A secure Search over Distributed Data

Rana J. Mohammed
Department of Computer Science, College of Education of Pure Science, Basrah University, Basrah, Iraq.

Abstract

In recent years, due to the economic benefits and technical advances of cloud computing, huge amounts of data have been outsourced in the cloud. To protect the privacy of their sensitive data, data owners have to encrypt their data prior outsourcing it to the untrusted cloud servers. To facilitate searching over encrypted data, several approaches have been provided. However, the majority of these approaches handle Boolean search but not ranked search; a widely accepted technique in the current information retrieval (IR) systems to retrieve only the top–k relevant files. In this paper, propose a distributed secure ranked search scheme over the encrypted cloud servers. Such scheme allows for the authorized user to search the distributed documents in a descending order with respect to their relevance documents and the contents of files be retrieved. To do so, each data owner builds its own searchable index, and associates with each document in that index its weight score, which facilitate document ranking. To ensure the privacy of these weights, utilized the ranking. While preserving their capability to perform the ranking process and to conducted several empirical analyses on a real dataset, all programs needed to the propose system written using Matlab language.

Keywords: Cloud Computing, searchable encryption, Index Terms—Cloud Computing, searchable encryption, filter.

البحث الامن على البيانات الموزعة

رنا جاسم محمد

قسم علوم الحاسوب ، كلية التربية للعلوم الصرفة ، جامعة البصرة ، بصرة ، العراق.

الخلاصة :

في السنوات السابقة, نتيجة للاستفادة الاقتصادية والتقدم التقني في الاحتساب السحابي , كمية كبيرة من البيانات تم نقلها الى هذا النموذج الجديد . لحماية خصوصية البيانات فان مالكي البيانات يشفرون بياناتهم قبل نقلها الى خادمات الاحتساب السحابي غير الموثوقة . ولتسهيل عملية البحث في البيانات المشفرة فقد تم تطوير العديد من المناهج لاجراء هذه العملية ولكن اغلب هذه المناهج تعالج مشكلة البحث البولياني وليس البحث الترتيبي , الذي يعد تقنية واسعة الاستخدام في انظمة استرجاع المعلومات الحديثة .

في هذا البحث تم تصميم نظام بحث توزيعي لبحث البيانات المشفرة في الخادمات السحابية بصورة امنة . النموذج المقدم يتيح امكانية بحث الملفات التوزيعية وارجاع الملفات التي لها اكثر تشابه مع الاستفسار المدخل, لاجراء هذه االمهمة فان مالك البيانات يبني دليل قابل للبحث ويربط مع كل ملف وزنه في الدليل لتسهيل عملية الترتيب ولحماية الاوزان فقد استخدمنا طريقة المقابلة الحافظة للخصوصية تم اجراء عدة تجارب على قواعد بيانات حقيقية , اللغة المستخدمة لكتابة البرامجيات للنظام المقترح هي لغة ماتلاب .

*Email:rana2005ra@yahoo.com

Introduction

Recently, due to the great advances in Internet technologies, large companies like Google, Yahoo, and Amazon have adopted the Cloud Computing technology to support their consumers with an efficient, large scale data management services [1]. However, outsourcing sensitive data such as health records, personal e-mail, photos, and government documents to the cloud servers represents a great barrier towards the wide adoption of Cloud Computing. To see way, in the cloud, the management of users' data will be under the control of untrusted cloud servers, and this may jeopardize data privacy. To alleviate concerns, users usually encrypt their sensitive data before outsourcing it to combat unsolicited access.

Searchable symmetric encryption systems (SSES) [2-7] offer secure and efficient solution to selectively retrieve the encrypted data through keyword search. Typically, these systems build a secure index structure and outsource it along with the encrypted documents to the remote server. Authorized users submit their requests as secret trapdoors that are integrated properly with the stored indexing information. The server uses the received trapdoor to search over the stored index, and retrieves the matching encrypted documents.

However, almost all the previous SSES systems suffer from two main drawbacks. First, these systems only support the search over a single source of encrypted documents, i.e. a document storage server. To scale for a large amount of data, however it is desirable to search over multiple sources, i.e. several distributed servers. Second, these systems are limited to the case of Boolean search model, which uses the presence or absence of queried keywords as a measure to decide whether the searched documents are relevant or not to the search request. Under such systems it is very cumbersome for most users to express complex requests and control the number of retrieved documents. In contrast, modern IR systems [8] utilized ranking model to score all retrieved documents according to some relevant criteria. Such model enhances the efficiency of the system and provides precise answer by retrieving only the top- k relevant documents from the whole document collection.

This paper explores a ranked searchable encryption scheme of multi-keyword queries over distributed cloud servers. Searching over distributed servers enhances the ability to move for large-scale collections, and improve the search response time. In such setting, the computation required to evaluate user query is distributed among several servers. All the available servers cooperate to evaluate the same query. A ranker server is used to manage user's request, which accepts search request, broadcasts it to all the parallel servers, and then merges all partially responded results into a final list. Searchable encryption schemes usually use a searchable structure to index their text collection for speeding the search task. To facilitate the ranking task, we need to upgrade the task. Searchable index into rank-oriented searchable index. To do so, we need to associate numerical weighting scores with each keyword-document entry in the searchable index. However, unless secured well, such scores leak important statistical information about the underlying keywords to the adversary server. Thus, the main issue here is how we can encrypt these scores while preserving their ability to rank the relevant documents. In the literature of secure keyword ranked search, the presented schemes [9], [10] always use the order preserving symmetric encryption (OPSE) [11,12] primitive to encrypt the numerical weighting scores. Unfortunately, such primitive leaks the relative order of the scores to the adversary server. To tackle this problem, we instead utilized the privacy preserving mapping (PPM) primitive [13] to secure the these values without leaking anything, while still having the ability to order the relevant documents according to their secure weight scores.

Our contributions can be summarized as the follows. Firstly, to provided a secure ranked symmetric encryption search over distributed cloud servers. Secondly, for using the PPM primitive for preventing the cloud servers from learning anything about the score values. Thirdly, provided new mechanism to perform ranked search with multiple keywords. Fourthly, to develop a simple yet secure protocol for collecting certain statistics from the private datasets of the individual data owners.

The rest of this paper is organized as follows. Related works are reviewed and discussed in Section II. Section III introduces the problem definition and the privacy requirements. Section IV introduces the preliminary techniques, while Section V introduces the proposed scheme. Security analysis and performance investigation are provided in Section VI, and conclusions are drawn in Section VII.

