A Unified Framework for Secure Search Over Encrypted Cloud Data

Cengiz Orencik*[1], Erkay Savas†[2] and Mahmoud Alewiwi‡[2]

[1]Beykent University, Istanbul, Turkey
[2]Sabanci University, Istanbul, Turkey

May 26, 2017

Abstract

This paper presents a unified framework that supports different types of privacy-preserving search queries over encrypted cloud data. In the framework, users can perform any of the multi-keyword search, range search and k-nearest neighbor search operations in a privacy-preserving manner. All three types of queries are transformed into predicate-based search leveraging bucketization, locality sensitive hashing and homomorphic encryption techniques. The proposed framework is implemented using Hadoop MapReduce, and its efficiency and accuracy are evaluated using publicly available real data sets. The implementation results show that the proposed framework can effectively be used in moderate sized data sets and it is scalable for much larger data sets provided that the number of computers in the Hadoop cluster is increased. To the best of our knowledge, the proposed framework is the first privacy-preserving solution, in which three different types of search queries are effectively applied over encrypted data.

Keywords:encrypted cloud data, multi-keyword search, k-nearest neighbor, range search, privacy preservation, scoring.

*cengizorencik@beykent.edu.tr
†erkays@sabanciuniv.edu
‡alewiwi@sabanciuniv.edu

1 Introduction

Recently, the need for reliably storing and efficiently processing massive amount of data experienced a sudden surge. The required computing and storage infrastructure to deal with the problem, however, is prohibitively expensive, thus most probably non-existent altogether for most small and medium-sized enterprises (SMEs). Even for large corporations, technical competency for the challenging task of big data management is often lacking or at least a matter of investment outside the core business area. Data outsourcing in the form of cloud computing is, therefore, a viable solution to relieve enterprises of the associated burdens.

Nevertheless, outsourced data often contains sensitive information, which can result in privacy violations [1, 2]. Performing any operation over the outsourced data in a secure and privacy-preserving manner, not only requires the encryption of data, but also of queries submitted to the cloud, operated by a third party.

While data encryption prior to outsourcing ensures the confidentiality of data content, the classical encryption methods do not allow even simple operations over the ciphertext. For instance, searching over the encrypted data for documents containing a particular set of keywords (multi-keyword search), finding similar documents to a given document (similarity search), or accessing a set of database elements that fall in a range in the n-dimensional space of attributes (range queries), which are very common queries, cannot be supported unless novel techniques, which enable these operations, are developed.

In recent years, several solutions have been proposed that are based on a searchable index structure which, succinctly represents the data without revealing the sensitive information. However, all existing work in the literature propose techniques that target a single type of search operation over the encrypted data. On the other hand, supporting different types of search operations is especially important for data sets composed of different data types (e.g., numeric, currency, name, date) as it increases data utilization.

Data and query confidentiality and hiding access patterns often rely on prohibitively expensive techniques in terms of computation complexity, which hinder their adoption in real applications. Thus, the efficiency of proposed solutions is a fundamental necessity. Also, another important concern is effectiveness, which evaluates the success rate of a method for matching relevant data items with a given query.

In this paper, to address all the expressed and implied concerns, we propose a novel, secure and searchable index generation framework that can

be utilized for three main query types, namely multi-keyword, k-nearest neighbor and range search operations. The proposed framework utilizes locality sensitive hashing (LSH) and homomorphic encryption for searchable index structure to achieve the privacy of user queries and applies symmetric encryption for data confidentiality mainly against the owner/operator of cloud storage. In the adopted index generation method, numeric data is represented by a set of terms and handled in the same manner as are keywords. Therefore, the same secure index structure is used for both range searches over numerical data and equality and similarity searches over text data. To the best of our knowledge, the proposed approach is the only solution, which enables efficient and effective applications of all three search operations over encrypted data within the same framework in a privacy-preserving manner.

The rest of this work is organized as follows. The related work in the literature is summarized in Section 2. Section 3 illustrates the proposed framework. Preliminary definitions are provided in Section 4. The security model used is defined in Section 5. Section 6 provides the construction of the privacy preserving search methods. Security analysis are given in Section 7. Section 8 presents the experimental results and Section 9 concludes the paper.

2 Related Work

Searchable encryption is a generic methodology that allows secure and privacy-preserving search over encrypted data. Many searchable encryption methods are proposed over the recent years in the literature [3, 4, 5, 6, 7, 8]. Curtmola et al. provide strong security definitions to formalize the security requirements of searchable encryption methods incorporating adaptive adversaries [9].

Bilinear pairings are proposed for keyword-based search over encrypted data [4, 5]. However, their computational costs are prohibitively high for practical applications in many circumstances. In [3], an encrypted inverted index structure is utilized to represent sensitive data. However, as user interaction is required during the search process, the practicality of the system is limited. Orencik and Savas [10] propose a multi-keyword search scheme that maps sensitive information to constant length binary arrays using hash functions. The method incorporates a limited ranking capability. Cao et al. [6] propose a multi-keyword search scheme that encodes sensitive data into two binary matrices with randomization techniques and uses inner product similarity during matching. Another multi-keyword search method that

returns the matching data items in a ranked ordered manner using sublinear search time is proposed by Strizhov and Ray [7]. The authors of this paper previously proposed a multi-keyword search method with ranking capability [8]. While that work only focused on multi-keyword search, the current work addresses a unified framework that supports a variety of different privacy-preserving search operations.

The problem of secure k-Nearest Neighbor (k-NN) or secure similar document detection aims to find top k most similar documents in an outsourced database to a given document or a relatively small set of documents without revealing document and query contents. Wong et al. [11] formalize security and execution requirements for secure k-NN search and propose a method called SCONEDB (Secure Computation ON Encrypted Data Base). The method utilizes a scheme referred as the asymmetric scalar-product-preserving encryption (ASPE) scheme, in which the query points and database points are encrypted in a different manner. While the encryption scheme ensures that the distances between data points are not recoverable in encrypted domain, the scalar product of a data point and a query point can be computed using their ciphertexts. Yao et al. [12] investigate the secure nearest neighbor problem and rephrased its definition. Instead of finding the exact nearest neighbor in the encrypted domain directly, the proposed method returns a partition of the encrypted data set, which is guaranteed to contain the exact nearest neighbor. Elmehdwi et al. [13] also consider the secure k-NN problem and propose a method that provides users' pattern confidentiality in addition to the security guarantees given by other works. Here, pattern confidentiality refers to hiding the access patterns of a user, which is regarded as a more challenging than the classical security requirements such as data, query and response confidentiality. Overall, the method provides high security guarantees, but the computation costs are significantly higher than the methods that reveal pattern confidentiality.

While one of the first methods on secure range search over encrypted data is proposed by Shi et al. [14], the model has security weaknesses such that value localization can be possible. Later, Hore et al. [15] propose a multidimensional range search based on bucketization techniques. In the method, data item are partitioned into a number of different multidimensional buckets depending on their attribute values. A query is transformed into a subset of buckets, whose contents are returned as the matching items. The disadvantage of this scheme is that the query results contain a large number of false positives that need to be eliminated by the user. Samanthula and Jiang propose a range query method based on secure multi-party computation. The data owner encrypts the database attribute-wise using

a homomorphic encryption scheme. The query is then executed based on a secure bit decomposition protocol. In another work, similar to our approach, Demertzis et al. [16] propose a method that reduce range search to multi-keyword search using range covering techniques with tree-like indexes.

