beta} \in \mathcal{Q}_i$ for the parameters $V = 30$ and $U = 60$.

To demonstrate the usefulness of our analysis, we conducted an experiment using synthetic data for the case, where adversary does not know the number of genuine keywords in a query. We generate a synthetic data for a set of query indices with the parameters $V = 30$ and $U = 60$ being fixed. The set contains a total of 250 query indices, where the first 50 indices contain 2 genuine keywords each, the second 50 indices contain 3 genuine keywords each, and so on. And finally, the last set of 50 indices contains 6 genuine keywords each. We create another set which contains only 6 query indices, which include 2, 3, 4, 5, and 6 genuine keywords, respectively. The distances between pairs of query indices, in which one query is chosen from the former set and the second one from the latter, are measured to obtain a histogram as shown in Figure 2(a). Consequently, a total of $250 \times 5 = 1250$ distances are measured. We also obtain another histogram featuring a total of 1250 distances between pairs of query indices, whereby query indices in a pair contain the same genuine search terms with different random keywords. Both of the histograms are given in Figure 2(a), where it can be observed that adversary basically needs to make a random guess whether given two query indices contain the same keywords or not.

We also conducted a similar experiment, in which we assume that the adversary has the knowledge of the number of search terms in a query index. We generate a set containing a total of 1000 indices, whose subsets with 200 indices each



(a) Histogram for the distances for two arbitrary query indices and for two query indices that are generated from the same search terms



(b) Histogram for the distances for two arbitrary query indices and for two query indices that are generated from the same search terms where the number of search terms in the query is 5

Figure 2: Histograms

contain 2, 3, 4, 5 and 6 genuine keywords, respectively. We then create a single query index with 5 genuine keywords. We measured the distances of the single query to all 1000 indices in the former set of indices to create a histogram (i.e. a total of $200 \times 5 = 1000$ distances are measured). We compared this with the histogram for 1000 measurements of the distance between two query indices with five identical search terms as shown in Figure 2(b). As can be observed from the histogram in Figure 2(b), 20% of the time, distances between two queries are 150 and they are totally indistinguishable. In 45% of the time, the distances are smaller than 150, where the adversary can guess $Q_j \in \mathcal{Q}_i$ with 0.6 confidence. In 35% of the time, the distances are greater than 150 and the adversary can guess $Q_j \notin \mathcal{Q}_i$ with 0.7 confidence. In accordance with these results, one can guess whether the query indices are generated from the same search terms or not correctly with 0.6 confidence under the assumption that the number of genuine keywords in the query is known. Hence, this information should be kept secret which is the case in our proposed method. Note that the query randomization does not change the response, which needs to be sent over a secure channel.

6.1 Error Rates

The indexing method that we employ includes all the information on keywords in a single r-bit index file. Despite the fact that the hash function employed in the implementations is collision-free, after the reduction and bitwise product operations there is a possibility that index of a query may wrongly match with an irrelevant document which is called as a false accept. The false accept rates given in Figure 3 are calculated as

$$FAR = \frac{number\ \ of\ \ incorrect\ \ matches}{number\ \ of\ \ all\ \ matches}$$

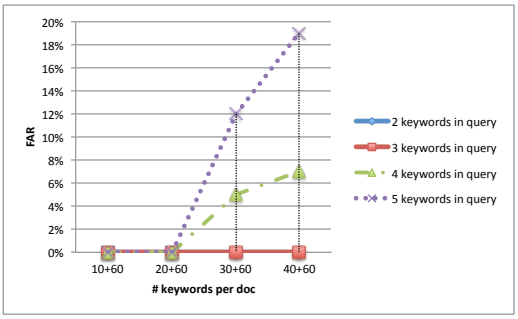


Figure 3: False Accept Rates: $d = 6$, $r = 448$ **bits,** $U = 60$, $V = 30$

for queries with 2, 3, 4 and 5 keywords where index size (r) is 448 bits. The false accept rates rapidly increase after 40 keywords per document; but we assume that there are less than 40 keywords per document in our experiments. If more keywords are required per document, false accept rates can be reduced by increasing the reduction parameter d while keeping the final index size r constant (i.e. choosing a longer HMAC function). Although computing longer HMAC functions will also increase the cost of the index generation, since the index size r is constant the communication cost and storage requirements do not increase. Optimized value for index size should be chosen considering the requirements of the applications, for which the proposed method is used.