Related Works

Searchable symmetric encryption systems (SSES). The first practical kind of such systems is due to Song *et al*. [2], where each word in the document is encrypted with a two-layer encryption scheme.

Goh *et al*. [3] introduced the secure index notion, where a single index is dedicated for each file in the collection. Chang *et al*. [4] provided new security definition of searchable symmetric encryption systems. Later on, strong security definitions along with their efficient constructions have been presented in [5], where a single index is constructed for the entire document collection. However, the above listed schemes are neither suitable to search multiple servers, nor consider the ranked search with multiple keywords. On contrary, our work addresses the above listed shortcomings figure 1.

Figure 1: The Basic design of the proposed framework.

Secure ranked search systems. Unlike searching plaintext data, much less attention has been paid to explore the secure ranked search problem over the encrypted data. For efficiency, all presented schemes for ranked search weaken the security guarantee of current SSES by leaking the relative relevance order to the adversary server. Both [9] and [10] have used OPSE primitive to encrypt the weight scores. However, OPSE primitive leaks to the cloud server important information with respect to the score values. Such leakage may lead to serious frequency attacks if that adversary has certain background knowledge about the underlying collection. Furthermore, the work of [10] allows only searching with single keyword queries. Our work replaces OPSE with PPM primitive to encrypt the score values without leaking anything about the underlying scores, and allow users to provide multi-keyword queries.

Problem definition and privacy requirements

This section gives an overview about our problem and illustrate its goals along with its privacy requirements.

A. *Problem Definition*

Consider that we have n connected yet distinct data owners (individuals or companies) DO_i (i = 0, n - 1 $\wedge$ n > 2), each owner holds a document collection $D_i = \{d_1, d_2, \dots, d_{n_i}\}$ of n_i text documents. We also have m ($m \leq n$) cloud servers, where the n data owners' collections are mapped to these m cloud servers. Figure.1 illustrates the overall architecture of our scheme. Each data owner extracts, in the off-line stage, the keyword set $Kw_i = \{w_1, w_2, \dots, w_g\}$ from the sub-collection D_i assigned to it, then it builds its searchable index Ind_i from Kw_i set. Finally, each data owner encrypts its own collection D_i and its searchable index Ind_i and then uploads them together to their dedicated cloud serverCs_i. To search the m remotely cloud servers with a multi keyword query $Q = \{sw_1, sw_2, \dots, sw_z\}$, the authorized user generates the secret trapdoor $Tr = \{Tsw_1, Tsw_2, \dots, Tsw_z\}$from the above query, and then sends Tr to the ranker server. The latter broadcasts the received trapdoor set to all the m parallel cloud servers. Once receiving Tr, each server searches its secure indices Ind_i (one or more) with the presented trapdoors. Once finding some matching, each server retrieves the posting list together with the label of the corresponding keyword to

the ranker server. The ranker server collects the local lists from all the participant servers, merges them together, sorts the final list according to the private weights, and finally, it returns the encrypted files of the top - *k* relevant scores to the cloud user.

B. *Privacy Requirements*

For enhancing the efficiency, the security definition of all presented ranked searchable encryption schemes allows to leak some important information to the adversary server. Such information includes the relative-order of the weight scores, access pattern (the identifiers of the relevant files), and search pattern (whether the keyword *w* has been searched before or not). Our scheme tightens the privacy requirements by preventing the cloud servers from learning anything about the weight scores. The privacy requirements of our searchable encryption scheme can be illustrated as follows:

1. (Documents privacy) the proposed encryption scheme does not leak anything to the cloud server pertaining to the stored documents more than their number, sizes, and file ids.

2. (Trapdoor privacy) without learning the secret key, cloud servers could not be able to generate a valid trapdoor.

3. (Index Privacy) secure index does not leak anything about its contents.

4. (Global weights privacy) To get globally consistent document scores, the participant data owners need to get the union set of their private weight scores. For such a process, no data owner is allowed to learn the source of each item in the resulted union set.

Assuming that all participant parties (servers and data owners) are semi-honest parties and not colluding with each other. In this model, all parties are supposed to follow the protocol faithfully, but they are curious to the private information of the others.

Preliminary Techniques

A. Information Retrieval (IR)

Most of the current IR systems [8] use an index on the document collection to speedup search. The inverted index structure is the most popular one. This index is represented as a set of entries, where each entry includes a keyword (term) and the set of file identifiers (posting list) that contain the corresponding keyword. To make the ranking process more feasible, each file in the posting lists is assigned with a numerical weight. Such weights are used to decide which file is more relevant to the corresponding keyword. The TF-IDF scheme is widely used to compute the weighting scores. For the given term w and the document d, TF (term frequency) refers to the number of times in which w appear in the document d. While IDF (inverse document frequency) denotes the ratio of the whole number of files in the collection to the number of files that include the term w. figure. 2 illustrates a simple example of the inverted index.

B. Privacy Preserving Mapping (PPM)

PPM [13] is a deterministic encryption scheme that allows for mapping numbers into encrypted images, such that the <, =, > relations among the plaintext numbers are preserved. The author has presented three privacy definitions along with their constructions, namely ideal privacy, level-2 privacy, and level-3 privacy. We adapt the first one, which prevents leaking any additional information to the adversary server beyond the comparison result. Suppose that we have a set $X = \{x_1, x_2, \dots, x_r\}$ of distinct integer numbers, key-based hash function $\mathrm{H1}_{kp}()$ with a secret key kp, another H2 hash function. The PPM encryption is run in the data owner side and goes as follows:

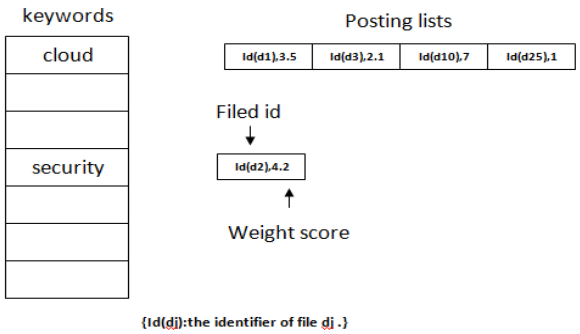

Figure 2: Rank-oriented inverted index.

1) Generate the mapping list:

$$Mlist = \{H2(H1_{kp}(x_j)\|H1_{kp}(x_i))|x_i < x_j \wedge 1 \le i,j \le r\} \quad (1)$$

2) For each number $x_i \in X$ generate its secret image as:

$$T_i = H1_{kp}(x_i) \quad (2)$$

3) Send the public parameters (H2, Mlist) along with the secret images T_i ($1 \le i \le r$) to the adversary server.

