Received: 09 June 2021 | *Revised: 11 July 2021* | *Accepted: 11 July 2021*
DOI: 10.37917/ijeee.17.2.1 *Vol. 17| Issue 2| December 2021*

Open Access

Iraqi Journal for Electrical and Electronic Engineering
Original Article

Secure Multi-keyword Similarity Search Over Encrypted Data With Security Improvement

Hussein M. Mohammed, Ayad I. Abdulsada*
Department of Computer science, Education College for Pure Sciences,
University of Basrah, Basrah, 61004, Iraq

Correspondence
* Ayad I. Abdulsada
Department of Computer science
Education College for Pure Sciences,
University of Basrah, Basrah, Iraq
Email: ayad.abdulsada@uobasrah.edu.iq

Abstract
Searchable encryption (SE) is an interesting tool that enables clients to outsource their encrypted data into external cloud servers with unlimited storage and computing power and gives them the ability to search their data without decryption. The current solutions of SE support single-keyword search making them impractical in real-world scenarios. In this paper, we design and implement a multi-keyword similarity search scheme over encrypted data by using locality-sensitive hashing functions and Bloom filter. The proposed scheme can recover common spelling mistakes and enjoys enhanced security properties such as hiding the access and search patterns but with costly latency. To support similarity search, we utilize an efficient bi-gram-based method for keyword transformation. Such a method improves the search results accuracy. Our scheme employs two non-colluding servers to break the correlation between search queries and search results. Experiments using real-world data illustrate that our scheme is practically efficient, secure, and retains high accuracy.

KEYWORDS: Cloud computing, privacy preserving, searchable encryption, multi-keyword search, similarity search.

I. INTRODUCTION

Due to the appealing services provided by cloud computing, clients and even business are encouraged to upload their private data (health records, e-mails, private accounts, bank, and financial accounts) into the cloud servers for lowering the storage and computing costs at local devices. However, cloud servers, are usually assumed as untrusted entities. So, private data should be encrypted before being uploaded to preserve data confidentiality. However, this treatment disables the majority of data management services such as searching and indexing. To solve this problem, researchers have developed *searchable encryption* (SE) schemes that enable clients to search their encrypted data at the server-side without the need for decryption. Search operation should protect the privacy of data and even the search queries as they may contain private keywords.

SE schemes work by generating a searchable index *I* from the entire file collection. Such index could be either forward index, which builds a separate index for each file a reverse index that maps each keyword *w* within file collection with the set of file identifiers (IDs) whose corresponding files include *w*. Data owner uploads the encrypted file collection along with the encrypted index into the remote server. During the search, a client provides a keyword (or even multi-keywords) and generates a search token by using a secret key. Such a token is used by the server to search the encrypted index to find the matched identifiers without revealing the keywords of neither the file collection nor the query. For efficiency purposes, the majority of SE schemes leak the *access pattern* (the search results) and the *search pattern* (whether or not some keyword is searched before) to the cloud server. However, some studies have illustrated that such leakage may leak the client queries when combined with some background knowledge [1].

The earlier SE schemes support only a single keyword search (e.g., [2-5]), which returns too many items with only a few relevant items. The first SE scheme was proposed by Song et al. [2], which used two layers to encrypt each keyword of a textual file. During the search, the encrypted file is scanned sequentially to find matching. Goh [3] has built a forward index (as a Bloom filter) for each file. This means that search time is linear to the number of files. Curtmola et al.'s scheme [4] is considered one of the most efficient SE schemes, where the reverse index is used to improve search complexity into sublinear search (proportional to the number of files that

share a keyword). Chang et al.'s scheme [5] has the same search complexity of [3] but with a better security definition. However, these schemes do not support the ranked search. The schemes of [17, 18] have addressed the ranked search problem, where the frequencies of keywords are considered during the search. Unfortunately, these two schemes still support only a single keyword search. Later, many SE schemes have been proposed to support multi-keyword search (e.g., [6-8]), where queries incorporate several keywords and results are refined and returned according to their relevance scores. However, all of the above-mentioned schemes support only exact search, where no files are returned when incorrect (misspelled) keywords are provided. Thus, several schemes have been proposed to support fuzzy search (e.g., [9-12]), that handle the spelling errors in searched keywords. For instance, the solution of [9, 10] builds a keyword dictionary for each indexing keyword. For example, the dictionary of keyword "car" within edit distance 1 is {car, *car, c*a, ca*}. However, this method supports only a single keyword search at the expense of more storage costs.

Wang et al. [13] proposed the first scheme (we called it MKFS) to solve the problem of multi-keyword similarity search over encrypted data without the need for a predefined dictionary. In MKFS, each file (and query) is indexed as a Bloom filter (binary vector). Such index is constructed by extracting the keyword set of that file and hashing those keywords several times to determine the bit locations that will be set to one. Keywords are transformed into a bi-gram vector to be hashed using LSH functions. Under such a setting the similarity between two files will be captured by applying the inner product for their corresponding Bloom filters. Since LSH functions are used, the misspelled keywords are mapped into the same locations of the correct keyword with a high chance. The most important property of MKFS was ignoring the need for a predefined dictionary. However, this scheme suffers from many problems. From a security perspective, it leaks the access and the search patterns, which lead to disaster consequences on the data security [1, 14, 15]. In addition, this scheme loss the ability to match the keywords of the same roots such as "compute" and "computing". Finally, MKFS does not consider the weight of keywords during ranking.