In summary, a common characteristic of the works in the literature is that they target only a single type of query. This is a drawback and possibly decreases the data utilization as the outsourced data contains a rich variety of data types and therefore, necessitates different types of search queries. In this paper, we demonstrate that different types of queries can be supported in an efficient, effective and privacy-preserving manner within the same framework.

3 The Framework

In our framework, we provide privacy-preserving search in three different search models: multi-keyword search, k-NN search for documents (i.e., document similarity) and range query search. We consider a data outsourcing scenario that consists of three entities: data owner, two non-colluding semi-honest servers and users. The big picture for the interactions between the entities is illustrated in Figure 1.

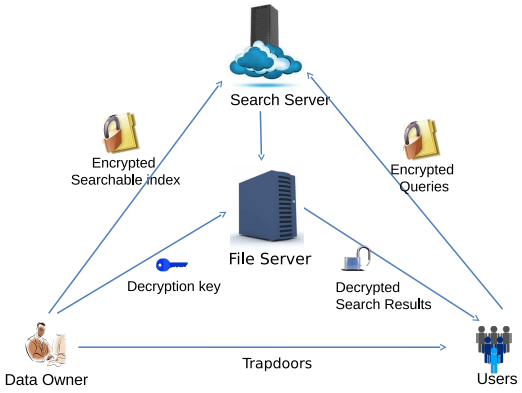

Figure 1: The framework

Data Owner is a person or organization that owns a data set and wants to outsource the database functionality along with the data to the cloud. Since the data may contain sensitive information, it is encrypted prior to its outsourcing to the cloud. Beside the encrypted data, a searchable index is also generated by the data owner for enabling secure search operations

and shared with the cloud. It is important to note that, searchable index and encrypted data are outsourced to two different non-colluding servers as explained below.

Cloud Server is a party that offers storage and computational services. We assume the cloud servers are semi-honest (i.e., honest but curious), a common model for cloud environment. In our settings we utilize two non-colluding servers: *file server* and *search server*.

The search server takes a query from a user, applies the requested search operation over the secure index and returns encrypted intermediate results to the file server. Then, the file server decrypts and sorts the intermediate results and sends the final results to the user. The aim of using two non-colluding servers is to hide the correlation between the queries and final results. The search server only learns the encrypted queries but not what they match with. Similarly, the file server learns the requested matching data but not the corresponding encrypted query itself.

Users are the authorized entities that have the right to query the cloud data. A user employs certain cryptographic techniques to protect the query confidentiality. A query generated in this manner is then sent to the search server. The user receives the search results from the file server.

As three different queries are supported in our scheme, it requires the execution of different types of operations over the encrypted data or rather its secure index. We will henceforth refer to them as the data mining operation for sake of simplicity. Then, the secure data mining operation can be formalized as follows. Let $\mathcal{D}$ be a database with n data records and Δ be the set of all features (or terms), which are included in data entries or queries. There are five main phases in the method, namely: *Setup*, *Trapdoor Generation*, *Index Generation*, *Query Generation* and *Search*.

1. *Setup (ψ)*: Given a security parameter (ψ), it generates a symmetric key and a public-private key pair as $\mathcal{K} = \{K_{id}, (K_{pub}, K_{pr})\}$.

2. *Trapdoor Generation (Δ)*: Given the list of all possible features Δ, it generates λ random permutations of Δ as $\mathcal{T} = \{P_1, \ldots, P_\lambda\}$, where λ is a precision parameter.

3. *Index Generation ($\mathcal{K}$,$\mathcal{D}$)*: Given a database $\mathcal{D}$, a secure searchable index $\mathcal{I}$, that represents each entry $D_i \in \mathcal{D}$, is generated by using the features of the entries D_i and the key $\mathcal{K}$.

4. *Query Generation ($\mathcal{T}, F$)*: Given the set of trapdoors $\mathcal{T}$ and a set of features to be queried, it generates a secure query Q for the given features that does not reveal the content information.

5. *Search* $(\mathcal{I}, Q)$: The given query Q is compared with the searchable index $\mathcal{I}$ and the encrypted entries from $\mathcal{D}$ that match with Q are returned.

4 Preliminaries

In this section, we provide a brief introduction to the building blocks used in our construction of the proposed scheme.

4.1 Locality Sensitive Hashing (LSH)

The main goal of locality sensitive hashing (LSH) [17] is to represent data items with arbitrary number of features using constant sized sets that are called signatures. The idea is to hash each feature set F_i into a constant size (and preferably small) signature that preserves the similarity between the data entries. More precisely, signatures provide an approximation for measuring the similarity between two data entries and the accuracy of the approximation is directly related with the length of the signatures, whereby longer signatures yield better accuracy in similarity. However, while very small signatures are usually sufficient for detection of either almost identical or totally unrelated data entries, relatively longer signatures are required for similarity levels that fall somewhere in between being identical and completely dissimilar.

In a LSH mapping, the requirement is that, given similar inputs, the mapping returns outputs that are likely to be similar. Conversely, if the inputs are dissimilar, the outputs are dissimilar as well with high probability. Note that, this requirement is completely different from that of cryptographic hash functions, whereby finding two different inputs that result in the same output is a very hard problem.

The signatures are represented as sets. A well known metric for representing the similarity between two sets is the Jaccard similarity.

Definition 1 (Jaccard Similarity)**.** *Let A and B be two sets, the Jaccard similarity of A and B is defined as in Equation 1.*

$$J_s(A, B) = \frac{|A \cap B|}{|A \cup B|}. \tag{1}$$

The elements of a signature are computed using MinHash functions [17], which is defined as follows.

Definition 2 (MinHash)**.** *: Let Δ be a set of elements, P be a permutation on Δ and $P[i]$ be the i^{th} element in the permutation P. MinHash of a set $F \subseteq \Delta$ under permutation P is defined as:*

$$h_P(F) = min(\{i | 1 \leq i \leq |\Delta| \wedge P[i] \in F\})$$

Each data element signature is represented by λ MinHash functions, each of which is applied using a different, randomly chosen permutation. The resulting signature for a data set element D that contains the set of features F is:

$$Sig(D) = \{h_{P_1}(F), \ldots, h_{P_\lambda}(F)\}, \tag{2}$$

where h_{P_j} is the MinHash Function under permutation P_j.

The MinHash functions are used while generating the signatures as there is a perfect correlation between the Jaccard similarity and MinHash function outputs. The probability that a MinHash function provides the same output for two inputs A and B is equal to the Jaccard similarity between the sets A and B, as shown in Equation 3. [17]

$$Pr[h(A) = h(B)] = \frac{|A \cap B|}{|A \cup B|} = J_s(A, B). \tag{3}$$

As each MinHash function with a different permutation provides an independent experiment, using longer signatures (i.e., larger λ) provides more accurate results.

4.2 Term Relevancy Score

The signatures obtained using MinHash functions as described capture the similarity between two database entries in accordance with the number of common elements shared by both of them. However, they cannot represent the importance of the common terms. Therefore, we also utilize tf-idf weighting factor [18], which is derived from term frequency (tf) and inverse document frequency (idf) values, to capture the importance of each term in an entry. While term frequency represents the occurrence rate of a term within a database entry, idf measures the rarity of the terms within the whole data set.

