International Research Journal of Advanced Engineering and Science
ISSN (Online): 2455-9024

A Secure and Dynamic Multi-Keyword Ranked Search Scheme over Encrypted Cloud Data

Prof. P. Gnanasekaran[1], C. Mareswari[2]
[1]Research Guide, Department of Computer Science, Jairams Arts & Science College, Attamparappu, Kakkavadi (PO), Karur, Tamilnadu, India-639003
[2]MPhil Scholar, Department of Computer Science, Jairams Arts & Science College, Attamparappu, Kakkavadi (PO), Karur, Tamilnadu, India-639003

***Abstract*—** *Mobile cloud computing has entered a new era of technology which would help in keeping large quantity of data in a lesser cost. Platforms for accessing and computing of large amount of data are much more viable than previous days. Researchers have been using large volumes of data for producing out puts and they consider data as an asset. Now a days data is being outsourced for conducting research in different sectors like market study, military purposes, tackling terrorist attacks ,whether forecasting etc this increased the value of data and it has to be kept safe. To preserve the nature of data high level encryption should be done while transferring the data but this may cause problems when multiple users try to access data from a cloud source. Researchers are trying to develop new encryption technique to reduce the risk of security while transferring the data. Solutions that are being discussed are a multi keyword search scheme supporting result ranking by adopting k nearest neighbors knn technique and another is a dynamic searchable encryption scheme through blind storage to conceal access pattern of the search user.*

Searching the encrypted data should be easy and should provide an easy feeling like using a search engine in the internet. It should also support multi key word search easy identification of relevant search result. The data base would be large the searchable encryption scheme should be in such a way that it would result in minimum delay. The ideal scheme for providing better performance is a multi keyword ranked search.

Keywords*—Cloud Computing, Encryption, Ranked search scheme, cipher, Data Vector, Stemming algorithm, Efficiency.*

I. INTRODUCTION

Multi Keyword Ranked Search over Encryption

Cloud computing is an important concept now a days multiple users can access and share their data in a single platform this provides an on demand high quality request and service from a single pool of data. Data may be of different type, it may be financial, digital, email or any other thing which may be private or public and this data may be out sourced for many purposes. For providing safety and security of this data it should be encrypted before out sourcing. Downloading and decryption of the entire data is not possible. Ranked search allows data users to discover the most appropriate information quickly and avoid redundant network traffic. It is important for such ranking systems to support multiple keyword searches. While index construction each document is associated with a binary vector as a sub index where each bit signifies whether matching key is contained in the document. The search key is also illustrated as a binary vector where each bit means whether corresponding key word appears in this search request so the resemblance could be exactly calculated by the inner product of the query vector with the data vector on the other hand directly outsourcing the data vector will go against the privacy

Contribution

We propose a coordinate matching when defining thread models in different cases

Ranking searching will improve the result and would provide a better solution in an expected time span.

When tested I n real time data it proves its ability and the proposed model also maintains more search Semitics

Proposed System

1) Cloud setup
2) Cryptography cloud storage
3) Vector model

Cloud Setup

We should posses large quantity of data in a cloud setup. The data owners would push the data to the server. The inflow and outflow of data from the server should be secure. The service providers should check the data and the communication between the user and cloud will be on the basis of multi keyword ranked search

Crypto Cloud Storage

Since we have to keep the privacy of data it has to be encrypted before outsourcing the data

Vector Model

We use a series of searchable symmetric encryption system which allows searching on cipher text time cost to generate a query mainly depends on the number of keywords in the dictionary, since the common main operation or time consuming operation in all the schemes is query encryption. So the time cost will become large as increasing the number of key words in the dictionary, the difference between PRSE 1 and PRSE 2 is that query semantic extension needs to be carried out during the step of search in the user interest model.

II. PROBLEM FORMULATION

We formulate the privacy problem of the multi-keyword fuzzy ranked search over encrypted data in this section.

Prof. P. Gnanasekaran and C. Mareswari, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *International Research Journal of Advanced Engineering and Science*, Volume 2, Issue 3, pp. 70-75, 2017.

