Chaotic Searchable Encryption for Mobile Cloud Storage

Abir Awad, Adrian Matthews, Yuansong Qiao, Brian Lee

***Abstract*—This paper considers the security problem of outsourcing storage from user devices to the cloud. A secure searchable encryption scheme is presented to enable searching of encrypted user data in the cloud. The scheme simultaneously supports fuzzy keyword searching and matched results ranking, which are two important factors in facilitating practical searchable encryption. A chaotic fuzzy transformation method is proposed to support secure fuzzy keyword indexing, storage and query. A secure posting list is also created to rank the matched results while maintaining the privacy and confidentiality of the user data, and saving the resources of the user mobile devices. Comprehensive tests have been performed and the experimental results show that the proposed scheme is efficient and suitable for a secure searchable cloud storage system.**

***Index Terms*—Cloud, Security, Searchable encryption, Chaos, Locality sensitive hashing.**

I. Introduction

CLOUD computing is a model to enable convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) [1]. In the current Internet, people can easily access their data stored in the cloud with their mobile devices from anywhere e.g., check emails, read the history of online chatting applications, view previously saved photos, videos or other kind of documents. To provide security in all such scenarios, it is essential to store and access the outsourced data in a secure and efficient manner. For the protection of data privacy and control, data is usually encrypted before outsourcing, which makes its effective utilization a challenge. In particular, indexing and searching the outsourced encrypted data becomes problematic.

Searchable encryption (SE) allows searching over encrypted data in the cloud and returns to the user the data that correspond to the given keywords, without having to reveal the keywords. It is thus a critical enabler for securing outsourced data. Traditional searchable encryption [2]-[7] schemes allow a user to securely search over encrypted data through keywords but only support 1) exact keyword matching, which is not a practical requirement for current mobile phone input methods and 2) boolean search without capturing the relevance of data files. The system usability can be greatly enhanced by the use of fuzzy keyword search [1], [8]- [10] instead of traditional searchable encryption. Fuzzy, or error tolerant, searchable encryption returns to the user the files that match not only the exact predefined keywords but also the closest possible matched files based on keyword similarity semantics. Similarly, system usability is greatly enhanced by ranked search [11], [12] which returns the matched files in a ranked order determined by appropriate relevance criteria. This paper investigates the problem of supporting both ranked and fuzzy keyword search in a single scheme to achieve effective utilization of remotely stored encrypted data in mobile cloud computing applications.

Many approaches are proposed to enable fuzzy search. Researchers in [8] consider the use of wildcards to enlarge the range of possible similar keywords searched, but this technique only covers part of the possible close keywords. A wildcard only permits capturing of errors provided we know where they are located in the keyword [1]. In [9], the authors proposed a new cryptographic primitive called Public Key Error Tolerant Searchable Encryption (PKETS) which is based on public key encryption with keyword search proposed in [2]. This algorithm was applied to the biometric data in [13]. Acceptable erroneous keywords did not have to be specified in advance in their algorithm. However, this approach was designed for a special type of data i.e. iris code. This technology is useful at airports as a replacement for passports but it is not designed for text documents. The authors in [14], proposed to embed edit distance (Levenshtein distance) into Hamming distance to obtain a fuzzy keyword search suitable for strings and then text files. This method uses existing locality sensitive hashing (LSH) to enable the fuzziness in the search method and has a very low distortion. However, this method is mainly theoretical and the proposed embedding technique introduces a lot of redundancy, which increases the dimension of the stored data, and is not suitable for the case of mobile usage because of the small amount of memory available. Another method, proposed in [15], uses bloom filters and Jaccard similarity to perform the translation and the LSH. It also introduces ranking of the retrieved encrypted data. However, the ranking has to be performed by the user himself and not automatically by the server which can add unwanted burden for a mobile user's device.

The paper is submitted the October 2014.

This work is made within the framework of the Irish Centre for Cloud Computing and Commerce (IC4) which is an Irish government and Enterprise Ireland supported technology research centre established in 2012.

A. Awad. B. Lee, A. Matthews and Y. Qiao are within the Irish Centre for Cloud Computing and Commerce Software Research, Athlone Institute of Technology, Software research Institute, Athlone, , Ireland.

e-mail: aawad@research.ait.ie, blee@ait.ie, amatthews@research.ait.ie, ysqiao@research.ait.ie

Actually, very few searchable encryption schemes support the ranking of matched items though this problem has recently attracted the attention of some researchers [11], [12], [16]. Fuzziness and ranking are currently two different research axes and very few researchers have considered combining them [15], [17]. However, these methods are either not practical for mobile usage as is the case in [15] or they suffer from security problems as is the case in [17].

In this paper, we propose a new fuzzy transformation by introducing chaos and enhance the fuzziness through amplification of the LSH, which significantly improves both the security and the efficiency of the fuzzy searching process compared to the existing solutions. Furthermore, comprehensive tests on different LSH methods are performed in order to select the best one to be used in our algorithm. Chaotic systems are widely used in the cryptography domain and have attracted the attention of many researchers [21]-[23] due to the interesting characteristics of chaos. However, to the best of our knowledge, this is the first paper proposing to use chaos in the searchable encryption schemes. Our proposed system is, in addition, designed to support fuzzy and ranking mechanisms and is proven to be practical for mobile usage.

The paper is organized as follows: Sections 2 and 3 provide background information and related works. Section 4 presents the proposed and tested chaos based locality sensitive hashing methods. Section 5 describes the proposed chaotic searchable encryption solution. The simulation results and security analysis of the proposed algorithm are given in section 6. Finally, we summarize our conclusions in section 7.

II. USEFUL TOOLS

A. Locality sensitive hashing

The differences occurring between similar data are reduced by LSH functions with high probabilities. Then, similar results are obtained for data with close proximity. However, distant data remain remote.

Let B be a metric space, $\lambda_{\min}, \lambda_{\max} \in \mathbb{R}$ with $\lambda_{\min} < \lambda_{\max}$ and $\epsilon_1, \epsilon_2 \in [0,1]$ with $\epsilon_1 > \epsilon_2$

A family $H = \{h_1, \dots, h_\mu\}$ is an LSH family if for all $x, x' \in B$:

$$\Pr_H[h(x) = h(x')] > \epsilon_1, \text{if } d(x,x') < \lambda_{\min} \quad (1)$$

$$\Pr_H[h(x) = h(x')] < \epsilon_2, \text{if } d(x,x') > \lambda_{\max} \quad (2)$$

d is the distance metric utilized (e.g. Hamming distance).

The choice of a suitable LSH depends on the data types (Binary, Euclidean space, biometric...). A survey of existing LSH families can be found in [24].

The Jaccard coefficient is usually used to measure the similarity between two sets *A* and *B* containing words from two documents [25]. It is defined in (3) as follows:

$$s(A,B) = \frac{|A \cap B|}{|A \cup B|} \quad (3)$$

The distance between these sets can be obtained by (4) as follows:

$$d(A,B) = 1 - s(A,B) \quad (4)$$

For this measure, some researchers [26] proposed an LSH family called min-hash. If π is a random permutation, the hash value is defined by (5) as follows:

$$h_\pi(A) = \min\{\pi(a) \ / \ a \in A\} \quad (5)$$

and the probability that the two hashed values are equal, is equal to the Jaccard distance (6):

$$\Pr[h_\pi(A) = h_\pi(B)] = s(A,B) \quad (6)$$

In our proposal, min-hash is used to support the fuzzy transformation applied on the keyword indexing the outsourced files.

B. Bloom filters

A Bloom filter is a data structure used for answering set membership queries.

Bloom filter with storage

A Bloom filter with storage is an extension of Bloom filter [9], [27]. It gives not only the result of the set membership test but also an index associated with the element. It has an array of subsets called buckets $T_1, \dots T_m$which are initially empty. For each element y to be indexed, we add to the bucket T_α all the tags associated with y as follows: $T_\alpha \leftarrow T_\alpha \cup \psi(y)$ where ψ is the tagging function and $\alpha = h'_j(y), j \in [1, v]$ (where v is the number of hash functions).The set of tags associated with y is obtained by computing the intersection between the corresponding buckets $\bigcap_{j=1}^{v} T_{h'_{j(y)}}$.