In this work, we propose a multi-keyword similarity search over encrypted data that recover the problems of MKFS. Our proposed scheme uses the same index of MKFS but it enjoys a better security level, where it hides both search and access patterns along with protecting file confidentiality and query privacy. To do so, we encrypt each file vector (and query vector) by using the homomorphic encryption method. Furthermore, our two-server solution hides the search pattern leakage which is common in the literature. In this work, we do not consider the security of the actual files as the can be protected using any secure cryptographic method like AES. Our contributions can be summarized as follows:

1) A secure scheme is proposed and implemented that supports multi-keyword similarity search.
2) Each keyword is converted to its root before being indexed. This contributes to reduce the number of distinct keywords in the file and thus reducing the cost of building the indexes. Furthermore, stemmed keywords have higher chance to be matched.
3) Returned results are ranked using the term frequency (TF) of each keyword. Thus, relevant files to the provided query will have large scores.
4) The proposed scheme ensures high-security guarantees by exploiting Paillier encryption [16] to encrypt file and query vectors.
5) The accuracy and efficiency of our proposed scheme were investigated by experimental results on real data set.

The remaining sections for this paper are: Section II reviews the most relevant works. Section III describes problem formulation. Section IV presents the theoretical background for techniques used within the paper. Section V describes our proposed scheme, discussion. Section VI describes security analysis. Section VII reports performance comparison and experimental results. Section VIII draws the paper conclusion.

II. Related Works

Related works on SSE schemes can be divided into the following topics.

A. Multi-keyword search

The single keyword search has limited usability. Cao et al. [19] proposed the first scheme that supports multi-keyword ranking search. They used a vector space model to describe files as numerical vectors and adopt the "coordinate matching" similarity measure to evaluate the relevance between files and query. The relevance scores are captured by calculating the inner product between files and query vectors. The search results are ranked by these relevance scores. The main drawback of Cao et al.'s scheme is its efficiency, where files are described as very long vectors. Sun et al. [20] improve the work of [19], where a tree data structure is used to store the searchable index. They used term-frequency, vector space model, and cosine similarity for search ranking. Li et al. [21] exploit the access time of each keyword to enhance the ranking method. However, these aforementioned schemes only support multi-keyword ranking search for exact keywords. In contrast, our scheme can handle misspelled keywords.

B. Fuzzy keyword ranked search

The previously mentioned schemes support exact keyword search (i.e., results are returned just when the same keywords are provided), and hence such schemes fail if typos keywords are searched for. To solve this problem, li et al. [9] considered the first scheme that supports fuzzy keyword search. Particularly, they used a wildcard technique to obtain the keyword fuzzy sets, which list all the possible forms of

the given keyword. This scheme takes a lot of storage. li et al. [10] solve this drawback by using the gram method to obtain smaller keyword fuzzy sets. Wang et al. [11] also used wildcard with edit distance to support fuzzy search but using the tree data structure. Kuzu et al. [12] used the minhash technique to construct an efficient reverse index. These schemes only support a single keyword ranking search. Later, the Bloom filter is used in [13, 22] to improve the search and storage efficiency. Furthermore, such schemes utilized LSH to insert the fuzzy set of each keyword in the Bloom filter. Wang et al. [23] use a score table to support fuzzy ranking search. Fu et al. [24] enhance the accuracy of Wang et al.'s scheme [13]. In their work, keywords are transformed by using uni-gram vectors. Semantic search was introduced by Moataz et al. [25] and Fu et al. [26], where keywords are extended by their semantic forms.

III. Problem Formulation

In this section, we formalize the problem of multi-keyword similarity ranked search over encrypted data. Table 1 collects the notations that are used within this paper.

A. System model

Our proposed scheme (which is illustrated in Figure 1) consists of four different entities: data owner, data user, file server, and search server. The two servers are assumed to be non-colluding servers and both servers constitute the cloud server. The cloud server has huge storage service and unlimited computational power. The reason behind using two servers is to cut the link between queries and the matching file identifiers, which hides the search pattern.

1) *Data owner* (DO): a party that has n textual files $F = (f_1, f_2, \dots, f_n)$. DO is assumed to not has sufficient storage so DO outsources his data to the cloud server. DO encrypts his data to get the encrypted collection $C = (c_1, c_2, \dots, c_n)$ and outsource C to the cloud server. Do also constructs from F a searchable index I that is outsourced also to the cloud server to enable an efficient search. In this work, we consider the forward index setting, where each file has its own index. Thus, I is the union of the file indexes.
2) *Data Users* (DU): An entity (authorized entity) that has query q of multi-keywords, DU generates the search token T from q using a secret key, then sends T to the search server SS. The latter measures the similarity scores with the help of file server FS and return the ranked scores to FS, who forwards their corresponding files to DU.
3) *Search server* (SS): This server stores index I. When it receives the search token T from DU, it computes the encrypted similarity scores between T and all files and sends the encrypted scores to the file server FS.
4) *File server* (FS). This server stores the encrypted collection C and helps SS to compute the encrypted similarity scores. FS decrypts the scores and sends them back to DO, who correlate each score with its proper file identifier and selects only the best k matching identifiers. DO asks FS to download the actual files in another round.