A. System Model In this paper, we consider a cloud system consisting of data owner, data user and cloud server. In our system model, data owner has a collection of n data files F = (F1, F2, F3,⋯, Fn) and outsources them to the cloud server in the encrypted form C. To enable efficient search operation on these encrypted files, data owner will build a secure searchable index I on the keyword set W extracted from F. Both the index I and the encrypted data files C are outsourced to the cloud server. To search the encrypted data files for t given keywords, an authorized user computes a corresponding trapdoor T and sends it to cloud server. Upon receiving the trapdoor, the cloud server is responsible to search the index I and return the corresponding set of the encrypted documents. To improve the file retrieval accuracy and save the communication cost, the search result should be ranked by the cloud server and return the top-K relevant files to the user as the search results.

B. Threat Model In our threat model, both data owners and data users are trusted. However, the cloud server is honest-but-curious as in [2]–[4]. Even though data files are encrypted, the cloud server may try to obtain other sensitive information from user search requests while performing keyword-based search over C. So the search should be performed in a secure manner that allows data files to be securely retrieved while revealing as little information as possible to the cloud.

C. Design Goals • Support more spelling mistakes: Our multi-keyword fuzzy search scheme should support more spelling mistakes. For example, "network security" related files should be found for a misspelled query "netward security","netwrok security","netrwork security" and "netwrk security". • Privacy guarantee: The cloud server should be prevented from obtaining additional information from the encrypted data files and the index. • No Predefined Dictionary: No predefined dictionary is a great contribution of original scheme, so our scheme should not have predefined dictionary. • Support updating: The same as original scheme, our scheme should support dataset updating, such as file adding, file deleting and file modifying. • Ranked results according to the relevance score: To make users more satisfied with search results, the return results should be ranked according to relevance score. • Efficiency and Accuracy: The efficiency of our scheme should be same as the original scheme. And our scheme should be as accurate as possible and keep high accuracy.

D. Preliminaries three important techniques are used in our design: Stemming algorithm, Bloom Filter, Locality-Sensitive Hashing (LSH) and p-stable LSH. 1) Stemming Algorithm: A stemming algorithm is a process of linguistic normalisation, in which the variant forms of a word are reduced to a common form. A stemmer for English, for example, should identify the string "cats" (and possibly "catlike", "catty" etc.) as based on the root "cat", and "stems", "stemmer", "stemming", "stemmer" as based on "stem". A stemming algorithm reduces the words "fishing", "fished", and "fisher" to the root word, "fish". On the other hand, "argue", "argued", "argues", "arguing", and "argus" reduce to the stem "argu" (illustrating the case where the stem is not itself a word or root). It is widely adopted in Information Retrieval systems to improve performance. 2) Bloom Filter: Bloom filter is a kind of data structure with very high space efficiency. It makes use of the m-bit array to represent a collection, and can determine whether an element belongs to the collection. It is initially set to 0 in all positions and for a given set S ={ a1,a2,⋯,an}, usel independent hash functions from H ={ hi | hi : S → m,1 ≤ i ≤ l} to insert an element a ∈ S into the Bloom filter by setting the positions to be 1. To check whether an element q is in S, feed it to each of the l hash functions to get l array positions. If the bit at any position is 0, q / ∈ S; otherwise, either q ∈ S or q yields a false positive. The false positive rate of a m-bit Bloom filter is approximately (1−e−ln m)l. The optimal false positive rate is (1/2)l when l = m n ·ln2. 3) Locality-Sensitive Hashing (LSH): LSH is an algorithm for solving the approximate or exact Near Neighbor Search in high dimensional spaces. LSH hashes input items so that similar items are mapped to the same buckets with high probability. A hash function family H is (r1,r2, p1, p2)sensitive if any two points x, y and satisfy: if d(x, y)≤r1;Pr [h(x)=h(y)]≥ p1 (1) if d(x, y)≥r2;Pr [h(x)=h(y)]≤ p2 (2) where d(x, y) is the distance between the point x and the point y. 4) P-Stable LSH: The original scheme uses the p-stable LSH [18]. we will introduce it here. • When p=1, it is Cauchy distribution, defined by the density function fp(x)= 1 x 1 1+x2 , is 1-stable; • When p=2, it is Gaussian distribution, defined by the density function fp(x)= 1 √2π e−x2 2 , is 2-stable. The p-stable LSH function is: ha,b(v) =a·v +b w
Where a is a d-dimensional vector, b∈[0, w] is a real number and w is a fixed constant for one family.

