I. J. Computer Network and Information Security, **2016, 11, 65-71**
Published Online November 2016 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2016.11.08

An Efficient Multi-keyword Symmetric Searchable Encryption Scheme for Secure Data Outsourcing

Vasudha Arora
Manav Rachna International University, Department of Computer Science & Engineering
Faridabad, India
E-mail: vasudharora6@gmail.com

S.S. Tyagi
Manav Rachna International University, Department of Computer Science & Engineering
Faridabad, India
E-mail: shyam.fet@mriu.edu.in

Abstract—Symmetric searchable encryption (SSE) schemes allow a data owner to encrypt its data in such a way that it could be searched in encrypted form. When searching over encrypted data the retrieved data, search query, and search query outcome everything must be protected. A series of SSE schemes have been proposed in the past decade. In this paper, we are going to propose our an efficient multi-keyword symmetric searchable encryption scheme for secure data outsourcing and evaluate the performance of our proposed scheme on a real data set.

Index Terms—SSE, TRSE, Searchable encryption, homomorphic encryption, Information Retrieval. Data outsourcing.

I. Introduction

Cloud computing provides the data owners a privilege to store their data on pay per use basis. Searchable encryption can be defined as a technique that allows the organizations or any data owner to store their confidential documents in encrypted manner. Documents stored can be retrieved in encrypted form by single or multiple clients while leaking as little information as possible to the server. When outsourcing, we must keep in mind that, we need to protect the retrieved data, the query sent for searching the data, and the outcome of the search query received by the client. When searching the data there can be a single query as well as multiple queries, which can be adaptive or non-adaptive in nature. A series of queries that are independent of each other are non- adaptive whereas multiple queries where next query is based on the previous results are adaptive queries. Further, in the cloud based scenario, there can be a single user that can query the data outsourced or there can be multiple concurrent data users that can query the data, who are possibly given the access rights defined by the owner. To implement SSE scheme, we consider the following scenario: the client organization or data owner has a collection of files F that consists of a set of words. Data owner encrypts the document collection; together with some additional data structure and sends everything to the cloud server. The data user, authorized by the data owner, must have the capability to encrypt the query in searchable manner and send that query to the cloud server to make a search over the encrypted data. The honest-but-curious server, on the other hand, must be capable to find all the documents that contain a particular keyword but reveal the information as little as possible. In this paper we propose a multi-keyword top-k retrieval symmetric searchable encryption scheme and evaluate and compare the performance of already existing scheme as well as proposed scheme.

The rest of the paper is organized as follows: we define the architecture of the problem in section 2. Section 3 discusses the existing schemes and their performance. It also discussed the problems with existing schemes. Section 4 discusses detailed description of our proposed scheme. Section 5 gives the performance evaluation of our proposed scheme based on different parameters. Section 6 discusses the conclusions.

II. Architecture

Many solutions have been proposed to solve the problem of secure search over outsourced data to the honest-but-curious cloud server model. Symmetric searchable encryption scheme has been defined to secure the outsourced data by encrypting at the data owner's end and make the retrieval of data possible in encrypted manner only. Data when outsourced to cloud could be secured on the network channel using number of existing network security schemes but when data is kept on the cloud how to protect it from the cloud server itself is a major question. As data resides on the cloud server everything is visible to the server. Hence, symmetric searchable encryption scheme provides a way out to

secure the data from the server itself.

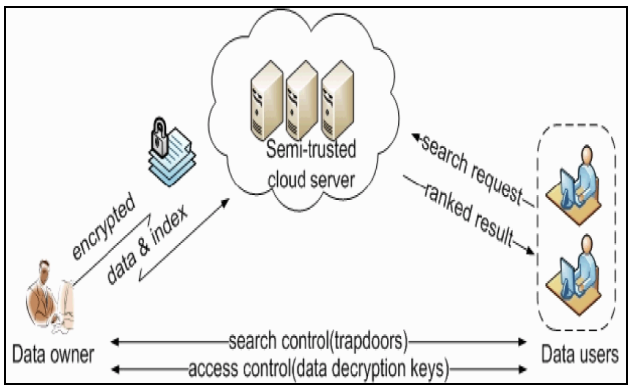

Fig.1. Architecture for Search over Encrypted Cloud Data.

Fig.1 shows the cloud computing scenario which involves three basic objects or entities. The architecture discussed in [10][7][9][12][4] shows the complete scenario of retrieval of encrypted cloud data. The three entities involve the data owner, the data user & the cloud server. The data owner entity has the huge amount of data to be outsourced, so that it could be used by its large number of authorized users at anytime and anywhere. The cloud server hosts the outsourced data to its storage. As the data may be sensitive, so the honest-but-curious cloud servers cannot be trusted for protecting data.