TABLE 1
COMMON NOTATIONS

Symbols	Descriptions
W	The keyword set, $W = \{w_1, \dots, w_m\}$
m	The number of keywords
F	The file collection, $F = \{f_1, \dots, f_n\}$
n	The number of files in F
C	The encrypted file collection, $C = \{c_1, \dots, c_n\}$
I	Searchable index
pk_p	Public key for encryption
sk_p	Private key for decryption
q_i	i[th] query in the query set.
T	The search trapdoor of keywords
k	Number of returned files
$BF(f)$	The Bloom filter of file f
$[\![BF(f)]\!]$	The encrypted Bloom filter
γ	Bloom filter size
l	Number of hash functions
PPT	Probabilistic polynomial time

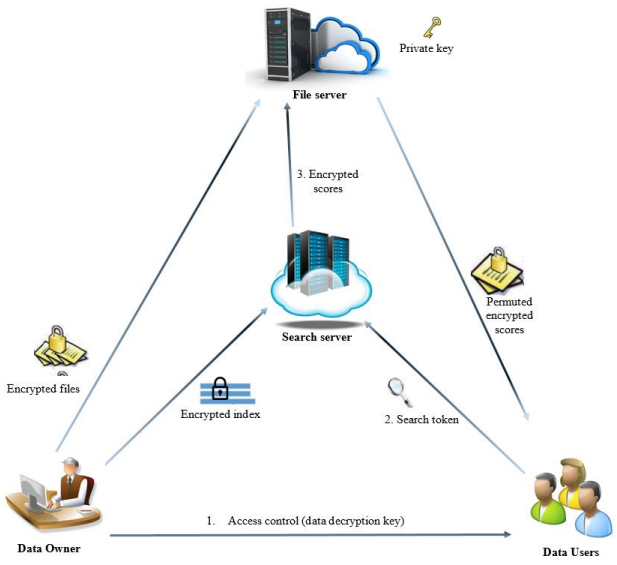

Fig. 1: The system model

B. Security Model

In this paper, we follows the same model of [24, 27, 28], which propose that both DO and DU are trusted parties. However, SS and FS are assumed to be "honest-but-curious" servers as in [3, 9, 23]. This means that they honestly perform search and ranking operations but try to obtain some additional sensitive information through the received messages. Therefore, the file collection, index, and queries should be encrypted before outsourcing to servers.

Additionally, we consider that the two involved servers there are non-collusion, as adopted by most previous solutions [28-30].

C. Design Goals

Our scheme is designed to achieve the following goals:

1) *Multi-Keyword similarity search:* The primary goal of our scheme is to search with multi-keyword misspelled queries. For instance, "netward securaty" should return a file that includes "network security".
2) *No predefined dictionary*: Our scheme does not require any keyword dictionary in contrast to many previous works [4, 9, 20].
3) *Support ranking results*: Our scheme should return the top k-relevant files.
4) *Result efficiency and accuracy*: Our scheme should obtain the results of a similarity search as efficiently and accurately as possible.
5) *Security requirements*: our design prevents the adversary servers from getting useful information about the files, indexes, and search queries. The security requirements are defined as follows:
   A. *Index confidentiality and query privacy*: The index representation of files and queries should protect the underlying keywords from the cloud server.
   B. *Search pattern privacy (unlinkability):* when two search queries q_1, q_2 are issued by users then the cloud server should not deduce that such queries include common keywords.
   C. *Access pattern privacy*: when search results (file identifiers) are known only to the owner of the query, we say that the searchable scheme supports access pattern privacy. This means that the cloud server, should not learn the search results.

IV. Background

This section describes briefly the techniques employed in this work.

1) *Keyword stripping*: stripping [31, 32] is a technique that aims to reduce the variant forms of a given keyword into a fixed form. A stemming technique for English, for the keywords "searches", "searched", and "searching" would be "search". Information retrieval systems adopt widely this technique to enhance search efficiency.
2) *Bloom Filter (BF)*: Bloom filter is an efficient data structure used for storing elements and checking their membership. It is used to represent a set of elements as γ-bit vector of zero elements. To represent a set $S = \{s_1, \dots, s_g\}$ in BF, we use l independent hash functions from $H = \{h_i \mid h_i : S \to [1, \gamma], 1 \le i \le l\}$, and BF inserts an element $s \in S$ by setting the l positions h_i of the vector BF to 1. To check if an element s' is in S, we check all the bits at positions $\{h_i(s'), 1 \le i \le l\}$. The s' is present in S if all bits at these positions are 1. However, in some probability, there is a false positive, in this case s' seems to be in S but indeed it is not. False positive occurs since each position may be set to 1 by some other elements. However, the rate of false positive of γ -bit BF is nearly $(l - e^{-\frac{lg}{\gamma}})^l$. The optimal rate of false positive is $1/2^l$, where $l = \frac{\gamma}{g} \cdot ln2$.
3) *Locality-Sensitive Hashing (LSH)*: LSH is a hashing algorithm used within the nearest neighbor search [33, 34]. An LSH function uses a set of hash functions to hash data items s.t close items produces with high probability the same outputs. LSH uses a hash function family H which is (r_1, r_2, p_1, p_2)-sensitive that holds:

$$\text{if } dist(s,t) \le r_1;\ Pr[h(s) = h(t)] \ge p_1 \quad (1)$$
$$\text{if } dist(s,t) \ge r_2;\ Pr[h(s) = h(t)] \le p_2 \quad (2)$$

where $dist(s,t)$ is the distance metric between the two points, r_1, r_2 are two distances defined by users, and p_1, p_2 are two properties s.t $p_1 > p_2$. Particularly, we employed the p-stable LSH function which hash the vector v as follows:

$$h_{a,b}(v) = \left\lfloor \frac{a \cdot v + b}{\omega} \right\rfloor \quad (3)$$

Where a is a numerical vector, $b \in [0, \omega]$ and ω are fixed constants.

4) *Homomorphic Encryption:* We use homomorphic encryption to perform certain computations on ciphertexts. Particularity, we utilized Paillier cryptosystem [16], which provides the following homomorphic properties on field Z_N:

$$E(a + b) = E(a) \cdot E(b) \bmod N^2$$
$$E(ab) = E(a)^b \bmod N^2$$

Paillier encryption is secure against chosen-plaintext attack (CPA) and its security is proved under the assumption of decisional composite residuosity [16]. Paillier cryptosystem uses the public key $pk_p = (N = pq, g)$ for encryption, where $g = N + 1$, and p and q are two primes of equivalent length chosen randomly and independently, and uses the private key $sk_p = (\phi(N), \phi(N)^{-1} \bmod N)$ for decryption, where $\phi(N) = (p-1)(q-1)$, And $\phi(N)^{-1}$ stands for the modular multiplicative inverse. The encryption of the message $x \in Z_N$ is denoted by $[\![x]\!]$, which is computed as $[\![x]\!] = g^x . r^N \bmod N^2$ for some random $r \in Z_N^*$. A ciphertext is decrypted as $x = L([\![x]\!]^{\phi(N)} \bmod N^2) . \phi(N)^{-1} \bmod N$, where $L(u) = \frac{u-1}{N}$.