7. PRIVACY OF THE METHOD

The privacy-preserving multi-keyword search (MKS) method must provide the user and data privacy requirements specified by definitions in Section 3.1. This section is devoted for the formal proofs that the proposed method satisfies these privacy requirements.

THEOREM 1 (DATA PRIVACY). *The proposed MKS method satisfies data privacy in accordance with Definition 1.*

PROOF. The security of the encrypted data relies on the underlying symmetric-key encryption method. The secret key of the encryption is known by the data owner and it is privately shared with the user by employing blinding as explained in Section 4.4. The adversary can listen the channel and learn the RSA encrypted secret key ($y = RSA_e(sk)$), the blinded version of y ($z = c^e y \mod N$) and the blinded version of the secret key ($\bar{z} = c(sk) \mod N$). Since c is a random number, z and $\bar{z}$ are treated as random values. Assuming that the data owner's private key d and the chosen random number c are not compromised, the knowledge of z and $\bar{z}$ does not reveal any information on the symmetric-key sk. Without the knowledge of sk, adversary cannot learn any information about the actual data. Additionally, the data owner cannot learn sk without knowing c. Given ($\bar{z} = c(sk) \mod N$), without the knowledge of the blinding factor c, it is a random looking number, and hence is indistinguishable from any $\bar{z_2} = c_2(sk_2) \mod N$ where $c \neq c_2$. □

THEOREM 2 (INDEX PRIVACY). *The proposed MKS method satisfies index privacy in accordance with Definition 2.*

PROOF. Assuming that the adversary A is not an authorized user, A cannot ask the data owner for the trapdoors of keywords in L_1 and L_2. Additionally, since the trapdoor information is sent to the authorized users in an encrypted form, A cannot learn that information through channel listening. First assume $L_1 = L_2$ such that they have exactly the same genuine keywords with different random keywords. As shown in Section 6, indices for such two keyword lists are indistinguishable from indices with different keywords. Assume $L_1 \neq L_2$ such that they have different genuine keywords and different random keywords. In the worst case, let L_1 and L_2 include only a single genuine keyword. The HMAC function uses a randomly chosen 128 bit key which implies A needs to try 2^{127} different hash functions for each $\binom{U}{V}$ possible random choice on average to guess b with a confidence greater than 0.5, which is infeasible for all practical purposes. □

THEOREM 3 (TRAPDOOR PRIVACY). *The proposed MKS method satisfies trapdoor privacy in accordance with Definition 3.*

PROOF. Assuming that the adversary A is not an authorized user, A is not allowed to acquire query index for his search terms. Specifically, let A learns a query index $Q_{(w_i,w_j)}$ for two keywords (w_i, w_j) and V random keywords, A cannot deduce the search index Q_{w_i} for a single keyword w_i from the received index. Let x of the r bits are 0 in $Q_{(w_i,w_j)}$ where x_i, x_j and x_R are the number of 0 bits resulting from keywords w_i, w_j and random keywords, respectively. Note that A does not know the values of x_i or x_j, but without loss of generality, let $x_i = x_j = \frac{r}{2^d}$ and furthermore let they do not overlap. On average, there will be 20 times more 0's resulting from random keywords than x_i (i.e. $F(V)/F(1)$). A valid search index Q_{w_i} must contain all the x_i 0 bits and must not contain any of the x_j 0 bits corresponding to the trapdoor of w_j. The probability of finding a valid trapdoor Q_{w_i} which is denoted as $P(v_T)$, is smaller than choosing x_i 0's from x where none of them are from the x_j 0's.

