Distrib Parallel Databases (2014) 32:119–160
DOI 10.1007/s10619-013-7123-9

An efficient privacy-preserving multi-keyword search over encrypted cloud data with ranking

Cengiz Örencik · Erkay Savaş

Published online: 7 March 2013

Abstract Information search and retrieval from a remote database (e.g., cloud server) involves a multitude of privacy issues. Submitted search terms and their frequencies, returned responses and order of their relevance, and retrieved data items may contain sensitive information about the users. In this paper, we propose an efficient multi-keyword search scheme that ensures users' privacy against both external adversaries including other authorized users and cloud server itself. The proposed scheme uses cryptographic techniques as well as query and response randomization. Provided that the security and randomization parameters are appropriately chosen, both search terms in queries and returned responses are protected against privacy violations. The scheme implements strict security and privacy requirements that essentially disallow linking queries featuring identical search terms. We also incorporate an effective ranking capability in the scheme that enables user to retrieve only the top matching results. Our comprehensive analytical study and extensive experiments using both real and synthetic datasets demonstrate that the proposed scheme is privacy-preserving, effective, and highly efficient.

Keywords Privacy-preserving keyword search · Multi-keyword · Cloud storage · Ranking

Communicated by Elena Ferrari.

Part of this work is presented in PAIS 2012 [1].

C. Örencik (✉) · E. Savaş
Sabanci University, Istanbul, Turkey
e-mail: cengizo@sabanciuniv.edu

E. Savaş
e-mail: erkays@sabanciuniv.edu

1 Introduction

Due to the increasing storage and computing requirements of users, everyday more and more data is outsourced to 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) [2] 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].

Cloud computing has the potential of revolutionizing the computing landscape [7]. Indeed, many organizations that need high storage and computation power tend to outsource their data and services to clouds. Clouds enable its customers to remotely store and access their data by lowering the cost of hardware ownership while providing robust and fast services [8]. Forecasts [7] indicate that by 2014, the cloud services brokerage (CSB) providers, that assist organizations to consume and maintain cloud services, will rise from dozens to hundreds. It is also expected that by 2015, more than half of Global 1000 enterprises will utilize external cloud computing services and by 2016, all Global 2000 will benefit from cloud computing to a certain extent [7].

While its benefits are welcomed in many quarters, some issues remain to be solved before a wide acceptance of cloud computing technology. The security and privacy are among the most important issues (if not the most important). Particularly, 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 Cloud Platform [9], Amazon Elastic Compute Cloud [10] or Microsoft Live Mesh [11] may access the sensitive data such as search and access patterns, hiding the query and the retrieved data has great importance in ensuring the privacy and security of those using cloud services.

In this paper, we propose an efficient system where any authorized user can perform a search on an encrypted remote database with multiple keywords, without revealing neither the keywords he searches for, nor the information of the documents that match with the query. The only information that the proposed scheme leaks is the access pattern which is also leaked by almost all of the practical encrypted search schemes due to efficiency reasons. A typical scenario that benefits from our proposal is that a company outsources its document server to a cloud service provider. Authorized users or customers of the company can perform search operations using certain keywords on the cloud to retrieve the relevant documents. The documents may contain sensitive information about the company, and similarly the keywords a user searches for may give hints about the content of the documents, hence both must be hidden. Furthermore, search terms themselves may reveal sensitive information about the users as well, which is considered to be a privacy violation by users if learned by others.

Our proposed system differs from majority of the previous works which assume that only the data controller queries the database [3, 12, 13]. In contrast to previous works, our proposal facilitates that a group of users can query the database provided that they possess so called *trapdoors* for search terms that authorize the users to

include them in their queries. Another major superiority of the proposed method is the capability of hiding the search pattern which is the equality among the search requests. 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 most relevant matches in an ordered manner.

The contributions of this paper can be summarized as follows. Firstly, we introduce formal definitions for the security and privacy requirements of keyword search on encrypted cloud data including hiding the search pattern.We show that linking a query or a response to another leads to correlation attacks that may result in violation of privacy. Secondly, we show how a basic multi-keyword search scheme can be improved to ensure the given privacy and security requirements in the most strict sense. Besides cryptographic primitives, we use query and response randomization to avoid correlation attacks. We provide an extensive analytical study and a multitude of experimental results to support our privacy claims. Thirdly, we propose a ranking method that proves to be efficient to implement and effective in returning documents highly relevant to submitted search terms. Fourthly, we give formal proofs that the proposed scheme is secure in accordance with the defined requirements. Lastly, we implement the proposed scheme and demonstrate that to the best of our knowledge, it is more efficient than existing privacy-preserving multi-keyword search methods in literature.

The rest of this paper is organized as follows. In Sect. 2, we discuss the related previous works. The privacy requirements are defined in Sect. 3. Section 4 gives the system model and the basics of the scheme. Then we provide a detailed description of the proposed scheme in Sect. 5. Query randomization, along with its formal analysis is presented in Sect. 6. Section 7 presents the method used in hiding the pattern of query responses together with its analysis. Section 8 explains the ranking method and verifies its accuracy. In Sect. 9, 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 Sect. 10. Finally, Sect. 11 gives the concluding remarks of the paper.

2 Related work

The problem of Private Information Retrieval (PIR), which is a related topic with privacy-preserving keyword search, was first introduced by Chor et al. [2]. In PIR system, user is assumed to know the identifier of the data item he wants to retrieve from the database and receive the data item without revealing what he retrieves. Recently Groth et al. [5] propose a multi-query PIR method with constant communication rate. However, the computational complexity of the server in this method is very inefficient to be used in large databases. On the other hand, PIR does not address as to how the user learns which data items are most relevant to his inquiries. For this, an efficient privacy-preserving keyword search scheme is needed.

Efficient privacy-preserving keyword search methods are extensively studied in literature. Traditionally, almost all such schemes have been designed for single key-

word search. Ogata and Kurosawa [14] show privacy-preserving keyword search protocol in the random oracle model, based on RSA blind signatures. The scheme requires a public-key operation per item in the database for every query that must be performed on the user side. Freedman et al. [15], propose 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 high since every search term in a query requires several homomorphic encryption operations both on the server and the user side. Liu et al. [13] propose an efficient keyword search scheme utilizing bilinear maps that is based on the public key encryption with keyword search (PEKS) scheme proposed by Boneh et al. [16]. The scheme is designed for a single user and more importantly, queries in the scheme are generated in a deterministic way, and therefore, cannot hide search pattern. A work proposed by Wang et al. [17] allows ranked search over an encrypted database by using inner product similarity. However, this work is also limited only to single keyword search queries. Recently, Kuzu et al. [18] propose another single keyword search method utilizing locality sensitive hashes (LSH), which reveals search and access patterns but nothing else. Different from the other works, this scheme is a similarity search scheme such that any typo that exists in the query can be handled by the matching algorithm.

A number of privacy-preserving multi-keyword search schemes are also proposed in literature. One of the most related methods to our solution is proposed by Cao et al. [19]. Similar to our approach presented here, it proposes a method that allows multi-keyword ranked search over encrypted database. In this method, the data controller 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 is determined essentially by the size of the set of keywords used in searches, which can be in the order of several thousands.

Zhang and Zhang [20] propose a conjunctive keyword search scheme using bilinear pairing based crypto system that does not require keyword fields. Due to the complex bilinear mapping operations and the trapdoor generation operation that must be performed on the client side, this scheme is not practical. Moreover, this work is not implemented by the authors therefore, cannot be compared with our proposed work.

Wang et al. [21] also propose a trapdoorless private multi-keyword search scheme that is proven to be secure under the random oracle model. The scheme uses binary check to test whether the secure index contains the queried keywords, therefore, search is efficient. However there are some security issues that are not addressed in the paper. We adapt their indexing method to our scheme, but we use a different encryption methodology to increase the security and address the security issues that are not considered in [21].

3 System and privacy requirements

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 (e.g, cloud server). We consider three roles consistent with previous works [19, 21]:

- *Data Controller* is the actual entity that is responsible for the establishment of the database. The data controller 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 server) that offers information services to authorized users. It is often required that the server be oblivious to content of the database it maintains, the search terms in queries and documents retrieved.

Given a query from the user, the server searches over the database and returns a list of ordered items. Note that this list does not contain any useful information to the third parties. Upon receiving the list of ordered items, the user selects the most relevant data items and retrieves them. The details of the framework are presented in Sect. 4.

The privacy definition for search methods in the related literature is that the server should learn nothing but the search results (i.e., access pattern) [19]. 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. (Query Privacy) The query does not leak information of the corresponding search terms it contains.
2. (Search Term Privacy) Given a valid query for a set of genuine search terms, no one can generate another valid query for a subset of the genuine search terms in the former query.
3. (Search Pattern Privacy) Equality between two search requests cannot be verified by analyzing the queries or the returned list of ordered matching results.
4. (Non-Impersonation) No one can impersonate a legitimate user.

Definition 1 (Query Privacy) A multi-keyword search protocol has query privacy, if for all polynomial time adversaries A that, given two different keyword lists L_1 and L_2 and a query Q_b generated from the keyword list L_b where $b \in_R \{1, 2\}$, the advantage of A in finding b is negligible.

Definition 2 (Search Term Privacy) A multi-keyword search protocol has search term privacy, if for all polynomial time adversaries A that, given a valid query for a set of search terms K, A cannot generate a valid query for any $S \subset K$ where $0 < |S| < |K|$ (i.e., $S \neq \varnothing$ and $S \neq K$).

Definition 3 (Search Pattern Privacy) A multi-keyword search protocol has search pattern privacy, if for all polynomial time adversaries A that, given a query Q and all previous queries and corresponding matching results that return, A cannot find the list of equivalent searches (i.e., previous queries that include exactly the same set of search terms with Q).

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.

4 Framework of the proposed method

The previous section introduces the three roles that we consider: Data Controller, Users and Server. Due to the privacy concerns that are explained in Sect. 5.4, we utilize two servers namely: index server and file server. The overview of the proposed system is illustrated in Fig. 1. We assume that the parties are semi-honest ("honest but curious") and do not collude with each other to bypass the security measures; two assumptions which are consistent with most of the previous works.

In Fig. 1, steps and typical interactions between the participants of the system are illustrated. In an off line stage, the data controller creates a search index element 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 controller.

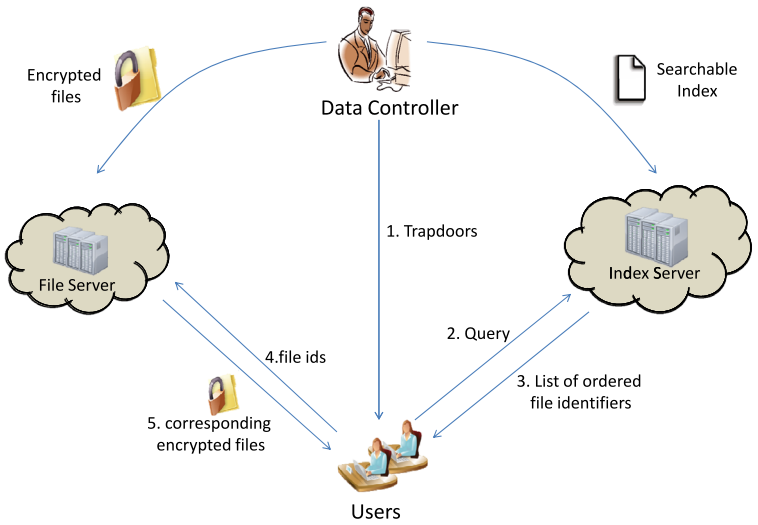


Fig. 1 Architecture of the search method

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

Then, the data controller uploads this search index file to the index server and the encrypted documents to the file server. 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 one of its steps.

When a user wants to perform a keyword search, he first connects to the data controller. He learns the trapdoors (cf. Step 1 in Fig. 1) for the keywords he wants to search for, without revealing the keyword information to the data controller. 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. Alternatively, the user can request all the trapdoors in advance and never connects again to the data controller for trapdoors. One of these two methods can be selected depending on the application and the users' requirements. After learning the trapdoor information, the user generates the query (referred as *query generation* henceforth) and submits it to the index server (cf. step 2 in Fig. 1). In return, he receives meta data[2] for the matched documents in a rank ordered manner as will be explained in subsequent sections. Then the user retrieves the encrypted documents from the file server after analyzing the meta data that basically conveys *a* relevancy level of the each matched document, where the number of documents returned is specified by the user.