V. The Proposed Scheme

This section presents our construction for multi-keyword similarity search over encrypted data.

A. Overview

Our scheme builds a searchable index for each file f. This index represents all the keywords of f, and it is represented as a γ-bit Bloom filter (BF). To build the file index, we first extract each distinct keyword w in f and convert it into a single vector. Then we use several functions to map this vector. Hash values determine the positions of BF that will be set to one. However, traditional hash functions are suitable only with exact keywords. Our scheme is designed to capture similar keywords, thus we replace standard hash functions with LSH functions that has the ability to hash similar keywords into the same positions. To enhance ranking functionality, we store the keyword frequency at each computed position. For protecting privacy, we uses Paillier encryption to encrypt the elements of BF before uploading it to the cloud server. Search queries are generated and encrypted in the same way of the document files. During search we measure the secure inner product between two encrypted vectors. To solve this challenging problem, we used a secure protocol with two-not colluding servers. The main steps of this scheme are explained as follows.

B. Basic steps of the proposed scheme

Figure 2 illustrates the basic steps of the proposed scheme. Notice that encryption steps are omitted for clarity purposes.

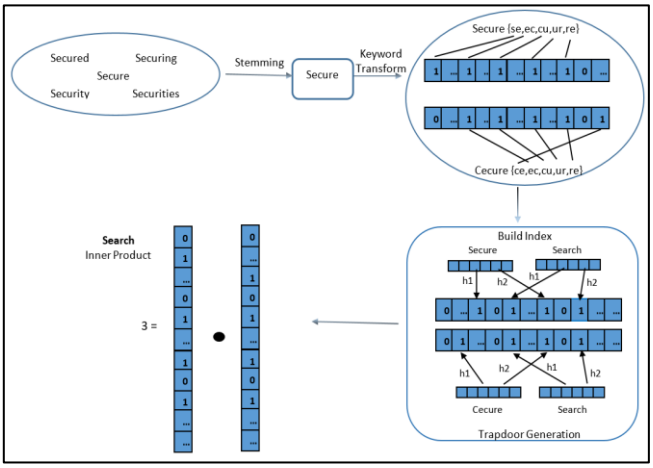

Fig. 2: The main steps of our scheme.

1) *Feature Extraction*: Suppose we have the file , we extract indexing keywords from the file f. Let $\{w_1, \dots, w_z\}$ be the set of distinct keywords in f.
2) *Stemming Algorithm*: We perform Porter algorithm [31] to transform each keyword into its original form.
3) *Keywords transformation:* Keywords are transformed from string type into bigram vector representation in order to be used by the LSH functions. Bigram vector is generated for each keyword as bellow. First, we extract a bigram set from keyword *w*. This set contains all the adjacent two letters existed in the keyword. For example, a keyword "secrecy" has the bigram set {se, ec, cr, re, ec, cy}. Then a bigram set is represented as a 26^2-bit long vector. If the bigram appears in the bigram set of a given keyword, the relevant coordinate in the bigram vector will be set to one. Otherwise, it will be zero. This representation is unaffected by misspelling location. Furthermore, it is also unconcerned about which letter was misspelled. For example, "sacrecy", "swcrecy", and "secresy" will be hashed to a bigram vector that varies by some elements from the main vector. Under this setting, misspelled keywords still be expressed in a vector that is very similar to the right one. This representation is stable, inclusive, and enables the application of LSH functions.
4) *Bloom filter construction*: In this step, the bigram vectors of distinct keywords of f are mapped into a single Bloom filter $BF(f)$ that represents the index of f. Traditional Bloom filters utilize standard hash functions. In this paper, we use l LSH functions instead. LSH functions ensure that that "close" vectors will produce, with high probability, the same output. To construct the $BF(f)$, we first define l LSH functions which use bigram vectors as input and hash them several times. Then we initialize an γ-zero bit $BF(f)$. Each bigram vector is hashed l times, so that the corresponding locations in the $BF(f)$ are set 1. In this way, misspelled keywords are mapped to identical locations in $BF(f)$. Hence, the similarity keyword search can be realized. For example (as illustrated in Fig. 2.), a misspelled keyword "securite" is mapped into the same output of the proper keyword "security" so, during the search process, a match can be realized. Thus, the key to applying similarity search is to use LSH functions to build a Bloom filter index for each file. The index $BF(q)$ of query file q is also constructed in the same way.
5) *File matching*: As shown in Figure 2, a Bloom filter represents files and queries as vectors. Hence, a similarity score between file and query can be achieved by computing a simple inner product between their corresponding vectors. If a file contains the query keywords, then both corresponding vectors will share 1 at the same locations, so a high value will be assigned to the matching score.
6) *Ranking improvement*: To guarantee that the results meet the user's requirements well, term frequency tf (keyword frequency) is used to represent the relevance of files. To do so, we set the locations of Bloom filer, which are determined by hash functions, to the tf of the corresponding keyword. If multiple keywords are mapped to identical locations, we store the average of their term frequencies. Hence, if a particular file includes a large number of keywords from the provided query, then it will receive a great chance to be included in the best matching results. If two files have the same keywords, the file with the highest matching score will appear first in the returned results.
7) *Index protection.* Bloom filter hides the underlying keywords and their frequencies but such filters are constructed in a deterministic procedure. A