III. Basic Idea of our Scheme

In this section, we first describe the main steps of our scheme and subsequently discuss the differences between our scheme and the original scheme. Finally, we will present additional details of our scheme.

A. Main Steps of Our Scheme The main steps of our proposed scheme are described in this section and illustrated in Data Preprocessing: For a data set F ={f1, f2,..., fn}, we first extract keywords from F to build a keyword set W ={w1,w2,...,wn}. We apply the Porter Stemming Algorithm to ascertain the root of the word.

For example, for the following set of words: "walk", "walks", "walking" and "walked" all have a similar meanings, and their stem is "walk". For the constructed stem set, we compute the relevance between the files and stems. Keyword Transformation: Keyword transformation is an important step in our scheme. Because the LSH functions take a vector as the input, we use the uni-gram vector to represent the keyword. A keyword is first transformed into the unigram-based set. For example, the uni-gram set of the keyword "secure" is {s1, e1, c1, u1, r1, e2}, in which s1 indicates the first s in the word. Similarly, e2 denotes the second e in the word. We use a 160-bit long vector to represent the uni-gram set. The element set contains 26*5 letters, 30 numbers and commonly used symbols, and thus the length of the vector is 160. The element is set to 1 if the corresponding uni-gram exists in the uni-gram

Prof. P. Gnanasekaran and C. Mareswari, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *International Research Journal of Advanced Engineering and Science*, Volume 2, Issue 3, pp. 70-75, 2017.

set of a given keyword. In practice, the length of the vector is changeable, and users can set up the uni-gram set according to their own needs. Using this representation, a keyword that might be misspelled in many different ways can still be represented in a vector that is highly similar to the correct one, and this closeness (distance) is measured by the Euclidean distance, the well-known metric for distance between vector-type data items. This uni-gram vector representation is a key step in enabling the use of LSH functions.

Algorithm 1 Generate Uni-Gram Based Keyword Vector Input: a plain-text keyword set, a null vector {0,1}160 Output: uni-gram based vector 1: for Wi from W1 to Wv in the set do 2: Stem Wi to STi whose length is sti, Spilt STi to STi[j] and generate a vector{y | y[j]=1,0 < j < sti}3: for STi[j] in STi do4: for k=1 to k=j-1do 5: if STi[j]=STi[k] then 6: y[j]++ 7: end if 8: end for 9: set STi[j] and y[j] to a new element STY[j] 10: end for 11: for STY[1] to STY[st] do 12: Set all corresponding position in {0,1}160 to 1 13: Set the rest position to 0 14: Output the vector VWi for the keyword Wi 15: end for 16: Output the V ={VWi | Wi ∈ W} for all keywords in the set 17: end for

3) Construction of the Bloom-Filter-Based Index/Query: We first generate an m-bit Bloom filer in which the initial

Algorithm 2 Search Procedure Input: Encrypted query vector q, q, Encrypted index, Threshold T Output: Encrypted document ID 1: for each document Di in the set do 2: for j=1 to j=m do 3: S = S+q∗ij +q∗i j 4: end for 5: if S > T then 6: Output S and IDDi 7: end if 8: end for 9: Ranked IDDi according to S 10: Output top-K document ID={IDDi |i ∈[1, K]}value of each bit is 0.