4) Given two images T_x and T_y, the adversary server can compare between them as follows:

$$\text{Cpmpare}(Tx\ Ty) \begin{cases} 0 & \text{if} \quad Tx = Ty \\ 1 & \text{if} \quad H2(Tx\|Ty) \in M\ list \\ -1 & \text{if} \quad \text{outher wise} \end{cases} \quad (3)$$

C. Bloom filter

Brie fly, Bloom filter BF [14] is a space-efficient data structure for storing a large amount of items. It is defined as a bit array of q bits, where all bits are initially set to 0. It also defines t independent hash functions, $\varphi_1, \varphi_2, \ldots, \varphi_t$, to map items into a domain between 0 and q - 1. In order to store the item b_1, all the indices obtained from the t hash functions are set to 1. To check whether the item b_1 is found in the filter, we feed it to all the t hash functions to generate t positions. If all these positions are set to 1, then the item is found. Otherwise, it is not found. A false positive is possible which can be controlled by changing the filter length q as follows:

$$q = \frac{-r\ln(p)}{(\ln 2)^2} \quad \ldots\ldots\ldots\ldots\ldots\ldots(4)$$

Where r is the total number of items to be stored, p is the user defined false positive probability.

The Distributed Ranked Searchable Symmetric Encryption Scheme (Drsse)

The proposed DRSSE scheme is composed of seven algorithms that are scattered between two phases, namely the indexing phase and the retrieving phase. The indexing phase includes four algorithms: **Keygen, Index construction, Weight collection,** and **Document encryption,** while the retrieving phase includes three algorithms: **Trapdoor generation, Search,** and **Ranking.**

A) Indexing phase

- *Key generation*: The system administrator (any one of the involved data owners) runs the Keygen algorithm to initialize the system. This algorithm takes the security parameter (λ) as input, and generates the secret key *K* that is securely shared among all the n participant data owners.
- *Index construction*: Each data owner DO_i runs Index construction algorithm to construct its secure document index Ind_i. This algorithm is composed of two steps, index building, and index protection.
-

$$Weight_{i,j} - \frac{1}{|d_j|}\left(1 + \ln(tf_{i,j})\right) \quad (5)$$

Index building step: In this step, each data owner preprocess its local collection and creates its own searchable index Ind_i . In this paper, we use a multiple-keyword query *Q* to search multiple servers. Among several TF-IDF formulas, we choose the following one to compute the weight of the document d_j with regards to the keywordw_i:

Where $|d_j|$ is the length of the document d_j , which represents the total number of its terms, functioning as a normalization factor, $tf_{i,j}$ is the term frequency of term w_i in document d_j. To rank-order the documents, during the retrieval phase, we need to measure the overall relevant score for each file d_j with respect to the query *Q*, hence we need to sum the weight values of each keyword in *Q* correspondig to the file d_j :

$$Score(Q, d_j) = \sum_{w_i \in Q} Weight_{i,j} \quad \ldots\ldots\ldots\ldots\ldots(6)$$

Index protection step: In this step, each data owner turns its own searchable index into a secure index before shipping it to the cloud server. Such process is achieved by encrypting the individual components of the inverted index. The inverted index is protected through the following steps:

1. (Keyword set protection) keyword set has to be encrypted such that only the authorized users can generate them during the trapdoor generation step. We suggest using a key-based hash function, $\pi_{sk}(.)$, where *sk* is a secretly shared key among all the involved data owners. In practice we have used the SHA-1 one-way function as instance of π with output of 160 bit.

2. (File IDs protection) For protecting the file IDs we suggest encrypting these values with any semantically secure encryption scheme such as (Paillier cryptosystem) [15]. Such scheme guarantees that if the same file ID is encrypted multiple times, it will produce different ciphertexts but all decrypted to the same value.
3. (Posting list length concealing) The number of file IDs in the posting lists should also be concealed to prevent the leakage of important information, which may be used by some statistical attacks. To do so, we can pad all the posting lists with faked pairs of (random file IDs, zero weight) to fit the *MAX_ L*, where *MAX_ L* is the length of the longest posting list in the collection.
4. (Keywords number hiding) Similarly, we can add some faked records composed of faked keywords along with their faked posting lists to hide the actual number of stored keywords.
5. (Weight scores protection) However, the major problem is to encrypt the weight numbers while preserving their functionality to rank the retrieved documents. To solve this problem, we utilize the appealing feature of the recently developed PPM primitive.

- *Global weights collection*: The participant data owners cooperate together to run **Weight collection** algorithm to get the union set of their private weights . To keep the globally consistent document scores, we have to collect all the weight values from all the distributed data owners, and merge them together into a single set. Thus we perform the following modifications:

1) Each data owner prepares its unique set of weight values $Xi = unique\{(Weight_{j,k})\}$.

2) Collect the set X of global weight values from all the n distributed $Do_i\ s.t\ X = \cup\ X_i, \forall i = 1, \dots, n$.

3) Generate the mapping list *Mlist* from the set X as in (1).

4) Each data owner encrypts its own set $X_i = \{x_i\}$ to generate their secure images as in (2).

5) Send *(H2, Mlist)* to the ranker server as public parameters .

Similarly, the ranker server uses the *Mlist* together with H2 to compare any received pair of images as in (3). However, the privacy requirement for this step is how the distributed DO_i can find the union set X of their local sets X_i without revealing which weight belongs to which data owner. We preset a simple approach to calculate the union of the individual weight values belonging to distributed owners without revealing the source of these values.

Given n data owners $DO_0, DO_1, \dots, DO_{n-1}$ of n local weigh sets X_0, X_1, X_{n-1}. We wish to compute $\cup X_i$ while preserving the privacy of the participant data owners. Initially, the involved parties arrange themselves into a ring. Then, each party generates its local random vector $R_i$of length $|X_i|$, and then performs the term-wise addition between the generated vector and the original vector weights X_i to produce the new vector Rx_i. The proposed protocol goes into two rounds. In the first round, DO_0 starts the work by sending its Rx_i vector to its successor. The from its predecessor party, merges it with its local set X_i, permutes the result, and passes the resulted vector to the next party. To prevent DO_1 from learning the number of scores in DO_0, DO_0 pad its random vector Rx_0 with a fake vector of c random values. At the end of the first round we get the union of noisy weights. When the DO_0 gets back a vector, it starts the second round, which removes the noise from the resulted union set.

Algorithm 1 Global weights collection

Input: $DO_0, DO_1, \dots, DO_{n-1}$: n data owners, each one have a local set X_i of weight value.