deterministic process discloses the search pattern. Specifically, the server can figure out if the same query vector has been queried previously. Moreover, because the file indexes (Bloom filters) are stored at the server in plaintext, the server can reveal the files' identifiers that are similar to the queried file, this is the access pattern. To hide this leakage, we encrypt the Bloom filters before being uploaded to the cloud server. However, because the server should perform a similarity search, the encryption should be homomorphic encryption since it enables performing some arithmetic operation without the need for decryption. We use the Paillier cryptosystem [16] as the instance of homomorphic encryption to protect both the file index and query index. Recall that the similarity between two encrypted vectors is computed using their inner product. Hence, the server should produce the encrypted score by utilizing the inner product among ciphertext vectors. Observe that, the inner product between two numerical vectors is the addition of the sub-multiplications corresponding to the coordinates of the two involved vectors. The secure multiplication of two numbers is denoted by $[\![xy]\!]\ mod\ N^2$. However, the multiplication of two encrypted numbers is not supported by the homomorphic properties of Paillier encryption and hence such an operation requires special treatment. To perform the secure multiplication, we use a second server that has access to the Paillier private key. This second server is assumed not to collude with the first server and hence it does not need to be trusted. The first server utilizes the homomorphic addition property to mask the two multiplication numbers by using random numbers. The masked results are moved to the second server who decrypts the received numbers, multiplies the masked number, and then encrypts the result. The first server homomorphically subtracts the random numbers and obtains the result of encrypted multiplication. Algorithm 1 illustrates how to multiply two encrypted numbers.

Algorithm 1 Secure Multiplication ($[\![x.y]\!]$)

SS:
Input: x, y
Output: x', y'
pick random $n_1, n_2\ mod\ N$
$x' = [\![x]\!] \times [\![n_1]\!] = [\![x + n_1]\!]$
$y' = [\![y]\!] \times [\![n_2]\!] = [\![y + n_2]\!]$
send x', y' to FS
FS:
Input: x', y'
Output: h'
$Dec(x') = x + n_1$
$Dec(y') = y + n_2$
$h = Dec(x') \times Dec(y')\ mod\ N$
$h = xy + yn_1 + xn_2 + n_1 n_2\ mod\ N$
$h' = [\![h]\!]$
send h' to SS
SS:
Input: h'
Output: $[\![x.y]\!]$
$$[\![x.y]\!] = h' \times [\![x]\!]^{N-n_2} \times [\![y]\!]^{N-n_2} \times [\![r_1 r_2]\!]^{N-1}\ mod\ N^2$$

The dot product can be estimated by using a secure multiplication algorithm between the encrypted numbers of Bloom filters and then perform homomorphic additions on the partial results. All items of the Bloom filters are protected using the method of Paillier encryption, with pk_p as $[\![BF(f)]\!] = ([\![BF(f)[1]]\!], \dots, [\![BF(f)[\gamma]]\!])$. The secure dot product is calculated as:

$$\left[\!\left[secure_{DP}\left(BF(f_i), BF(f_j)\right)\right]\!\right] = \sum_{p=1}^{\gamma} [\![(BF(f_i)[p]).(BF(f_j)[p])]\!] \quad (4)$$

Consider the encrypted Bloom filters $BF = \{[\![BF(f_1)]\!], \dots, [\![BF(f_n)]\!]\}$ and a query Bloom filter $[\![BF(q)]\!]$, the search server computes the encrypted dot product between the file query and each other file. Calculating dot products entails the computation of secure multiplication between encrypted Bloom filters, which is carried out with the help of the file server. The search server permutes the encrypted scores by using a random permutation π provided by the search user. This permutation conceals the relation between the similarity scores and the real file identifiers from the file server. Since a random permutation is provided by the user for each query, the file server is not able to perform any possible inference attack. The permuted encrypted scores are then uploaded to the file server, which uses the private key to decrypt them and send them back to the user. The user performs the inverse of the permutation to get the actual file identifiers with the maximum dot product. Algorithm 2 shows the secure similarity search.

Algorithm 2 Secure similarity Search

SS:
Input: $[\![BF(f_1)]\!], \dots, [\![BF(f_n)]\!]$: n encrypted Bloom filters
- $[\![BF(q))]\!]$: query Bloom filter, π : random permutation
- for each $[\![BF(f_i)]\!] \in BF$ do
  - $[\![dot]\!] = [\![0]\!]$
  - for $j = 1\ to\ \lambda$ do
    $[\![dot]\!] = [\![dot]\!].$
    $[\![secure_{DP}(BF(f_i), BF(q))]\!]$
  - $[\![Scores[i]]\!] = [\![dot]\!]$
- Permute $\{[\![Scores[1]]\!], \dots, [\![Scores[n]]\!]\}$ with random permutation π and send the result to FS

FS:
Input: permuted encrypted scores
- Decrypt Scores and send the results to the user

DU:
- Perform the inverse permutation π on the similarity scores to associate them with their actual identifiers
- Sort the scores to get file identifiers with maximum dot product.

C. Scheme Details

This section shows more details for our scheme. Our scheme is based on public key based encryption and it is composed of four PPT algorithms:

• **KeyGen**(λ): This algorithm receives the security parameter λ, and outputs public key pk_p for Paillier encryption and private key sk_p for Paillier decryption.