As the keywords of the files and query have been transformed into vector, we can use l LSH functions to hash the vector. For each keyword, we have l hash values and the set the corresponding bit in Bloom filter to 1. Due to the nature of the locality-sensitive hashing functions, two similar inputs within a certain distance are mapped into the same output with high probability. In this manner, a misspelled keyword can be hashed into the same bucket of the Bloom filter. Finally, the fuzzy keyword search can be achieved. 4) Inner Product Based Matching Algorithm: As shown in the final secure index for each file is a Bloom filter that contains all of the keywords in the file in which the keywords are first transformed into its bi-gram vector representation and subsequently inserted into the Bloom filter by LSH functions. Because the query and the index are constructed in the same way, we can compute the relevance of the query to each file. If a document contains the keyword(s) in the query, the corresponding bits in both vectors are 1, thus the inner product is a high value. This inner product result is a good measure for evaluating the number of matching keywords. Finally, the top-K files will be returned if their inner product is larger than the threshold T.

B. Differences Between Our Scheme and the Original Scheme 1) Stemming Algorithm: For English keywords, the same stems have different forms of expression. For example, the following set of words: “walk”, “walks”, “walking” and “walked”, they all have a similar meaning of “walk”, but they also display certain distinctions. In this case, if we query the keyword “walking”, but the keyword in index is “walk”, the probability of finding the keyword “walking” is low because the distance between “walk” and “walking” is too large. In fact, what we need is to denote the keywords with the same root into the same form. The stemming algorithm is one of the most commonly used algorithms for this purpose.

After applying the Porter stemming algorithm, we can ascertain the root of the word “walk” and find the corresponding files. Keyword Transformation: Keyword transformation is a key step of our scheme. In the original scheme, a keyword is first transformed to a bi-gram set and then the bi-gram set is mapped into a 262-bit long vector. Each element in the vector represents one of the 262 possible bi-grams. The element is set to 1 if the corresponding bi-gram exists in the bi-gram set of a given keyword. In our scheme, we still use the vector to represent the keyword. However, we use the uni-gram based method to transform the keyword. For example, the uni-gram set of keyword “secure” is {s1, e1, c1, u1, r1, e2}, where s1 indicates the first s in the word. Similarly e2 denotes the second e in the word. In this work, we will compare our method with Wang’s method under conditions of different spelling mistakes and then discuss the disadvantage of our method and those of Wang’s method.

a) Comparison under different spelling mistakes: In actual retrieval, a keyword can be misspelled into many forms and all spelling mistakes should be considered by the fuzzy keyword search system. Compared with MFSE, our keyword transformation method exhibits a smaller Euclidean distance for each type of spelling mistake. We present selected concrete examples for better understanding. • Misspelling of a letter: For example, the keyword “secure” is misspelled into “secare”. In this case, the keyword “secare” is transformed into {s1, e1, c1, a1, r1, e2} by our transformation, which changes only 1 uni-gram, and the Euclidean distance between the correct keyword and misspelled keyword is √2. The bi-gram set of misspelled keyword “secare” is{se,ec,ca,ar,re}in MFSE, which changes2 bi-grams and the Euclidean distance is 2. Obviously, our method has a smaller Euclidean distance.

• Missing a letter or adding a letter: For example, the keyword “secure” might be misspell as “secre” and is transformed to {s1, e1, c1, r1, e2}. The Euclidean distance of this situation is 1, and the Euclidean distance is √3 if we use the bi-gram. If the keyword “secure” is misspelled into “secuure”, the Euclidean distance is 1 for use of the uni-gram and the Euclidean distance is √2 for use of the bi-gram. • Reversing the order of two letters: For example, the keyword “secure” is misspelled into “secrue”. The misspelled keyword “secrue” is transformed into {s1, e1, c1, r1,u1, e2} and the generated vector is the same as the correct one. If we use the bi-gram, the bi-gram set is {se,ec,cr,ru,ue}, and the Euclidean distance is 3. b) Disadvantages of our method and Wang’s method: Although our method offers advantages, it still contains certain disadvantages. The proposed uni-gram removes the order dimension. Because the order is removed, the anagram is