$$P(v_T) < \frac{\binom{x_i}{x_i}\binom{x_j}{0}\binom{x-(x_i+x_j)}{y}}{\binom{x}{x_i+y}} \approx \frac{\binom{18x_i}{y}}{\binom{20x_i}{x_i+y}} \approx 2^{-9} \qquad (7)$$

where y is the number of 0's chosen from random keywords. Note that the adversary cannot verify whether a trapdoor is valid for w_i, hence brute-force search is not possible. □

THEOREM 4 (NON-IMPERSONATION). *The proposed MKS method satisfies non-impersonation property in accordance with Definition 4.*

PROOF. All the communication that is from the user to the data owner is authenticated by a signature with the user's private key. We assume that the private key information of the authorized users is not compromised and we further assume that the server is semi-honest. In order to impersonate an authorized user with an RSA public-key e_u, A needs to learn the private key d_u where $e_u d_u = 1 \mod \phi(N)$. Therefore, the probability of impersonating an authorized user is ϵ where ϵ is the probability of breaking the underlying RSA signature scheme. The communication between the user and the server does not require any authentication. Since the user does not provide his identity during the communication with the server, impersonation is not an issue. □

8. COMPLEXITY

In this section, we present an extensive cost analysis of the proposed technique. The communication and computation costs will be analyzed separately. Especially, low costs on the user side are crucial for rendering the proposed technique feasible for mobile applications, where the users usually perform the search through resource-constrained devices such as smart phones.

- Communication Costs:

  Three steps in the proposed method are identified, where communication is required: i) for learning the trapdoor, ii) for sending the query and receiving the results, and iii) finally for learning the decryption key.

  1. *Between the user and the data owner, for learning the trapdoor*: To build the query, the user first determines the bin IDs of the keywords he wants to search for and send these values to the data owner. Let γ be the number of the keywords the user wants to search for. Then the user sends at most $32 \cdot \gamma$ bits to the data owner together with a signature since each bin ID is represented by a 32 bit integer. The data owner replies with the HMAC keys that belongs to those bins. The reply is encrypted with the user's public-key, so the size of the result is $\log N$. Note that if two query keywords happen to map to the same bin, then sending only one of them will be sufficient since their responses will be the same.
  2. *Between the user and the server, for query*: After learning the trapdoor keys, the user calculates the query index and transmits it to the server. The size of the index is r bits, independent from γ, so the user transmits only r bits. Let α be the number of documents matched with the query and N be the RSA modulus. The server returns the indices of the matching documents which is $\alpha \cdot r$ bits in total. If the user requests θ of those documents where $\theta \leq \alpha$, then the server returns the symmetric-key encryption of the documents and the RSA encryption of the corresponding symmetric keys. The size of the former is approximately the same as document size itself, while the latter is $\log N$ bits. Therefore, the total amount of data sent from the server is $\theta \cdot (doc\_size + \log N)$. Note that the additional communication cost of the proposed method is $\theta \cdot \log N + \alpha \cdot r$ bits and the rest is the data part which must be sent even no security is used. In the case where the ranking is used, only the top τ matches are returned to the user by the server instead of α where $\tau \leq \alpha$.
  3. *Between the user and the data owner, for decryption*: In order to decrypt the symmetric-key encryption of the document, the user should first blinds the corresponding key which is encrypted with RSA. The user first transmits the blinded ciphertext to the data owner, which is $\log N$-bit long and receives decryption of it, which is again $\log N$-bit long.

The communication costs are summarized in Table 1.

	Trapdoor	Search	Decrypt
User	$32 \cdot \gamma + \log N$	r	$\log N$
Data Owner	$\log N$	0	$\log N$
Server	0	$\alpha \cdot r +$ $\theta \cdot (doc\_size + \log N)$	0

Table 1: Communication costs incurred by each party (in bits)

- Computation Costs:

  Among the three parties participated in the protocol, computation cost of the user is the most crucial one. The data owner and the server can be implemented on quite powerful machines, however the users may be using resource-constrained devices.