Output*: X*: the global (union) set of the local sets X_i.

1: Set $X = \emptyset$, Set $= X_{noisy} = \emptyset$

2: Each data owner DO_i generates its own local vector R_i of $|X_i|$ random numbers.

3: DO_0generate the faked vector *F* of c random numbers.

4: Each DO_i computes: $Rx_i = X_i + R_i$

5: DO_0 merge its vectors Rx_0 and *F* as : $Rx_0 = \{Rx_0, F\}$

6: DO_0 sends Rx_0 to DO_1.

7: { round 1}

8: **for** i=1 to *n* - 1 **do**

9: Receive : Rx_{i-1} from DO_{i-1}

10: $X_{noisy} = \{X_{noisy}, Rx_{i-1} || Rx_i\}$

11: $X_{noisy} = permute(X_{noisy})$

12: Send *X noisy* to DO_{i+1} mod *n*.
13: **end for**
15: **for** *i=0* to *n - 1* **do**
16: Remove Rx_i from *X noisy*
17: Set *X = permute*{ $X, X_i \| X\ noisy$ }
18: Send *X* to DO_{i+1} mod *n*
19: **end for**
20: DO_0 refines *X* as: *X = unique(X)*

Here each party subtracts the added random values R_i from its random set Rx_i in the received union set, and then passes the new set to its successor. This work repeats until DO_0 receives the pure union set. Having done that, it removes the duplicated values to get the set *X*. Algorithm 1 depicts the operations of this protocol. Once doing that, DO_0 generates *Mlist* and then sends it to the ranker server.

- *Security improvement and efficiency enhancement:*

Paillier-PPM: As we said before, PPM primitive is a deterministic encryption. Such property leaks additional information that may be utilized from the cloud servers to conduct some statistical attacks. This is because, the distribution of the original weights will be preserved in the encrypted images. To overcome this problem, we need to make PPM a probabilistic primitive. Such new setting encrypts the duplicated weights into different ciphers. To do so, we encrypt the secure images T_i of each party with another layer of probabilistic encryption *enc*. So that $T_i = enc_{kprob}(H1_{pk}(x_i))$, where *kprob* is a securely shared key among all the *n* parties. Particularly, we utilize the probabilistic asymmetric Paillier cryptosystem to implement *enc* encryption function. The ranker server has to decrypt each received image before performing its comparison process.

Efficiency enhancement: The global *Mlist* list needs a storage capacity of $\frac{r(r-1)}{2}$ to store the generated hash values, where $r = |X|$. Particularly, for some large *r* value, this amount of data seems to be problematic. Our solution to solve this problem is to use a single Bloom filter for storing the *Mlist* list. For this new setting, secure images of *X* set are stored in the Bloom filter *BF* as:

$$BF(\varphi_i(H2(H1_{kp}(x_j)\|H1_{kp}(x_k))) = 1 | x_k < x_j \wedge 1 \le k, j \le r, \forall_i = 1, \dots, t \quad (7)$$

Algorithm 2 Secure indices generation

Input: λ : the security parameter, DO_l $(l = 0, \dots, n-1)$: data owners, each with its own document collection D_l, $\varphi_1, \dots, \varphi_t$: hash functions for the Bloom filter *BF*, $H1_{kp}$ and H2: two hash functions for the PPM primitive, *prop*: the desired false positive probability, $Enc_2()$: AES symmetric encryption function, *MAX_ L*: the maximum length of posting lists, *Max_keyword* : the maximum number of keywords that could occur in a collection, (Enc, Dec): the encryption and decryption functions of Paillier cryptosystem.

Output: Ind_l: secure index for each data owner DO_l, *Mlist*: the global mapping list.

Keygen

- Use λ to generate *sk*, *pk*, and *Kcoll* secret keys, two key pairs *(pai_enc_key$_1$; pai_dec_key$_1$), (pai_enc key$_2$, pai_dec_key$_2$)* for Paillier cryptosystem.

for all DO_l, *l* = 0, . . . , n - 1 **do**

- Set $X_l = \emptyset$.
- Extract the distinct keyword set $KW_l = \{w1, w2, \dots, w_{nterms}\}$.

for all $w_i \in kw_l$ **do**

- Build the ID_i set, which include the set of files ids that include the keyword w_i, ID_i ={ $id_1, id_2, \dots, id_{nfile}$}.

for all $id_j \in ID_i$ do

- Calculate the weight score $x_{i,j}$ as in (5).
- $X_l = \{X_l, x_{i,j}\}$
- $Encrypt\ id_j\ as: Eid_j = Enc_{pai\_enc\_key1}(id_j)$
- $Encrypt\ x_{i,j}\ as: EncEnc_{pai\_enc\_key2}(H1_{pk}(x_{i,j}))$

end for

- Set $Posting_i = \{Eid_j, Ex_{i,j}\}, \forall j = 1, \dots, |ID_i|$.

- Pad $Posting_i$ with $MAX_L - nfile$ pairs:
$Enc_{pai-enc-key1}(rn), Enc_{pai\_enc\_key2}(o))$, where rn is a fresh random number
- Store $\{\pi_{sk}(wi), Posting_i\}$ in Ind_l

end for

- Pad Ind_l with *Max_keyword* faked records.

- Encrypt the document collection D_l with $Enc2_{kcoll}()$ encryption to get the encrypted collection C_l
- Upload C_l collection along with Ind_l to the CS_l server that is dedicated to DO_l party.
- X_l = unique(X_l)

end for

- Use Algorithm 1 to get the global set *X* of the local sets X_l (*l* = 0, . . . , *n*-1).
- DO_0 compute *Mlist* from *X* as in (1).
- Compute the length *q* of the Bloom filter *BF* as in (4).
- Initialize the Bloom filter *BF* of *q* zero-bits.
- Store *Mlist* items into *BF* as in (7).
- Send *(H2, BF,*$\varphi 1, \dots, \varphi t, pai\_dec\_key_2$*)* to the ranker server.
- Publish *(sk, kcoll)* keys to the authorized users.

The comparison function will be modified as follows:

$$CBF(T_x, T_y) \begin{cases} 0 & if \quad T_x = T_y \\ -1 & if \quad BF(\varphi i(H2(T_x \| T_y))) = 1, i = 1, \dots, t \\ 1 & if \quad otherwise \end{cases} \qquad (8)$$

Algorithm 2 illustrates our protocol for generating the secure indices.