Bloom filter encoding

Bloom filter encoding is described by the authors in [15]. A bloom filter is a bit array that is affiliated with some hash functions. Each hash function maps an element to a bit location with a uniform probability. The bloom filter in this case is used to embed a string *S* into the filter in order to obtain an array of numbers which can be used as an input for the minhash method. Each n-gram of a keyword is subject to each hash function and the corresponding bit locations are set to 1. The indices of the "1" values in the bloom filter provide the array of numbers which can be then used as an input for the minwise permutation to obtain the minhash value.

C. Order Preserving Symmetric Encryption

The OPSE [28] is a deterministic encryption scheme in which the numerical order of the plaintexts is preserved by the encryption function and a comparison operation can then be performed without revealing the plaintext values. In our proposal, we use OPSE to encrypt the relevance score of each keyword in the related files. These values need to be stored in the cloud in order to perform ranking in the search phase and must be secured as they can reveal information about the keywords and the files. Traditional methods such as AES are not appropriate for this case as the relevance score ranking need to be achieved on the encrypted values. In this situation, encryption schemes like OPSE that preserve the numerical ordering should be used.

D. PWLCM Chaotic Map

Chaos has certain distinct characteristics, e.g. good pseudo-randomness and sensitivity to its control parameters, that can be directly linked to the properties of confusion and diffusion in cryptography. In addition, these systems are deterministic, meaning that their future behavior is fully determined by their parameters, with no random elements involved. However, the chaotic signal is pseudo-random and may appear as noise for unauthorized users. Chaotic values are often generated with simple iterations, which make chaos suitable for designing strong and high speed systems.

PWLCM (Piece Wise Linear Chaotic Map) [21], [23] is one of the simplest chaotic systems and has good properties [29]. A piecewise linear chaotic map is a map composed of multiple linear segments and can be described in (7) as follows:

$$x(n) = F[x(n-1)] = \begin{cases} x(n-1) \times \dfrac{1}{p} & if\ \ 0 \le x(n-1) < p \\ [x(n-1) - p] \times \dfrac{1}{0.5 - p} & if\ \ p \le x(n-1) < 0.5 \\ F[1 - x(n-1)] & if\ \ 0.5 \le x(n-1) < 1 \end{cases} \tag{7}$$

where the positive control parameter and the initial condition are respectively $p \in (0; 0.5)$ and $x(\text{i}) \in (0; 1)$.

III. RELATED WORK

In this section, we briefly explain some existing searchable encryption methods. We classify these methods into three groups; Fuzzy SE methods, ranking based SE methods and combined fuzziness and ranking based SE methods.

A. Fuzzy SE methods

In their papers [9], [13], Bringer et al. proposed a new scheme permitting search over encrypted data with an approximation of a keyword. An application in the biometric domain is also proposed. A biometric identification scheme arises from this construction; it permits identification of a person using his biometrics in an encrypted way. A specific difficulty concerning biometrics is their fuzziness. It is nearly impossible for a sensor to obtain the same image from biometric data twice. The classical way to solve this problem is to use a matching function, which basically tells if two measures represent the same biometric data or not, but these methods do not meet the privacy requirements that someone can expect from an such identification scheme. The Bringer et al. algorithm resolves this issue and provides the privacy missing in the existing algorithms. This method uses a combination of LSH method specific for an iris code (beacon indexes) to enable the fuzziness and a Bloom filter with storage to accelerate the search on the encrypted data.

In [14], the authors modified the above mentioned algorithm to allow its usage for text messages. The changes entail on applying embedding and sketching methods on the message which enables the application of the above mentioned algorithm in [9], [13] that was previously used for the biometric information. However, the algorithm is still theoretical and no implementation or test is provided.

The authors in [1], proposed an Effective Error-Tolerant Keyword Search for Secure Cloud Computing. They propose a scheme based on a fuzzy extractor. Their method is able to transform the servers' search for error-tolerant keywords on cipher texts to the search for exact keywords on plaintexts using an index table. Their method is tested on the Digital Bibliography & Library Project (DBLP) dataset, which was developed and maintained by a team from Germany Trier University. The algorithm seems promising but it does not take the ranking problem into consideration.

B. Ranking based SE method

In [11], the authors are the first to propose a ranked keyword search over encrypted cloud data that enables effective utilization of remotely stored encrypted data in the cloud. They embed weight information (relevance score) of each file during the establishment of a searchable index before outsourcing the encrypted file collection. They also used Order Preserving Symmetric Encryption (OPSE) to protect this sensitive information. Experimental evaluation is conducted on the Request For Comments (RFC) database [30]. This scheme allows the ranking of the searched files but does not take into account the fuzziness of the keyword.

C. Combined fuzziness and raking based SE methods

In [15], the authors proposed a symmetric scheme for similarity search over encrypted data and their algorithm allows a fuzzy keyword search over text documents. First, a translation is used to embed strings into a Bloom filter. In this case, each keyword is represented by a set of substrings of length n or n-grams. Then, each substring is hashed and the corresponding bit locations set to one. The other buckets of the Bloom filter are null. The encoding, R, of the keyword is an array of the bit locations in the Bloom filter. If Δ is the domain of all possible elements of the encoding set R and P is a random permutation on Δ, $P[i]$ is the element in the i^{th} position of P and min is a function that returns the minimum of a set of numbers. Then, the minhash of a keyword A under P is as follows in (8):

$$minhash_P(R) = min(\{i \mid 1 \le i \le |\Delta| \Lambda P[i]\}) \tag{8}$$

This method also permits the user to perform the ranking by means of the encrypted bit vectors returned by the server as an answer to the user's query. Once the ranking is performed, the user sends the identifiers of the data items with top t high scores to the server which, in turn, returns the encrypted items corresponding to the provided identifiers. The user decrypts these to obtain their plaintexts. The authors provide an implementation and test of the method on the Enron dataset [31]. As can be seen, this method combines ranking of the returned results and fuzziness of the search. However, this method requires additional work i.e. the ranking must be performed by the user which is not practical for a mobile device. In our proposal, the user is relieved from this task and the ranking is calculated automatically by the server while maintaining the privacy and the confidentiality of the user.

In [17], we proposed a new solution enabling fuzzy search and ranking together on encrypted data in the cloud. This method uses a murmur hash function to enable the fuzziness. The used LSH takes a word as an input and then hashes each letter of this word. The locality sensitive hash value of this word is then the minimum of the letters' hashes. In Fig. 1, we give an example of the word "minhash".

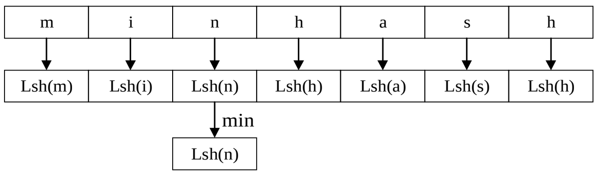

Fig. 1. A simple minhash example

Even though this method is quite fast and gives good results for the fuzzy search, it suffers from security issues because of the simple fuzzy transformation used.

IV. PROPOSED LOCALITY SENSITIVE HASHING METHODS

In this section, we describe the locality sensitive hashing methods proposed in this paper. In the first chapter A, we describe the two proposed minhash methods: Grp and Omflip minhashes which will be compared to Kuzu minhash [15] later in the paper. In chapter B, we describe the amplification that we apply on the proposed minhashes to obtain the amplified Grp minhash and amplified Omflip which is also compared to the amplified Kuzu minhash in section VI.

The modification of all the mentioned minhashes with the chaos that we propose in this paper is explained in chapter C and the obtained chaotic minhashes are introduced: chaotic Kuzu minhash, chaotic Grp minhash, chaotic Omflip minhash, amplified chaotic Kuzu minhash and amplified chaotic Grp and Omflip minhashes.

A. Minhash methods

1) Grp minhash

In this method, the same scheme of Kuzu is used but the random permutation is replaced by the Grp permutation method [18], [19]. GRP permutation is defined in (9) as follows:

$$R3 = Grp(\, R1, R2) \tag{9}$$

R1 is the source array, *R2* is the configuration array which is generated by a pseudo random generator and *R3* is the destination array for the permuted values.

The basic idea of the Grp is to divide the values from the source *R1* into two groups according to the values in *R2*. For each bit in *R1*, we check the corresponding bit in *R2*. If the bit in *R2* is 0, we move this bit from *R1* into the first group. Otherwise, we put this bit into the second group (see Fig.2).