In this work we claim that the proposed scheme satisfy query, trapdoor and search pattern privacy requirements as defined in Sect. 3 provided that the parameters are set accordingly. For appropriate setting of the parameters, the data controller needs to know only the frequencies of the most commonly queried search terms for a given database. By performing a worst case analysis for these search terms, the data controller can estimate the effectiveness of an attack and take appropriate countermeasures. The necessary parameters and the methods for their optimal selections are elaborated in the subsequent sections.

We use both a real dataset and synthetic datasets in our analysis. The real dataset used in this work is only a small part (10.000 documents) of the RCV1 (Reuters Corpus Volume 1), which is a corpus of newswire stories made available by Reuters, Ltd. [22].

5 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.

5.1 Index generation (basic scheme)

Recently Wang et al. [21] 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 construction scheme.

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

The original scheme uses forward indexing, which means that a searchable index file element for each document is maintained to indicate the search terms existing in the document. In the scheme of Wang et al. [21], a secret cryptographic hash function, that is shared between all authorized users, is used to generate the searchable index. Using a single 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. The following example illustrates a simple attack against queries with few search terms.

Example 1 There are approximately 25000 commonly used words in English [23] and users usually search for a single or two keywords. For such small input sets, given the hashed trapdoor 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 instead propose a trapdoor based system where the trapdoors can only be generated by the data controller through the utilization of multiple secret keys. The keywords are mapped to a secret key using a public mapping function named `GetBin` which is defined in Sect. 5.2. The usage of secret keys eliminates the feasibility of a brute force attack. The details of the index generation algorithm which is adopted from [21] are explained in the following and formulized in Algorithm 1.

Let $\mathcal{R}$ be the document collection where $|\mathcal{R}| = \sigma$. While generating the search index entry for a document $R \in \mathcal{R}$ that has keywords $\{w_1, \ldots, w_m\}$, we take HMAC (Hash-based Message Authentication Code) of each keyword with the corresponding secret key K_{id} 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 trapdoor of a keyword w_i be denoted as I_i where I_i^j represents the jth bit of I_i, (i.e., $I_i^j \in GF(2)$ where GF stands for Galois field [24]). The trapdoor 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.

Algorithm 1 Index generation

Require: $\mathcal{R}$: the document collection, K_{id}: secret key for the bin with label id
  for all documents $R_i \in \mathcal{R}$ **do**
    for all keywords $w_{i_j} \in R_i$ **do**
      $id \leftarrow \texttt{GetBin}(w_{i_j})$
      $x_{i_j} \leftarrow \text{HMAC}_{K_{id}}(w_{i_j})$
      $I_{i_j} \leftarrow Reduce(x_{i_j})$
    end for
    index entry $\mathcal{I}_{R_i} \leftarrow \odot_j I_{i_j}$
  end for
  return $\mathcal{I} = \{\mathcal{I}_{R_1}, \ldots, \mathcal{I}_{R_\sigma}\}$

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 jth 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 (1).

$$I_i^j = \begin{cases} 0, & \text{if } x_i^j = 0 \\ 1, & \text{otherwise} \end{cases} \tag{1}$$

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

$$\mathcal{I}_R = \bigodot_{i=1}^{m} I_i, \tag{2}$$

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

In the next subsection, we explain the technique used to generate queries from the trapdoors of each search term.

5.2 Query generation

The search index file of the database is generated by the data controller using secret keys. A user who wants to include a search term in his query, needs the corresponding trapdoor from the data controller 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 controller, therefore a technique is needed to hide the trapdoor asked by the user from the data controller.

Bucketization is a well-known data partitioning technique that is frequently used in literature [15, 25–27]. We adopt this idea 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 map 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 keywords that exist in a document are mapped by the data controller to one of those bins using the `GetBin` function. Note that δ is smaller than the number of 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. Each bin in the index generation phase has a unique secret key used for all keywords in that bin.

Algorithm 2 Query generation

Require: a set of search terms $\{w'_1, \ldots, w'_n\}$
  for all search terms w'_i **do**
    $id \leftarrow \texttt{GetBin}(w'_i)$
    if $K_{id} \notin$ previously received keys **then**
      send id to Data Controller
      get K_{id} from Data Controller
    end if
    $x_i \leftarrow \text{HMAC}_{K_{id}}(w'_i)$
    $I_i \leftarrow Reduce(x_i)$
  end for
  query $Q \leftarrow \odot_i I_i$
  return Q

The query generation method whose steps are given in Algorithm 2, works as follows. When an authorized user connects to the data controller to obtain the trapdoor for a keyword, he first calculates the bin IDs of keywords and sends these values to the data controller. The data controller 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 controller can send trapdoors of all 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 in a similar manner to the method used by the data controller to compute the search index. 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., $I_1, \ldots, I_n$) are available to the user:

$$Q = \bigodot_{j=1}^{n} I_j .$$

Finally the user sends this r-bit query Q to the index server. The users' keywords are protected against disclosure since the secret keys used in trapdoor generation are chosen by the data controller and never revealed to the server. In order to avoid impersonation, the user signs his messages.

5.3 Oblivious search on the database

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

[3] In fact, I_i, which is calculated for the search term w_i as explained in Sect. 5.1 is the trapdoor for the keyword w_i.

in (3).

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

Note that given a query, it should be compared with search index entry of each document in the database. The following example clarifies the matching process.

Example 2 Let the user's query be $Q = [011101]$ and two document index entries be $I_1 = [001100]$ and $I_2 = [101101]$. The query has the 0 bit in 0th and 4th bits therefore, those bits must be 0 in the index entry of a document in order to be a match. Here the query matches with I_1, but do not match with I_2 since 0th bit of I_2 is not 0.

Then the index server sends a rank ordered list of metadatas of the matching documents to the user. The meta data is the search index entry of that document, which the user can analyze further to learn more about the relevancy of the document. After analyzing the meta data, the user retrieves ciphertexts of the matching documents of his choice from the file server.

To improve security, the data controller can change the HMAC keys periodically whereby each trapdoor will have an expiration time. After the expiration, 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.

5.4 Document retrieval

The index server returns the list of pseudo identifiers of the matching documents. If a single server is used for both search and file retrieval, it can be possible to correlate the pseudo identifiers of the matching documents and the identifiers of the actual encrypted files retrieved. Furthermore, this may also leak the search pattern that we want to hide. Therefore, we use a two-server system similar to the one proposed in [18], where the two servers are both semi-honest and do not collaborate. This method leaks the access pattern only to the file server and not to the index server, hence prevents any possible correlation between search results and encrypted documents retrieved.

Subsequent to the analyzes of the meta data retrieved from the index server, the user requests a set of encrypted files from the file server. The file server returns the requested encrypted files. Finally user decrypts the files and learns the actual documents. Key distribution of the document decryption keys can be performed using state-of-the-art key distribution methods and is not within the scope of this work.

In case access pattern needs also to be hidden, traditional PIR methods [2–6] or Oblivious RAM [28] can be utilized for the document retrieval process instead. However these methods are not practical even for medium sized datasets due to incurred polylogarithmic overhead.

6 Query randomization

Search pattern is the information of equality among the keywords of two queries that can be inferred by linking one query to another. If an adversary can test the

equality of two queries, he may learn the most frequent queries and correlate with frequently searched real keywords that may be learned from statistics such as Google Trends [29]. The proposed basic scheme fails to hide the search pattern since the search index entries are generated in a deterministic way. Any two queries 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 query generation phase. This process is known as *query randomization*, which should be carefully implemented so that the queries do not leak information about the search patterns. In this section, we analytically demonstrate the effectiveness of the proposed query randomization method. Note that the query randomization does not change the response to a given query.

For introducing non-determinacy into the search index generation, we generate a set $\mathcal{U}$ with $|\mathcal{U}| = U$, whereby the elements of $\mathcal{U}$ are dummy keywords that do not exist in the dictionary (i.e., they are simply random strings). We add these U dummy keywords in every index entry along with the genuine search terms. While generating a query, the user first randomly creates a set $\mathcal{V}$ where $|\mathcal{V}| = V$ and $\mathcal{V} \subset \mathcal{U}$. Then the query is composed using all elements of $\mathcal{V}$ together with the genuine search terms. The number of different choices of $\mathcal{V}$ from $\mathcal{U}$ is calculated as $\binom{U}{V}$.

We can formalize the discussion as follows. Let

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

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

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

Definition 5 (Scarcity function ($F(x)$)) The scarcity function $F(x)$ is the expected number of 0's in a query with x keywords.

Definition 6 (Overlap function ($C(x)$)) The overlap function $C(x)$ is the expected number of 0's that coincide in the corresponding bit positions of an x keyword query (Q_a) and a single keyword query (Q_b).

Recall that r is the size of a query, d is the reduction value (cf. Sect. 5.1) and $Q[i]$ is the ith bit of Q. The functions are calculated as follows:

Proposition 1 *For the Scarcity and Overlap functions we can write,*

$$F(x) = F(x-1) + F(1) - C(x-1)$$
$$C(x) = \sum_{i=0}^{r-1} P\big(Q_a[i] = 0, Q_b[i] = 0\big)$$

where $P(a, b)$ is a joint probability distribution.

Note that initial case for $F(x)$ is $F(1) = \frac{r}{2^d}$ and $C(x)$ is calculated with the following derivation:

$$C(x) = \sum_{i=0}^{r-1} P\big(Q_a[i] = 0, Q_b[i] = 0\big)$$
$$= r\frac{F(x)}{r}\frac{F(1)}{r} = \frac{F(x)}{2^d}.$$

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

Proposition 2 *The Expected Hamming distance between two queries can be calculated as follows*:

$$\Delta(Q_1, Q_2) = \frac{(F(x) - F(\bar{x}))(r - F(x))}{r} + \frac{F(x)(r - F(x))}{r}.$$

This can be seen by the simple derivation:

$$\begin{aligned}
\Delta(Q_1, Q_2) &= \sum_{i=0}^{r-1} P\big(Q_1[i] \neq Q_2[i]\big) \\
&= r\,P\big(Q_1[1] \neq Q_2[1]\big) \\
&= r\big[P\big(Q_1[1] = 0\big)P\big(Q_2[1] = 1|Q_1[1] = 0\big) \\
&\quad + P\big(Q_1[1] = 1\big)P\big(Q_2[1] = 0|Q_1[1] = 1\big)\big] \\
&= r\left[\frac{F(x)}{r}\left(\frac{F(\bar{x})}{F(x)} \cdot 0 + \frac{F(x) - F(\bar{x})}{F(x)} \cdot \frac{r - F(x)}{r}\right)\right. \\
&\quad \left. + \frac{r - F(x)}{r}\left(\frac{F(x)}{r}\right)\right] \\
&= \frac{(F(x) - F(\bar{x}))(r - F(x))}{r} + \frac{F(x)(r - F(x))}{r} \qquad (4)
\end{aligned}$$

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

Each query chooses V keywords out of U dummy keywords. While comparing two arbitrary queries, the expected number of dummy keywords that exist in both queries (E_O) is calculated as in the following.

Proposition 3 *Expected number of dummy keywords that both queries include can be calculated as follows*:

$$E_O(V) = \sum_{i=0}^{V} \frac{\binom{V}{i}\binom{U-V}{V-i}}{\binom{U}{V}} i. \qquad (5)$$

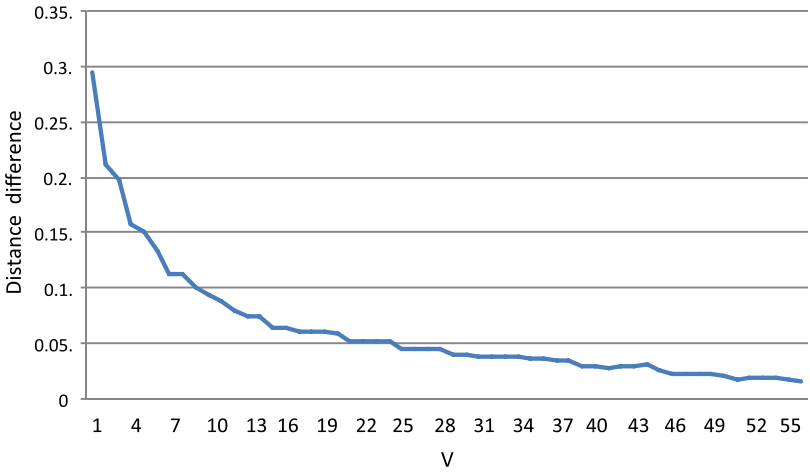


Fig. 2 Normalized difference of Hamming Distances between two arbitrary queries and two queries with same genuine search terms, for 3 genuine search terms per query where $U = 60$

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 one is calculated as follows: i keywords are chosen from the set of keywords that is selected also by the first query and $(V - i)$ keywords are chosen from the set of unselected keywords. Then we use summation to calculate the expected value in (5). Note that $E_O(V)$ is a monotonically increasing function (i.e., $V \geq V' \Leftrightarrow E_O(V) \geq E_O(V')$).