  1. *User:* After receiving trapdoor keys from the data owner, query index is generated as explained in Section 4.1, which is essentially equivalent to performing hash operations[5]. Subsequent to the retrieval of encrypted documents, the user should perform one blinding operation over an RSA encryption, one signing for authentication purposes and symmetric-key decryption operations as explained in Section 4.4. These operations are equivalent to 3 modular exponentiations, 2 modular multiplications and one symmetric-key decryption operation per document retrieved.
  2. *Server:* The server only does the search operation, which is binary comparison of r-bit query index with σ database indices, each of which is again r-bit binary sequence. If the ranking is used, the query index should also be compared with indices of the matching documents. So the server performs η additional binary comparison of r-bit indices for each matching document, where η is the number of levels.
  3. *Data Owner:* The data owner creates the indices and symmetric-key encryptions of all documents; but these operations are performed only once in the initialization phase. Other than this, data owner is active while the user learns the trapdoors and decryption keys, which requires 2 modular exponentiations for each.

  The computation costs are summarized in Table 2.

User	1 hash and bitwise product 2 modular multiplication 3 modular exponentiation 1 symmetric-key decryption
Data Owner	initialization phase 4 modular exponentiation per search
Server	$\sigma\,\eta$ binary comparison over r bit indices

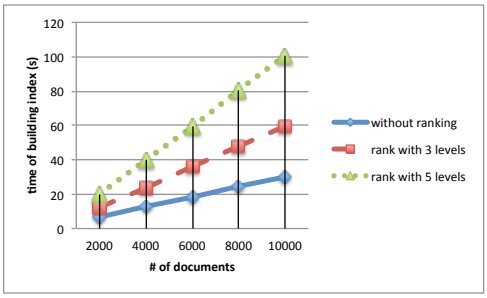
Table 2: Computation costs incurred by each party

[5]Computing bitwise product is negligible compared the overall operations the user performs.

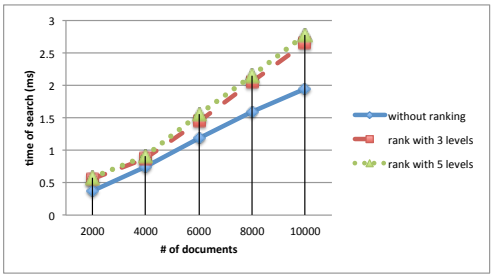
8.1 Implementation Results

The entire system is implemented by Java language using socket programming on an iMac with Intel Core i7 processor of 2.93 GHz. Considering analyzing a document for finding the keywords in it, is out of the scope of this work, a synthetic database is created by assigning random keywords with random term frequencies for each document. The HMAC function produces outputs, whose size (l) is 336 bytes (2688 bit), which is generated by concatenating different SHA2-based HMAC functions. We choose $d = 6$ so that after the reduction phase the result is reduced to one-sixth of the original result; therefore the index size (r) is 56 bytes (448 bits). In the RSA encryption, modulus N is chosen as a 1024-bit integer which is the product of two randomly chosen 512-bit prime numbers.

In our experiments, we used different databases with different number of documents (from 2000 to 10000 documents). The timing results for creating the search indices are obtained for documents with 20 genuine and 60 random keywords each using ranking technique with different rank levels in Figure 4(a). Considering that index generation is performed only occasionally (if not once) by the data owner and that index calculation problem is of highly parallelized nature, the proposed technique presents highly efficient and practical solution to the described problem.



(a) Timings for index construction with 20 genuine and 60 random keywords per document (on data owner side)



(b) Timings for query search (on server side)

Figure 4: Timing results

Figure 4(b) demonstrates the server timings for a search with different rank levels. As can be observed from the graphic in Figure 4(b), time spent by the server per query is quite low rendering high-throughput processing of user queries possible. By parallelization and native language support, the throughput can be increased several orders of magnitude.

In the work of Cao et al. [3] index construction for 6000 documents takes about 4500 s where we need 60 s in the

highest rank level. Similarly they require 600 ms to search over 6000 documents where we need only 1.5 ms. The tests in [3] was done on an equivalent computer, Intel Xeon processor 2.93 GHz.