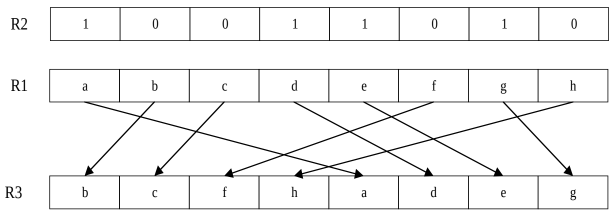

Fig. 2. GRP permutation method

2) Omflip minhash

This method is also a variation of Kuzu but the random permutation is replaced this time by the Omflip permutation method [23]. The OMFLIP (OMega-FLIP) permutation is basically a concatenation of two permutation stages – an Omega stage and a Flip stage.

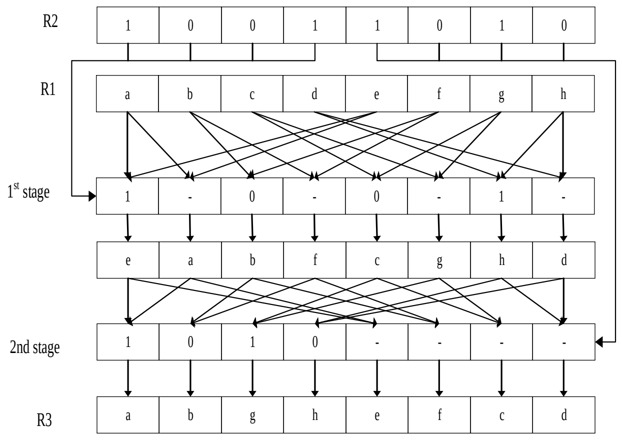

Fig. 3. OMFLIP permutation method

In Fig. 3, $R1$ is the source array that contains the values to be permuted. $R2$ is the configuration array that is generated randomly and which should be of the same length as $R1$. $R3$ is the resultant permuted sequence. The values are permuted in 2 stages. In the first stage, the values of $R1$ are placed as shown in Fig. 3. The first $N/2$ values of the control sequence $R2$ control the permuted array. If the value of the box i of $R2$ is 1, the two adjacent cells i and $i+1$ are permuted. If not, nothing is done. In a second stage, the bits of the array resulting from the first stage are placed as shown below in Fig. 3. The last $N/2$ values of the control sequence $R2$ control the resulting array. If the value of $R2$ is 1, the two adjacent cells i and $i+N/2$ are permuted. If not, nothing is done. At the end, the permuted OMFLIP array $R3$ is obtained.

B. Amplified minhash methods

To amplify a locality-sensitive hashing family a AND-OR construction can be used [15].
The AND construction is formed with k random functions from H: $g_i = \left(h_{i_1} \Lambda h_{i_2}, \dots \Lambda\, h_{i_k}\right)$. In this context, $g_i(x) = g_i(y)$ if and only if $\forall j \left(h_{i_j}(x) = h_{i_j}(y)\right)$ where $1 \le j \le k$. The OR construction is formed with λ different AND constructions such that $g(x) = g(y)$ if and only if $\exists i\left(g_i(x) = g_i(y)\right)$ where $1 \le i \le \lambda$.With such a construction, we can turn an (r_1, r_2, p_1, p_2) sensitive family into an (r_1, r_2, p'_1, p'_2) sensitive family where $p'_1 = 1 - \left(1 - {p_1}^k\right)^\lambda$ and $p'_2 = 1 - \left(1 - {p_2}^k\right)^\lambda$.

We applied the AND-OR construction on the three explained minhash methods in order to obtain: the amplified Kuzu minhash, amplified Grp minhash and amplified Omflip minhash that we compare in section VI of this paper.

C. Chaotic minhash methods

The idea of taking advantage of digital chaotic systems and of constructing chaotic cryptosystems has been extensively investigated and attracted many researchers [18]- [23] but to the best of our knowledge, it has not been previously considered for searchable encryption methods. In this paper, we propose new minhash methods based on Piece Wise Linear Chaotic Map (PWLCM) presented in section II. In these methods, the translation i.e. the encoding of the keyword, is performed by the chaotic map instead of the Bloom filter used by Kuzu et al. [15]. PWLCM is then used to transform the keyword to a set of numbers that will be used as input for the minwise permutation method in order to obtain finally the minhash value.

A 1-gram shingling is applied on each keyword and the ASCII code of each letter is mapped to the interval [0,1] and then encoded by the chaotic map. For each shingle, a number of iterations are performed and the obtained chaotic values are then mapped to integers in the interval $[0,m]$, where m is a secret parameter for the minhash. Finally, the keyword is represented by an array of values that are used as an input for the minhash method. The usage of chaos instead of a Bloom filter in the translation phase in the above mentioned minhashes gives the following chaotic minhashes: chaotic Kuzu minhash, chaotic Grp and Omflip minhashes. When the amplification method i.e. the AND-OR construction is also applied on each one in addition with chaos, the amplified chaotic minhashes are obtained: amplified chaotic Kuzu minhash and amplified chaotic Grp and Omflip minhashes. A comparison is performed on these locality sensitive hashing methods in order to determine the best one to use in our searchable encryption method.

V. THE PROPOSED ALGORITHM

In this section, we propose a new chaotic searchable encryption algorithm. The proposed approach allows searching over encrypted data stored in the cloud and returns the relevant files to the queries in a ranked order. This scheme permits search not only with the exact keyword used during the storage process, but also with an approximate keyword. Fig. 4 introduces the considered scenario for the proposed algorithm.

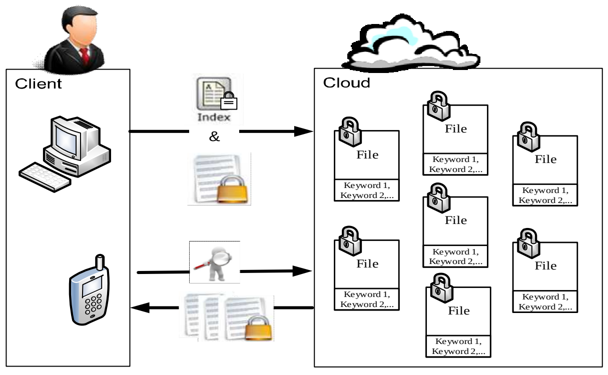

Fig. 4. Usage scenario of the proposed algorithm

It consists of two different parts: the sending process and the searching process. In the sending part, the user's computer encrypts the files that he wants to store in the cloud. It creates the meta-data necessary for the cloud to search these files later. In the searching process, the user queries the cloud through his mobile phone. The cloud receives the hashed query, performs the search, retrieves and returns the required documents in a ranked order. We give below the detailed description for both processes.

A. Sending a message x

This phase consists, basically, of two transformations; the fuzzy and the ranking calculation. The obtained keyword index (from the fuzzy calculation) and the ranking score, sc, are stored in the cloud in addition to the encrypted file itself.

Assume that N files need to be added to the cloud. The following steps of the algorithm will be performed:

1- The cloud attributes to each file $F_j, (j = 1 \dots N)$ a unique identifier F_{d_j} and sends it to the user device.

2- The user device sends the encrypted file $Enc(F_j)$ to the cloud which stores it in a storage cell that depends on F_{d_j}.

3- The user device adds F_{d_j} (the ID of the file) to the posting list I_{w_i} of the corresponding keyword w_i and adds also the relevance score sc of this keyword for the added file. A posting list (also referred to as inverted index) is an index data structure storing a mapping from content, such as words, to the location of a file in a set of documents. The purpose of a posting list is to allow fast searches over the database. An example of the posting list for a keyword w_i is given in Fig. 5 [17].

File ID F_d	F_{d_1}	F_{d_2}	...	F_{d_N}
Relevance score	$OPSE\left(sc(w_i, F_{d_1})\right)$	$OPSE\left(sc(w_i, F_{d_2})\right)$		$OPSE\left(sc(w_i, F_{d_N})\right)$

Fig. 5. Posting list $I_{w_i}$of the keyword w_i

Note that the size of the posting list varies from one keyword to another one depending on the number of files where this keyword occurs. The definition of the relevance score sc is given in (10) as follows:

$$sc(W, F_{d_j}) = \sum\nolimits_{w_i \in W} \frac{1}{\left|F_{d_j}\right|} \cdot \left(1 + ln f_{d_j,w_i}\right) . ln\left(1 + \frac{N}{f_{w_i}}\right) \qquad (10)$$

W denotes the set of the keywords in each index.

f_{d_j,w_i} is the term frequency (TF) of the term w_i in the file identified by F_{d_j}.