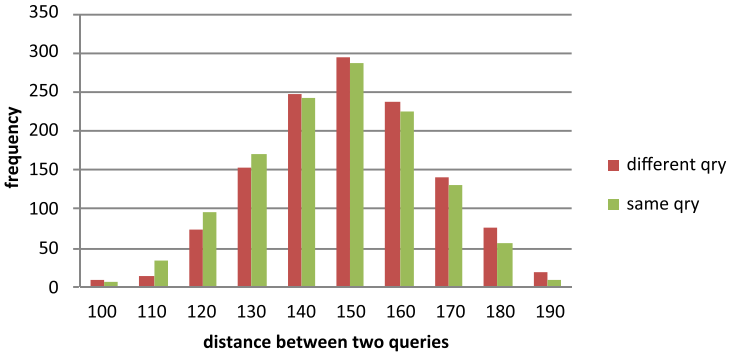
A possible way of choosing an optimum parameter setting is shown in the following example.

Example 3 We use 448 bits as the query size (r) and the largest U for this query size that provides sufficient accuracy (i.e., small false accept rate; cf. Sect. 6.4) is found as 60. Any further increase in U necessitates increasing the query size, which causes an increase in communication, computation and storage requirements (cf. Sect. 10). Using the formulae in (4) and (5), the normalized differences between $\Delta(Q_i, Q_j)$ for two arbitrary queries $\{Q_i, Q_j\}$ and $\Delta(Q_{i\alpha}, Q_{i\beta})$ where $\{Q_{i\alpha}, Q_{i\beta}\} \in \mathcal{Q}_i$ are given in Fig. 2. The normalized difference is calculated using

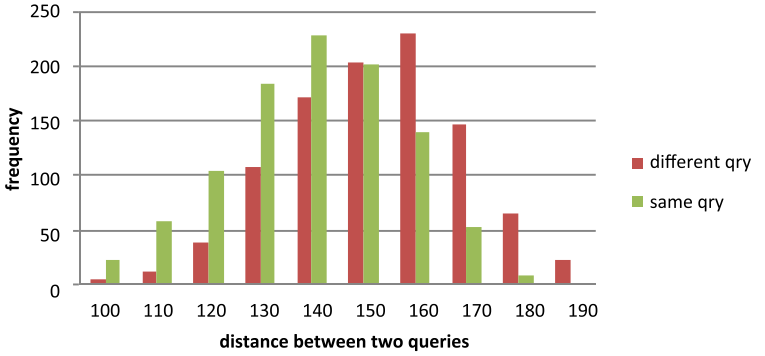
$$\frac{\Delta(Q_i, Q_j) - \Delta(Q_{i\alpha}, Q_{i\beta})}{\Delta(Q_i, Q_j)}.$$

One can demonstrate from Fig. 2 that when U is fixed as 60, $V = 30$ is the smallest value, which ensures that the distance between two queries $Q_i \in \mathcal{Q}_i$ and $Q_j \notin \mathcal{Q}_i$ is sufficiently close to the distance between $Q_{i\alpha} \in \mathcal{Q}_i$ and $Q_{i\beta} \in \mathcal{Q}_i$. Note that any $V \geq 30$ can also be used. The parameter setting that we used in our tests, is discussed more formally in Sect. 10.

To demonstrate the usefulness of our analysis, we conducted an experiment using synthetic query data for the case, where adversary does not know the number of



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



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

Fig. 3 Histograms for the Hamming distances between queries

genuine search terms in a query. We generate a synthetic data for a set of queries with the parameters $V = 30$ and $U = 60$ being fixed. The set contains a total of 250 queries, where the first 50 queries contain 2 genuine search terms each, the second 50 queries contain 3 genuine search terms each, and so on. And finally, the last set of 50 queries contains 6 genuine search terms each. We create another set that contains only 5 queries, which include 2, 3, 4, 5 and 6 genuine search terms, respectively. The distances between pairs of queries, 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 Fig. 3(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 queries, whereby queries in a pair contain the same genuine search terms with different dummy keywords. Both histograms are given in Fig. 3(a), where it can be observed that adversary cannot do better than a random guess to identify whether given two queries contain the same genuine search terms 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. We generated a set con-

taining a total of 1000 queries, whose subsets with 200 queries each contain 2, 3, 4, 5 and 6 genuine search terms, respectively. We then created a single query with 5 genuine search terms. We measured the distances of the single query to all 1000 queries in the former set of queries 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 queries with five identical search terms as shown in Fig. 3(b). As can be observed from the histogram in Fig. 3(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 queries are from the same search terms or not correctly with 0.6 confidence under the assumption that the number of genuine search terms in the query is known. Hence, this information should be kept secret, which is the case in our proposed method.

6.1 Correlation attack

It is possible that the attacker may have some prior knowledge on the statistical model of the search terms in the queries (e.g., search frequency of the most frequently queried search terms). In this case, the attacker may use this information to deduce a set of queries that all include a search term w. Then the trapdoor for w may be revealed with some error provided that the adversary obtains sufficient number of queries all featuring a search term w. In this section, the proposed method is analyzed against this attack that is referred as correlation attack. Note that the wisest choice of w for the adversary to attack will be the most commonly used search term which has the highest occurrence rate in the previous queries. The adversary may have prior knowledge of the most frequently queried search terms or may guess using the public statistics such as [29].

In order to analyze whether the attacker can identify a group of queries that all possess the same genuine search terms from other queries, we define a Distinguisher function $H(\mathcal{A}_k, Q_{k+1})$. This function takes a set $\mathcal{A}_k = \{Q_1, \ldots, Q_k\}$ that has k queries and a single query Q_{k+1} and returns the number of 0's that coincides the set $\mathcal{A}_k$ and Q_{k+1} (i.e., the number of query bit positions where all $k+1$ queries has 0 in that bit position). Let Q_{k+1} have x_{k+1} search terms, where $\bar{x}$ of them are common with all the queries in $\mathcal{A}_k$. The expected result for $H(\mathcal{A}_k, Q_{k+1})$ function is estimated as:

Definition 7 (Distinguisher function) The distinguisher function, $H(\mathcal{A}_k, Q_{k+1})$, is the number of 0's that coincide in the corresponding bits positions of the set $\mathcal{A}_k$ with k queries and the query Q_{k+1}.

Proposition 4 *Expected Value of Distinguisher Function can be estimated as*

$$H(\mathcal{A}_k, Q_{k+1}) = \begin{cases} F(\bar{x}) + \frac{(F(x_1)-F(\bar{x}))(F(x_2)-F(\bar{x}))}{r} \\ \quad \textit{if } \mathcal{A}_k = \{Q_1\} \textit{ (i.e. } k = 1) \\ F(\bar{x}) + \frac{(H(\mathcal{A}_{k-1}, Q_k)-F(\bar{x}))(F(x_{k+1})-F(\bar{x}))}{r} \\ \quad \textit{otherwise.} \end{cases} \tag{6}$$

Let a search term w be an element of all the queries in $\mathcal{A}_k$ and further assume $w \in Q_{k+1}$ and $w \notin Q'_{k+1}$. We define a dissimilarity function $\Omega(\mathcal{A}_k, Q_{k+1}, Q'_{k+1})$, as in the following.

Definition 8 (Dissimilarity Function)

$$\Omega\big(\mathcal{A}_k, Q_{k+1}, Q'_{k+1}\big) = \frac{|H(\mathcal{A}_k, Q_{k+1}) - H(\mathcal{A}_k, Q'_{k+1})|}{H(\mathcal{A}_k, Q_{k+1})} \tag{7}$$

If $\Omega(\mathcal{A}_k, Q_{k+1}, Q'_{k+1}) \leq z$, where z is a sufficiently small threshold parameter, we say that distinguishing the set of queries possessing the same search term w from other queries that do not, will be hard.

This function is analyzed for various values of inputs that are used to find the optimum choice for the parameters of randomness (i.e. U, V) that minimize $\Omega(\mathcal{A}_k, Q_{k+1}, Q'_{k+1})$ in (7). We present the results in Fig. 4.

Figures 4(a), 4(b), and 4(c) indicate that for a fixed value of U, increasing V decreases the dissimilarity of queries when compared with the set $\mathcal{A}_k$. In other words, it will be difficult to distinguish queries that possess w from those that do not. Note that since V is introduced for obfuscating queries it increases the similarity between unrelated queries as expected. Another issue that can be observed from the figures is that increasing U, which enables larger values for V, also decreases the dissimilarity of queries as expected.

Let the adversary be able to access all the search history (i.e., all previous queries from all users). If the adversary can find k queries that all feature an arbitrary search term w, where the dissimilarity function $\Omega(\mathcal{A}_k, Q_{k+1}, Q'_{k+1})$ is greater than z for that k, then the adversary can identify that all k queries include the same search term with a high confidence level and may learn the trapdoor of the search term w with a small error. Therefore, the probability of finding such k queries must be negligible.

We provide an example using the Reuters news dataset [22] that shows the difficulty of finding a trapdoor of a search term. Without loss of generality, we assume the adversary tries to find the trapdoor of the most commonly queried search term, which is the easiest choice.

Example 4 In large databases, the occurrence frequencies of real search terms are considerably small. For instance, in the Reuters dataset we use, the most frequent search term occurs in 7 % of all the documents. We assume that the same statistics apply to the real search patterns, which implies one search term w can occur at most 7 % of all the queries.

Let there be 1000 queries in the history where 70 of them expectedly feature the search term w. The probability of finding such k queries where the most frequent search term occurs in $p\%$ of the queries in a database of n queries, is

$$\frac{\binom{np/100}{k}}{\binom{n}{k}},$$

which is approximately 2^{-19} for $k = 5$, 2^{-39} for $k = 10$, 2^{-60} for $k = 15$ and 2^{-81} for $k = 20$ in a database with $n = 1000$ and $p = 7$. In other words, one has to try 2^{81}