▪ *Document encryption:* To protect the privacy of their document contents, each DO_i runs Document encryption algorithm to encrypts its collection D_i with a pseudo-randomness against chosen plaintext attack (PCPA) encryption $\mathsf{Enc2}_{kcoll}()$ before uploading it to the cloud. In particular, we use (AES with CTR mode) as an instance of $\mathsf{Enc2}_{kcoll}()$.Given the document d_i and its identifier $id(d_i)$, the encrypted collection will be $C = \{(id(d_i), \mathsf{Enc2}_{kcoll}(d_i)\} \forall d_i \in D_i$.

Algorithm 3 Ranking

Input: *Merge List:* a set of encrypted posting lists of length *L*.

Output: *FileID*: a set of relevant file id_s,*Pscore:* a set of weight scores, both sets of length *L*.

- Set *Efileid* to capture the file ids in *MergeList: Efileid* = $\{Enc_{pai-enckey1}(id)\}$of length *L*.
- Use *pai_enc_key1* to decrypt each file id in *Efileid* to get: *FileID* = $\{id_1, id_2, \dots, id_L\}$.
- Define *Escore* set to capture all the weight values in *MergeList:*

Escore = $\{Enc_{pai_{enckey2}}\left(H1_{pk}(x_i)\right)\}$ of length *L*.

- Initialize the zero matrix *M* of (L*L) dimensions.
- Decrypt each item in the *Escore* set with the *pai_dec_key2* to get the *Image set: Image* = $\{H1_{pk}(x_i)\}$ of length *L*.

for *i*1=1 to *L* **do**
  for *j*1=1 to *L* **do**
    if *i*1 < *j*1 **then**
      - *M*(*i*1; *j*1)=**CBF**(Image(*i*1),Image(*j*1)) as in (8)
    else if *i*1 > *j*1 **then**
      M (*i*1,*j*1)=-M(*j*1,*i*1)
    end if
  end for
end for

- *Pscore* = {*sm1, . . . , smL*}, where sm_j is the summation of row *j* in *M* .

B. Retrieving phase

• *Trapdoor generation:* System administrator shares some secret information with the authorized users to enable them searching the encrypted documents. Such information includes :

1) *sk:* For generating the secure trapdoors for the given keywords search.

2) *Kcoll:* For decrypting the retrieved encrypted documents.

Given the query $Q = \{ws_1, ws_2, \dots, ws_z\}$ of z keywords, the authorized user U runs **Trapdoor generation** algorithm to generate the secret trapdoor $Tr = \left(\{\pi_{sk}(ws_j)\}, k\right), \forall j = 1, \dots, z$ for his multiple keyword query, where *k* is a user defined parameter controlling the number of retrieved files.

• *Distributed searching*: The ranker server accepts the search request Tr from the end users and broadcasts the trapdoor set $\pi_{sk}(ws_i), \forall j = 1, \dots, z$ to all the *m* distributed cloud servers. Each cloud server Cs_i runs in parallel the search algorithm on its searchable index Ind_i to find the matched entries. If some matching exists, the server Cs_i returns each matched posting list together with the label of the corresponding trapdoor to the ranker server. More formally, Cs_i returns the pairs (*j*,PL), for each PL s.t. $\left(\pi_{sk}(ws_j), PL\right) \in Ind_i$

• *Distributed ranking:* Once receiving the posting lists, ranker server collects and merges them together according to their keyword labels. Given the merged posting list $Mergelist_j$ of the keyword label *j*, the ranker server runs Ranking algorithm to return the file identifier set *FileID* together with their corresponding partial score set *Pscore*. Once doing this, ranker server have to sum the partial score values of the relevant files. Finally it retrieves the encrypted files of the *k* highest score values. Algorithm 3 shows the ranking algorithm. Algorithm 4 shows our proposed protocol for retrieving the *top-k* file among multiple cloud servers.

Algorithm 4 Distributed rank based search over multi encrypted cloud servers.

Input: *k:* number of retrieved files, π_{sk}: key based hash function with its secret key *sk*, *kcoll:* decryption key of the encrypted file collection, $Q = \{ws_1, ws_2, \dots, ws_z\}$: multiple keywords query.

Output: The set of top-*k* encrypted files.

{**user side:** }

for all $ws_j \in Q$ **do**

- Compute $\pi_{sk}(ws_j)$

end for

- Send the trapdoor set $Tr = (\pi_{sk}(ws_j), k), j = 1, \dots, z$ to the ranker server.

- Use *Kcoll* to decryp the received top-*k* encrypted files.

{**Cloud servers side:**}

for all cloud server Cs_i(*i* = 1, . . . , *m*) **do**

for all $\pi_{sk}(ws_j) \in Tr$ **do**

if $\pi_{sk}(ws_j) \in Ind_i$ **then**

- Send (*j*,PL) to the ranker server, where PL is the posting list corresponding to $\pi_{sk}(ws_j)$

end if

end for

end for

{**Ranker side:**}

Input: $P = \{(lab_1, PL_1), (lab_2, PL_2), \dots, (lab_{np}, PL_{mp})\}$: a set of *mp* pairs of (label, posting list) from the parallel *m* cloud servers, *Pai_dec2* is the decryption key of Paillier decryption function.

Output: the set of the encrypted files of the top-*k* relevant scores.

- Initialize an empty set *RelFile* = $\emptyset$;

- Initialize an empty set Score = $\emptyset$;

- stor = 0.

for *j*=1 to *z* **do**

{*z* = |Q|}

- Set *Psup* = $\bigcup$ PL to capture all the posting lists *PL* ϵ *P* that satisfy:*(lab, PL)* $\in P \wedge lab = j$

- [*FileID, Pscore*]=Ranking(*Psup*)

for all *id* $\in$ *FileID* **do**

- Location = find(*RelFile*==*id*)

if *Location* $\neq -1$ **then**

{found}

- *Score*(*Location*) = *Score*(*Location*) + *Pscore(id)*

else

{not found}
- *RelFile(stor) = id*
- *Score(stor) = Pscore(id)*

end if

end for

end for

- Set HS vector to capture the indices of the k-minimum numbers in *Score* vector, $HS = \{hs_1, hs_2, \dots, hs_k\}$.
- Set *Relevant* = { $RelFile(hs_1), RelFile(hs_2), \dots, RelFile(hs_k)$} be the identifier set of the top-*k* encrypted documents.
- Ranker server broadcasts *Relevant* set to the *m* cloud servers, which send back the encrypted documents corresponding to the provided identifiers to the ranker server. Finally, the latter, provide the cloud user with the encrypted documents .