Data Owner has a collection of m files F = {f1,f2,f3,…….,fm} which he wants to outsource on the cloud server in encrypted form, for which cloud server must be able to store and have the capability to provide keyword based retrieval to both the data owner and the authorized data users. For this purpose data owner creates a secure searchable index I in encrypted form from the file collection F. A collection of n tokens or keywords T = {t1, t2, t3,……., tn} is extracted from F encrypted with some cryptographic scheme. Both the encrypted file collection F' and encrypted Index I' are then outsourced to the cloud server.

The authorized data users must have the capability and required authorization to search the outsourced data by processing a multi-keyword query. For this purpose multi-keyword requests are also encrypted and then sent to the cloud server to process the query in encrypted form. The cloud server then processes the query, searches the keywords, then based on relevance scores rank order the files and sends the top-k files as required by the user in encrypted form. The data user then decrypts the files for his use.

To create a searchable index an inverted index is created [13][4] which is designed to allow very fast full text searches. An inverted index consists of a list of all the unique words that appear in any document, and for each word, a list of the documents in which it appears. To create an inverted index, we first split the content field of each document into separate words (which we call terms, or tokens), create a sorted list of all the unique terms, and then list in which document each term appears.

For information retrieval a ranking function used [14] called *tf-idf* rule. Each term in the file is assigned a weight for that term, that depends on how many times a term or keyword occurs in the file or document. We basically calculate a score between the query term and the document. Let it be denoted as *tf(t,f)* for term frequency of a queried keyword *t* in a file *f*. For term frequency calculations the exact ordering of terms in a document is ignored but the number of occurrences of each term is taken. A document frequency *df(t)* is defined as number of files in the file collection C that contain the keyword *t*. If we have total m number of files in file collection C, an inverse document frequency of a term *t* can be defined as *idf(t)*of a term which is not repeated very frequently is high and otherwise for a frequent term is likely to be low. Now *tf-idf* rule assigns to a term *t* a weight in a file *f* using the following equation.

$$tf - idf(t,f) \;=\; tf(t,f)\; x\; idf(t) \qquad (1)$$

III. Symmetric Searchable Encryption Schemes

Based on searching on encrypted data, number of symmetric searchable encryption schemes has been defined. In this section we have analyzed the schemes on different parameters.

In [17] A. Boldyreva et al. defined an order preserving encryption scheme to search over encrypted data. OPE scheme preserves the numerical ordering of plaintext. It allows the indexing and query processing on encrypted data as efficient as on unencrypted data.

Basically, OPE is a method of encrypting data so that it is possible to make inequality comparisons on encrypted data without decrypting it. It is a deterministic (i.e. it is known well in advance) symmetric encryption scheme whose encryption algorithm preserves numerical ordering of plaintexts. For example, let M and N be finite ordered sets. We say that OPE is an order preserving encryption scheme with plaintext space M, ciphertext space N, and key space K if for any k ϵ K,and for any inputs x1, x2 ϵ M, the following condition holds: If x1 < x2 then OPE (k, x1) < OPE (k, x2), where OPE(k,x1), OPE(k,x2) ϵ N.

As OPE preserves the order of plaintexts therefore it is not a perfectly secure encryption scheme since ciphertexts leak the order information of plaintexts. There are various constructions of OPE schemes. [16] Proposed an OPE algorithm which first generates a sequence of random numbers and then encrypts an integer x to the sum of first x random numbers. [20] defines an OPE algorithm based on a sequence of strictly increasing polynomial functions. The encryption of an integer x is the output of iterative operations of those polynomial functions on x.

OPE has several loop holes. The most problematic is the adversary's ability to guess approximately where the underlying plaintext of a ciphertext lives in the plaintext space. And it sometimes also reveals to certain attackers half the bits of plaintext given its ciphertext. The security of an encryption scheme depends on how precisely the adversary can predict the bits in the plaintext. Another problem [18] against the OPE scheme based on IND-CPA is to reverse the order of chosen plaintext attack, i.e.

the adversary is given the ciphertext and subsequently chooses the plaintexts.

C. Wang et al. in [4] in 2012 modified the OPE scheme and came up with one-to-many order preserving mapping to reduce the amount of information leakage from the deterministic property of OPE scheme. In OPE scheme, a plaintext t in a Domain D is always mapped to some numeric value in a random interval range R. A ciphertext c for plaintext t is chosen from that range using t as seed value, which always deterministically assigns same ciphertext value to a plaintext.