4.3 Hadoop and MapReduce Framework

The Hadoop framework [19] is a well known and widely used parallel processing framework, which employs a cluster of computers for parallel processing.

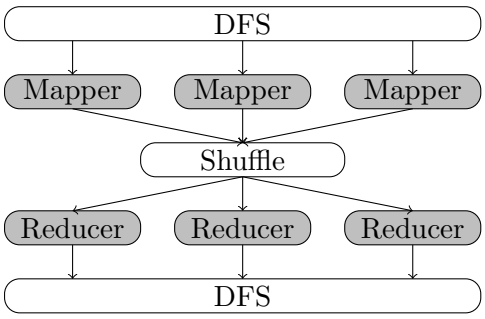

Figure 2: MapReduce Job Execution

The Hadoop framework works utilizing the MapReduce programing model, which employs a parallel execution and coordination method that can be used to manage complex computations over massive data sets [17].

Data replication is one of the key factors that improves the effectiveness of the Hadoop framework, which survives node failures while utilizing huge number of cluster nodes for data intensive computations. In the Hadoop framework, the MapReduce model is successfully implemented such that the details of the network communication, process management, inter-process communication, efficient massive data movements and fault tolerance are all transparent to the user. Typically, a developer needs to provide only configuration parameters and several high-level routines.

The MapReduce model is based on two functions, *Map* and *Reduce*. The Map function is responsible for assigning a list of data items, represented as key-value pairs, to cluster nodes. It receives key-value pairs, and sends the results as intermediate data to Reduce function. The Reduce function receives the intermediate data as a combination of a key and a list of values as `(key, [values])` and applies the user defined processing operations on the value part of the data. The shuffling process between the Map and Reduce functions, as illustrated in Figure 2, is responsible for designating all data items with the same key values to the same computation node in the cluster. The Reduce function performs desired operations for the records that share a common property and sends the final results to the user. Figure 2 depicts the working principle of the MapReduce framework.

5 Security Model

In this section, we analyze the security of the proposed scheme in the passive adversary model as it is a common adversarial model considered in secure data outsourcing scenario [15]. We assume the adversary is anyone on the search server side (e.g., a database administrator) as it is the entity that has the access to the secure index and secure queries and may try to extract any sensitive information leaking from them. In this model, the server is considered honest but curious, i.e., it implements all storage and processing functions correctly, but learns from any information leak.

The main privacy requirements that need to be satisfied are data and query confidentiality, whose intuitive definitions are given in Definitions (3) and (4). The privacy can further be extended to hide some side information such as access pattern confidentiality. Violation of access pattern confidentiality may cause the system be subject to some adversarial analysis [20]. Although access pattern confidentiality can be provided with oblivious RAM techniques [21, 22], due to efficiency concerns, we allow to leak that information in this work, which is in parallel with most of the related work in the literature.

Definition 3 (Data Confidentiality)**.** *A secure search scheme provides data confidentiality if the outsourced data (i.e., encrypted data and searchable meta-data) does not leak the information of the actual content or features of the data set elements.*

Definition 4 (Query Confidentiality)**.** *A secure search scheme provides query confidentiality if the given query does not leak the information of the actual queried terms or features.*

6 Proposed Method

This section provides the details of the proposed unified framework for the three search methods. Overall, the secure search operations are executed as follows. Initially, a secure (encrypted) search index structure is generated by the data owner, over which the search operation is performed. The secure search index is then sent to the search server and the actual encrypted data is sent to the file server. While it is still possible for an authorized user to search for the desired features, the secure index prevents the cloud service provider (i.e., search server) from learning sensitive information about the actual data (see Figure 1).

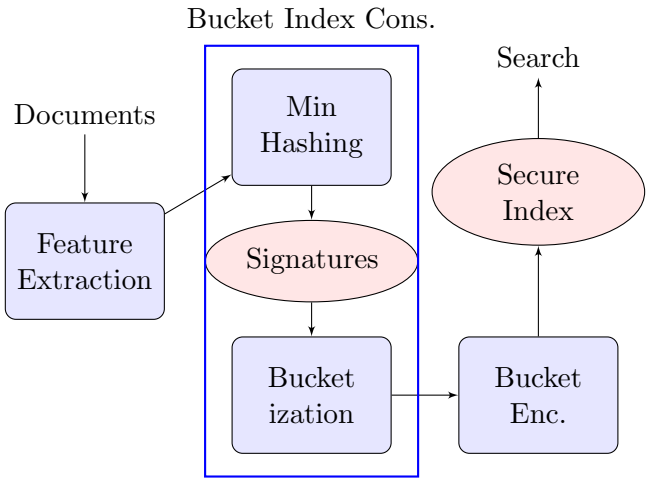

Figure 3: Flowchart of secure index generation

6.1 Secure Search Index Generation

In all three of our search methods, we utilize the bucketization technique developed by Kuzu et al. [23] for similar purposes. Bucketization is a partitioning technique which distributes each database entry (e.g., documents in our case) into a constant number of buckets according to the outputs of the MinHash functions (i.e., signatures). Due to the principle of locality in MinHash functions, the probability of two documents be assigned to the same bucket is the same as their Jaccard similarity, as shown in Equation (3). Hence, more similar documents share higher number of common buckets.

The main idea of our work is to create a signature based on MinHash functions to represent each document in the data set. The signatures are used to compare the corresponding documents. The proposed method for constructing the secure index consists of three phases, namely: feature extraction, bucket index construction and bucket index encryption. These phases, as illustrated in Figure 3 and explained in the next sections, are performed by the data owner in an offline stage.

6.1.1 Feature Extraction

For each data element $D_i \in \mathcal{D}$ the corresponding feature set $F_i = \{f_i^1, \ldots, f_i^y\}$ is extracted.

In the case of documents of text data, which are used for k-Nearest

Neighbor (k-NN) search and multi-keyword search in this work, each feature is composed of pairs $f_i^j = (w_{ij}, rs_{ij})$, where w_{ij} is the term j in document i, and rs_{ij} is the relevancy score of that term in the corresponding document. In this work, the tf-idf value is used for the relevancy score of a term for the document. This relevancy score is then used to rank the results of the queries in the search process. Since relevancy scores (rs_{ij}) are sensitive information, they are uploaded to the cloud only after encryption. There is a family of cryptographic algorithms, known as homomorphic cryptosystems that allow certain operations to be performed over the encrypted data (ciphertext) and generates an encrypted result which, when decrypted, matches the result of operations performed on the plaintext. In order to take advantage of these homomorphic properties, we use Paillier encryption [24], which is a well known additive homomorphic encryption method. The relevancy scores are encrypted with the Paillier scheme and all the numerical calculations on the server are performed over these encrypted values, thanks to its homomorphic properties.

In a feature set F_i of a document, there can be several terms with very low relevancy scores. Representing all those terms with low importance has an adverse effect on the accuracy of the method, since they may obstruct terms with high relevancy score in the signature. Therefore, we prune the feature set and use only the terms with tf-idf values higher than a predefined threshold σ. In the case, where there is no or only very few terms with tf-idf values higher than σ, we select t_{min} terms with the highest relevancy scores. By using this constraint, we guarantee that each document is represented by at least t_{min}, but possibly more, terms.