Prof. P. Gnanasekaran and C. Mareswari, “A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data,” *International Research Journal of Advanced Engineering and Science*, Volume 2, Issue 3, pp. 70-75, 2017.

mapped to the same vector, and our scheme cannot distinguish the keyword like this. If using the bi-gram, the keyword set is completely different because the order has not been changed. Wang's scheme also contains its disadvantages in that the original method cannot represent the same bi-gram. For example, the word "representation" has 2 "re" elements in the bi-gram set but only 1 is used when it is transformed into a vector. However, our method is able to handle this problem. Threshold T: The threshold is a measure used to determine whether the documents are relevant to the user's query. Each document is considered relevant only if its inner product is greater than the threshold T. Usually, the threshold is 0, if the result is the inner product of the query vector and index vector, such as in [2] and [3]. However, the threshold of our scheme is not 0 because of the false positive and false negative. A false positive is h(w) =h(w) when d(w,w)>r2 for two different keywords w, w. A false negative is d(w,w)<r1 but h(w) =h(w). Thus our threshold T is as follows. For t query keywords, the threshold T is a random number such that max(T1)<T < min(T2), whereT1 is the inner product of the t irrelevant keywords and the T2 is the inner product of the t−1 irrelevant keywords and one relevant keyword. For security consideration, we set the threshold T to a random number. For two queries with same number of query keywords, the threshold T is different which can protect the number of the query keywords. To improve the accuracy of the scheme, we set the threshold to T > min(T2) 4) Random Number ε and t: Because the cloud server computes the result scores of each document and the query, such information should not be revealed to the cloud. If the cloud knows such valuable information, it can use this information together with selected background knowledge to deduce additional information. Therefore, in building the query vector, we introduce the random number ε and t to protect the final similarly result scores and threshold T. We replace all of the elements 1 in the query vector with εi. In this work, εi is a random number that follows a normal distribution N(μ,σ2).

The element inserted into the query vector is not 1 but εi. In addition, each individual vector is extended to the (m+1)-dimension, where we assigned the random t to the extended dimension in each query vector. Therefore, for two queries with the same keywords, the final similarly result scores are different, and makes the cloud server cannot distinguish between the two generated vectors. Additional details can be found in section IV. 5) Relevance Ranking: To ensure that the result better satisfies the user's demand, we use the keyword frequency as a weight to reflect the relevance between the files and keywords. To produce the relevance ranking, we first compute the relevance between the files and keywords. Second, we replace the elements in the index Bloom filter with the relevance score.

Formerly, we set the corresponding position of hash value to 1, but at this point, we set it to the relevance score. Because there are l points for each keyword, so we assign the score to each point on average. If different keywords are hashed into the same point, and we use their average value as the insert. For each document, if it contains a greater amount of keywords that the user queried, it should have a higher priority in the returned top-K file list. For two documents, if they contain same number of keywords, the document with the higher relevance score of the keywords is the better matching result

IV. Security Analysis

In this section, we will analyze the security of our scheme. Inspired by Wang's work, we will show the process of proof in detail for known ciphertext and a known background. Before the proof, we introduce some notations. • History: H = (, I,Wk) where is a file set, I is a searchable index and a series of queries W = (w1,⋯,wk) is submitted by users. • View: V (H) = (Encsk(), Encsk(I), Encsk(W)) which is obtained by encrypting H with some secret key sk. Note that the cloud server can only see the views. • Trace of a history: Contains the sensitive information learned by the cloud server. A trace of the history H is the set of the trace of queries Tr(H)={Tr(w1),⋯,Tr(wk)} Specifically, Tr(wi)= {(δj,sj)wi⊂δj ,1 ≤ j ≤ ‖}, wheresj is the similarity score between the query wi and the file δj.