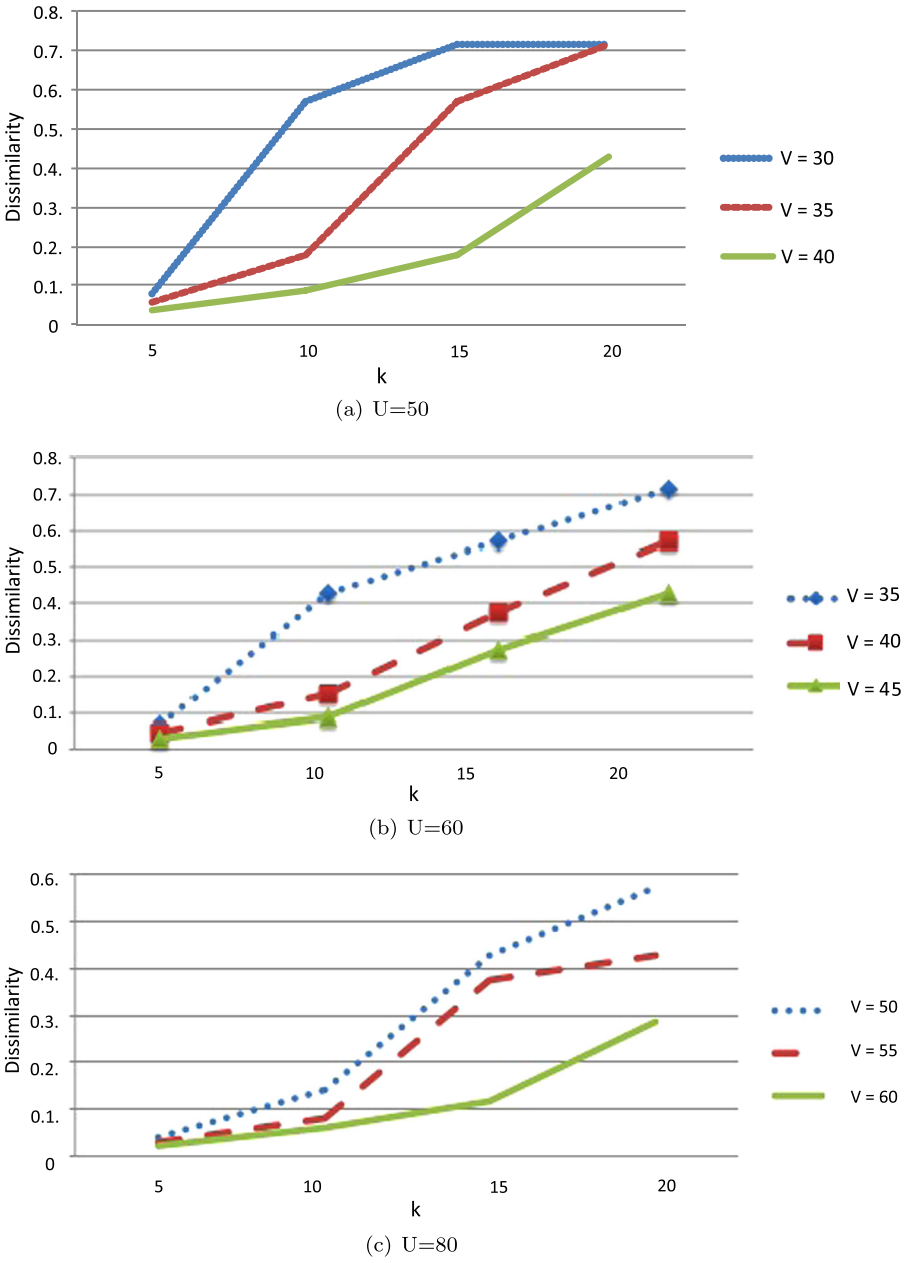


Fig. 4 Values of Dissimilarity Function (7) for different parameters

combinations of queries to find a correct set when $k = 20$. Note that increasing n has a very minimal effect on the calculated probability. For practical purposes, $k > 15$ satisfies sufficient security level, which implies that it is not feasible to find k queries

featuring the same search term. The attacker may find queries featuring the same search term for smaller values of k, but this time identifying whether they all include the same search term or not will be hard as shown in the next section.

6.2 Experiments

In order to demonstrate that our analyses are valid, we conducted experiments using synthetic data. Given a set of queries, we want to analyze the probability of identifying whether all the set elements contain a common genuine search term or not. We generate histograms that compare the number of 0's coinciding in two sets of queries. All queries in both sets have k genuine and V dummy search terms. While the first set has a common genuine search term w that exists in all the queries, the second set does not have any common genuine search term. We further compute the confidence levels indicating the reliability of the guess. While a confidence level of 0.5 means that one cannot do better than a random guess out of the two sets, a confidence level of 1.0 means that one can certainly identify the correct set from the given two sets. For the case where $U = 60$ and $V = 40$, the histograms that compare the number of coinciding 0's for a set of k queries are given in Fig. 5. In Table 1 we enumerate the confidence levels calculated from histograms.

We want the confidence level of the attacker to be less than 0.6. Our experiments indicate that setting the security threshold z for the dissimilarity function (Ω) as $z = 0.4$ gives sufficient level of obfuscation that satisfies the low confidence level required. From Fig. 4, the candidates that satisfy the required security level are $\{U = 50, V = 40\}$, $\{U = 60, V = 40\}$ and $\{U = 80, V = 55\}$.

6.3 Hiding dummy elements

During the query randomization process, we add $U \in \mathcal{U}$ dummy keywords in all the entries in the index. Similar to the genuine keywords, those dummy keywords are processed with HMAC and *Reduce* functions following the steps of Algorithm 1, which eventually maps some of the d-bit digits to single 0 bits. Since these U dummy keywords exist in all the entries, the bits that are assigned by those keywords are 0 in all the index entries $\mathcal{I}_{R_i} \in \mathcal{I}$. If the adversary has access to the searchable index file (e.g., cloud service provider), he can trivially identify bits set by dummy keywords by just marking bits that are 0 in all the index entries.

In order to hide those bits set by dummy keywords, we add fake index entries, where their bits are deliberately set. The adversary cannot distinguish bits that are set by genuine keywords, from the bits set by dummy keywords, if the distributions of the number of 0's are equivalent in both cases. Figure 6 shows the number of 0's in each bit location for, only genuine search terms (cf. Fig. 6(a)), after adding U dummy keywords (cf. Fig. 6(b)) and after the addition of fake index entries (cf. Fig. 6(c)). Figure 6 indicates that prior to the addition of fake elements (cf. Fig. 6(b)), the bits set by the dummy keywords are obvious, since all index entries contain the 0 bits in the same places. However, after the addition of fake entries (cf. Fig. 6(c)), they become indistinguishable from other bits. The number of fake entries is chosen to be equal to the number of genuine entries leading to doubling index size. However,

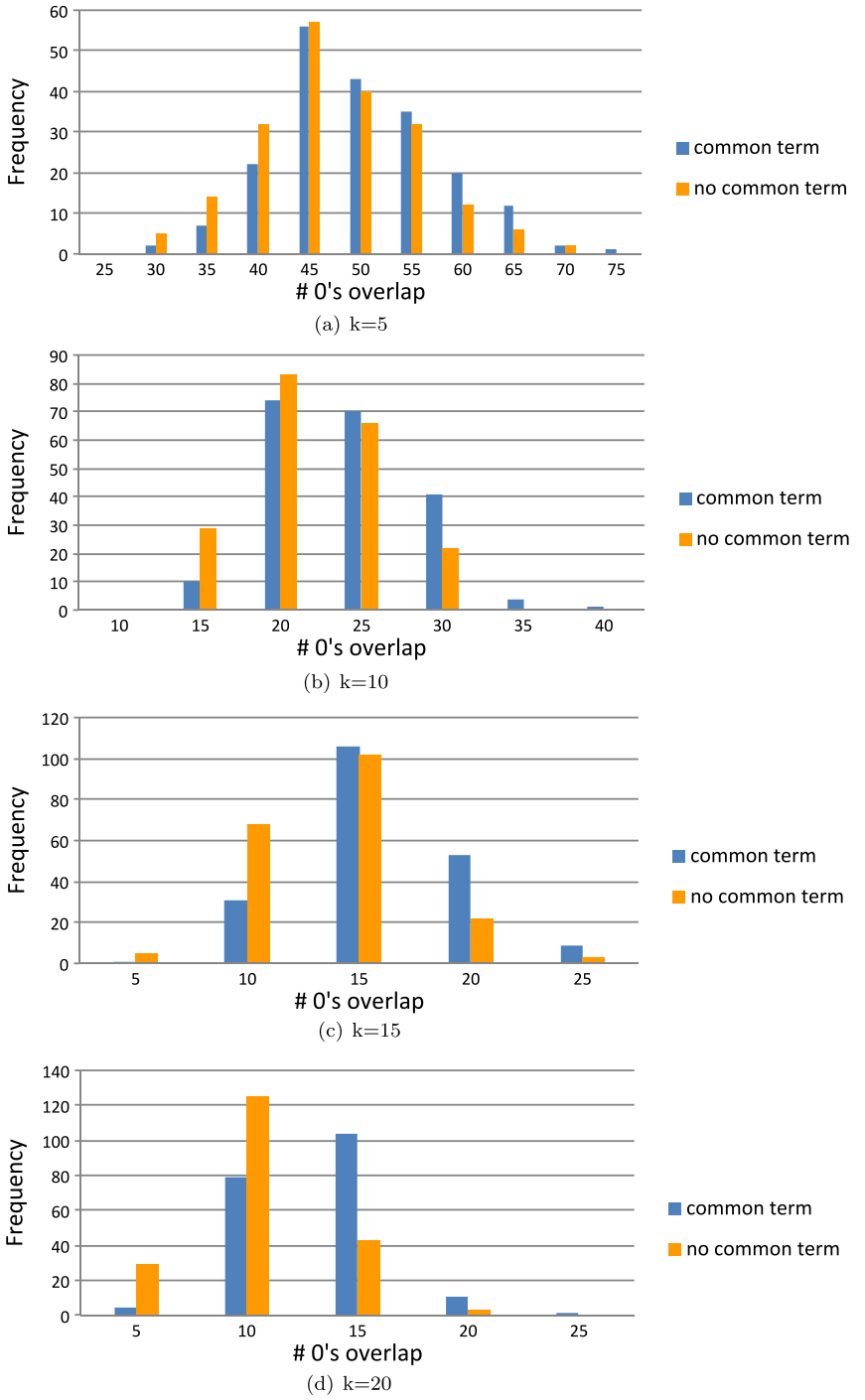
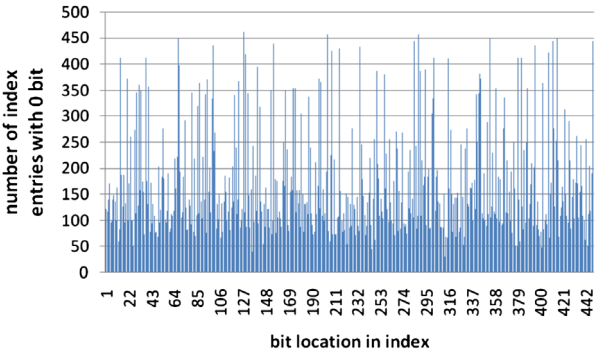


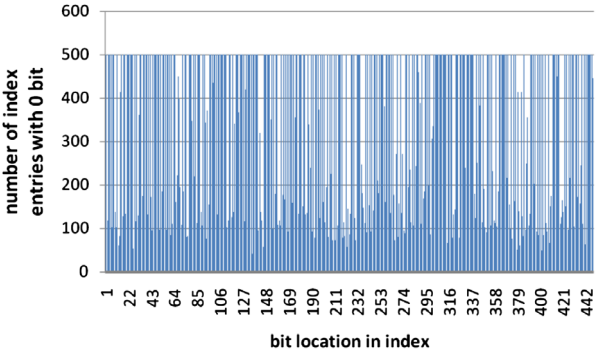
Fig. 5 Histograms that compare the number of 0's coinciding in k queries with a common search term and those with no common search term where $U = 60$ and $V = 40$

Table 1 Confidence levels of identifying queries featuring the same search term

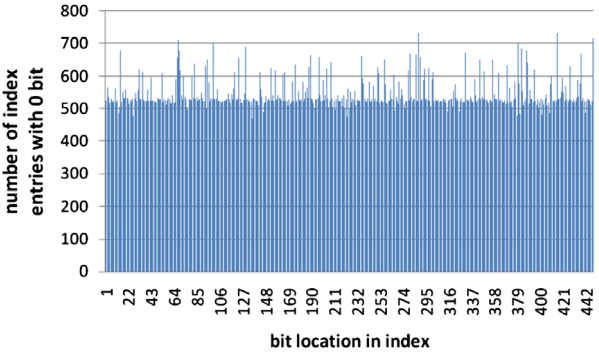
k	Confidence level
5	0.55
10	0.57
15	0.59
20	0.67



(a) before addition of dummy keywords



(b) with dummy keywords



(c) after addition of fake index entries

Fig. 6 Number of 0's in each bit location for 500 genuine and 500 fake entries (for (**c**))

since size of an index entry is very small (constant r bits) this is not a burden for the cloud server. Note that, the additional fake entries do not have any effect on the number of false positives (i.e., false accept rates) and hence, precision of the method is unaffected. Nevertheless, search time increases due to the increased number of index entries. However, the increase in search time is not significant due to the fact that it can be performed very efficiently.