• **IndexConstruction** (f, pk_p, l, γ): This algorithm is run by the data owner to build the encrypted Bloom filter $[\![BF(f)]\!]$ for the file f, This algorithm uses l LSH hash functions $h: \{0,1\}^{26^2} \rightarrow \{0,1\}^{\gamma}$ to hash the keywords of f and builds $BF(f)$, then it encrypts $BF(f)$ by pk_p to get $[\![BF(f)]\!] = ([\![BF(f)[1]]\!], \dots, [\![BF(f)[\gamma]]\!])$. Note that, since Paillier encryption is semantically secure, the encryption of the same numbers will produce with high probability (almost 1) a unique ciphertext.

• **TokenGeneration** (q, pk_p, l, γ): This algorithm is run by the authorized user, who knows the value of pk_p, to generate the search token $[\![BF(q)]\!] = ([\![BF(q)[1]]\!], \dots, [\![BF(q)[\gamma]]\!])$ for the file q as the same way for generating the encrypted Bloom filters.

• **SecureSearch** $([\![BF(q)]\!], [\![BF]\!], \pi)$: This protocol is run between the SS and the FS to answer the search token $[\![BF(q)]\!]$ as explained in Algorithm 2.

Discussions. When a file f appears in the search results for a given query q, this means that the keywords of that file match the keywords of q. If the keywords of q are a subset of the keywords of f, then f should be included in the result set. This is because keywords of both files will set 1 to the same positions of the corresponding Bloom filters as we use the same l hash functions for all files. So the inner product between q and f will be a higher value. Hence, the file f appears in the result set. When keyword w of q is slightly different from the word w of f (i.e., $d(w', w) \leq r_1$, where r_1 is a threshold defined in LSH), our scheme also can return f with high probability. File f will be included in the results when for all the $i = 1, \dots, l$, we have $h_i(w) = h_i(w')$. A missing case happens only when $d(w', w) \leq r_1$ but $h_i(w) \neq h_i(w')$. Observe that when $d(w', w) > r_1$, then there is a low probability for hashing the two keywords in the same positons.

Now suppose that keyword w in f matches with one keyword of q but $d(p, q) > r_2$, this case is called false positive and it is introduced to our scheme due to the use of $BF(f)$ and the locality sensitive hashing method. Notice that the false positive rate of a Bloom filter of size γ that stores n elements by using l hash functions is $(1 - {}^{1}/_{m})^{ln}$. While ${p_2}^{l}$ is the false positive introduced by LSH.

Dynamic scheme. Our scheme generates a separate index for each file, so to add new files, we only need to generate their own indexes and upload them to the search server. Removing files is trivial.

VI. Security Analysis

In this section, we discuss the security properties of our proposed scheme. Recall that the actual files are encrypted and uploaded to the file server. However, such files are usually encrypted using and CPA-encryption method that does not reveal any information to the server beyond the number of files and their actual sizes. Thus we do not consider the leakage of actual files on our discussion about the security of the proposed scheme. We consider only the security of the index and search queries.

During our scheme, the index of files and search queries are generated in a similar way, so the argument about file confidentiality and query privacy will be identical.

Bloom filter representation provides file confidentiality. Since all files are mapped into a fixed bloom vector. Recall that file keywords are transformed into bi-gram vectors and hashed several times by using secret LSH hash functions to determine the locations at which keywords' frequencies are stored. The only leaked information to the server is numerical vectors. This leakage is true in the case of unencrypted vectors. However, all vectors (Bloom filters) are encrypted in our scheme before being outsourced. Thus no information is leaked about files or queries.

Our proposed scheme supports also the privacy of search and access patterns. This is because; each element of the Bloom filter is encrypted with the Paillier encryption scheme which is a semantically secure encryption method. This ensures that SS is not able to distinguish each element from a random number. Therefore, the server will not be able to notice whether two queries are generated from identical keywords, thus the privacy of the search pattern is preserved. In the encrypted Bloom filters, the search server deals only with encrypted values so the similarity scores do not reveal anything to SS. In this case, SS will not be able to know the access pattern. Recall that FS owns a copy from Paillier private key and so it can compute the actual score values, but since those scores are permuted by the SS randomly, FS losses the ability to know the access pattern.

VII. Performance Evaluation

In this section, we test the performance of our proposed scheme. We implement our scheme with a real-data collection from Wikipedia articles. Particularly, we picked 100 articles; with each page we selected also its nine history versions. Therefore, we have in total a collection of 1000 files. To get better results, we first preprocess the files using some methods from the literature of information retrieval: stop words removal, lower case transformation, and keywords stemming (specifically, we adopt Porter algorithm [31]). In our collection, there are 4733 keywords, with an average 102 keywords per file. For LSH we use $l = 30$ hash functions and set $\gamma = 700$ for Bloom filter size. Experiments were executed on a PC of 1.6 GH Intel corei5 CPU, and 8 GB running on 64-bits Windows 10. Code was written by JAVA programming language. For handling big numbers we used BigNumber Java library. The security parameter is set to1024-bits, which is the standard value for current cryptographic applications [35]. The underlying representation method of our scheme is compared with two famous state-of-the-art method: Term vector (TV) of [36], enhanced term vector (ETV) [37], and Simhash of [38].

1) Efficiency measurement

We measure the time cost and communication overhead of our proposed scheme. Figure 3 reports the elapsed time for constructing encrypted Bloom filters for the entire collection when the size γ varies between 100 to 1000. Obviously, the execution time grows linearly as the value of γ increases. Search tokens take the same time as they construct in the same manner.

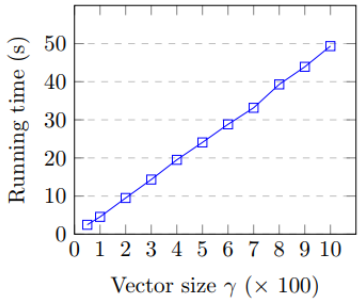

Fig. 3: Encrypted Bloom filter construction time.