$\left|F_{d_j}\right|$ is the length of file having the identifier F_{d_j}. It is obtained by counting the number of indexed terms.

f_{w_i} is the number of files that contain the term w_i.

N is the total number of files in the collection.

In our algorithm, we consider the case of a single keyword search. In this case, (11) could be used instead of (10) for the ranking purpose:

$$score(w_i, F_{d_j}) = \frac{1}{\left|F_{d_j}\right|} \cdot \left(1 + ln f_{d_j,w_i}\right) \qquad (11)$$

As directly outsourcing relevance scores (without encryption) will leak sensitive frequency information, thereby weakening the keyword privacy, order preserving encryption (OPE) is used to protect sensitive weight information (scores).

4- The locality sensitive hashing function (minhash), which is effective for the Jaccard measure, is applied on each keyword.

5- The user sends the minhash value(s) $lsh\,(w_i)$ and the posting list I_{w_i} to the cloud where they are added to a hash map. Note that the AND-OR construction is applied on the locality sensitive hashing method. Then, we have $k \times \lambda$ hash values that will be used as pointers for the posting list of the corresponding keyword as follows:

Each array of k hashes will be inserted in a hash map which is a variant of the Bloom filter with storage. Each of the λ arrays of size k (k hashes) can then point to this posting list. This is how the AND-OR construction is performed. The same approach is also used in the retrieval process below.

B. Retrieving a message x

During this process, the client mobile phone needs to perform the fuzzy transformation on the keyword and query the cloud with it. In its turn, the cloud uses this index (hashed keyword) to find the corresponding posting (encrypted) list and then retrieve the most relevant files for this keyword/query.

We explain below the algorithm in steps:

1. Retrieve the items that contain a keyword w_i. The user applies the amplified locality sensitive hashing on this keyword in order to construct the query of $kx\lambda$ hash values. Then the cloud uses this array of hashes as follows:

Each k hashes are used to find the corresponding posting list(s) in the hash map. This process is done for the λ sets. Then, the most frequently found posting list is considered as the most similar to the queried keyword and will be put in the first rank. The ranking is continued in descending order of the frequency of occurrence of retrieved posting lists until completed.

2. The cloud uses the retrieved and ranked posting lists in order to find the required file IDs corresponding to the query starting with the posting list with the highest rank.

3. The cloud returns the requested number of matched files in a ranked order based on the (encrypted) keyword scores in the files of each posting list i.e. the desired encrypted data items are returned to the user starting with the most relevant one (starting with the file having the highest score to the file having the lowest score). This comparison between the encrypted scores is possible thanks to the property of the OPSE encryption which conserves the numerical order of the scores' values. The files of the posting list of higher rank are first returned to the user device in order. Then, following the number of required files, the posting lists having lower ranks are used to return more files to the user device.

4. Once the encrypted items corresponding to the search request are retrieved, the user device decrypts them to obtain the plain versions of the requested files.

VI. EXPERIMENTAL RESULTS AND SECURITY ANALYSIS

The system architecture consists of three components that are implemented as software modules: The Client PC, Cloud Manager and Smartphone App. The Client PC software sends encrypted files and keyword indices to the cloud. The Cloud manager stores the encrypted files. The Smartphone App searches for files from the cloud. While a file can have many keywords as an index, the Smartphone will only be able to search with one keyword each time. It is assumed that there are secure communications between the different components of the system.

The hardware required for the implementation comprises:

- A PC acting as a cloud. The used PC is a Dell XPS 8500, Intel(R) Core (TM) i7-3770 CPU, 3.40 GHzx4, memory12 GB.

- A PC representing the Client PC in the storage process. The used PC is a Dell, Intel Core i7-3770 CPU, 3.40 GHzx8, memory 11.8 GB.

- An Android phone representing the Smartphone client in the search process. The used phone is a Samsung Galaxy S3.

The test is performed over WiFi (IEEE 802.11g) network. The client PC Software and the Cloud Manager are implemented in Java. OPSE is C based and the Smartphone app is Android based. All the tests are performed on selected files from the Request For Comments database (RFC) [30].

A. Choice of the LSH

In this section, a comparison between a number of locality sensitive hashing methods with, and without, the amplification method i.e. the AND-OR construction is conducted in order to choose the best method to be used in our storage and search scheme.

To perform this test, we applied the locality sensitive hashing methods on 1000 selected keywords from the RFC database. In order to calculate the failure rate, we inserted random misspelling errors on these keywords (1 error/keyword) after which we applied the locality sensitive hashing methods and compared the resulted hashes.
The parameters of the AND-OR construction are $k = 3$, $\lambda = 37$. The secret parameter for the chaotic minhash is m=70, the number of chaotic iterations performed on each shingle of the keyword is equal to the length of the keyword and the chaotic control parameter is p=0.3.

It is assumed that failures are the number of times the locality sensitive hashing method does not lead to a similarity between the original and the misspelt word (erroneous word). Table I shows the failure ratio of the following locality sensitive hashing methods: Kuzu minhash, Grp minhash, and Omflip minhash and their amplified versions.

TABLE I
FAILURE RATE FOR LSH METHODS WITH AND WITHOUT AMPLIFICATION

	Kuzu minhash	Amplified Kuzu minhash	Omflip minhash	Amplified Omflip minhash	Grp minhash	Amplified Grp minhash
Failure (‰)	570	250	534	365	540	385

As we can see, the failure is more than 500‰ for the locality sensitive hashing methods Kuzu, Omflip and Grp without amplification. Omflip minhash gives slightly better results (534‰) comparing to Grp minhash (540‰) and Kuzu minhash (570 ‰) which gives the biggest failure in this case. However, applying the AND-OR construction i.e. the amplification method with ($k = 3$ and $\lambda = 37$) reduces the failure rate by around 30% (for Omflip and Grp lsh methods) and better results are obtained for the amplified Kuzu method where the failure rate is reduced by more than 56% comparing to Kuzu LSH without amplification., Amplified Kuzu is the best compared to the Amplified Omflip minhash and Grp minhashes as it has the lowest failure rate between the three amplified minhashes. The failure rate can be obviously reduced further by increasing the number of the generated minhashes for each keyword i.e. by increasing the AND-OR parameters' values. However, increasing these parameters will add more complexity and require more computation time to calculate the LSH of each keyword which will respectively affect the efficiency of the whole storage and search scheme. The effect of each of the amplification parameters is shown later on in the paper.

Table II shows the failure ratio of the chaotic locality sensitive hashing methods: chaotic Kuzu and Omflip minhashes and their amplified versions.

TABLE II
FAILURE RATE FOR CHAOTIC LSH METHODS WITH AND WITHOUT AMPLIFICATION

	Chaotic Kuzu minhash	Amplified Chaotic Kuzu minhash	Chaotic Omflip minhash	Amplified Chaotic Omflip minhash
Failure (‰)	139	0	117	18

The chaotic locality sensitive hashing methods shown in Table II give better failure rates than the values shown in Table I. As we can see, the introduction of the chaos reduced the failure rate for Kuzu LSH (75%) and Omflip LSH (78%) compared with the original Kuzu and Omflip LSH methods and also gives better results than the amplification of the original LSH methods. However, the amplification of the chaotic methods is also more effective e.g. the failure rate is reduced from 25% to 0% by introducing chaos into the amplified Kuzu minhash.

The following tests are performed on the storage and search schemes using the amplified chaotic Kuzu locality sensitive hashing method as a fuzzy transformation and the same chaotic parameters values of this section are used.

B. Time of index construction

To allow the fuzzy ranked search, an index object (or posting list) is created for each indexed keyword on the client PC. This posting list contains the hashed keyword, the file IDs and the encrypted scores (encrypted with OPSE).
We built the indexes using 100 indexed keywords and 100 selected files from the RFCs. The used AND-OR construction parameters are $k = 3$ and $\lambda = 37$.

1) Effect of the indexed and stored files

Table III shows the effect of the indexed and stored files number on the index construction time. As we can see, the average index construction time increases with the number of stored files as the posting list size increases. The same keyword can be used, in this case, to index more files if this keyword occurs inside these files. The indexing and the score calculation are file content based.

TABLE III
FILES' NUMBER EFFECTS ON THE INDEX CONSTRUCTION TIME

Number of files / Time (ms)	20	100	300	500	800	1000	3000
Average Index construction time	56	182	484	786	1443	2073	11769

Average OPE time	50	163	429	697	1256	1781	9472
Rank calculation time	5	18	54	88	185	290	2295

As can be seen also, the OPSE requires the majority of the time of the index construction. It is argued here that it is implemented in a different language, C, that is called as an executable by the Java program leading to an increase in the processing time. A Java implementation of the OPSE may enhance the speed of the indexing construction process. The average time to perform the minhash on a keyword is 10 ms. This value is considered in the index construction time measurement but it is independent from the number of performed files and depends on the size of the keyword itself.