The index server may have access to excessive number of search results from various users. Utilizing the search results, the server can identify some of the fake entries with high confidence via analysis of the number of matches with each index entry. Note that the expected number of matches with fake index entries is smaller than the genuine index entries. We propose two methods to prevent the correlation of fake index entries. Firstly, the data controller can change the HMAC keys and the pseudo identifiers of index entries periodically. This will alleviate the risk by limiting the index server's access to search results. Note that index generation can be done efficiently within a few minutes (cf. Sect. 10.1). Alternatively, a trusted proxy can be utilized to occasionally send fake queries that match with the fake index entries. If number of matches of fake index entries has a similar distribution with the genuine index entries then fake index entries are indistinguishable from genuine index entries.

6.4 False accept rates

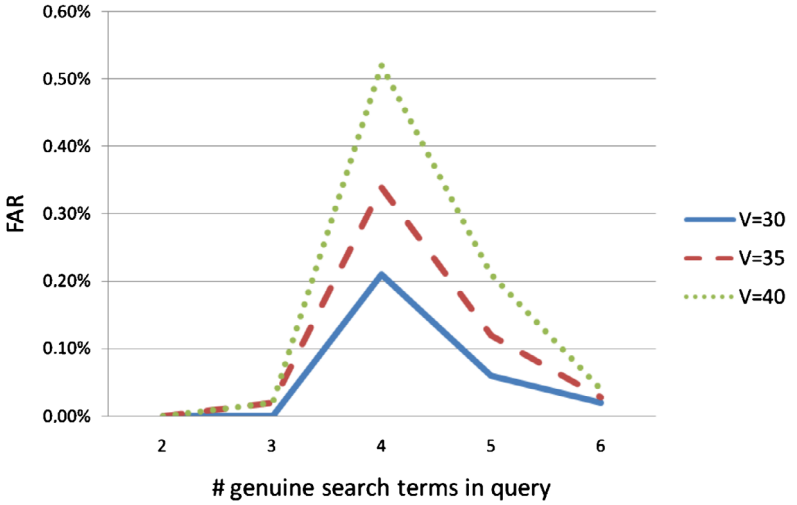
The indexing method that we employ includes all the information on search terms in a single r-bit index file per document. Despite the fact that the hash function employed in the implementations is collision-free, after the reduction and bitwise product operations there is a possibility that index of a query may wrongly match with an irrelevant document, which is called as a *false accept*. The false accept rates given in Figures 7 and 9 are defined as:

Definition 9 (False accept rate (FAR))

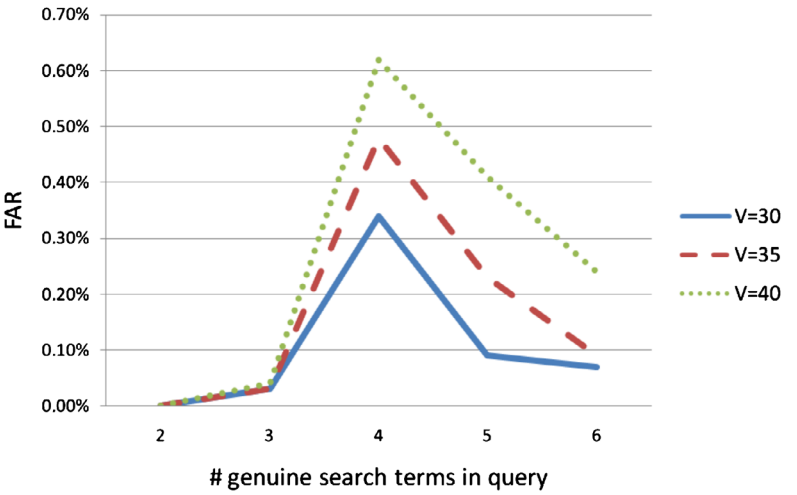
$$\text{FAR} = \frac{\text{number of incorrect matches}}{\text{number of all non-matches}} = \frac{fp}{fp + tn}$$

where fp is number of false positives (match) and tn is the number of true negatives (non-match).

Let m be the number of genuine search terms in a document. In Fig. 7, FAR is measured for queries with 2, 3, 4, 5 and 6 search terms for index size $r = 448$ bits and $U = 60$, whereby in Figs. 7(a) and 7(b) each document in the database has 30 and 40 genuine search terms (i.e., $m = 30$ and $m = 40$), respectively. When the number of genuine search terms in a query is small (less than 4), the noise in the query is limited, which results in 0 false positives (fp). When the number of search terms in a query is high (greater than 4), the number of matching documents is very small, which results in very high true negatives (tn) and therefore, very small false accept rate. While the false accept rates have a peak at 4 genuine search terms, the false accept rates are always smaller than 0.7 %.



(a) number of genuine search terms per doc is 30



(b) number of genuine search terms per doc is 40

Fig. 7 Effect of V in FAR, where $U = 60$

Figure 7 also indicates that an increase in the number of search terms in documents (m) also increases FAR. Note that FAR increases from Figs. 7(a) to 7(b) as m increases from 30 to 40. This is a result of the increase in the number of 0's in the index entries of the documents. If larger number of keywords is required for a document, we can increase the reduction parameter d and choose a longer HMAC function. Recall that as shown in (1) in Sect. 5.1, reduction maps a d-bit digit to a single bit where output is 0 with probability $1/2^d$ and 1 with probability $1 - 1/2^d$. Therefore, the ratio of the number of zeros in an index entry with m genuine and U dummy keywords with respect to index size can be estimated as $\frac{m+U}{2^d}$. Similarly

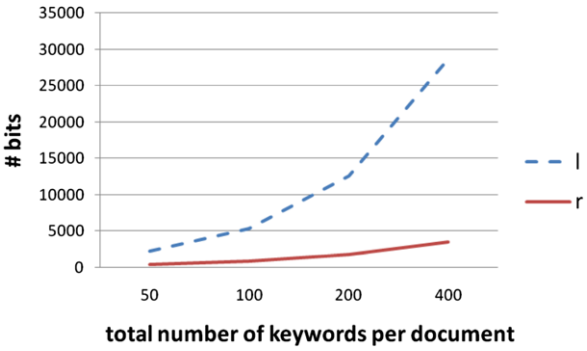


Fig. 8 Effect of increase in the total number of keywords $(m + U)$ per document on HMAC size (l) and index entry size (r)

each keyword is approximately represented with $\frac{r}{2^d}$ zero bits in an index entry. If d gets larger and r is kept constant, the number of zeros in the index decreases which may cause some keywords being not represented in the index. Provided that the ratios $\frac{m+U}{2^d}$ and $\frac{r}{2^d}$ do not change, the false accept rates will expectedly remain constant. If the number of genuine keywords in a document (m) doubles, the number of dummy keywords (U) also doubles and d is incremented by 1 to keep the ratio of $\frac{m+U}{2^d}$ constant. Due to the increase in d, r is also doubled to keep $\frac{r}{2^d}$ constant. In Fig. 8, we present the required output sizes for HMAC functions (l) and index entry (r) with respect to the total number of keywords (genuine and dummy) in documents such that maximum false accept rate does not exceed 0.7 %.

Figure 8 indicates that the increase in index entry size r is quite limited and can still be efficiently applied for large number of keywords per document. Although computing longer HMAC functions will also increase the cost of the index generation, since the index size r slightly increases, the communication cost, storage requirements and search time will remain at acceptable levels, without affecting the overall efficiency of the proposed scheme. Optimized value for the index size should be chosen considering the requirements of the applications.

Increasing the level of obfuscation (i.e., addition of more dummy keywords) increases the security of the method against an adversary trying to identify queries that feature the same genuine search term. Nevertheless, this will also increase the false accept rates. The results presented in Fig. 9 compare the false accept rates of the three parameter settings found in Sect. 6.2 that satisfy the security requirements.

Figure 9 indicates that the false accept rates increase with the number of dummy keywords (U) added. While FAR results for $U \leq 60$ are acceptable, the FAR for $U = 80$ is not suitable for various applications since it immediately incurs additional communication cost.

For a fixed U, the level of obfuscation increases as V gets larger (Fig. 4), however the false accept rates (FAR) also increase as V gets larger (Fig. 7). Therefore, we use the smallest V that satisfies sufficient obfuscation (i.e., satisfy $\Omega(\mathcal{A}_k, Q_{k+1}, Q'_{k+1}) < z$) as the optimum choice for V.

Given two queries with the same genuine search terms, the probability of having exactly the same set of dummy keywords must be very small. Otherwise the generated queries may be exactly same which leaks the information that the genuine search terms for the both queries are the same. This probability is $\binom{U}{V}^{-1}$ which is

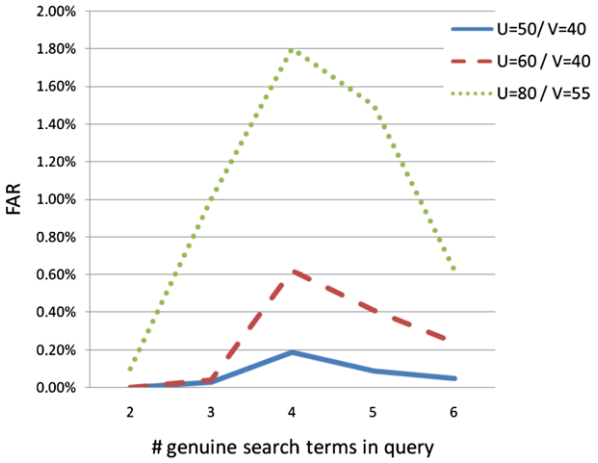


Fig. 9 FAR comparison, where number of genuine search terms per document is $m = 40$

maximized if $U = 2V$. For the two parameter settings that satisfy both low FAR and $\Omega(\mathcal{A}_k, Q_{k+1}, Q'_{k+1}) < z$, the probability of having the same set of dummy keywords is 2^{-52} for $U = 60$, $V = 40$ and it is 2^{-33} for $U = 50$, $V = 40$, which makes $U = 60$, $V = 40$ the optimum setting for our case. Note that the number of genuine keywords in each document is set as $m = 40$ in the experiments. Utilizing these tests on the Reuters dataset [22], we can generalize an optimum setting of U, V and m as follows:

$$\frac{m}{U} = \frac{2}{3} \quad \text{and} \quad \frac{V}{U} = \frac{2}{3}.$$

7 Hiding response pattern

In the previous section, we demonstrate that it is not feasible to link queries that feature the same genuine search terms by using the query information provided that the randomization parameters (i.e. U and V) are set appropriately. However, if the attacker has access to the database (e.g., cloud service provider) it will be possible to correlate queries with same search terms since the list of matching results will be almost the same except for some false accepts. Note that different queries can match with the same index entries due to different keywords in those documents. Nevertheless, if the list of matching documents is the same, then the attacker can guess with some confidence that the queries are also the same. Similar to the randomization method we use in Sect. 6.3, we propose to add some fake index entries[4] to the database such that the lists of matching documents for two queries with the same genuine search terms will be different.

In the basic scheme, other than the genuine keywords, index entry of each document possesses U random dummy strings, where random V of them are added to

[4]Users, but not the server, can identify the fake index entries. Since there is no document corresponding to fake entries, they will be discarded by the user.

each query index. In the modified method, similar to the real documents, the fake entries include both genuine search terms and dummy strings. The genuine search terms are placed in the fake entries according to the distribution in the real dataset, but with a constant factor more frequent. We define a frequency enhancer constant c as in the following.

Definition 10 (Frequency enhancer (c)) Fake entries in the index file include genuine search terms more frequently than real documents with a factor of frequency enhancer.

Namely, if a genuine search term w occurs in p % of the real documents in the database, it occurs in $c \cdot p$ % of the fake entries.

While the dummy strings are chosen with uniform distribution from the set $\mathcal{U}$, the number of dummy strings selected for a fake entry, which we denote as V', must be carefully set. The next sections show the analysis on as to how the values of V' and c are set.

Note that the fake entries we add here are generated in a different way from the ones we use for hiding positions of dummy keywords in Sect. 6.3. The fake entries generated in this section include both genuine search terms and dummy strings hence, can match with queries. However, the fake entries generated in Sect. 6.3 do not possess the dummy keywords and cannot match with the queries.