In the case of numeric data, which is used for range search, the numeric values are converted to text values that represents the underlying numeric data. Let the range of the numeric data be $[v_{min}, v_{max}]$. This range is discretized to t values as

$$[v_{min} < v_1 < v_2 < \ldots < v_t < v_{max}].$$

Any numeric data x such that $v_i < x \leq v_{i+1}$, is represented by the following t features:

$$\text{“}x > v_1\text{”}, \ldots, \text{“}x > v_i\text{”}, \text{“}x < v_{i+1}\text{”}, \ldots, \text{“}x < v_t\text{”}.$$

As each numeric data is represented by a set of features, the setting is almost identical to the setting of multi-keyword search. The only difference is in the scoring part. In the feature set of a data x, each feature like “$x > v_i$” either occurs a single time or does not occur at all. Therefore a binary scoring is

chosen instead of a term frequency based scoring. Hence, for range search, the features of data are still represented as pairs $f_i^j = (w_{ij}, rs_{ij})$ but the score values rs_{ij} are constant and equals to 1.

6.1.2 Bucket Index construction

In the bucket index construction there are two phases. First, using the feature sets and the MinHash functions, a constant length signature is generated for each data set entry. Each feature set is hashed with λ MinHash functions, where each function is under a random permutation P_j of the set of all possible features Δ. Let F_i be the feature list of document $D_i \in \mathcal{D}$, then the signature of D_i is calculated as

$$Sig(D_i) = \{h_{P_1}(F_i), \ldots, h_{P_\lambda}(F_i)\}. \tag{4}$$

The signatures are sets of λ elements. An important property of signatures is that, the similarity between signatures of two data items represents the similarity of the underlying feature sets of the corresponding data items as discussed in Section 4.1.

After the generation of signatures, the identifier of each data item is distributed to λ different buckets according to their signatures. Let B_k^i be the bucket identifier for the i^{th} MinHash function with output k, the content vector $V_{B_k^i}$ that stores the contents of the corresponding bucket is defined as

$$(id(D_j), rs_{jk}) \in V_{B_k^i} \iff id(D_j) \in B_k^i. \tag{5}$$

Note that, independent of the number of its features, each data element is mapped to exactly λ different buckets, but the total number of different buckets depends on the set of features Δ, the feature set of each data element (F_i) and the randomly chosen MinHash functions.

6.1.3 Bucket Index Encryption

Both the bucket identifiers and the bucket content vectors contain some sensitive information, which need to be protected prior to outsourcing. The bucket identifier represents the MinHash function used and its output. This may reveal some important information of the input feature set such as the terms it does not contain and one term that it definitely contains. Therefore, bucket and query contents should be protected using a cryptographic primitive. However, the server also needs to be able to match the queried

encrypted buckets to the buckets stored in the server. This requirement necessitates using a deterministic cryptographic scheme for protecting the bucket identifiers. HMAC functions can be considered as cryptographic hash functions that take a secret key as input besides the message. We use HMAC functions to hide the information that may leak from bucket identifiers. As the process is deterministic, comparison of buckets is still possible. The HMAC secret key (K_{id}) is not revealed to any of the servers, hence it is secure against any brute-force attack. The encrypted bucket identifier is denoted as

$$\pi_{B_k^i} = HMAC_{K_{id}}(B_k^i). \tag{6}$$

On the other hand, the bucket content vector stores the document identifiers and relevancy scores of the data elements, which are mapped to that bucket. In order to get the score of a document, it is required to map the bucket elements with the same document identifier. Therefore, the document identifiers are again hashed with a cryptographic hash function that is denoted as $H(id(D_i))$. For the relevance scores, we use additive homomorphic encryption methods that permit homomorphic addition operation over the encrypted data without requiring decryption. More specifically, we use the Paillier encryption scheme [24], which is a very efficient additive homomorphic encryption technique. Formally, Paillier encryption satisfies the following fundamental homomorphic property,

$$E(m_1, r_1) \cdot E(m_2, r_2) = E(m_1 + m_2, r_3),$$

where m_1 and m_2 are two messages and r_i values are random numbers.

A public encryption key (K_{pub}) for the Paillier cryptosystem is used for encrypting each bucket element before outsourcing the data to the server. The encryption of the bucket content vector $V_{B_k^i}$ is denoted as $\mathcal{V}_{B_k^j}$ and the result of this encrypted vector is a list of pairs as

$$\mathcal{V}_{B_k^i} = \{(H(id(D_j)), E_{K_{pub}}(rs_{jk})\} \forall id(D_j) \in B_k^i. \tag{7}$$

Paillier encryption provides semantic security against chosen plain text attacks, which means multiple encryptions of the same message result in different ciphertexts, which cannot be linked.

The secure index generation method is summarized in Algorithm 1.

Finally, the data owner outsources the secure index $\mathcal{I}$ to the search server and the private key K_{priv} to the file server (i.e., the two non-colluding servers).

Algorithm 1 Index Generation

Require: Δ: set of possible terms, $\mathcal{D}$: database, h: *MinHash* function, Ψ: security parameter

Ensure: $\mathcal{I}$: Secure index for $\mathcal{D}$

$K_{id}, K_{pub}, K_{priv} \leftarrow Setup(\Psi)$
$\mathcal{L} \leftarrow \emptyset$ {$\mathcal{L}$: Bucket identifier list}
for all $D_i \in \mathcal{D}$ **do**
  $F_i \leftarrow$ extract features of D_i
  $Sig(D_i) \leftarrow \{h_{P_1}(F_i), \ldots, h_{P_\lambda}(F_i)\}$
  for $j = 1$ to λ **do**
    $B_k^j \leftarrow Sig(D_i)[j-1]$
    if $B_k^j \notin \mathcal{L}$ **then**
      add B_k^j to $\mathcal{L}$
      create $\mathcal{V}_{B_k^j}$ as an empty vector
    end if
    add $(H(id(D_i)), E_{K_{pub}}(rs_{ik})$ to $\mathcal{V}_{B_k^j}$
  end for
end for
for all $B_k^j \in \mathcal{L}$ **do**
  $\pi_{B_k^j} \leftarrow HMAC_{K_{id}}(B_k^j)$
  add $(\pi_{B_k^j}, \mathcal{V}_{B_k^j})$ to secure index $\mathcal{I}$
end for
return $\mathcal{I}$

6.2 Secure Query Generation

This section provides the query generation process using the trapdoors. Again there are three types of queries; keyword search, k-NN search and range search. All those query types are represented as multi-term search operations, where all the queried terms are combined and encoded in a single secure query. However, there are still some minor differences in between each specific query.

- In the case of keyword search, the query is a set of any number of terms. All the terms in the query have equal relevancy scores (i.e., tf-idf values) such that no term has more significance than others.
- In the case of k-NN search, we search for the k most similar documents to a given document, hence the query is the set of all terms in a

document. Similar to the keyword search, the query is represented as a set of terms, with the only difference that each term in the set have different relevancy score depending on its tf-idf value.

- In the case of range queries, the range boundaries (i.e., $[v_{min}, v_{max}]$) are also represented as a set of terms as $\{x > v_{min}, x < v_{max}\}$. The search query can also include multiple ranges from different attributes such as "$salary > w_1$" and "$age > v_1$", "$age < v_2$". Similar to the keyword search, all queried range terms have the same relevancy score.