Table IV shows how the index construction time increases when increasing the number of indexing keywords.

TABLE IV
KEYWORDS' NUMBER EFFECTS ON THE INDEX CONSTRUCTION TIME

Number of keywords / Time	10	100	300	500	800	1000
Index construction time	5 sec 755 ms	18 sec 223 ms	45 sec 291 ms	1 min 6 sec 468 ms	1min 16 sec 56 ms	1 min 21 sec 521 ms
OPE time	5 sec 196 ms	16 sec 344 ms	39 sec 485 ms	56sec 981 ms	1 min 3 sec 633 ms	1 min 6 sec 395 ms
Rank calculation time	494 ms	1 sec 787 ms	5 sec 663 ms	9 sec 298 ms	12 sec 180 ms	14 sec 855 ms
Minhash time	23 ms	44 ms	85 ms	121 ms	171 ms	195 ms

2) The effect of the LSH parameters

In this section, we measure the effect of the AND-OR construction parameters on the index construction time for 100 keywords used to index 100 files, which is shown in Table V and VI.

The effect of k, λ parameters is not significant on the index construction time. Actually, the AND-OR construction parameters affect only the locality sensitive hashing time which is relatively fast. The Rank calculation and the OPSE time depend on the number of files and keywords and are independent of the number of the hashes for each keyword.

TABLE V
EFFECTS OF K ON THE INDEX CONSTRUCTION TIME WHEN $\lambda = 37$

k / Time	1	3	7	9
Index construction time	18 sec 391 ms	18 sec 471 ms	18 sec 512 ms	18 sec 675 ms
OPE time	16 sec 531 ms	16 sec 596 ms	16 sec 620 ms	16 sec 791ms
Rank calculation time	1 sec 789 ms	1 sec 781 ms	1 sec 779 ms	1 sec 765 ms
Minhash time	31 ms	44 ms	68 ms	77 ms

TABLE VI
EFFECT OF λ ON THE INDEX CONSTRUCTION TIME WHEN $k = 3$

λ / Time	15	26	37	60
Index construction time	17 sec 443 ms	18 sec 189 ms	18 sec 471 ms	18 sec 531ms
OPE time	15 sec 590 ms	16 sec 338 ms	16 sec 596 ms	16 sec 630 ms
Rank calculation time	1 sec 780 ms	1 sec 776 ms	1 sec 781 ms	1 sec 806 ms
Minhash time	34 ms	38 ms	44 ms	58 ms

C. Search time

The search time means the average time between the search request and the identification of the files' identifiers.
This time can be split into three distinct slots:

The first slot T_1 is the time for processing the minhash over the phone. The second slot T_2 is the processing and search time on the Cloud Manager side. Finally, the third slot T_3 is the phone round trip time which includes the communication time between the phone and the cloud and the displaying of the file IDs on the phone.

To perform this measurement, 100 text files from the RFC indexed by 100 keywords are used. We show in Table VII, the value of each time slot T_j, $j = 1, 2, 3$ for 100 queries. The AND-OR construction parameters are $k = 3$ and $\lambda = 37$.

TABLE VII
SEARCH TIME FOR 100 QUERIES

	Minhash Time T_1	Cloud Manager processing Time T_2	Communication Time T_3
Time (ms)	792	60	306

The average search time for one query is shown in Table VIII.

TABLE VIII
SEARCH TIME FOR 1 KEYWORD

	Minhash Time T_1	Cloud Manager processing Time T_2	Communication Time T_3
Time (ms)	10	2	59

Notice that in Table VII, we presented the time when the cloud is performing the search for 100 queries and returning the corresponding file IDs in one round trip which reduces the latency of the wireless transmission and the networking delays.

1) The effects of the number of stored data items

In this test, we show the effects of the number of stored data items on the search time. The number of indexed keywords is 100. Table IX presents the changes in the processing and search time T_2 on the cloud manager side and the phone round trip search time T_3. In this test, $k = 3$, $\lambda = 37$ and the number of stored keywords are 100.

As we can see, the search time increases linearly with the number of stored data because the size of the posting list becomes bigger when increasing the number of stored files. In this case, the cloud manager needs to search and rank more files for each keyword. The number of relevant file IDs also increases which affects also the communication time.

TABLE IX
EFFECT OF THE NUMBER OF STORED FILES

Number of files / Time (ms)	20	100	200	500	800	1000	2000	3000
Time on the cloud/keyword	1	2	3	13	22	28	58	146
Communication Time/keyword	51	59	76	120	135	168	291	431

The search performance does not depend on the number of keywords. In fact, the cloud manager does not have to traverse every posting list in the stored indices, but instead uses a hash map to fetch the corresponding posting list which makes the search independent of the number of stored posting lists or indexed keywords.

2) The effect of the LSH parameters

Fig. 6 and 7 respectively show how the choice of k and λ affects the search time for 100 keywords used to index 100 files.

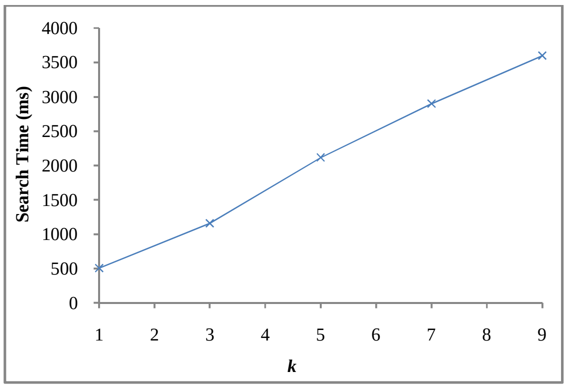

Fig. 6. Search time (ms) over k for $\lambda = 37$

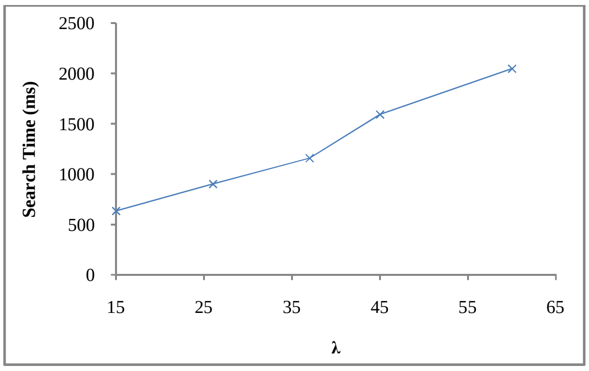

Fig. 7. Search time (ms) over λ for $k = 3$

In this test, we assumed that the user is looking only for the files indexed by the keyword most similar to the query. In this case, the search time is increasing with k and λ. Otherwise, if the user is requesting more files, the keywords with less similarity to the query will be considered. In this case, the search time will increase when increasing λ, because more similar items are retrieved, and decrease when increasing k because fewer similar items will be found and returned to the user device.

D. Precision/recall

Precision and recall metrics are used to evaluate the retrieval performance of our algorithm [15], [16].
To do the evaluation, 1000 random keywords are selected from the database and used to index the files. Then, we inserted spelling errors in 25% of these keywords and queried with the new set of keywords.

If N_{f_w} is the number of positive files in the cloud, R_{f_q} is the number of retrieved files for a query q and R_{f_w} is the number of positive retrieved files i.e. the number of retrieved files for the original keyword w.

Then, the precision $\mathrm{prec(q)}$ and recall $\mathrm{rec(q)}$ of a query q and the averages $\mathrm{avprec(Q)}$ and $\mathrm{avrec(Q)}$ for the query set $Q = \{q_1, q_2, \dots, q_n\}$ are described in (12), (13), (14) and (15) as follows:

$$\mathrm{prec(q)} = \frac{R_{fw}}{R_{fq}} \tag{12}$$

$$\mathrm{avprec(Q)} = \sum_{\mathrm{i=1}}^{\mathrm{n}} \frac{\mathrm{prec(q_i)}}{\mathrm{n}} \tag{13}$$

$$\mathrm{rec(q)} = \frac{R_{fw}}{N_{fw}} \tag{14}$$

$$\mathrm{avrec(Q)} = \sum_{\mathrm{i=1}}^{\mathrm{n}} \frac{\mathrm{rec(q_i)}}{\mathrm{n}} \tag{15}$$

In this section, we calculate the precision and recall depending on the number of files t required by the user. Once the query is issued, if many posting lists are retrieved, these posting lists are firstly ranked according to their similarity with the query. The exact match is ranked first then the similar positing lists are ranked respectively. The retrieved documents identifiers are ranked according to their scores. Finally, if number of requested files t is smaller than the number of retrieved files, just the top t files are returned to the client (and considered in the calculation as the number of retrieved files) otherwise all the retrieved files (by the cloud provider) are returned to the user. Average precision and recall for 1000 queries when changing t are shown in Table X.