Consider the following example to illustrate the work of our protocol. For ease of exposure we use unencrypted posting lists. Suppose that $k = 4$, $Q = \{cloud, security\}$, and three cloud servers. cloud server 1 returns {(1, {(1, 4)(3, 7)}), (2, {(2, 5)(1, 3)})}, cloud server 2 returns {(1, {(5, 3)(6, 2)}), (2, {(5, 1)(6, 1)})}, and cloud server 3 returns {(1, {(9, 2)(10, 2)}), (2, {(11, 3)(12, 6)})}

Ranker server collects the posting lists for each keyword label as:

- $Psub_1$ = {(1, 4)(3, 7)(5, 3)(6, 2)(9, 2)(10, 2)}
- $Psub_2$ = {(2, 5)(1, 3)(5, 1)(6, 1)(11, 3)(12, 6)}

and use the ranking algorithm to return the $FileID_1$ ($FileID_2$) and $Pcscore_1$($Pcscore_2$) set for Pub_1 (Pub_2)

- $FileID_1$ = {1, 3, 5, 6, 9, 10}
- $FileID_2$= {2, 1, 5, 6, 11, 12}
- $Pcscore_1$ is computed inside ranking algorithm as follows :

✓) Image = {4, 7, 3, 2, 2, 2} of length 6.

✓) Compute matrix M as follows:

0	1	-1	-1	-1	-1
-1	0	-1	-1	-1	-1
1	1	0	-1	-1	-1
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0

Thus $Pcscore_1$ = {-3,-5,-1, 3, 3, 3}

- $Pcscore_2$ = {-3, 0, 4, 4, 0,-5} is computed as$Pscore_1$.
- *RelFile* = {1, 3, 5, 6, 9, 10, 2, 11, 12}
- *Score* = {-3,-5, 3, 7, 3, 3,-3, 0,-5}
- *HS* = {2, 12, 1, 7}, where 2 is the location of the minimum number -5 in *Score*, and so on.
- *Relevant* = {3, 12, 1, 2}

C. Document decryption

Once the encrypted documents corresponding to the secure trapdoor *Tr* are retrieved, user decrypts them with *Kcoll* key to obtain their plaintext documents.

Security analysis and performance investigation

In this section evaluating the security and the performance of our proposed scheme

A. *Security analysis*

In the security analysis, investigate to which extent our scheme fulfills the privacy requirements described in section III-B

1. For protecting the document files, each party applies a PCPA-encryption on these documents before outsourcing . The appealing feature of such encryption is that its output can not be distinguished computationally from random values. However, only the file ids, number of files, and the size of each encrypted document are allowed to be leaked in our scheme.

2. To prevent the cloud servers and even the ranker server from generating a valid trapdoor we suggest using a key-based hash function $\pi_{sk}()$ with secret key shared among the data owners and the authorized users to generate the valid trapdoors. Such that without knowing the secret key *sk*, it seems impossible for such servers to generate a valid trapdoors. According to Oxford's dictionary [16], English language includes about 25000 distinct words. Such limited number of keywords makes it is easy for the cloud server to perform its brute-force attack.

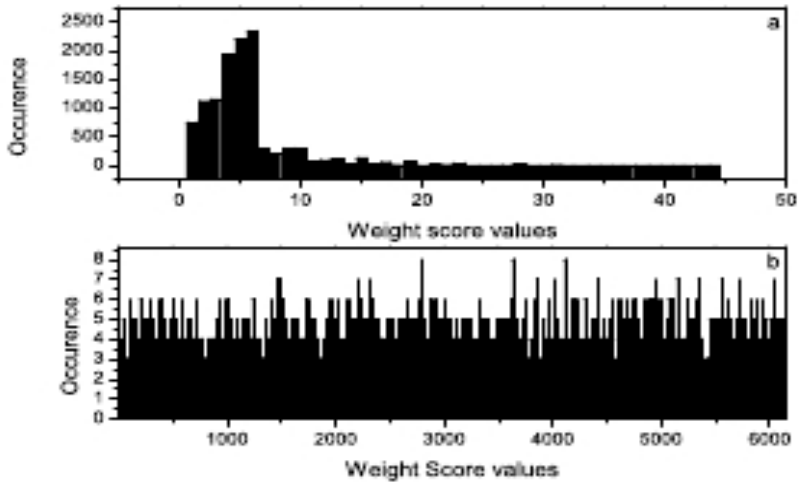

Figure 3-Weight scores distribution.

For this reason, we used a key-based hash function to make it more difficult for the cloud server to conduct its brute-force attack.

3. Secure index privacy: To protect the keyword set, our scheme uses a one-way key-based hash function. Furthermore, for preventing some statistical attacks, suggest adding some faked keywords to hide the actual number of the stored keywords in each index. Similarly, the posting lists are padded with faked pairs of (file id, weight score) to hide the actual length of the individual lists. What remains is to encrypt the individual file ids and their corresponding weight values. Both weight values and file ids are encrypted with the randomized Paillier encryption. Such scheme not only protects the privacy of these values, but also prevents leaking the distribution of these values to the cloud servers.

4. Global weights privacy: it is clear from Algorithm 1 that no data owner knows the real weigh scores of the other owners during the union process.

B. *Performance analysis*

In this section, to examine the performance of our scheme by conducting several experimental results on the publicly available real dataset: Mini_newsgroups [17], which contains 2000 text files. Selecting randomly 1000 file and distributed them equally among three data owners (*n=3*). Each data owner is provided with a single cloud server (*m = 3*). For reducing the index structure, we exclude the high frequency keywords (stop-words) from the keyword set, convert all keywords into lower-case strings, returning the keywords to their root (stemming), and then refine the generated set to include only pure English words. Our experiments have been conducted on a 2.61GHz Pentium processor, Windows 7 operating system, with a RAM of 1GB. We used MATLAB (R2008a) to implement the experiments. Some algorithms are programmed by Java such as Paillier encryption and inverted index construction.

1) *The effectiveness of Paillier-PPM*: In this experiment, explain how proposed Paillier-PPM can improve the security of the encrypted weight scores. Figure 3-a shows the histogram of 11348 encrypted weight scores under the original PPM. It is easy to see that PPM does not randomise the encrypted scores very well. Such peaky histogram exposes more frequency information to the adversary cloud server. For reducing the leaked amount of information, we have to flatten such distribution by encrypting each weight score with a randomised encryption. Such method ensures that repeated weights will be mapped into different values. Figure 3-b shows the resulted histogram after applying our method, which seems like a uniform distribution.