The total time the user spends on all computations after learning the trapdoor information is 10 ms per document retrieved. The time that the data owner spends for decryption and signature check is about 5 ms per query. The low time requirements on the data owner side enables processing multiple requests with high-throughput. Note that the programs used in the experiments are developed in Java language for portability reasons and unoptimized. Further optimization or support of native code will further increase the performance of the proposed system.

9. CONCLUSION AND FUTURE WORK

In this paper, we motivate and solve the problem of efficient and secure ranked multi-keyword search on remotely stored encrypted database model where the database users are protected against privacy violations. We first define the security requirements for the given problem. We then employ a secure usage of the method given in [14] for practical application scenarios where total number of keywords that can be searched is relatively limited and there are only few search terms in a query by using a trapdoor based system where the trapdoor can only be generated by the data owner. We appropriately increase the efficiency of the scheme by using symmetric-key encryption method rather than public-key encryption for document encryption. We also propose to use the blinded encryption technique in accessing the contents of the retrieved documents without revealing them to other parties. We prove that our proposed method satisfies the security requirements. The proposed ranking method proves to be efficient to return highly relevant documents corresponding to submitted search terms. We implement the entire scheme and extensive experimental results on the implementation demonstrate the effectiveness and efficiency of our solution.

Following the current research, there are possible improvements and undergoing efforts that will appear in the future work. Firstly, the user side of proposed system will be implemented on mobile devices running Android and iOS operating systems since the potential application scenario envisions that users access the data anywhere and anytime. And secondly, the proposed method will be tested on a real dataset in order to compare the performance of our ranking method with the ranking methods used in plain datasets that do not involve any security or privacy-preserving techniques.

10. ACKNOWLEDGMENTS

The work was in part supported by the European Union project UBIPOL (Ubiquitous Participation Platform for Policy Making). We would like to thank TUBITAK (The Scientific and Technological Research Council of Turkey) for the Ph.D. fellowship supports granted to Cengiz Orencik.

11. REFERENCES

[1] Oxford dictionaries, the oec: Facts about the language. `http://oxforddictionaries.com/page/oecfactslanguage/the-oec-facts-about-the-language`, June 2011.

[2] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. Skeith. Public key encryption that allows pir queries. In *Advances in Cryptology - CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 50–67. Springer Berlin / Heidelberg, 2007.

[3] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou. Privacy-preserving multi-keyword ranked search over encrypted cloud data. In *IEEE INFOCOM*, 2011.

[4] Y.-C. Chang and M. Mitzenmacher. Privacy Preserving Keyword Searches on Remote Encrypted Data. In *Applied Cryptography and Network Security*, pages 442–455. Springer, 2005.

[5] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of CRYPTO'82*, pages 199–203, 1982.

[6] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45:965–981, November 1998.

[7] L. E. Dickson. *Linear Groups with an Exposition of Galois Field Theory*. Dover Publications, New York, 2003.

[8] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography Conference - TCC 2005*, pages 303–324, 2005.

[9] J. Groth, A. Kiayias, and H. Lipmaa. Multi-query computationally-private information retrieval with constant communication rate. In *PKC*, pages 107–123, 2010.

[10] W. Ogata and K. Kurosawa. Oblivious keyword search. In *Journal of Complexity, Vol.20*, pages 356–371, 2004.

[11] J. T. Trostle and A. Parrish. Efficient computationally private information retrieval from anonymity or trapdoor groups. In *ISC'10*, pages 114–128, 2010.

[12] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39:50–55, December 2008.

[13] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou. Secure ranked keyword search over encrypted cloud data. In *ICDCS'10*, pages 253–262, 2010.

[14] P. Wang, H. Wang, and J. Pieprzyk. An efficient scheme of common secure indices for conjunctive keyword-based retrieval on encrypted data. In *Information Security Applications*, Lecture Notes in Computer Science, pages 145–159. Springer, 2009.

[15] J. Zobel and A. Moffat. Exploring the similarity space. *SIGIR FORUM*, 32:18–34, 1998.