TABLE X
PRECISION AND RECALL FOR $k = 3$ AND $\lambda = 37$ WHEN THE NUMBER OF REQUESTED FILES t CHANGES

t	1	5	10	20	50	100	500	1000
avprec(Q)	0.87	0.83	0.80	0.77	0.74	0.72	0.69	0.69
avrec(Q)	0.12	0.31	0.41	0.52	0.68	0.78	0.87	0.87

As we can see, the recall is increasing when the number, t, of required files increases. The number of positive files in the cloud does not change when t changes then the denominator in the recall equation will remain the same. However, the number of positive retrieved files (which the user receives) increases when the number of required files t is increasing. For this reason, the average recall curve is increasing with t. When the number of required files t is equal to the number of positive files in the cloud, the average recall becomes constant as we can see in Table X.

For the precision, the number of retrieved files is increasing with t. When t is small the method is most likely finding the exact match to the query and all the retrieved files are positive thus the precision is high. When t increases the method is either (a) retrieving other files (in addition to the files with exact match to the query) that contain similar keywords to the query and then the number of retrieved files is bigger but the number of positive retrieved files (with exact match to the query) remains the same thus the precision is smaller or (b) the method is not retrieving any further files and then the precision is constant which explains the behavior of the precision curve.

In the following subsection, we study the effect of the AND-OR construction parameters, the chaos parameters and the error types on the precision and recall.

1) Effect of the AND-OR parameters

To perform this test, we first fix the AND construction parameter k and change the OR-construction parameter λ, then we fix λ and we change k. In Table XI, we show the effect of the OR construction parameter λ on the precision when the number of requested files t changes. We choose $k = 3$ and we show the precision for λ=15, 37 and 60. As it can be seen, the precision decreases slightly when λ increases.

The recall does not change because the algorithm is still able to retrieve the positive keyword posting list and then the positive files. The number of retrieved posting lists (and respectively files) increases when λ increases but this list always contains the correct posting list (and respectively positive files) for $k = 3$ and the corresponding positive files are returned first to the user.

TABLE XI
PRECISION WHEN $k = 3$ AND WHEN THE NUMBER OF REQUESTED FILES t AND THE OR CONSTRUCTION PARAMETER λ CHANGE

t	1	5	10	20	50	100	500	1000
$\lambda = 15$	0.87	0.85	0.83	0.81	0.79	0.78	0.77	0.77
$\lambda = 37$	0.87	0.83	0.80	0.77	0.74	0.72	0.69	0.69
$\lambda = 60$	0.87	0.83	0.79	0.75	0.71	0.67	0.64	0.64

The number of retrieved files affects just the precision and not the recall as long as the positive files are still retrieved.

In Table XII and XIII, we show the effect of the AND-construction parameter k respectively on the precision and recall. We choose $\lambda = 37$ and we show the retrieval ratio for k=1, 2 and 3.

TABLE XII
PRECISION WHEN $\lambda = 37$ AND WHEN THE NUMBER OF REQUESTED FILES T THE AND-CONSTRUCTION PARAMETER K CHANGE

t	1	5	10	20	50	100	500	1000
$k = 3$	0.87	0.83	0.80	0.77	0.74	0.72	0.69	0.69
$k = 2$	0.87	0.79	0.73	0.66	0.56	0.48	0.41	0.40
$k = 1$	0.37	0.31	0.28	0.24	0.21	0.19	0.18	0.18

TABLE XIII
RECALL WHEN $\lambda = 37$ AND WHEN THE NUMBER OF REQUESTED FILES T THE AND-CONSTRUCTION PARAMETER K CHANGE

t	1	5	10	20	50	100	500	1000
$k = 3$	0.12	0.31	0.41	0.52	0.68	0.78	0.87	0.87
$k = 2$	0.12	0.31	0.41	0.52	0.68	0.78	0.87	0.87
$k = 1$	0.07	0.16	0.20	0.25	0.30	0.33	0.36	0.36

Changing k has more effects on the precision values i.e. the precision increases with k. When k is higher, the retrieval method is better able to find the exact match for the query i.e. to find just the correct files without more items in addition. Then, the precision is higher. The recall is low for $k = 1$ and it is almost the same for $k > 2$.

2) Effect of chaos parameter

In this section, we show how the precision and recall change with the chaos parameter p (Fig. 1. The precision slightly increases when p decreases, as we can see in Table XIV. However, the obtained recall values are the same when p changes.

TABLE XIV
PRECISION FOR $k = 3$ AND $\lambda = 37$ AND WHEN THE CHAOTIC PARAMETER p AND THE NUMBER OF REQUESTED FILES t CHANGE

t	1	5	10	20	50	100	500	1000
$p = 0.4$	0.87	0.83	0.80	0.77	0.72	0.70	0.67	0.67
$p = 0.3$	0.87	0.83	0.80	0.77	0.74	0.72	0.69	0.69
$p = 0.1$	0.87	0.85	0.83	0.80	0.78	0.77	0.76	0.76

3) Effect of the error type

Tables XV and XVI show respectively how the precision and recall values change with the type of error that occurs in the query. To do the evaluation, 100 erroneous queries are used to search over the database containing 100 files.

The type of error does not have a big effect on the precision/recall. As we can see in Tables XV and XVI, we obtain very close precision and recall values with all types of error with a maximum difference of 0.04 for the precision and 0.01 for the recall.

TABLE XV
PRECISION VALUES, WHEN THE NUMBER OF REQUESTED FILES t CHANGES, WITH THE TYPE OF ERROR OCCURRED IN THE SEARCH QUERY

t	1	5	10	20	50	100
Deletions	0.98	0.96	0.94	0.92	0.91	0.91
Permutations	0.99	0.95	0.92	0.91	0.90	0.90
Insertions	0.99	0.95	0.92	0.90	0.89	0.89
Substitutions	0.99	0.95	0.92	0.89	0.87	0.87

TABLE XVI
RECALL VALUES, WHEN THE NUMBER OF REQUESTED FILES t CHANGES, WITH THE TYPE OF ERROR OCCURRED IN THE SEARCH QUERY

t	1	5	10	20	50	100
Deletions	0.24	0.60	0.76	0.89	0.95	0.98
Permutations	0.24	0.61	0.77	0.9	0.96	0.99
Insertions	0.24	0.61	0.77	0.9	0.96	0.99
Substitutions	0.24	0.61	0.77	0.9	0.96	0.99

4) RFC vs Enron database

Tables XVII and XVIII show the precision and recall values for the RFC and Enron databases. For each database, 100 keywords are selected to index 100 files then 100 erroneous keywords are used as queries for each dataset to perform test.

As we can see, similar curves are obtained for both databases.

TABLE XVII
PRECISION VALUES , WHEN THE NUMBER OF REQUESTED FILES t CHANGES FOR RFC AND ENRON DATABASES

t	1	5	10	20	50	100
RFC	0.99	0.95	0.94	0.92	0.90	0.90
Enron	0.99	0.96	0.95	0.94	0.93	0.93

TABLE XVIII
RECALL VALUES , WHEN THE NUMBER OF REQUESTED FILES t CHANGES FOR RFC AND ENRON DATABASES

t	1	5	10	20	50	100
RFC	0.24	0.61	0.77	0.9	0.96	0.99
Enron	0.27	0.59	0.74	0.84	0.93	0.99

E. Average retrieval ratio

Let w be the keyword used to index a document and q be the query. If $\mathrm{W_d}$ is the number of keywords within a distance d from a query q and $\mathrm{R_{F_{W_d}}(q)}$ is the number of files indexed by these keywords i.e. the number of retrieved files within a distance d from the query q, which is a subset of $\mathrm{N_{F_{W_d}}(q)}$, the number of files within a distance d in the database, then the retrieved ratio $\mathrm{rrd(q_i)}$ of a query q_i and the average retrieved ratio $\mathrm{arrd(Q)}$of the query set $Q = \{q_1, q_2, \dots, q_n\}$ are described in (16) and (17) as follows [15], [16]:

$$\mathrm{rrd}(q_i) = \frac{R_{F_{W_d}}(q_i)}{N_{F_{W_d}}(q_i)}, \, i = 1 \dots., n \tag{16}$$

$$\mathrm{arrd(Q)} = \frac{\sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{rrd}(q_i)}{\mathrm{n}} \tag{17}$$

where n is the number of queries.