7.1 Analysis on selecting number of fake entries

Note that search index entry of each real document contains all U dummy strings while queries have only V out of those U strings. Let the fake document entries possess the dummy strings in a set $\mathcal{V}' \subseteq \mathcal{U}$ where $|\mathcal{V}'| = V'$. It is clear that $V < V' < U$. In order to match a query with a fake document entry, all the genuine search terms and the dummy strings in the query should also exist in that fake document entry which implies that $V < V'$ must hold for a match with a fake document index entry. If $V' = U$ as in the real document index entries, the lists of matching index entries of documents for two queries with same genuine search terms would be identical, a case which we want to avoid. Small V' reduces the number of fake document entries that match with a query. However, it increases the probability that the sets of matching fake entries to queries with same genuine search terms are different.

Given a query and a fake entry that possesses all the genuine search terms of that query, the probability that the query matches with the fake entry (p_F) (i.e., $\mathcal{V}' \subseteq \mathcal{U}$):

$$p_F = \prod_{i=0}^{U-V'-1} \frac{U-V-i}{U-i} \tag{8}$$

In Sect. 6.1, we show that an optimal choice for the parameters of dummy strings in our setting is when we set $U = 60$ and $V = 40$. While fixing U and V, we plot the values of p_F with respect to V' in Fig. 10. The figure shows that for values of $V' \leq 57$, p_F is very low, which drastically reduces the number of fake matches and

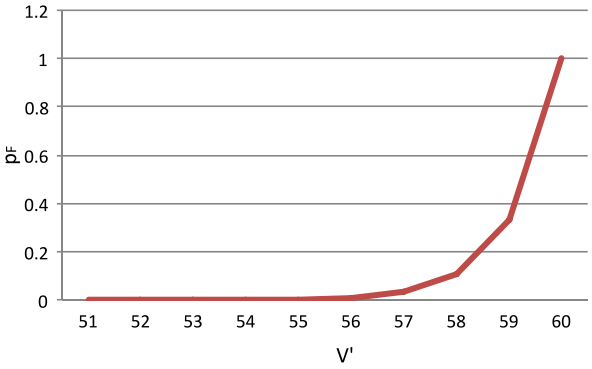


Fig. 10 p_F values with respect to V', where $U = 60$ and $V = 40$

thus decreases obfuscation. Therefore, the only two possible choices of V' for this setting are 58 and 59.

The security of the system is analyzed in the next section.

7.1.1 Correlating match results

Given a query, the number of matching fake document entries should be larger than the number of matching real documents. Otherwise, correlating two queries with the same genuine search terms can be possible. Let σ be the number of real documents in the database, we add $q \cdot \sigma$ fake entries to the index file. Also let f be the fake match enhancer such that if a query matches to a real document with probability p, it matches with a fake entry with probability $f \cdot p$, which is calculated as $f = c \cdot p \cdot p_F$ where p_F is as defined in (8) and c is the frequency enhancer as defined in Sect. 7. Then, the number of matching fake document entries will approximately be $f \cdot \sigma$. While this method increases the storage requirement for the database index by a factor of q, since the index entry size is very small (constant r bits) this is not a burden for the cloud server. Nevertheless, search and index generation times also increase with a factor of q, therefore q needs to be minimized. We set $q = 1$ and increase c to satisfy $c \cdot q \cdot p_F = f$.

The server can correlate two queries with the same genuine search terms if the number of index entries that both queries match is significantly larger than the average number of entries that any two arbitrary queries both match. We provide a theoretical analysis on the number of common matching entries for a given pair of queries. Note that there are three ways a document index entry can match to a query:

- *Case 1*. A real document entry can match if the document possesses all the genuine search terms in the query
- *Case 2*. A real document entry can falsely match due to the false accept rate
- *Case 3*. A fake document entry can match if the fake document entry possesses all the genuine search terms and dummy strings in the query

Let x be the number of genuine search terms in the query and p_i is the frequency of the ith search term in the database, assuming that the occurrences of search terms in a document are independent events, the expected number of index entries that match

from case 1 ($E(M_1)$) is:

$$E(M_1) = \sigma \prod_{i=1}^{x} p_i$$

Let FAR_x be the false accept rate of a query containing x genuine search terms and q be the multiplicative factor for the number of fake index entries as defined in Sect. 7.1. Then, the expected number of index entries that match from case 2 ($E(M_2)$) is:

$$E(M_2) = \text{FAR}_x \cdot (q+1)\sigma$$

Let p_F be as defined in (8), the expected number of index entries that match from case 3 ($E(M_3)$) is:

$$E(M_3) = cq\sigma p_F \prod_{i=1}^{x} p_i = f\sigma \prod_{i=1}^{x} p_i$$

Note that $E(M_1)$ is the expected number of true positive matches while $E(M_2)$ and $E(M_3)$ are the expected numbers of false positives (accidental and intentional respectively). Therefore, we denote $E(M_1)$ as $E(T^+)$ and $E(M_2) + E(M_3)$ as $E(F^+)$.

The expected total number of index entries that match to a query with x genuine search terms ($E(M)$) is:

$$E(M) = E\left(T^+\right) + E\left(F^+\right)$$

Note that $E(T^+) < \frac{E(F^+)}{f}$ and also note that $E(M_2)$ is reasonably small due to very small false accept rates given in Fig. 9. This implies that

$$E(M) \leq (f+1)E\left(T^+\right). \tag{9}$$

Given two arbitrary queries Q and Q', the expected number of common index entries that both queries match, denoted as $E(C_{arb})$, is estimated as:

$$\begin{aligned} E(C_{arb}) &= \frac{E(T_Q^+)}{\sigma}\frac{E(T_{Q'}^+)}{\sigma}\sigma + \frac{E(F_Q^+)}{q\sigma}\frac{E(F_{Q'}^+)}{q\sigma}q\sigma \\ &\approx \frac{E(T_Q^+)E(T_{Q'}^+)}{\sigma} + \frac{f^2E(T_Q^+)E(T_{Q'}^+)}{q\sigma} \end{aligned} \tag{10}$$

Given two queries Q and Q' that have the same genuine search terms, the expected number of common index entries that both queries match is estimated as:

$$\begin{aligned} E(C_{same}) &= E\left(T^+\right) + \frac{E(F_Q^+)}{q\sigma}\frac{E(F_{Q'}^+)}{q\sigma}q\sigma \\ &\approx E\left(T^+\right) + \frac{f^2E(T^+)^2}{q\sigma} \end{aligned} \tag{11}$$

We assume that the two queries have equal number of search terms in the queries compared (i.e., they have similar number of index entries matched) and therefore, $E(T_Q^+) \approx E(T_{Q'}^+)$. Note that otherwise they can easily be identified as different queries. We define an identifiability function $\mathcal{S}(Q_1, Q_1', Q_2)$ that takes three queries where Q_1 and Q_1' have the same genuine search terms and Q_2 is an arbitrary query and returns a value indicating the identifiability of Q_2 from Q_1.

Definition 11 (Identifiability function)

$$\mathcal{S}(Q_1, Q_1', Q_2) = \frac{E(C_{same}) - E(C_{arb})}{E(C_{same})}$$

The identifiability function can be calculated as:

$$\begin{aligned}
\mathcal{S}(Q_1, Q_1', Q_2) &= \frac{E(C_{same}) - E(C_{arb})}{E(C_{same})} \\
&= \frac{(E(T^+) + \frac{f^2 E(T^+)^2}{q\sigma}) - (\frac{E(T_Q^+)E(T_{Q'}^+)}{\sigma} + \frac{f^2 E(T_Q^+)E(T_{Q'}^+)}{q\sigma})}{(E(T^+) + \frac{f^2 E(T^+)^2}{q\sigma})} \\
&= \frac{E(T^+) - \frac{E(T^+)^2}{\sigma}}{(E(T^+) + \frac{f^2 E(T^+)^2}{q\sigma})} \\
&= \frac{1 - \frac{E(T^+)}{\sigma}}{(1 + \frac{f^2 E(T^+)}{q\sigma})} \\
&= \frac{1 - \frac{\sigma \prod_{i=1}^{x} p_i}{\sigma}}{(1 + \frac{f^2 \sigma \prod_{i=1}^{x} p_i}{q\sigma})} \\
&= \frac{1 - \prod_{i=1}^{x} p_i}{(1 + \frac{f^2 \prod_{i=1}^{x} p_i}{q})} \\
&= \frac{1 - \prod_{i=1}^{x} p_i}{(1 + f^2 \prod_{i=1}^{x} p_i)} \quad \text{since we set } q = 1 \qquad (12)
\end{aligned}$$

If we have the following inequality,

$$\mathcal{S}(Q_1, Q_1', Q_2) \leq \check{\epsilon}$$

where $\check{\epsilon}$ is a security threshold, we say that the attacker cannot identify whether two queries are from same search terms or not, from the information of matching index entry ids. Note that f and $\mathcal{S}(Q_1, Q_1', Q_2)$ are inversely proportional and therefore, we set f as large as possible by adjusting parameters q and c.

7.2 Experimental results

We conducted tests on the real data set [22] of 30, 000 database index entries (10,000 real, 10,000 fake from Sect. 6.3 and 10,000 fake from Sect. 7) to demonstrate the success in hiding the response patterns. We randomly generate four groups of genuine search term sets which contain from 2 up to 5 search terms. Each group has 200 elements with total of 800 sets of search terms. For each search term set, we generate 2 queries (i.e., same genuine search terms with different dummy keywords) and measure the similarity between the sets of matching index entry ids. Similarly we also generate another test group. This time, we apply 200 tests for each group by generating two different queries within the same group and measure the similarity between the sets of matching index entry ids. The results are illustrated in Fig. 11 where $U = 60$, $V = 40$ and $V' = 59$. In order to decrease c, we use the largest possible V' which is $U - 1$. Note that although higher f implies lower identifiability which is desirable, increasing f necessitates increasing c, which also increases number of genuine search terms in fake index entries. In our tests we set $f = 5$, which is the largest f that keeps the number of keywords in fake index entries the same as the number of keywords in genuine index entries.

The number of commonly matched index entries decreases as the number of search terms increases since increasing the constraints decrease the number of matching index entries. We observe that the number of matching index entries for a query with 2 genuine search terms is significantly higher than queries with more search terms. One can observe from Fig. 11 that when the number of common matching index entries is larger than 150, which occurs with probability 0.075, it can be identified that the two queries have exactly same genuine search terms. However, for lower values attacker cannot decide on the equality between those two queries. Note that by increasing f, the identifiability of queries can be further reduced in order to satisfy the required security of the database.

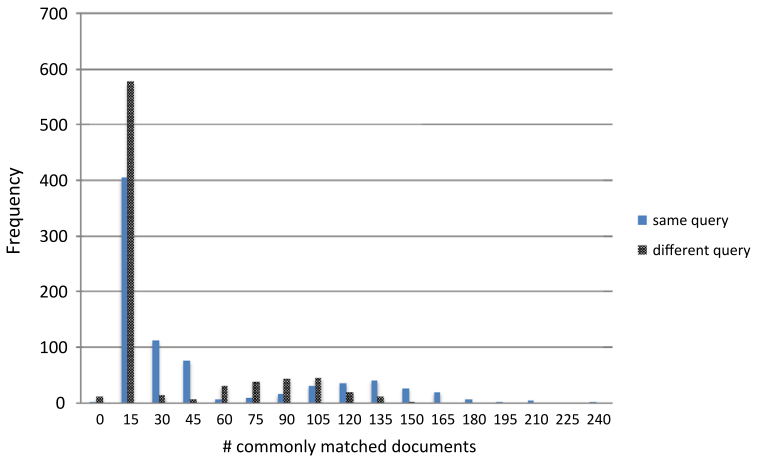


Fig. 11 Comparison of histograms between queries with same and different search terms for the number of commonly matched index entries

8 Ranked search

The multi-keyword search method explained in Sect. 4 checks whether queried keywords exist in an index entry or not. If the user searches for a single or a few 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 entries for frequently occurring keywords in a document. 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 index entry corresponding to a given search query.

There are four main metrics that are widely used in information systems [30]:

- *Term frequency* is defined as the number of times a keyword appears in a document. Higher term frequency implies more relevant document.
- *Inverse document frequency* measures rarity of a search term within the database collection. Intuitively a search term that is rare within the database but common in a document is given a higher relevancy. The inverse document frequency of a search term w is obtained as:

$$idf_w = \log\left(\frac{M}{df_w}\right)$$

  where M is the total number of document entries (fake and real) and df_w is document frequency of w (i.e., total number of documents containing w).
- *Document length (Density)* means given two documents that contain equal number of search terms, the shorter one is more relevant.
- *Completeness* means the more search terms the document contains, the more relevant that document is.