Secure query generation is very similar to the index generation process. As in the flowchart given in Figure 3, after feature extraction MinHash functions are applied on the query terms which create a query signature. Let the set of query terms be F_q, then the query signature is calculated as

$$Sig(F_q) = \{h_{P_1}(F_q), \ldots, h_{P_\lambda}(F_q)\}.$$

Using query signature $Sig(F_q)$, the λ buckets corresponding to the query are determined. The bucket identifiers are then hidden with the HMAC function used in the index generation phase as

$$\pi_i = HMAC_{K_{id}}(B_i), \tag{8}$$

using the same key K_{id} of Equation 6.

The list of λ bucket identifiers is the secure query, Q. Depending on the query type (i.e., k-NN search), a relevancy score may accompany the list of bucket identifiers. Note that, the same set of λ MinHash functions are used both in index and query generations. Therefore, authenticated users need to know the permutations generated in index generation for MinHash functions. The set of permutations, which is referred as the *trapdoor*, is shared with all authenticated users, but hidden from the both non-colluding cloud servers.

In k-NN search, the tf-idf values of the terms in the query document are also used in the query generation phase. Recall that, output of a MinHash function, which determines the corresponding bucket identifier, is one of the terms in the query set. Therefore, tf-idf value of the output term is used in the query together with the bucket identifier. However, tf-idf values are also sensitive information that may reveal the corresponsing term. In order to hide the tf-idf values in secure queries, we apply an order preserving hashing (h_{op}) to each tf-idf value such that if $x > y$ then $h_{op}(x) > h_{op}(y)$. The order preserving function used in the protocol is defined as,

$$h_{op}(x) = MSB_\zeta(rx + r_2), \tag{9}$$

where r and r_2 are two random numbers ($r_2 < r$) and MSB_ζ is a function that returns the most significant ζ bits of the given input. While the same random r is used for all λ calls of h_{op} for a given query, r_2 is randomly chosen in each call of the function. If a deterministic function were used for hiding the tf-idf scores, for different MinHash functions that produce the same output, the corresponding scores would also be the same. This would eventually reveal bucket identifiers that correspond to the same terms.In order to avoid such leakage, we use a randomized order preserving function that outputs different, but close values for the same input with high probability.

The extracted ζ bit values are combined with HMACed bucket identifiers, resulting a vector of λ pairs as

$$V_q = \{(\pi_1, h_{op}(rs_{\pi_1}), \ldots, (\pi_\lambda, h_{op}(rs_{\pi_\lambda}))\}.$$

Note that, scoring is only used for k-NN queries and for keyword and range queries only the bucket identifiers protected by HMAC functions are used in the query.

Algorithm 2 describes the process of query generation.

Algorithm 2 Query Generation

Require: F: feature set of keywords to be queried,
h: λ MinHash functions, K_{id}: HMAC key
Ensure: V_q: Secure query

```
Find Sig(F) as in Algorithm 1
Generate an odd random r
for i = 1 to λ do
  π_i ← HMAC_{K_id}(Sig(F)[i])
  if k-NN search then
    Generate random r_2 < r
    rs_{π_i} ← score of Sig(F)[i]
    h_op(rs_{π_i}) ← MSB_ζ(r(rs_{π_i}) + r_2)
    set V_q[i] ← (π_i, h_op(rs_{π_i}))
  else
    set V_q[i] ← π_i
  end if
end for
return V_q
```

6.3 Secure Search

Search over the encrypted data is performed over the secure index stored on the search server. As briefly mentioned previously, two non-colluding servers are employed in the search process, namely search server and file server. While the search server stores the secure index generated by the data owner as explained in Section 6.1, the file server stores the actual encrypted data elements and also knows the private key K_{priv} for the homomorphic encryption (i.e., Paillier cryptosystem) that is used in the encryption of the relevancy scores.

A user generates a secure query as explained in Section 6.2 and submits it to the search server. The search server works homomorphically over the secure index and calculates encrypted, unsorted relevancy scores for the list of the matching results.

Given the secure query V_q, the matching λ vectors of $\mathcal{V}_{B_k^i}$ (Equation 7) are retrieved by the search server. Note that, each vector $\mathcal{V}_{B_k^i}$ contains pairs of document id and corresponding encrypted score that match with that bucket. Then, for each document id in those vectors the accumulated encrypted relevancy score is calculated by summing the encrypted scores of the same document from different vectors. Due to the additive homomorphic property of the encryption method used in the encryption of the relevancy scores, accumulated encrypted score can be calculated without decrypting any of the individual scores. A data structure like hash table can be utilized to increase the performance of this operation.

In the case of k-NN search, the query also contains relevancy scores which are stored using an order preserving hash function (Equation 9). Utilizing the scalar multiplicative homomorphic property of the encryption method, the scores in the secure k-NN query are first multiplied with the scores of data items in the matching vectors of $\mathcal{V}_{B_k^i} \in \mathcal{I}$ and then the accumulated encrypted results are calculated. Note that, the relevancy scores in the index $\mathcal{I}$ are stored encrypted at all times and all the calculations are done over the encrypted values.

The accumulated encrypted results are then sent to the file server that possesses the private key of the homomorphic encryption. It decrypts and sorts the relevancy score list of the matching data items and sends those encrypted, top matching data items to the query owner.

7 Security Analysis

In the proposed method, all the actual data items that are stored in the file server are encrypted with a semantically secure symmetric encryption scheme (e.g., AES with CBC and CTR modes) prior to outsourcing. Therefore, an adversary can learn no information using the encrypted data elements. However, meta-data called the secure index, that is used for applying secure search operation, is also outsourced together with the encrypted data. In order to show the security of the search method, we need to formalize which information can and cannot be learned from the method. A common way to show the security of a search method is to formalize the leakage, and prove that the adversary learns nothing more than this leakage, which is introduced by Curtmola et al. [9].

Definition 5. ***History*** *(H_n) Let $\mathcal{D}$ be the collection of documents in the data-set and $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$ be a collection of n search queries. The n-query history is defined as $H_n(\mathcal{D}, \mathcal{Q})$.*

Definition 6. ***Search Pattern*** *(S_p) is the frequency of the queries searched, which is found by checking the similarity between two queries. In our case, a query is a collection of λ bucket identifiers. Hence, search pattern is defined as the frequency of the queried bucket identifiers (π_i).*

Definition 7. ***Access Patten*** *(A_p) is the collection of data identifiers that match with a user query (i.e., that are returned to the user). Let F_i be the feature set of Q_i and*

$$R(F_i) = \{H(id(D_i)) \mid F_i \in D_i\}$$

be the collection of hashed identifiers of data elements that match with feature set F_i, then $A_p(Q_i) = R(F_i)$.

Definition 8. ***Trace*** *($\gamma(H_n)$) Let $C = \{C_1, \ldots, C_l\}$ be the set of encrypted documents, $id(C_i)$ be the identifier of C_i, $|C_i|$ be the size of C_i and $|\mathcal{I}|$ be the number of buckets in the secure index $\mathcal{I}$. The trace of H_n is defined as:*

$$\gamma(H_n) = \{\forall_{C_i \in C}(id(C_i), |C_i|), S_p(H_n), A_p(H_n), |\mathcal{I}|\}.$$

We allow leaking the trace to an adversary.