A. In Known Ciphertext Model In the known ciphertext model, given two histories with the same trace, if the cloud server cannot distinguish which of them is generated by the simulator, it cannot learn additional information for the index and the dataset beyond the search result and the access pattern. Wang et al. proved the security of their scheme in their original paper. As the secure index Encsk(I()) and the trapdoor Encsk(Wk) generates the same trace as the one that the cloud server, we claim that no probabilistic polynomialtime (P.P.T.) adversary with more than 1/2 probability can distinguish between the view V and V(H). Particularly, due to the semantic security of the symmetric encryption, no P.P.T adversary can distinguish between Encsk() and. And the indistinguishability of indexes and trapdoors is based on the indistinguishability of the secure kNN encryption [17] and the random number introduced in the split processes. As we use the same encryption method as in the original scheme, so our scheme is also secure also under known ciphertext model. However, because we introduce the random number ε and t into the query vector, our scheme should be more secure under the known ciphertext model. The improved details are presented as follows: • BuildIndex (D, SK, l): Choose l independent LSH functions from the p-stable LSH family H ={ h :{ 0,1}160 → {0,1}m}. Construct a (m+1)-bit Bloom filter ID as the index for each file D.

1) Extract the keyword set WD ={ w1,w2,⋯} , wi ∈{ 0,1}160 from D.

2) For each keyword wi, insert the relevance score into the index ID using hj ∈ H,1≤ j ≤l.

3) Set the (m+1)-dimension in index Bloom filter to 1;

4) Encrypt the index ID using Index_Enc(SK,ID) and output the Encsk(ID) •

Trapdoor(Q, SK): Generate a (m+1)-bit long Bloom filter for the query Q.

1) For each search keyword qi, insert qi using the same l LSH functions hj ∈ H,1 ≤ j ≤ l into the bloom filter.

Prof. P. Gnanasekaran and C. Mareswari, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *International Research Journal of Advanced Engineering and Science*, Volume 2, Issue 3, pp. 70-75, 2017.

2) Set the (m+1)-dimension in the query Bloom filter to t;
3) Encrypt Q using Query_Enc(SK,Q), and output the Encsk(Q) and the threshold T.
Theorem 3: Our scheme is more secure under the known ciphertext model. Proof: We compare the security of our scheme with that of the original scheme in the query vector, the final similarity scores and the threshold T. For the query vector, the indistinguishability is based on the secure kNN encryption and the random number introduced in the split processes. Because we insert the random number ε and t into the query vector, our trapdoor is obviously more indistinguishable. Not that we introduce the random number ε and t into the query vector, the final similarity scores would be IT By doing so, even for two queries with the same query keywords, the final similarity scores are different which protects the scale relationship for two queries with the same query keyword.

In our scheme, the threshold T is a random number in a certain range. As we introduce the random number ε and t into the query vector, the final similarity scores would change the threshold will changes as well. The new threshold is μ·T+t. Because T, t and μ are all random numbers, for two queries with the same number of query keywords, the threshold is completely different. Due to μ and t, the cloud server will be more difficult to obtain the relationship between the number of query keywords and the threshold. Based on these three aspects, we observe that our scheme is more secure than the original scheme. For the other aspects, our method is the same as the original scheme. Therefore, the theorem.1 is has been proven.

B. In Known Background Model In the known background model, we follow the original scheme that uses a pseudo-random function f as an extra security layer to secure the linkage between the keywords and the bloom filter. In building the index, we choosel independent LSH functions from the hash family H and one pseudo-random function: f :{ 0,1}∗ ×{ 0,1}s →{ 0,1}∗. The new hash functions are {gi = fki · hi,hi ∈ H,1 ≤ i ≤ l} and we use the new hash functions to generate the query and index vector. The extra security layer does not affect the search result because the pseudo-random functions are collision free. Under the known background model, the cloud server obtains selected background information such as a certain amount of the keyword and trapdoor pairs, denoted as (wi,Ti). Intuitively, the cloud server should not be able to distinguish the view generated by the simulator from its own view through the keyword and trapdoor pairs.