The distance between a query and a file is the Jaccard distance between the encoding of the query and the encoding of the keyword that led to this file. As with the precision and recall tests, 1000 random keywords are selected from the database and used to index the files. When querying, 25% of these keywords have spelling errors.

In the following subsection, we study the effect of the AND-OR construction parameters, the chaos parameters and the error's types on the retrieval ratio.

1) Effects of the AND-OR parameters

To perform this test, we first fix the AND construction parameter k and change the OR-construction parameter λ then we fix λ and change k.

In Table XIX, we show the effects of the OR construction parameter λ on the retrieval ratio. We choose $k = 3$ and we show the retrieval ratio for λ=15, 37 and 60.

TABLE XIX
RETRIEVAL RATIO , WHEN THE DISTANCE D CHANGES, FOR $k = 3$ AND WHEN λ CHANGES

d	0	0.2	0.3	0.4	0.5	0.7	1
$\lambda = 60$	1	0.98	0.88	0.68	0.45	0.29	0.27
$\lambda = 37$	1	0.98	0.85	0.61	0.39	0.25	0.23
$\lambda = 15$	1	0.97	0.83	0.57	0.35	0.21	0.20

As we can see, the retrieval ratio is higher when λ increases. By increasing this value, more items are found and then more items within a given distance d are retrieved.

In Table XX , we show the effect of the AND-construction parameter k on the retrieval ratio. We choose $\lambda = 37$ and we show the retrieval ratio for k=2, 3 and 5.

TABLE XX
RETRIEVAL RATIO , WHEN THE DISTANCE D CHANGES, FOR $\lambda = 37$ AND WHEN k CHANGES

d	0	0.2	0.3	0.4	0.5	0.7	1
$k = 2$	1	0.99	0.93	0.76	0.55	0.36	0.33
$k = 3$	1	0.98	0.85	0.61	0.39	0.25	0.23
$k = 5$	1	0.90	0.66	0.30	0.09	0.01	0.01

As we can see, the retrieval value decreases when k increases. Actually, the algorithm is more selective, thus the precision is higher but the retrieval ratio is smaller.

2) Effect of the chaos parameter

The use of chaos in cryptography has attracted many researchers due to its randomness and high sensitivity to its control parameters. Even if the chaotic values change when modifying the control parameter p, this change doesn't significantly affect the overall behaviour of the system vis-à-vis the retrieval ratio as we can see in Table XXI. We obtain very close results for p=0.1,0.3 and 0.4 with a maximum difference of 0.05 between the obtained retrieved ratio values.

TABLE XXI
RETRIEVAL RATIO, WHEN THE DISTANCE D CHANGES, FOR DIFFERENT VALUES FOR THE CHAOTIC PARAMETER p

d	0	0.2	0.3	0.4	0.5	0.7	1
$p = 0.4$	1	0.99	0.88	0.66	0.43	0.26	0.24
$p = 0.3$	1	0.98	0.85	0.61	0.39	0.25	0.23
$p = 0.1$	1	0.98	0.85	0.61	0.38	0.22	0.20

3) Effect of the errors' types

In the previous tests, random errors are inserted in the query list of keywords. In this section, we test how the retrieval ratio changes with each type of error i.e. deletion of a letter of the keyword, insertion or substitution of a letter or permutation of two adjacent letters.

In this test, 100 erroneous keywords are used to query the database. As we can see in Table XXII, the retrieval ratio values are very close for all types of error with a maximum of 0.04 difference between them. Consequently, the test with a random error in the query is realistic.

TABLE XXII
RETRIEVAL RATIO, WHEN THE DISTANCE D CHANGES, FOR DIFFERENT TYPE OF ERRORS IN THE QUERY

d	0	0.2	0.3	0.4	0.5	0.7	1
Deletions	1	0.94	0.79	0.56	0.36	0.20	0.19
Insertions	1	0.96	0.81	0.55	0.34	0.19	0.19
Permutations	1	0.94	0.79	0.59	0.34	0.19	0.18
Substitutions	1	0.94	0.82	0.60	0.34	0.20	0.19

4) Enron database

The same retrieval ratio test is performed on Enron database and then compared with the RFC database in Table XXIII. The retrieval ratio values when the distance d increases between the query and the stored keyword are shown.

TABLE XXIII
RETRIEVAL RATIO, WHEN THE DISTANCE d CHANGES FOR RFC AND ENRON DATABASES

d	0	0.2	0.3	0.4	0.5	0.7	1
Enron	0.97	0.92	0.78	0.54	0.31	0.23	0.23
RFC	1	0.98	0.85	0.61	0.39	0.25	0.23

As we can see in Table XXIII , similar results are obtained for both databases RFC and Enron.

F. Security analysis

In this section, we analyse the security of the proposed algorithm. As assumed in all existing searchable encryption methods, we consider a semi-trusted server i.e. “honest-but-curious” server [1], [8]-[12], [15]-[17]. The security guarantee in this case is to prevent the cloud server from learning the plaintext of either the data files or the searched keywords, and achieve the “as strong-as-possible” security strength compared to existing searchable encryption schemes. In our scheme, the cloud provider can only see the encrypted files and indexes. The file content is clearly well protected due to the security strength of the file encryption scheme. Thus, we only need to focus on keyword privacy. In the following, we analyse the security of both fuzzy transformation and ranking supported by our algorithm.

- Concerning the fuzzy transformation: The security requirement is typically characterized that nothing should be leaked except the result of a search, which is referred to as an access pattern or similarity pattern in our case because we are developing a similarity searchable encryption scheme and not a standard one. Thus, similar to [15], to achieve this, we perform multiple LSHs (Amplified LSH) on each keyword to construct the query. Thus, the number of common components between distinct queries may leak relative similarity between them. And, an adversary might infer some information about the similarity of the indexes. In this case, the similarity pattern is leaked instead of the search pattern which is acceptable in the case of similarity matching. We conclude that our fuzzy searchable encryption scheme does not leak any information beyond the trace, which is the maximum amount of information the content owner is willing to leak. In addition, the usage of chaos based LSH increases the computational indistinguishability from a totally random permutation based transformation which results in queries computationally indistinguishable from random values and thus the fuzzy keyword search scheme is secure regarding the search privacy.

- Concerning the ranking scheme: To ensure the security guarantee, the cloud server should learn very little about the relevance criteria as they exhibit significant sensitive information against keyword privacy. Similar to [11], the proposed ranking scheme uses a posting list that embeds the encrypted relevance scores in addition to file ID containing this keyword. These scores are encrypted using order preserving encryption (OPSE) which gives only a sequence of order-preserved numeric values. Though adversary may learn partial information from the duplicates (e.g., ciphertext scores duplicates may indicate very high corresponding plaintext scores duplicates), OPSE makes it difficult for the adversary to predict the original plaintext score. Thus, the keyword privacy is also well preserved in our scheme.

VII. CONCLUSION

In this paper, we proposed the first chaos based searchable encryption approach which also allows both ranked and fuzzy keyword searches on the encrypted data stored in the cloud. Our approach guarantees the privacy and confidentiality of the user even *vis*-à-*vis* the cloud provider who is semi-trusted in our case. The proposed method is designed to achieve effective retrieval of remotely stored encrypted data for mobile cloud computing scenarios. This scheme is implemented and evaluated using two databases: RFCs and the Enron database. Comprehensive tests have been performed to prove the efficiency of our proposition. First, the chaotic locality sensitive hashing method with 0‰ failure is selected. Then, effects of different parameters of the amplification method (AND-OR construction) and the chaos, on the efficiency of the algorithm, are shown when different numbers of files are requested. The algorithm is also tested when different kind of errors (deletions, insertions, permutations and substitutions) occur in the query and similar precision, recall and retrieved ratio curves are obtained. Our proposed algorithm supports the search with only one keyword and an extension of the proposed algorithm to enable conjunctive and disjunctive multi-keywords search, will be considered in the future work.