Definition 9. ***View*** *($v(H_n)$) is the information that is accessible by an adversary. Let $\mathcal{I}$ be the searchable secure index and, $id(C_i)$ and $\mathcal{Q}$ be as defined above. The view of H_n is defined as:*

$$v(H_n) = \{(id(C_1), \ldots, id(C_l)), C, \mathcal{I}, \mathcal{Q}\}.$$

Definition 10. ***Semantic Security*** *[9] A cryptosystem is semantically secure, if for all probabilistic polynomial time algorithms (PPTA), there exists a simulator $\mathcal{S}$ such that, given the trace of a history H_n, $\mathcal{S}$ can simulate the view of H_n with probability $1-\epsilon$, where ϵ is a negligible probability.*

Theorem 1. *The proposed method satisfies semantic security in accordance with Definition 10.*

Proof. Let the real view $v(H_n)$ and trace $\gamma(H_n)$ be

$$\begin{aligned} v(H_n) &= \{(id(C_1), ..., id(C_l)), C, \mathcal{I}, \mathcal{Q}\} \\ \gamma(H_n) &= \{\forall_{C_i \in C}(id(C_i), |C_i|), S_p(H_n), A_p(H_n), |\mathcal{I}|\}. \end{aligned}$$

Further let the view simulated by a simulator S be $v^*(H_n) = \{(id^*(C_1), \ldots, id^*(C_l)), C^*, \mathcal{I}^*, \mathcal{Q}^*\}$. The proposed method is adaptive semantically secure if $v(H_n)$ is indistinguishable from $v^*(H_n)$.

- The first component of the view $v(H_n)$ is the pseudo identifiers of the documents, $id(C_i)$, which are also available in the trace. Hence, S can trivially simulate document identifiers as $id^*(C_i) = id(C_i)$. Since for all possible values of i, $id^*(C_i) = id(C_i)$, they are indistinguishable.

- Each document is encrypted using a PCPA-secure encryption method. Note that, the output of a PCPA-secure encryption method [9] is by definition indistinguishable from a random number that has the same size as the ciphertext. To simulate ciphertexts C, S assigns l random numbers to C^* such that $C^* = \{C_1^*, \ldots, C_l^*\}$, where $\forall i, |C_i^*| = |C_i|$ using the size information of each ciphertext, which is available in the trace. Considering that for all i, C_i and C_i^* are indistinguishable, C and C^* are also indistinguishable.

- Note that $\mathcal{I}$ is a collection of bucket identifier (π_i) and encrypted vector pairs where each vector is a list of pairs as

  $$\mathcal{V}_i = H(id(D_i)), E_{K_{pub}}(rs_i).$$

  Simulator S generates $|\mathcal{I}|$ index elements, $\mathcal{I}^*[i] = (\pi_i^*, \mathcal{V}_i^*)$ such that π_i^* is a randomly chosen element from the search pattern S_p using the given distribution. Note that, we allowed to leak S_p which shows the occurrence frequency of each bucket identifier π_i. Since π_i^* is actually chosen from possible bucket identifiers using the same frequency distribution, it has the same occurrence probability with π_i and hence, π_i^*

and π_i are indistinguishable. Similarly, the first part of $\mathcal{V}_i$ is a hashed document identifier which exist in the access pattern. Simulator S generates $H(id(D_i))^*$ as a randomly chosen element from the access pattern $A_p \in \gamma(H_n)$ using the given distribution of hashed document identifiers in A_p. Hence, $H(id(D_i))$ and $H(id(D_i))^*$ are indistinguishable. The second part of $\mathcal{V}_i$ is a ciphertext of a Paillier encryption method $(E_{K_{pub}}(rs_i))$, which provides semantic security and therefore can be simulated by a random number. Hence, $\mathcal{I}$ is indistinguishable from $\mathcal{I}^*$.

- $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$ is a set of n queries, where each query Q_i is composed of λ encrypted bucket identifiers. Hashed bucket identifiers (π_i^*) can be simulated by the S as shown in the simulation of $\mathcal{I}$. S can simulate the queries from the search pattern S_p using the given distribution. For each Q_i^*, λ random simulated bucket identifiers are chosen from S_p. Note that real and simulated bucket identifiers are indistinguishable from each other as they have exactly the same frequency distribution. Hence, for all i, Q_i is indistinguishable from Q_i^* and following from this, $\mathcal{Q}$ is indistinguishable from $\mathcal{Q}^*$.

The simulated view v^* is indistinguishable from the genuine view v since the components of v and v^* are indistinguishable. Henceforth, the proposed method satisfies adaptive semantic security. □

The adversary can learn the hashed document identifiers that match with a query (i.e., access pattern) and the hashed bucket identifiers (i.e., search pattern), but the corresponding features of the documents and the queries are protected. Hence, the method satisfies both data confidentiality and query confidentiality in accordance with definitions 3 and 4.

8 Experiments

In order to assess the performance of the proposed scheme, we build a small cluster of 3 nodes with Cloudera CDH4 for cluster management. The cluster has two nodes with Xeon processor E5-1650 3.5 GHz with 12 cores and one node with Core i7 3.07 GHz with 8 cores. All three nodes have 16 GB of RAM and on each node, Ubuntu 12.04 LTS, 64 -bit operating system is installed and Java 1.6 JVM is used.

In our experiments, we used the Enron data set [25] for evaluating the text based operations (i.e., multi-keyword and k-NN search) of the proposed

method. This data set contains a collection of 517,000 real e-mail files. The size of each file changes from 4 KB to 2 MB.

In the case of range queries, a numerical data is required instead of text data, hence we conducted experiments with a different data set. The TPC-H data set is utilized, which is a decision support benchmark widely used in the database community [26]. TPC-H data set contains 131,000 data elements with 4 numeric attributes (or features in the adopted terminology).

Initially, the entire data set is processed to determine the features of each data item in what is known as the feature extraction phase. First, the data set is cleaned from the mail headers and stop words and the terms are stemmed to their roots using a stemmer. The *tf-idf* values of each term in each file are then calculated and stored. After the feature extraction phase, secure index is generated as explained in Section 6.

For the accuracy experiments we consider two metrics namely *precision* and *recall*. Intuitively, precision is the ratio of correctly found matches over the total number of returned matches and recall is the ratio of correctly found matches over the total number of expected results (i.e., the ground truth). Both precision and recall are real numbers between 0 and 1, where a higher value means better accuracy.

We assume that the server receives several search requests at any time which is in line with the big data context. Therefore instead of processing each query individually, we group several individual queries and process a bulk search for all. Bulk search operations have a great positive effect on the throughput of the system as the distributed file systems and parallel programing models do not benefit relatively small number of such operations.

8.1 Experiments with Multi-keyword Search

The classical definitions of precision and recall metrics do not consider the importance of features for a data item, but only their mere presence. For example, if a query contains t terms and a data item contains $t-1$ of those terms with high tf-idf values, but fails to contain one term, then it is considered as a false match for the given query so far as precision and recall metrics are concerned. Although those metrics can be advantageous in several areas, we claim that they may not be highly suitable for search over unstructured data, where some features can be significantly more important than others. Therefore, we measure the accuracy of the method by calculating precision and recall in comparison to what we refer as *the ground truth*.