V. Experimental Results

In this section, we estimate the overall performance of our proposed scheme by implementing our proposed system using C# language on a Windows7 server with a Core2 CPU running at 2.93GHz. We used Request for comments database (RFC) [6] as our data set. We chose approximately 3000 files from the data set, and we extracted 3422 keywords in total. The maximum number of the files is 179 and the minimum is 92. We set k=8, l=30 and set m=8000 and build a (√3, 2, 0.56, 0.28)-LSH hash function. Similar to the original scheme, we randomly picked one letter from the keyword and replaced it with another letter. We allowed at most two fuzzy keywords in a query. The tests for other spelling mistakes are independent and not included in the precision of the result

A. Efficiency 1) Trapdoor generation: The trapdoor generation process contains three major steps: stemming, the Bloom filter generation and the encryption shows the total time of trapdoor stemming and Bloom filter generation. The generation time increased linearly with respect to the number of the inserted keywords. As the number of keywords grew, the trapdoor generation time also increased. 2) Index construction: The index construction time was the same as that of trapdoor generation. Because the stemming and Bloom filter generation were linear in the number of the keywords, the index vector generation time could be large, but it was just a one-time effort shows that the encryption time is linear in the size of files because the index structure we constructed was a per file based index. 3) Search time: One important parameter that affected the search time was the number of the files n. Because our index was a per file based index, the search time increased linearly in the number of files, as illustrated
(a) The Bloom filter generation time of trapdoor & a single index file; The stemming time of keywords. (b) The encryption time for all the indexes.. (a) The search time of different size of the file set. We set the query keyword number = 5; (b) The search time of different number of query keyword. We set the size of document = 3000
we note that the number of the query keywords had a small impact on the search time. This is because regardless of the number of keywords, all of them were mapped into a query bloom vector. Hence, the search time was independent of the number of query keywords to a large extent. Another important parameter is the length of the bloom filter. The search efficiency of our scheme was the same as that of the original scheme because both the index and the trapdoor were built in the same manner.

B. Result Accuracy We used precision to measure the result accuracy. We denoted the true positive by tp and the false positive by fp, and the precision was equal to tp tp+fp. To generate the fuzzy search, we randomly chose keywords and modified it into a fuzzy keyword. 1) Precision of Our Scheme: An important parameter in our proposed scheme is the number of the keywords in the query. For the exact search, the precision decreased slightly from 100% to 95% as the number of the keywords increased from 1 to 10. Although the accuracy of the fuzzy search was not greater than that of the exact match, it was still produce a high level of accuracy, greater than 85%. From we note that the precision of the exact match slightly decreased from 100% to 95% as the number of the query keywords increased from 1 to 10.

VI. Conclusion

In this paper, we investigate the problem of multi-keyword fuzzy ranked search over encrypted cloud data. We propose a multi-keyword fuzzy ranked search scheme based on Wang et al.'s scheme. Concretely, we develop a novel method of

Prof. P. Gnanasekaran and C. Mareswari, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *International Research Journal of Advanced Engineering and Science*, Volume 2, Issue 3, pp. 70-75, 2017.

keyword transformation and introduce the stemming algorithm. With these two techniques, the proposed scheme is able to efficiently handle more misspelling mistake. Moreover, our proposed scheme takes the keyword weight into consideration during ranking. Like Wang et al.'s scheme, our proposed scheme does not require a predefined keyword set and hence enables efficient file update too. We also give thorough security analyses and conduct experiments on real world data set, which indicates the proposed scheme's potential of practical usage.

REFERENCES

[1] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. of IEEE Symposium on Security and Privacy'00*, 2000.
[2] E.-J. Goh, "Secure indexes," Cryptology ePrint Archive, Report 2003/216, 2003, http://eprint.iacr.org/.
[3] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proc. of ACNS'05*, 2005.
[4] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proc. of ACM CCS'06*, 2006.
[5] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. of EUROCRYP'04*, volume 3027 of LNCS. Springer, 2004.
[6] P. Golle, J. Staddon, and B. R. Waters, "Secure conjunctive keyword search over encrypted data," in *Proc. of ACNS04*, pp. 31-45, 2004.