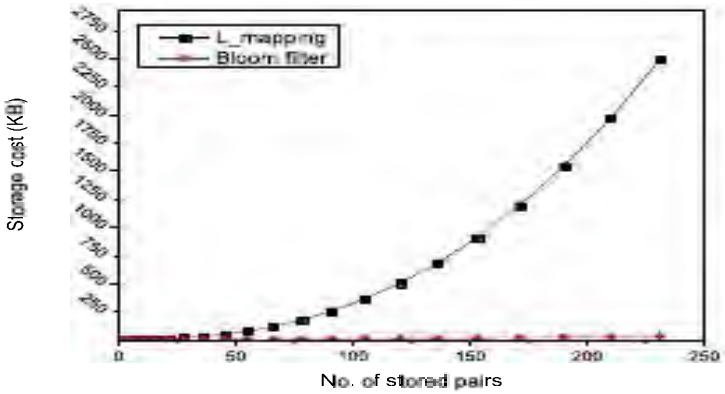

Figure 4-Storage cost.

2) *Paillier-PPM efficiency:* The original setting of PPM uses a hash table *L*_Mapping to store *Mlist* list of size $\frac{r*(r-1)}{2}$ For very large *r* values, such scheme seems to be impractical. To reduce the storage cost without compromising the effectiveness, we suggest using a single Bloom filter (7) to store all the elements of *Mlist.* In this experiment, we measure the efficiency of both schemes in terms of storage cost and time building cost. Figure 4 shows the storage cost for both hash table *L_Mapping* and Bloom filter schemes. From this figure, we see that Bloom filter enhances greatly the storage cost. To see why, Bloom filter specifies a fixed number of bits to store all the individual items, whereas, *L_Mapping* specifies a single entry of 160-bit to store each item. Figure 5 illustrates the time cost for generating both *L_Mapping* table and Bloom filter. Even both schemes take a linear time for their work; no scheme has advantage over the other. This case might be interpreted due to the massive operations of both schemes to store their data. Bloom filter needs to perform a number of hash functions for storing each item, where the number of these functions is adaptively changed according to the total number of stored elements. On the other hand, the time cost for *L_Mapping* depends greatly on the number of hash collisions and the time required resolving such collisions.

3) *Index construction:* To allow for efficient search, each data owner builds its own inverted index. Comparing with the none secure inverted index construction; our scheme consumes more storage and time overhead. Specifically, to allow for ranked search, we associate a weight score with each keyword document entry in posting lists. For security purposes, we need to replace these scores with their encrypted form. Furthermore we encrypt keyword strings, file id, pad the individual posting lists with faked entries, and finally add faked pairs of (keyword, posting lists) to hide the actual number of keywords. The time cost for building and encrypting the posting lists depend directly on the length of each posting list. Figure. 6 shows the time construction as the posting list length increased. Note that due to the padded entries, all the posting lists require fixed storage cost. Table 1 lists the whole index generation requirements for three data owners with different document collections.

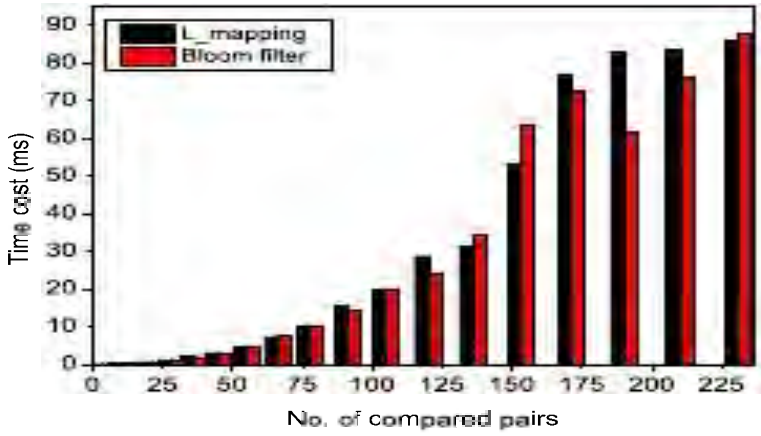

Figure 5-Time cost

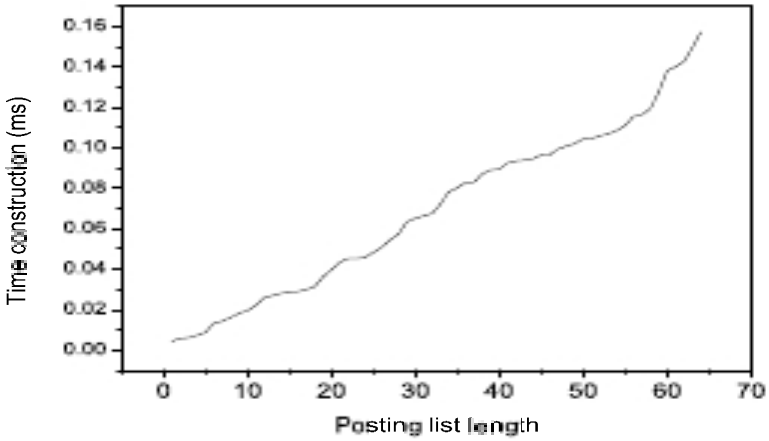

Figure 6-Effect of posting list length

Table 1-Indices parameters

Features per-data owner	DO_1	DO_2	DO_3
No. of documents	300	400	300
No. of distinct Keywords	13027	14355	13899
Unique weight scores	764	897	817
Longest posting list	76	69	67
Index storage cost (kb)	23204	23215	21826
Index building time (ms)	68.8873	77.6394	74.2347

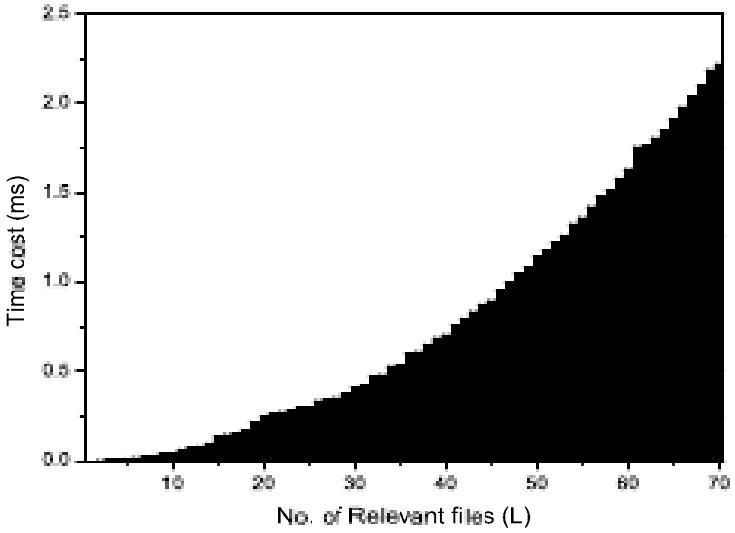

Figure. 7: Matrix M construction Time

4) *Efficiency of global weights collection algorithm:* We discuss here the efficiency of the proposed global weights collection algorithm in terms of computation cost and communication cost. . Recall that such protocol is executed onetime to find the union set of weight scores among the *n* participant data owners. Clearly Algorithm 1 states that we need two rounds for finding the final answer. Such that the communication cost of our protocol can be estimated as: $2\sum_1^n(g * wb)$, where *g* is the average number of weight scores provided by each party, and *wb* is the average number of bits that are required to encode the elements. Whereas the computation cost is estimated as: *n*(As+Ps+Ms+Rs+Ss*), where *As,*