REFERENCES

[1] B.Yang, X. Pang, Q. Du, and Dan Xie, "Effective Error-Tolerant Keyword Search for Secure Cloud Computing," *Journal of computer science and technology*, vol. 29, no.1, pp. 81-89, Jan. 2014.

[2] D. Boneh, G. D. Crescenzo, "Public key encryption with keyword search," in *C. Cachin and J. Camenisch, editors, Advances in Cryptology, Eurocrypt*, vol. 3027 of LNCS, pp. 506–522, Springer, 2004.

[3] S. Kamara, K. Lauter, "Cryptographic cloud storage, " in *Financial Cryptography and Data Security*, pp. 136-149, Springer Berlin Heidelberg, 2010.

[4] S. Kamara, C. Papamanthou, T. Roeder, "CS2: A searchable cryptographic cloud storage system," Microsoft Research, Tech. Report MSR-TR, 2011.

[5] Y. Earn, R. Alsaqour, M. Abdelhaq, T. Abdullah, "Searchable symmetric encryption: review and evaluation," *Journal of Theoretical and Applied Information Technology*, vol. 30, 2011.

[6] R. Koletka, A. Hutchison, "An architecture for secure searchable cloud storage," *IEEE, Information Security South Africa (ISSA)*, pp. 15-17, Aug., 2011.

[7] E. Stefanov, C. Papamanthou, E. Shi, "Practical Dynamic Searchable Encryption with Small Leakage," *IACR Cryptology ePrint Archive*, 2013.

[8] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," *INFOCOM, 2010 Proceedings IEEE, Dept. of ECE, Illinois Inst. of Technol.*, Chicago, IL, USA , Mar. 2010.

[9] J. Bringer, H. Chabanne, B. Kindarji, "Error-tolerant searchable encryption," *Communication and Information Systems Security Symposium, International Conference on Communications (ICC)*, Dresden, Germany, pp. 14-18, Jun. 2009.

[10] J. Yu, J. Li, X. Wang, W. Gao, "Conjunctive Fuzzy Keyword Search Over Encrypted Data in Cloud Computing," *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol.12, no.3, pp. 2104-2109, Mar. 2014.

[11] C. Wang, N. Cao, J. Li, K. Ren, W. Lou, "Secure ranked keyword search over encrypted cloud data," *ICDCS '10 Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems, IEEE Computer Society Washington, DC, USA*, pp. 253-262, 2010.

[12] R. Li, Z. Xu, W. Kang, K. Choong Yow, C. Z. Xu, "Efficient multi-keyword ranked query over encrypted data in cloud computing," *Elsevier, Future Generation Computer Systems*, vol. 30, pp. 179–190, 2014.

[13] J. Bringer, H. Chabanne, B. Kindarji, "Identification with encrypted biometric data, "*Security and Communication Networks*, vol. 4, no. 5, pp. 548–562, May 2011.

[14] J. Bringer, H. Chabanne, "Embedding edit distance to enable private keyword search," *Secure and Trust Computing, Data Management and Applications, Communications in Computer and Information Science*, vol. 186, no. 1, pp. 105-113, 2011.

[15] M. kuzu, M. S. Islm, M. Kantarcioglu, "Efficient similarity search over encrypted data," *ICDE's12 proceedings of the 2012 IEEE 28th International conference on data engineering*, pp. 1156-1167, IEEE computer society Washington, DC, USA, 2012.

[16] W. Lu, A. Swaminathan, A. L. Varna, and M. Wu, "Enabling Search over Encrypted Multimedia Databases," *In IS&T/SPIE Electronic Imaging, International Society for Optics and Photonics*, pp. 725418-725418, 2009.

[17] A. Awad, A. Matthews, and B. Lee, "Secure cloud storage and search scheme for mobile devices," in *the 17th*

IEEE Mediterranean Electrotechnical Conference (MELECON), pp. 144-150, Apr. 2014.

[18] A. Awad, A. Saadane, "New Chaotic Permutation Methods for Image Encryption", *IAENG International Journal of Computer Science*, vol. 37, no. 4, pp. 402-410, 2010.

[19] A. Awad, and A. Saadane, "Efficient chaotic permutations for image encryption algorithms," in *Proceedings of the World Congress on Engineering*, vol. 1, 2010.

[20] A. Awad, and D. Awad, "Efficient image chaotic encryption algorithm with no propagation error," *ETRI journal*, vol. 32, no. 5, pp. 774-783, 2010.

[21] A. Awad, and A. Miri, "A new image encryption algorithm based on a chaotic DNA substitution method," in *Communications (ICC), 2012 IEEE International Conference on*, pp. 1011-1015, 2012.

[22] M. Ismail, G. Chalhoub, and B. Bakhache, "Evaluation of a fast symmetric cryptographic algorithm based on the chaos theory for wireless sensor networks," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, pp. 913-919, 2012.

[23] R. Rostom, B. Bakhache, H. Salami, and A. Awad, "Quantum cryptography and chaos for the transmission of security keys in 802.11 networks," in the *17th IEEE Mediterranean Electrotechnical Conference (MELECON)*, pp. 350-356, Apr. 2014.

[24] A. Andoni, P. Indyk, "Near optimal hashing algorithms for approximate nearest neighbor in high dimensions, "*Communications of the ACM - 50th anniversary issue: 1958 - 2008, ACM New York, NY, USA*, vol. 51, no. 1, pp. 117-122, Jan. 2008 .

[25] A. Z. Broder, "On the resemblance and containment of documents," in *proceedings of Compression and Complexity of Sequences*, pp. 21-29, 1997.

[26] A. Z. Broder, M. Chaikar, A. M. Frieze, M. Mitzenmacher, "Min wise independent permutations," *Journal of Computer and System Sciences*, vol. 60, no. 3, pp. 630-659, 2000.

[27] B. Julien, H. Chabanne, and B. Kindarji, "Identification with encrypted biometric data," *Security and Communication Networks*, vol. 4, no. 5, pp. 548-562, 2011.

[28] A. Boldyreva, N. Chenette, Y. Lee, A. O'Neill, "Order preserving symmetric encryption," in *Proceedings of Eurocrypt*, vol. 5479 of LNCS, Springer, 2009.

[29] A. Awad, et al. "Comparative study of 1-D chaotic generators for digital data encryption." *IAENG International Journal of Computer Science* 35, no. 4, pp. 483-488, 2008.

[30] RFC, "Request for comments database," http://www.ietf.org/rfc.html.

[31] "Enron email dataset," http://www.cs.cmu.edu/enron, 2011.

Dr. Abir Awad received her PhD degree from Polytech' Nantes (France) in 2009. Later, she joined the Operational Cryptology and Virology Laboratory, ESIEA and Le Mans University in Laval in 2010 and the computer science lab. (LIFO), University of Orleans in 2011 as a lecturer-researcher. In 2012, she worked at the computer science dep., Ryerson University, Toronto (Canada). She is currently working as a researcher within the Irish Centre of Cloud Computing and Commerce (IC4) in Ireland. Her research interests include cloud security, searchable encryption, provenance, security analytics, trusted computing and chaotic cryptography.

Dr. Brian Lee is the Director of the Software Research Institute and holds a PhD from Trinity College Dublin the application of programmable networking for network management. He has over 20 years experience in fixed and mobile network management and has extensive experience of systems and software design and development for large telecommunications products He was previously director of research for LM Ericsson Ireland with responsibility for overseeing all research activities. His research interests include programmable networking for network management and distributed data analytics for infrastructure management.

Dr. Adrian Matthews is a Research Engineer at the SRI. He holds a BSc in Physics and Applied Mathematics (1994), an MSc in Opto-Electronics (1995) and a PhD in atomic physics (1999), all from the Queen's University of Belfast. After several years working in Ericsson, Athlone, he joined the AIT in 2006 and has worked on many Enterprise Ireland funded projects for the SRI.

Dr. Yuansong Qiao received his PhD in Computer Applied Technology from the Institute of Software, Chinese Academy of Sciences (ISCAS), Beijing, China, in 2007. As part of his Ph.D. research program he joined the SRI at Athlone I.T. in 2005. He continued his research in the SRI as a postdoctoral researcher in 2007. He is currently a postdoctoral researcher and the Principal Investigator on the COMAND Technology Gateway program in the Software Research Institute at Athlone Institute of Technology. His research interests include network protocol design and multimedia communications for the Future Internet.