A commonly used weighting factor for information retrieval is *tf–idf* weighting [30]. Intuitively, it measures the importance of a search term within a document for a database collection. The weight of each search term in each document is calculated using the *tf–idf* weighting scheme that assigns a composite weight using both term frequency and inverse document frequency informations. The *tf–idf* of a search term w in a document R is given by:

$$tf\text{–}idf_{w,R} = tf_{w,R} \times idf_w.$$

The $tf\text{–}idf_{w,R}$ value is highest when the keyword w occurs frequently in R but occurs within small number of documents in the database (i.e., has low document frequency). It is lower when w occurs in many documents or less frequently in R. We assign *tf–idf* weights to each search term in each document. Instead of using these weights directly, we assign relevancy levels based on the weights of search terms. Note that the proposed search scheme is conjunctive and requires the document to contain all the queried search terms for a match. Therefore, completeness metric is not used in our scheme.

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 entry for search terms

Algorithm 3 Ranked search

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

with higher weights of that document in a cumulative way in descending order. This basically means that ith level entry includes all search terms in the $(i+1)$th level and the search terms that have sufficient weight for the ith level. The higher the level, the higher the weight of the search term is. For instance, if $\eta = 3$, level 1 index entry includes keywords that occur at least once in the document while levels 2 and 3 include keywords that have $tf\text{–}idf_{w,R}$ values at least, say 0.1 and 0.2,[5] respectively. There are several variations for relevancy score calculations [31] and we use a very basic method. The relevancy score of a document is calculated as the number representing the highest level search index entry that the query matches.

All the keywords that exist in a document are included in the first level search index entry of that document as explained in Sect. 5.1. The other higher level entries include the frequent keywords that also occur in its previous level, but this time they have to occur the number of times, which should at least be equal to the *tf–idf* of the corresponding level. The highest level includes only the keywords that have the highest *tf–idf* values. In the oblivious search phase, the server starts comparing the user query against the first level index entries of each document. The matching documents found as a result of the comparison in the first level are then compared with the index entries in the other levels as shown in Algorithm 3.

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 [17], given in the following:

$$Score(\mathcal{W}, R) = \sum_{w \in \mathcal{W}} \frac{1}{|R|} tf\text{–}idf_{w,R} \tag{13}$$

where $\mathcal{W}$ is the set of search terms in a query and $|R|$ is the length of the document R.

We use the Reuters [22] dataset to compare the two ranking methods. We generate a database of 10.000 documents and test with 100 queries of 2, 3, 4 and 5 genuine

[5] The number of levels and the weights of each level can be chosen in any convenient way.

Table 2 Number of matching documents per level

η	lvl					
	6	5	4	3	2	1
5	NA	3.8	11.7	21.9	28.2	59.2
6	2.7	9.5	15.6	23.4	25.3	46.7

search terms each. Note that in our proposed ranking there is no ordering within the matches from the same level. The number of elements matched in each level set is correlated with the number of levels.

- For the case where $\eta = 5$, 94 % of the time the top match for the given relevance score, is also in the top match level (i.e., matches with highest level) for our proposed ranking method. Additionally, in 82 % of the time, at least 4 of the top 5 matches for the given relevance score are in the top match level in our method.
- For the case where $\eta = 6$, 90 % of the time the top match for the given relevance score, is also in the top match level (i.e., matches with highest level) for our proposed ranking method. Additionally, in 79 % of the time, at least 4 of the top 5 matches for the given relevance score is in the top match level in our method.

Note that as η gets larger, number of genuine search terms in each level decreases which causes less number of matches in top levels. Since a user asks for top k rank levels, higher η provides user to retrieve accurate information with less communication. Nevertheless, reducing the number of matches per level slightly increase the probability of missing some top relevant documents in the top match level. For the given experiment on 10000 documents, average number of matching document entries per level is given in Table 2.

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 entries are stored instead of a single search index entry 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 size of an index entry is usually negligibly small compared to actual document sizes.

9 Privacy of the method

The privacy-preserving multi-keyword search (MKS) method must provide the user and data privacy requirements specified by definitions in Sect. 3. This section is devoted to the proofs that the proposed method indeed satisfies these privacy requirements. In proofs, we assume that the randomization parameters are selected appropriately by taking into consideration of the database or search statistics.

The proposed method is semantically secure against chosen keyword attacks under indistinguishability of ciphertext from ciphertext (ICC). The formal proof is provided in Theorem 1 of [21] from which we adopt their indexing method; therefore, we omit this proof here. Intuitively, the proof is based on the property that, since the HMAC

function is a random function, hash values of any two different keywords will be two random numbers and be independent of each other. Therefore, given two keyword lists L_0, L_1 and an index $\mathcal{I}_b$ for the keyword list L_b where $b \in \{0, 1\}$, it is not possible for an adversary to guess the value b with probability greater than $1/2$.

The security against chosen keyword attacks is required, but not sufficient for the privacy-preserving search scheme we define in Sect. 3. We consider further privacy requirements that the work in [21] does not satisfy. The major difference of our work from [21] is capability of hiding the search pattern.

Lemma 1 *Given three queries Q_1, Q_2 and Q'_b where $b \in \{1, 2\}$ and Q_b and Q'_b are generated from the same genuine search terms, for a suitable parameter choice, it is infeasible to find b.*

Proof In Sect. 6, it is shown that $\Delta(Q_1, Q'_b) \approx \Delta(Q_2, Q'_b)$ for some values of U, V. Therefore, it is not feasible to find b if the parameters U and V in the index generation are securely chosen. □

Note that the values of those parameters depend on the structure of the database and any $V \geq 30$, where $U = 60$, is a secure candidate for our test database.

Theorem 1 (Query privacy) *The proposed MKS method, for a suitable parameter choice, satisfies query privacy in accordance with Definition 1.*

Proof Let the adversary A be an authorized user. Given the search term lists L_1 and L_2, the attacker can get the corresponding trapdoors from the data controller and generate corresponding queries. By Lemma 1 it is not possible to correlate the generated query with the given query Q_b unless all the V random dummy keywords are also same. There are $\binom{U}{V}$ possible random choices of V for each search term list therefore, A needs to generate and try $\binom{U}{V}$ queries and compare with Q_b. Generating a single query requires choosing a set of V dummy keywords from the set $\mathcal{U}$, combining the trapdoors of those chosen keywords by bitwise product operation and finally comparing with the given query Q_b. For appropriate choice of U and V, generating $\binom{U}{V}$ queries and applying bitwise comparison for each of them is infeasible. □

A numerical example that demonstrates the difficulty of the attack, for the parameters that we use in our implementation is shown in the following example.

Example 5 In our setting this operation should be repeated $\binom{U}{V} = \binom{60}{40} = 2^{52}$ times. Note that assuming generating a single index from the trapdoors that are given in advance followed with a binary comparison requires 0.1 ms, this brute force search takes 10^4 years which is infeasible for all practical purposes.

Theorem 2 (Search term privacy) *The proposed MKS method satisfies search term privacy in accordance with Definition 2.*

Proof Assume that the adversary A has no information on the search terms corresponding to the given query. Specifically, let A learn a query $Q_{(w_i, w_j)}$ for two search

terms (w_i, w_j) and V dummy keywords; A cannot deduce the query Q_{w_i} for a single search term w_i from the received query. Let x of the r bits be 0 in $Q_{(w_i,w_j)}$ where x_i, x_j and x_R are the number of 0 bits resulting from search terms 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 assume that they do not overlap. On average, there will be $F(V)/F(1)$ times more 0's resulting from random keywords than x_i. A valid query Q_{w_i} must have 0 in all the x_i bits corresponding to the 0 bits from the trapdoor of w_i and must not contain any of the x_j 0 bits corresponding to the trapdoor of w_j. The probability of finding a valid query Q_{w_i}, which is denoted as $P(v_T)$, is smaller than choosing x_i 0's from x, where none of them is from the x_j 0's. Thus, the proposed method satisfies search term privacy if

$$P(v_T) < \frac{\binom{x_i}{x_i}\binom{x_j}{0}\binom{x-(x_i+x_j)}{y}}{\binom{x}{x_i+y}}$$

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

We provide an example with the adopted parameter setting here.

Example 6 In our setting, $r = 448$ bits and $d = 6$, therefore, there will be $F(V)/F(1) \approx 20$ times more 0's resulting from random keywords than x_i, which leads to the following inequality,

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

Since brute-force search is not possible this probability result indicates the security of the setting.

Lemma 2 *Given the list of all previous queries $\mathcal{Q}$ and a single query Q, it is not possible to find the list of queries from the set of queries $\mathcal{Q}$ that are generated from exactly the same genuine keywords with Q.*

Proof The trivial approach will be one-by-one comparison of Q with each element of $\mathcal{Q}$. However, by Lemma 1, it is proven that adversary cannot identify equality of queries with one-by-one comparison.

An advanced approach will be applying correlation attack. The adversary may try to find a set of k queries that all possess a genuine search term w. If the adversary can extract genuine search term information from a set of queries, he can correlate Q with other queries that are generated from exactly the same genuine keywords with Q. However, we have shown in Sect. 6.1 that it is not possible to apply correlation attack in our proposed scheme. □

Lemma 3 *Given the database* (*i.e., the searchable database index*), *the list of all previous queries* $\mathcal{Q}$ *and a single query* Q, *it is not possible to find the list of queries from the set of queries* $\mathcal{Q}$ *that are generated from exactly the same genuine keywords with* Q.

Proof With the additional index file access information, the adversary (e.g., cloud server) can also use the information of the list of ordered matching items with the query Q while comparing with other queries in $\mathcal{Q}$. As shown in Sect. 7, due to the additional fake document index entries, it is not possible to correlate two queries using one-by-one comparison between the lists of matching documents. □

Theorem 3 (Search pattern privacy) *The proposed MKS method satisfies search pattern privacy in accordance with Definition* 3.

Proof By Lemma 2, it is shown that the query information do not leak useful search pattern information. By Lemma 3, it is shown that the information of retrieved matching entries with a given query does not leak search pattern information. Therefore, the proposed scheme satisfies search pattern privacy. □

Theorem 4 (Non-impersonation) *The proposed MKS method satisfies non-impersonation property in accordance with Definition* 4.

Proof All communication from the user to the data controller 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 $e_u \cdot d_u = 1 \mod \phi(N)$. Therefore, the probability of impersonating an authorized user is ϵ where ϵ is the probability of breaking the underlying RSA signature scheme. □

10 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:
  Two steps in the proposed method are identified, where communication is required: (i) for learning the trapdoor and (ii) for sending the query and receiving the results.

  1. *Between the user and the data controller, for learning the trapdoor*: To build a query, the user first determines the bin IDs of the keywords he wants to search for and send these values to the data controller. Let γ be the number of genuine

Table 3 Communication costs incurred by each party (in bits)

	Trapdoor	Search
User	$32 \cdot \gamma + \log N$	r
Data controller	$\log N$	0
Index server	0	$\alpha \cdot r$

search terms the user queries. Then the user sends at most $32 \cdot \gamma$ bits to the data controller together with a signature since each bin ID is represented by a 32 bit integer. The data controller replies with the HMAC keys that belong 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 search terms 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 index server, for query*: After learning the trapdoor keys, the user calculates the query and transmits it to the server. The size of the query is r bits, independent from γ, so the user transmits only r bits. Let α be the number of index entries matched with the query. The server returns the index entries of the matching documents whose size is $\alpha \cdot r$ bits in total. Note that size of the encrypted document depends on the size of the actual document. Independent of the used scheme, all privacy-preserving search methods return the encrypted document, therefore the communication cost of document retrieval is not considered here. 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 $\tau \leq \alpha$.

The communication costs are summarized in Table 3.