In the ground truth, for a given query, we consider the data items with top 50 scores in the data set as the actual matching results, instead of those

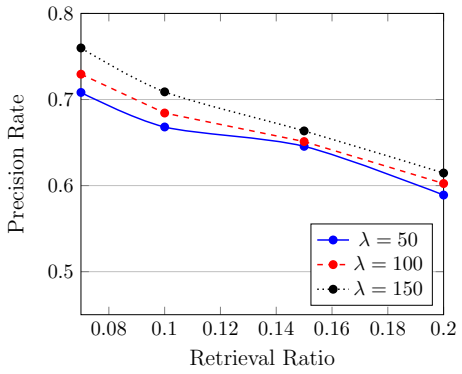

Figure 4: Average precision rates for multi-keyword search

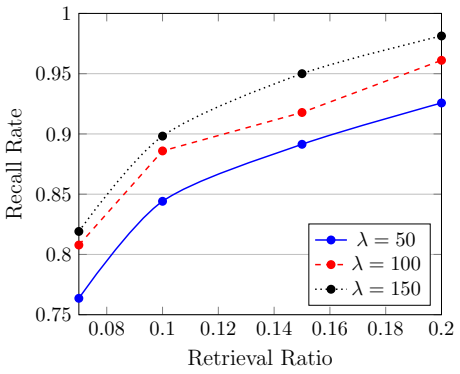

Figure 5: Average recall rates for multi-keyword search

that contain all the terms in a query. The top 50 matching results are found by evaluating the search over the plaintext data (i.e., without using the secure index) utilizing the full tf-idf scores. These actual results are then compared with the output of the proposed scheme to measure the accuracy of the method. The average precision and recall rates are calculated by taking the average of 50 queries, randomly generated with 2, 3 and 4 terms. Note that, in the proposed method all the data items that have non-zero scores (i.e., match candidates) contain a nonempty subset of the queried terms. Therefore, for single term queries, both precision and recall rates are 1.0 (i.e., without any false accept or false reject).

The average precision and recall rates are presented in Figures 4 and 5, respectively. The retrieval ratio in the figures represents the ratio of the data elements with non-zero scores that are considered as a match with the given query. As retrieval ratio increases, more data items are returned as a match and hence, it has a positive effect on recall, but a negative effect on precision. Recall that, as MinHash functions with different permutations provide independent experiments for approximating the Jaccard similarity, using longer signatures (i.e., larger λ) provides more accurate results. The results given in Figures 4 and 5 also support this fact as accuracy increases with λ.

We test the performance of the multi-keyword search with $\lambda = 150$ as it is an optimum value that provides recall rates very close to 1.0. The search operation has two major phases. In the first phase, the search server receives a query and homomorphically finds the encryption of an accumulated score for each data element that matches (at least partially) with the given

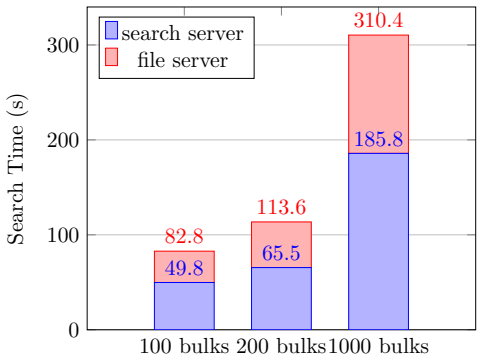

Figure 6: Average multi-keyword search time for bulks of 100, 200 and 1000 queries

query. The encrypted scores are then sent to the file server which applies the decryption operation. The file server then sorts the data items with respect to their relevancy scores and returns the top matches based on a pre-determined retrieval ratio.

In order to show the effect of bulk search over the throughput of the system we tested the scheme for three sets of 100, 200 and 1000 queries, respectively. As Figure 6 presents, 100 queries can be processed in about 83 seconds, whereas 200 queries can be processed in less than 114 seconds and 1000 queries can be processed in 310 seconds. Note that a 10 times increase in the number of queries only increases the process time less than 4 times. These results demonstrate that performing search operation with larger sets of queries has a significantly positive effect on throughput, but it may also increase latency of the queries; hence bulk size should not be too large. Therefore, we use bulks with 200 queries in the rest of the experiments. We also observe from Figure 6 that on average, about 60% and 40% of the total search time are spent in the search server and the file server phases, respectively.

8.2 Experiments with k-NN Search

Similar to multi-keyword, k-NN search operation is also tested using the Enron data set [25]. There are two main differences of a k-NN search from a multi-keyword search. The first difference is that, the number of queried terms is much larger in k-NN search since all the important terms of a data item is considered as a term in the query. In the case of multi-keyword search, we assume the number of queried terms will be small (e.g., less than

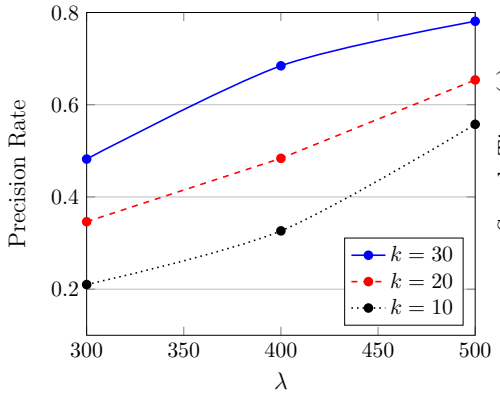

Figure 7: Average precision rates for k-NN search with different λ and k

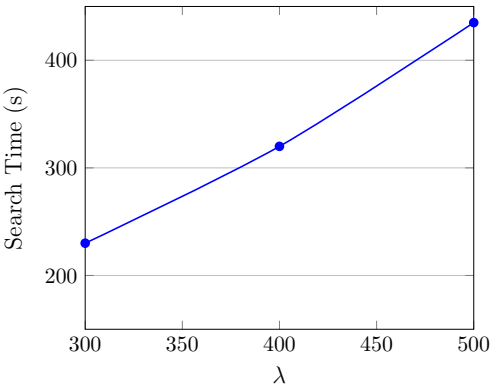

Figure 8: Average search time for k-NN search with bulks of 200 queries and different λ

5). The second difference is that, in k-NN search the queried terms have also different importance levels (e.g., tf-idf values) for the data item. Recall that in multi-keyword search, the terms in a query have equal importance.

As the number of terms in queries is significantly larger than the ones in multi-keyword search, a larger value of λ needs to be used in order to satisfy an equivalent level of accuracy. Furthermore, in the case of k-NN search, both precision and recall metrics are equivalent as the number of returned matches and the number of actual matches are both equal to k. We therefore used a single metric, namely precision, to evaluate the accuracy of the k-NN search method. The accuracy of k-NN search in our experiments is illustrated in Figure 7 for different values of k and λ.

Figure 7 demonstrates that increase in λ has a positive effect on accuracy as expected due to the properties of locality sensitive hash functions. Similarly, increase in k has also a positive effect on the accuracy. The proposed method can find the most similar data items (i.e., nearest neighbors) to the queried item, but possibly in a different order with respect to the actual results. Therefore, for small values of k, some of the actual most similar matches may not fall in the top k slots and leads to a negative effect on the accuracy. However, for relatively larger values of k, most of the top matching data items are usually accommodated in the top k slot and thus leads to high accuracy rates.

The k-NN search operation is the most expensive among the three operations studied in this work (see Figure 8). This has two main reasons. First of all, as the number of the terms in a query is significantly large, the size

of the signatures used is almost three times larger than the ones used for the other two types of search operation. Secondly, different from the multi-keyword search, the features of the queries also have scores in k-NN search; hence an additional homomorphic multiplication is applied as explained in Section 6.3. Similar to the experiments with the multi-keyword search, we empirically evaluate the computation costs of search operation using bulk queries of 200, as shown in Figure 8.