Authors in [4] define one–to-many OPM which also maps the plaintext value in a random interval range R but same plaintext values are no longer assigned the same ciphertext values instead a random value in the range R is assigned. To do so, OPM algorithm uses the unique file Ids, and plaintext value m as a random seed to choose the final ciphertext in a range R. Due to the use of unique file Ids as a part of random selection seed same plaintext m will be always mapped to the different ciphertext c, within a randomly assigned range R.

Hence, with the help of order-preserving mapping, the server can rank the encrypted files as efficiently as for unencrypted scores. This scheme embeds the encrypted relevance scores in inverted searchable index in addition to the file identifiers. Hence, the encrypted scores are the only additional information that could be utilized by the opponent. The content is already protected with the encryption scheme. Thus only keyword privacy needs to be focussed As long as the range size R chosen by the data owner is sufficiently large,an the information about order preserved numerical values of keywords is difficult to predict. Hence, the information leakage from an overall point of view is reduced and keyword privacy is always preserved.

The RSSE scheme [4] adds the relevance score entries to an ordinary inverted index where relevance scores are encrypted using OPM, which adds the mapping cost only to the original inverted index construction cost.

In 2013 J.Yu et al. in [10] proposed two round searchable encryption (TRSE) scheme. Authors employed the k-similarity relevance to avoid the information leakage due to order preserving techniques. Earlier techniques employ server-side ranking based on OPE to improve the efficiency of retrieval over encrypted cloud data. Authors introduced the concepts of k-similarity relevance, and employed vector space method and homomorphic encryption.

J. Yu. et al. used a Vector space model [19] to score a file on multiple keywords where earlier schemes scored only on the single keyword retrieval. Authors used this algebraic model to represent a file as a vector and each term extracted out as keyword in the index represents the each dimension of the file. That is, an m*n index is generated where each file represents a column and each term represents a row in the index. If a term occurs in a file the relevance score for the term is non zero otherwise, it is zero.

The data owner encrypts the searchable index with homomorphic encryption. A public key encryption scheme E is homomorphic if it permits encrypted data to be operated on without decrypting the ciphertext. Now, in order to perform operations on ciphertexts, a partial or fully homomorphic encryption scheme requires that for all k and all (public key, secret key) pair, where k denotes the security parameter required to generate the key pair, the following conditions hold:

For any m1, m2 that belongs to a plaintext space M and any c1, c2 belongs to ciphertext space C with $m1=Dec_{sk}(c1)$ and $m2 = Dec_{sk}(c2)$, we always have $Dec_{sk}(c1*c2) = m1 * m2$ where * can be any group operation.

Table 1. Index Structure for TRSE Scheme

Files / Term	F1	F2	F3	F4
Network	0.996	0.993	1.234	0.034
Species	0.087	0.876	0	0.456
Technology	0.017	0.444	0	0.254
Foundation	0.523	1.23	0.466	0
Domain	1.987	0	0.120	0.342

Authors in [10] used modified FHEI (fully homomorphic encryption over Integers) scheme [20] because vector space model for top-k retrieval only requires addition and multiplication operations over integers to calculate the relevance scores from the encrypted searchable index. This modified FHEI scheme is more simplified and efficient than fully homomorphic encryption scheme. Integer GCD used to provide the sufficient security over integers. But this modified FHEI scheme is relatively time consuming, therefore, authors only use it for encrypting the searchable index I, while the file set is encrypted using some symmetric encryption scheme.

Some of the areas that authors left for as a discussion and to be worked upon later on include efficiency improvement, the algorithm does not considers updates, communication overhead. The modified FHEI scheme that authors have used is although a more simplified form as proposed by Gentry et al. [19] but this simplicity is achieved at a cost of a large key size. Although authors have applied reductions and compressions still the key sizes generated for FHEI scheme are reasonably high for the practical system. When the user creates and encrypts his query trapdoor and sends to the cloud server the communication overhead would be too high. The algorithm also does not considers the data updates such as adding or deleting files or modifying already existing files that leads to another new challenge to the searchable encryption scheme. Since frequent updates may be there which requires updating the searchable index since TRSE scheme relies on tf-idf values and idf depends on number of documents that contain the keyword and tf depends on number of times a term occurs in a document which also needs to be updated with every modification in a file. The

cost of calculating tf-idf weights to get the relevance scores during each search request is too expensive.