– Computation Costs:

Among the three parties participated in the protocol, computation cost of the user is the most crucial one. The data controller 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 controller, query is generated as explained in Sect. 5.1, which is essentially equivalent to performing hash operations.[6]
2. *Index Server*: The index server performs only the search operation, which is binary comparison of r-bit query with $(q+2) \cdot \sigma$ database entries, each of which is again an r-bit binary sequence. Note that there are σ genuine database entries, σ fake entries added for hiding bit positions of dummy keywords (Sect. 6.3), and $q \cdot \sigma$ fake entries added for hiding response pattern (Sect. 7), which add up to a total of $(q+2) \cdot \sigma$ database entries. If the ranking is used, the query should also be compared with higher level index entries of the matching documents. So the server performs η additional binary comparison of r-bit index entries for each matching document, where η is the number of levels.

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

Table 4 Computation costs incurred by each party

User	1 hash and bitwise product
Data controller	initialization phase
	2 modular exponentiation per search
Index server	$(q+2)\cdot\sigma\cdot\eta$ binary comparison over r-bit index entries

3. *Data Controller*: The data controller creates the index file and symmetric-key encryptions of all documents; but these operations are performed only once in the initialization phase. Other than this, data controller is active while the user learns the trapdoors, which requires one encryption and one signature. This is equivalent to 2 modular exponentiations.

The computation costs are summarized in Table 4.

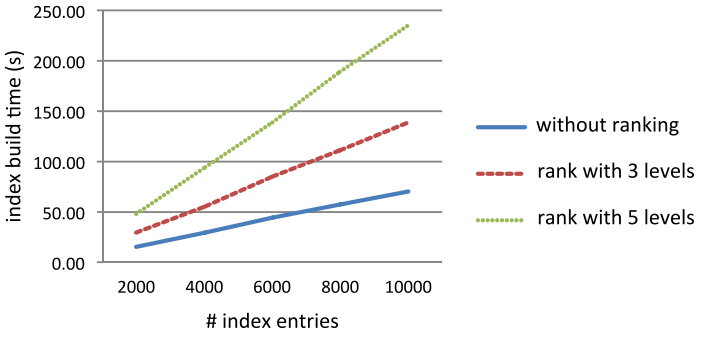
10.1 Implementation results

The entire system is implemented by Java language using socket programming on an iMac with Intel Core i7 processor of 2.93 GHz. Considering that initial document analysis for finding the keywords in the document 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 size of each database index entry and (r) is 56 bytes (448 bits).

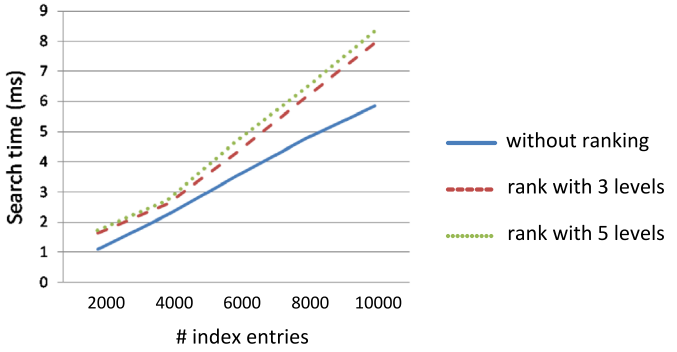
In our experiments, we used different datasets with different number of documents (from 2000 to 10000 documents). The timing results for creating the queries are obtained for documents with 30 genuine search terms and 60 random keywords each using ranking technique with different rank levels for parameters $q = 1$ and $f = 5$ in Fig. 12(a). Considering that index generation is performed only occasionally (if not once) by the data controller and that index generation problem is of highly parallelized nature, the proposed technique presents a highly efficient and practical solution to the described problem.

Figure 12(b) demonstrates the server timings for a search with different rank levels. As can be observed from the graphic in Fig. 12(b), time spent by the server per query is quite low, rendering high-throughput for the processing of user queries possible. By parallelization and native language support, the throughput can be increased by several orders of magnitude.

Most of the privacy-preserving search methods that exist in literature are only capable of single keyword search. The problem that we consider is multi-keyword search; therefore, we did not provide a comparison with the works that consider only single keyword search. A very recent work by Cao et al. [19] is the closest work to our proposed method. Our implementations show that our method is one to two orders of magnitude faster than the method in [19] in both off line and on line operations. The index construction method of [19], takes about 4500 s for 6000 documents



(a) Timings for index construction with 30 genuine and 60 random keywords per document (on data controller side)



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

Fig. 12 Timing results

while we need only 140 s in the highest rank level. Similarly the work in [19] requires 600 ms to perform a search over 6000 documents where we need only 4.7 ms. The tests in [19] were done on an equivalent computer, Intel Xeon processor 2.93 GHz. Among the other existing multi-keyword solutions, bilinear pairing based methods such as [20] provide only theoretical solutions. The method in [20] is not implemented due to its excessive computational requirements hence, cannot be compared with our proposed work. The work of Wang et al. [21], which is the inspiration for our proposed method, provides a faster solution than our work since they do not use additional fake entries or dummy keywords. However, that work does not satisfy some of the privacy requirements that we are interested in (cf. Sect. 9), such as hiding search pattern privacy.

The low time requirements on the data controller side enable 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 or parallel implementation will further increase the performance of the proposed system.

11 Conclusion

The proposed solution addresses the problem of privacy-preserving ranked multi-keyword search, where the database is outsourced to a semi-honest remote server. Our formal definitions pertaining to the privacy requirements of a secure search method are based on a comprehensive analysis of possible attack scenarios. One particular privacy issue concerning linking of queries featuring the identical search terms is often overlooked in literature. When an attacker is able to identify queries featuring the same search terms by inspecting the queries, their responses and database and search term statistics, he can mount successful attacks. Therefore, the proposed privacy-preserving search scheme essentially implements an efficient method to satisfy query unlinkability based on query and response randomization and cryptographic techniques. Query randomization cost is negligible for data controller and even less for the user. Response randomization, on the other hand, results in a communication overhead when the response to a query is returned to the user since some fake matches are included in the response. However, we show that the overhead can be minimized with the optimal choice of parameters. The true cost is due to the additional storage for extended index file and the actual search time. This can also be minimized by proper selection of parameters (i.e., the ratio of fake index entries to real index entries). On the other hand, the storage is usually not a real concern for cloud computers considering that index file is relatively small compared to document sizes. As for the search time, the proposed technique is extremely efficient that a relative increase in search time can easily be tolerated. Our implementation results confirm this claim by demonstrating search time over a database of $10,000$ documents, including ranking, takes only a couple of milliseconds. Considering that the search algorithm easily yields to the most straightforward parallelization technique such as MapReduce, the overhead in search time due to the proposed randomization method effectively raises no difficulty.

Selection of parameters involves some knowledge about the database and therefore, a priori analysis is required. However, our proposal needs only the frequency of the most used search terms and number of search terms used in queries. The formulation for parameter selection is simple and easy to calculate. Furthermore, we do not need to repeat the calculation process for different datasets. One can easily specify an upper bound on the frequency of the most used search terms and number of search terms that can be used for many cases.

Ranking capability is incorporated to the scheme which enables the user to retrieve only the most relevant matches. The accuracy of the proposed ranking method is compared with a commonly used relevance calculation method where privacy is not an issue. The comparison shows that the proposed method is successful to return highly relevant documents.

We implement the entire scheme and extensive experimental results using both real and synthetic datasets demonstrate the effectiveness and efficiency of our solution.

Acknowledgements 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 Örencik.

We would also like to thank Prof. Yucel Saygin for his valuable suggestions. Finally, we also would like to thank the anonymous reviewers for their valuable comments and suggestions.

References

1. Örencik, C., Savaş, E.: Efficient and secure ranked multi-keyword search on encrypted cloud data. In: Proceedings of the 2012 Joint EDBT/ICDT Workshops, pp. 186–195. ACM, New York (2012)
2. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. J. ACM **45**, 965–981 (1998)
3. Boneh, D., Kushilevitz, E., Ostrovsky, R., Skeith, W.: Public key encryption that allows PIR queries. In: Advances in Cryptology (CRYPTO 2007). Lecture Notes in Computer Science, vol. 4622, pp. 50–67. Springer, Berlin (2007)
4. Lipmaa, H.: First CPIR protocol with data-dependent computation. In: Information, Security and Cryptology (ICISC 2009), pp. 193–210. Springer, Berlin (2009)
5. Groth, J., Kiayias, A., Lipmaa, H.: Multi-query computationally-private information retrieval with constant communication rate. In: PKC, pp. 107–123 (2010)
6. Trostle, J.T., Parrish, A.: Efficient computationally private information retrieval from anonymity or trapdoor groups. In: ISC '10, pp. 114–128 (2010)
7. Cloud computing innovation key initiative overview (2012). http://my.gartner.com
8. Vaquero, L.M., Rodero-Merino, L., Caceres, J., Lindner, M.: A break in the clouds: towards a cloud definition. Comput. Commun. Rev. **39**, 50–55 (2008)
9. Google cloud platform (2012). http://cloud.google.com/
10. Amazon elastic compute cloud (Amazon ec2) (2012). http://aws.amazon.com/ec2/
11. Windows live mesh (2012). http://windows.microsoft.com/en-US/windows-live/essentials-other-programs?T1=t4
12. Chang, Y.-C., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: Applied Cryptography and Network Security, pp. 442–455. Springer, Berlin (2005)
13. Liu, Q., Wang, G., Wu, J.: An efficient privacy preserving keyword search scheme in cloud computing. In: Proceedings of the 2009 International Conference on Computational Science and Engineering, Vol. 02 (CSE '09), Washington, DC, USA, pp. 715–720. IEEE Comput. Soc., Los Alamitos (2009)
14. Ogata, W., Kurosawa, K.: Oblivious keyword search. J. Complex. **20**, 356–371 (2004)
15. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Theory of Cryptography Conference (TCC 2005), pp. 303–324 (2005)
16. Boneh, D., Franklin, M.K.: Identity based encryption from the Weil pairing. In: IACR Cryptology ePrint Archive, vol. 2001, p. 90 (2001)
17. Wang, C., Cao, N., Li, J., Ren, K., Lou, W.: Secure ranked keyword search over encrypted cloud data. In: ICDCS'10, pp. 253–262 (2010)
18. Kuzu, M., Islam, M.S., Kantarcioglu, M.: Efficient similarity search over encrypted data. In: Proceedings of the 2012 IEEE 28th International Conference on Data Engineering (ICDE '12), Washington, DC, USA, pp. 1156–1167. IEEE Comput. Soc., Los Alamitos (2012)
19. Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-preserving multi-keyword ranked search over encrypted cloud data. In: IEEE INFOCOM (2011)
20. Zhang, B., Zhang, F.: An efficient public key encryption with conjunctive-subset keywords search. J. Netw. Comput. Appl. **34**(1), 262–267 (2011)
21. Wang, P., Wang, H., Pieprzyk, J.: An efficient scheme of common secure indices for conjunctive keyword-based retrieval on encrypted data. In: Information Security Applications. Lecture Notes in Computer Science, pp. 145–159. Springer, Berlin (2009)
22. Lewis, D.D., Yang, Y., Rose, T.G., Li, F.: Rcv1: a new benchmark collection for text categorization research. J. Mach. Learn. Res. **5**, 361–397 (2004)
23. Oxford dictionaries, the OEC: facts about the language (2011). http://oxforddictionaries.com/page/oecfactslanguage/the-oec-facts-about-the-language
24. Dickson, L.E.: Linear Groups with an Exposition of Galois Field Theory. Dover, New York (2003)
25. Hacigümüş, H., Iyer, B., Li, C., Mehrotra, S.: Executing SQL over encrypted data in the database-service-provider model. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD '02), pp. 216–227. ACM, New York (2002)

26. Hore, B., Mehrotra, S., Tsudik, G.: A privacy-preserving index for range queries. In: Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB '04), vol. 30, pp. 720–731. VLDB Endowment, ??? (2004)
27. Hore, B., Mehrotra, S., Canim, M., Kantarcioglu, M.: Secure multidimensional range queries over outsourced data. VLDB J. **21**, 333–358 (2012)
28. Pinkas, B., Reinman, T.: Oblivious ram revisited. In: Proceedings of the 30th Annual Conference on Advances in Cryptology (CRYPTO '10), pp. 502–519. Springer, Berlin (2010)
29. Google trends (2012). http://www.google.com/trends/
30. Manning, H.S.C.D., Raghavan, P.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (2008)
31. Zobel, J., Moffat, A.: Exploring the similarity space. In: SIGIR FORUM, vol. 32, pp. 18–34 (1998)