8.3 Experiments with Range Search

Range search operation works with attributes that take on numerical values. Therefore, we used the **LineItem** table of TPC-H data set [26], instead of the Enron data set, which consists of text data. In the experiments we used four different features for range queries.

A range query can be thought of a predicate-based query, where the range boundaries for each attribute are tested. For instance, a query for determining the staff of a company whose salary is greater than \$2500 a month and whose age is between 20 and 30 can be found by testing whether the predicate $\{(Q_s > 2500), (Q_a > 20), (Q_a < 30)\}$ is satisfied, where Q_s and Q_a stand for salary and age features, respectively. From a broader perspective, a range query can be formulated very similar to a multi-keyword search, in such way that every predicate testing in the former is equivalent to a term in the latter. In fact, we use the word *term* to refer a predicate testing such as $(Q_a < 30)$ in this work. Consequently, the secure index for range queries can be constructed in an identical manner. The only difference between a range search and a multi-keyword search is that a basic scoring based on only the number of common buckets, is used, as tf-idf scoring is inapplicable in numerical data sets.

We evaluate the accuracy of the range search operation using queries of 2 and 4 features with both minimum and maximum boundaries (i.e., queries with 4 and 8 terms, respectively). A workload of 100 queries (50 for each case) is generated, where the range of each query was randomly selected from a uniform distribution which has the same range as the data sets. The results are presented in Table 1. In order to make the results comparable with the multi-keyword search, the value of λ is set as 150 and the accuracy results are calculated with respect to the ground truth as explained in Section 8.1.

The accuracy experiments demonstrate that both precision and recall results are significantly better than those of multi-keyword and k-NN search operations. This phenomenon is due to the fact that a structured data is used in range search, as opposed to the unstructured data set used for

the other two query types. Indeed, the data set for range search can be represented by only a few hundreds terms, whereas a text based data set such as Enron necessitates tens of thousands or even more terms. The results also show that increase in the number of queried terms has a positive effect on recall, but a negative effect on precision. As the number of queried terms increases, the number of elements in the data set that contain all those terms significantly decreases, which increases the recall values. However, the increase in the number of queried terms also increases the data elements with nonzero scores (i.e., that contain at least one of the queried terms) which causes a decrease in precision values.

Table 1: Accuracy of Range Search

	2 attributes	**4 attributes**
precision	0.88	0.81
recall	0.96	0.98

As the structure of the range search is identical to the multi-keyword search and they both utilize the same signature length (i.e., $\lambda = 150$), their timing results are also equivalent. The number of terms in a query has no effect in the cost of a search operation since, independent from the number of terms in a query, it has a constant length, which is λ.

9 Conclusion

In this work, we propose a general privacy preserving search scheme over encrypted cloud data. The proposed scheme provides privacy preserving search operations in large data sets with high performance by leveraging the MapReduce paradigm. The scheme is efficiently implemented for all three types of search operations on an experimental cloud computing environment using Hadoop MapReduce. Performance and accuracy of the search operations are evaluated using publicly available real data sets and shown to be efficient and effective. Moreover, it is shown that the scheme is scalable, and hence the performance can trivially be improved by providing larger Hadoop clusters.

Acknowledgment

This work was supported by TUBITAK under Grant Number 113E537.

References

[1] T. Jung, X. Mao, X. Li, S. Tang, W. Gong, and L. Zhang, "Privacy-preserving data aggregation without secure channel: Multivariate polynomial evaluation," in *Proceedings of the IEEE INFOCOM 2013, Turin, Italy, April 14-19, 2013*, pp. 2634–2642, 2013.

[2] Y. Yang, H. Li, W. Liu, H. Yao, and M. Wen, "Secure dynamic searchable symmetric encryption with constant document update cost," in *Global Communications Conference (GLOBECOM), 2014 IEEE*, pp. 775–780, Dec 2014.

[3] P. van Liesdonk, S. Sedghi, J. Doumen, P. Hartel, and W. Jonker, "Computationally efficient searchable symmetric encryption," in *Secure Data Management*, vol. 6358 of *Lecture Notes in Computer Science*, pp. 87–100, Springer Berlin Heidelberg, 2010.

[4] Z. Chen, C. Wu, D. Wang, and S. Li, "Conjunctive keywords searchable encryption with efficient pairing, constant ciphertext and short trapdoor," in *PAISI*, pp. 176–189, 2012.

[5] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rou, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Advances in Cryptology, CRYPTO 2013*, vol. 8042 of *Lecture Notes in Computer Science*, pp. 353–373, Springer Berlin Heidelberg, 2013.

[6] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 222–233, 2014.

[7] M. Strizhov and I. Ray, *Multi-keyword Similarity Search over Encrypted Cloud Data*, pp. 52–65. Springer Berlin Heidelberg, 2014.

[8] C. Orencik, A. Selcuk, E. Savas, and M. Kantarcioglu, "Multi-keyword search over encrypted data with scoring and search pattern obfuscation," *International Journal of Information Security*, vol. 15, no. 3, pp. 251–269, 2016.

[9] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in

Proceedings of the 13th ACM conference on Computer and communications security, pp. 79–88, ACM, 2006.

[10] C. Orencik and E. Savas, "An efficient privacy-preserving multi-keyword search over encrypted cloud data with ranking," *Distributed and Parallel Databases*, vol. 32, no. 1, pp. 119–160, 2014.

[11] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pp. 139–152, 2009.

[12] F. Li and X. Xiao, "Secure nearest neighbor revisited," *2014 IEEE 30th International Conference on Data Engineering*, vol. 0, pp. 733–744, 2013.

[13] Y. Elmehdwi, B. K. Samanthula, and W. Jiang, "Secure k-nearest neighbor query over encrypted data in outsourced environments," *CoRR*, vol. abs/1307.4824, 2013.

[14] E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, (Washington, DC, USA), pp. 350–364, IEEE Computer Society, 2007.

[15] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu, "Secure multidimensional range queries over outsourced data," *The VLDB JournalThe International Journal on Very Large Data Bases*, vol. 21, no. 3, pp. 333–358, 2012.

[16] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. Garofalakis, "Practical private range search revisited," in *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pp. 185–198, ACM, 2016.

[17] A. Rajaraman and J. D. Ullman, *Mining of massive datasets*. Cambridge: Cambridge University Press, 2012.

[18] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[19] "Apache Hadoop." http://hadoop.apache.org, Jan. 2015.

[20] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012.

[21] B. Pinkas and T. Reinman, "Oblivious ram revisited," in *Advances in Cryptology–CRYPTO 2010*, pp. 502–519, Springer, 2010.

[22] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path oram: An extremely simple oblivious ram protocol," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 299–310, ACM, 2013.

[23] M. Kuzu, M. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pp. 1156–1167, April 2012.

[24] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *ADVANCES IN CRYPTOLOGY - EUROCRYPT 1999*, pp. 223–238, Springer-Verlag, 1999.

[25] "Enron Dataset." http://www.cs.cmu.edu/∼./enron/, Jan. 2015.

[26] "TPC-H, Decision Support Benchmark." http://www.tpc.org/tpch, Aug. 2015.