2) Secure search

In this experiment, we measure the time cost that is required by our scheme to search the outsourced files. Figure 4 illustrates the CPU running time for searching the entire collection with variable values of Bloom filter sizes. It is easy to see that, longer Bloom filters entail immediately more search time.

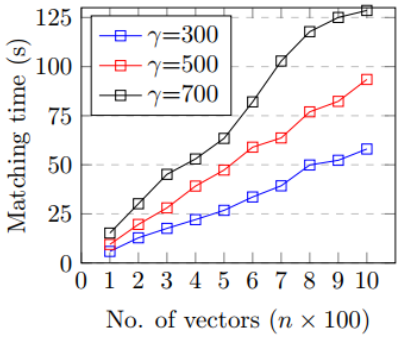

Fig. 4: Search time.

3) Accuracy investigation

The accuracy of results is investigated in terms of precision metric. Notice that, our scheme finds the top-k relevant file instead of finding all the relevant documents. Thus, we consider precision metric only without recall metric. When the set of relevant files is dented by A, and the set of retrieved files is denoted by B, then $|A \cap B|/|B|$ stands for the precision of results. In this experiment, each article is used as a query, the set A is those files in the same history of the query. The value of returned files $k \in \{5, \ldots, 15\}$ controls the size of B. Figure 5 shows the effect of the number of LSH hash functions l on the precision for the variable size of Bloom filters. Notice that more LSH functions cause to get variable positions in the generated Bloom filter, which leads to retrieve fewer non-relevant files, and hence obtain higher precision. Notice that the accuracy of results not lower than 40% under all settings.

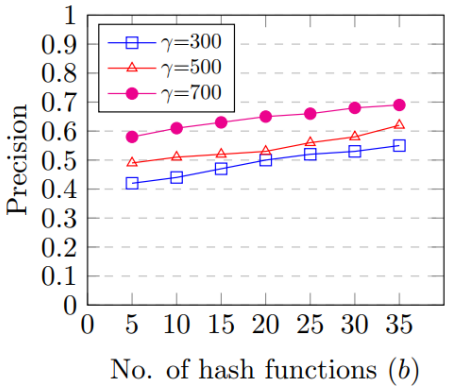

Fig. 5: Precision of results.

Figure 6 compares the average accuracy of our scheme against various schemes. The value of x-axes varies for different k values. The values of γ and l are set to 700 and 35, respectively. We set the binary vector of Simhash method to 64. Notice that the accuracy of all schemes decreases with large k values. Term vector method shows the best accuracy since it describes files with the classical way in the information retrieval arena. Our scheme lowers the effectiveness of results to enhance efficiency. The method of Simhash utilizes shorter vectors to represent files so it demonstrates the lowest accuracy.

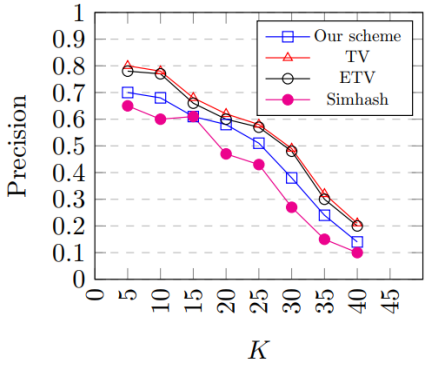

Fig. 6: accuracy comparison.

What remains to test the ability of our scheme to handle the misspelled keywords in the provided search query. In this experiment, we select a number of keywords from the query file and replace the occurrence of each selected keyword with its modified version. Figure 7 demonstrates the effect of fuzzy keywords on accuracy. In this experiment, we return only 5 files (i.e. k=5). Notice that, in all schemes, when more fuzzy keywords are provided less accuracy is obtained. Notice that our proposed method to transform file keywords gives our scheme the advantage over the competitor schemes to recover the misspelled keywords.

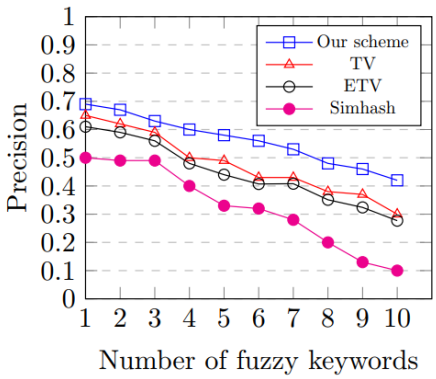

Fig. 7: Effect of misspelled keywords.

VIII. Conclusion

In this paper, we investigate the challenging problem of multi-keyword similarity ranked search over the encrypted cloud data. We introduced several innovative designs that are integrated to solve this problem efficiently. Our scheme adopts LSH functions in the Bloom filters to describe files and search queries. Concretely, we utilize a good keyword transformation method and applied stemming on file keywords. With these techniques, the proposed scheme can handle efficiently more misspelling mistakes. Our scheme enjoys enhanced security protection, where access and search patterns are protected. To do so, file vectors are encrypted with Paillier cryptosystem that has interesting homomorphic properties. Additionally, our proposed scheme takes keywords' frequencies into its consideration to support better ranking. Finally, we provide thorough security analyses and investigate performance by carrying out several experiments on real-world data set, which indicate that the proposed scheme is secure and suitable for practical usage.

Conflict of Interest

The authors have no conflict of relevant interest to this article.

REFERENCES

[1] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: ramification, attack and mitigation," in *Ndss*, 2012, vol. 20, p. 12: Citeseer.

[2] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, 2000, pp. 44-55: IEEE.

[3] E.-J. J. I. C. e. A. Goh, "Secure indexes," in Proc. Cryptol. ePrint Arch. vol. 2003, p. 216, 2003.