Ps, *Ms*, *Rs*, and *Ss* are costs of addition, permutation, merging, removing, and subtracting two-set operations, respectively.

5) *Ranking time:* Serving user queries includes two steps: searching step and ranking step. In searching step, each cloud server fetches its matched posting lists corresponding to the provided search requests. Such process does not need from the cloud servers to scan all the entire indices, but instead we use a hash tables for retrieving the posting lists with a constant time. Accordingly, searching step has very slight impact on search time. Ranking step, on the other hand, is a complex step and consumes more time. To rank the received posting lists, ranker server have to first merge them together into a global list of *L* relevant files, decrypt the outer layer of the weight score in the resulted list, fill the matrix *M(L*L)* with the comparison values between each pair of weight scores, and finally identify and retrieve the encrypted file ids of the top-*k* scores. Building the matrix *M* is the dominant factor for ranking step. This follows that, query processing time is varies from one query to another. This is because; each keyword query has a different number of relevant files (L). Figure. 7 illustrates ranking time time time as number of relvant files (*L*) increased.

Conclusions:

In this paper, proposed a secure ranked keyword search over encrypted document collections that are scattered among several cloud servers. Such scheme brings together the advanced distributed information retrieval and the cryptography primitives to retrieves the global top-*k* documents from all the distributed servers without revealing neither the contents of these documents, nor the provided search request. To do so, each data owner builds its own inverted index, integrate weight scores with each keyword-document entry in that index. Specifically, to utilize the recently presented cryptography primitive, PPM, for encrypting the weight scores while preserving their ability to perform the rank operation. Then we enhance the security and efficiency of PPM to suit the requirements of our distributed computing case. To get globally consistent document scores, we have developed a secure protocol for collecting the global set of all the distributed weights. Conducting several experiments on a real dataset to verify the performance of proposed scheme.

The suggestion for future work of the current system are: One promising idea is to integrate the IDF factor in the weight score formula and another idea is to support a similarity ranked search, which allows users to search even with misspelled secret trapdoors.

References

1. Mell,P. and Grance,T.**2011**. *The nist definition of cloud computing recommendations of the national institute of standards and technology*. Nist Special Publication, vol.145, no.6, p. 7.
2. Song, D. X., Wagner, D., and Perrig, A. **2000**. *Practical techniques for searches on encrypted data*. IEEE Computer Society, pp. 44–.
3. Goh, E.-J. **2003**. *Secure indexes*. Cryptology ePrint Archive, Report 2003/216.
4. Chang,Y.-C. and Mitzenmacher, M. **2005**. Privacy *preserving keyword searches on remote encrypted data*. in ACNS, pp. 442–455.
5. Curtmola, R., Garay, J., Kamara, S., and Ostrovsky, R. **2006**. *Searchable symmetric encryption: Improved definitions and efficient constructions*. .
6. Boneh, D., Crescenzo, G. D., Ostrovsky, R. and Persiano, G.**2004**. *Public key encryption with keyword search*. in EUROCRYPT, pp. 506–522.
7. Bellare, M., Boldyreva, A. and O'Neill, A. **2007**. *Deterministic and efficiently searchable encryption*. in CRYPTO, pp. 535–552.
8. Manning, H. S. C.D., Raghavan, P.**2008**. *Introduction to Information Retrieval*. Reading, MA: Cambridge UP.
9. Swaminathan, A., Mao,Y., Su, G.-M. Gou, H., Varna, A. L., He, S., Wu, M. and Oard, D. W.**2007**. *Confidentiality-preserving rank-ordered search*. in Proceedings of the 2007 ACM workshop on Storage security and survivability, ser. StorageSS '07. New York, NY, USA: ACM, pp. 7–12.
10. Wang, C., Cao, N., Li, J., Ren, K. and Lou, W. **2010**.*Secure ranked keyword search over encrypted cloud data*. in Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems, ser. ICDCS '10. Washington, DC, USA: IEEE Computer Society, pp. 253–262.

11. Agrawal, R., Kiernan, J., Srikant, R. and Xu, Y.**2004**. *Order preserving encryption for numeric data* . in Proceedings of the 2004 ACM SIGMOD international conference on Management of data, ser. SIGMOD '04. New York, NY, USA: ACM, pp. 563–574.

12. Boldyreva, A., Chenette, N., Lee, Y. and O'Neill, A. **2009**.*Order-preserving symmetric encryption.* in Proceedings of the 28th Annual International Conference on Advances in Cryptology: the Theory and Applications of Cryptographic Techniques, ser. EUROCRYPT '09. Berlin, Heidelberg: Springer-Verlag, pp. 224–241.

13. Tang, Q. **2010**.*Privacy preserving mapping schemes supporting comparison*. in Proceedings of the 2010 ACM workshop on Cloud computing security workshop, ser. CCSW '10. New York, NY, USA: ACM, pp. 53– 58.

14. Schnell,T. B. R. and Reiher, J**. 2009**. *Privacy-preserving record linkage using bloom lters . in BMC Medical Informatics and Decision Making*, vol. 9, no. 1, p. P. 41.

15. Paillier, P. **1999**. *Public-key cryptosystems based on composite degree residuosity classes* . in Proceedings of the 17th international conference on Theory and application of cryptographic techniques, ser. EUROCRYPT'99. Berlin, Heidelberg: Springer-Verlag, pp. 223–238.

16. "*The oec: Facts about the language*," Oxford dictionaries, June **2011**, http://oxforddictionaries.com/page/oecfactslanguage/the-oec-facts-about -the- language/.

17. http://kdd.ics.uci.edu/databases/20newsgroups/mini-newsgroups.tar.gz./.