[4] R. Curtmola, J. Garay, S. Kamara, and R. J. J. o. C. S. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in Proc. CCS, vol. 19, no. 5, pp. 895-934, 2011.

[5] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *International conference on applied cryptography and network security*, 2005, pp. 442-455: Springer.

[6] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *International conference on applied cryptography and network security*, 2004, pp. 31-45: Springer.

[7] L. Ballard, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data," in *International conference on information and communications security*, 2005, pp. 414-426: Springer.

[8] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Theory of cryptography conference*, 2007, pp. 535-554: Springer.

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

[10] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. J. I. C. e. A. Lou, "Enabling Efficient Fuzzy Keyword Search over Encrypted Data in Cloud Computing," in IACR Cryptol. ePrint. vol. 2009, p. 593, 2009.

[11] C. Wang, K. Ren, S. Yu, and K. M. R. Urs, "Achieving usable and privacy-assured similarity search over outsourced cloud data," in *2012 Proceedings IEEE INFOCOM*, 2012, pp. 451-459: IEEE.

[12] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *2012 IEEE 28th International Conference on Data Engineering*, 2012, pp. 1156-1167: IEEE.

[13] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, 2014, pp. 2112-2120: IEEE.

[14] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 668-679.

[15] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 707-720.

[16] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*, 1999, pp. 223-238: Springer.

[17] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *2010 IEEE 30th international conference on distributed computing systems*, 2010, pp. 253-262: IEEE.

[18] C. Wang, N. Cao, K. Ren, W. J. I. T. o. p. Lou, and d. systems, "Enabling secure and efficient ranked keyword search over outsourced cloud data," vol. 23, no. 8, pp. 1467-1479, 2011.

[19] N. Cao, C. Wang, M. Li, K. Ren, W. J. I. T. o. p. Lou, and d. systems, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in Proc. IEEE INFOCOM, vol. 25, no. 1, pp. 222-233, 2013.

[20] W. Sun *et al.*, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking,"

in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, 2013, pp. 71-82.

[21] R. Li, Z. Xu, W. Kang, K. C. Yow, and C.-Z. J. F. G. C. S. Xu, "Efficient multi-keyword ranked query over encrypted data in cloud computing," in Future Generation Computer Systems vol. 30, pp. 179-190, 2014.

[22] C.-M. Yu, C.-Y. Chen, and H.-C. J. I. s. j. Chao, "Privacy-preserving multikeyword similarity search over outsourced cloud data," in IEEE systems journal, vol. 11, no. 2, pp. 385-394, 2015.

[23] J. Wang, X. Yu, M. J. A. J. f. S. Zhao, and Engineering, "Privacy-preserving ranked multi-keyword fuzzy search on cloud encrypted data supporting range query," in Arabian Journal for Science and Engineering, vol. 40, no. 8, pp. 2375-2388, 2015.

[24] Z. Fu, X. Wu, C. Guan, X. Sun, K. J. I. T. o. I. F. Ren, and Security, "Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement," in IEEE Transactions, vol. 11, no. 12, pp. 2706-2716, 2016.

[25] T. Moataz, A. Shikfa, N. Cuppens-Boulahia, and F. Cuppens, "Semantic search over encrypted data," in *ICT 2013*, 2013, pp. 1-5: IEEE.

[26] Z. Fu, J. Shu, X. Sun, and D. Zhang, "Semantic keyword search based on trie over encrypted cloud data," in *Proceedings of the 2nd international workshop on security in cloud computing*, 2014, pp. 59-62.

[27] S. Bugiel, S. Nurnberger, A. Sadeghi, and T. Schneider, "Twin clouds: An architecture for secure cloud computing," in *Workshop on cryptography and security in clouds (WCSC 2011)*, 2011, vol. 1217889.

[28] G. Liu, G. Yang, S. Bai, Q. Zhou, and H. J. I. A. Dai, "FSSE: An effective fuzzy semantic searchable encryption scheme over encrypted cloud data," in IEEE Access, vol. 8, pp. 71893-71906, 2020.

[29] Z. Fu, L. Xia, X. Sun, A. X. Liu, G. J. I. T. o. I. F. Xie, and Security, "Semantic-aware searching over encrypted data for cloud computing," in IEEE Transactions, vol. 13, no. 9, pp. 2359-2371, 2018.

[30] K. Xue *et al.*, "Two-cloud secure database for numeric-related SQL range queries with privacy preserving," in IEEE Transactions, vol. 12, no. 7, pp. 1596-1608, 2017.

[31] M. F. J. P. Porter, "An algorithm for suffix stripping," Program, 1980.

[32] J. B. J. M. T. C. L. Lovins, "Development of a stemming algorithm," Mech. Transl. Comput. Linguistics, vol. 11, no. 1-2, pp. 22-31, 1968.

[33] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, 1998, pp. 604-613.

[34] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Vldb*, 1999, vol. 99, no. 6, pp. 518-529.

[35] F. Saad Muhi, "A Pseudorandom Binary Generator Based on Chaotic Linear Feedback Shift Register," *Iraqi Journal for Electrical And Electronic Engineering* vol. 12, pp. 155-160, 2016.

[36] W. Jiang, M. Murugesan, C. Clifton, and L. Si, "Similar document detection with limited information disclosure," in *2008 IEEE 24th International Conference on Data Engineering*, 2008, pp. 735-743: IEEE.

[37] M. Murugesan, W. Jiang, C. Clifton, L. Si, and J. T. V. J. Vaidya, "Efficient privacy-preserving similar document detection," The VLDB Journal, vol. 19, no. 4, pp. 457-475, 2010.

[38] S. Buyrukbilen and S. Bakiras, "Secure similar document detection with simhash," in *Workshop on Secure Data Management*, 2013, pp. 61-75: Springer.