Raghavendra et al. in [11] proposed a most significant index generation scheme which reduces the index generation time to the order of O(N*3). Authors proposed MSIGT scheme for secure and efficient single-keyword search over encrypted cloud data. The scheme creates an n* 3 index and uses MSD radix sort followed by counting sort.

MSD radix sort, sorts the index based on the most significant first character of every keyword extracted. After that counting sort groups the elements with same character into the same bucket. Then finally all the buckets are concatenated together in order. After sorting each keyword in the index is encrypted separately using the following equation:

$$\alpha\,(w_i) = \sum_{p=0}^{n} a_b \;\; x^{k-p} \qquad (2)$$

where w_i represents i^{th} keyword, k represents the number of characters in the keyword i.e the length of the keyword, a_b the ASCII code of character at $(k\text{-}p)^{th}$ position. And x represents a randomly generated real number but that should also be same for both the index keyword or queried keyword.

Authors proposed a scheme that generates index in much less time as compared to TRSE [10] scheme. But they didn't discuss anything about the retrieval phase and how searching should be done using this proposed index scheme.

IV. Proposed Scheme

In this section we define our proposed scheme that is more secure, accepts automatic updates and is simpler to implement. Here we first define the notions and preliminaries required to define the scheme and then we outline the structure of our proposed scheme.

A. Notations and Preliminaries

F = Set of m data files (f1, f2, ……, fm) in the file collection to be outsourced to the cloud server.

m = Represents total number of files to be outsourced.

T = Set of n identifiers or tokens (t1, t2, ……, tn) extracted from each file in the file collection F after filtering out stop words.

n = Total number of tokens extracted after removing stop words.

I = searchable Index built from the tokens extracted from file collection F. This index contains n records.

REQ = multi-keyword request generated by the data user.

Enc(t_i) = encrypted result for each encrypted keyword based on equation (2).

a = ASCII value of alphabets in the tokens extracted.

ID (f_i) = identifier generated for a file $f_i \in F$

SK = symmetric secret key generated and distributed to authorized users.

REQ' = Encrypted multi-keyword request generated by data user and sent as a trapdoor to cloud server for searching the data.

B. Framework for scheme

We here define our proposed scheme that describes a multi-keyword ranked search and uses an updatable index of the order O(n x 3). We use the following terminology to describe our scheme: we have a data owner that has a collection of *m* files *F= {f1, f2, f3, ….., fm)* to be outsourced to the cloud server. Before outsourcing, all the tokens are extracted from the files. The files are then encrypted using a symmetric encryption scheme. The tokens are then filtered to remove the stop words and rest n tokens are collected into an index table. The index table contains n rows and 3 columns that contain the tokens, file identifiers, relevance score of each file with respect to the token. The tokens are represented as a set of *n* values *T= (t1, t2, t3, …, tn)*. As soon as a file is added its identifier is generated and it is mapped to the index table. If an already existing file is modified it also generates a new identifier for the file and earlier record for that files are deleted from index table.

The tokens collected after filtering the stop words are then encrypted with symmetric encryption algorithm. A second level of encryption is also provided by encrypting the encrypted keywords with the following scheme [1]

$$Enc\,(t_i) = a_0 x^k + a_1 x^{k-1} + \ldots\ldots\ldots + a_n x^{k-n} \qquad (3)$$

$$Enc\,(t_i) = \sum_{l=0}^{n} a_i \;\; x^{k-l} \qquad (4)$$

Where x is a randomly generated large real number, which needs to be same for both indexed token and queried token. k Represents the length of the token, and l represents the position of the alphabet in the token.

After encryption, the encrypted files and doubly encrypted index is outsourced to the cloud server. On the user's side when queried keywords are encrypted using the same encryption methodologies as for indexed keywords there is a possibility of frequency analysis attack. To overcome this attack as soon as a search is initiated from a user the value of x is automatically changed after every search. Therefore every time a search is processed the same token will have different value for ciphertext and hence frequency analysis attack could be controlled.

Hence the entire procedure takes place in three phases: one on the data owner's end, second on the data user's end and third on the cloud server's end. The prposed scheme works on the following algorithms:

GenKey (r) - It Generates the symmetric encryption key *SK* using the security parameter r. This symmetric key is shared between the data owner and the authorized users.

Preprocessing (F, SK, x) – This algorithm extracts the tokens from the file collection F, then filters the stop words, calculates the relevance scores for files for every extracted token based on *tf-idf* rule. The tokens are then encrypted using equation (2). The index, I, is the re-encrypted using symmetric key, SK. The x is randomly generated and shared with the authorized user. The file collection F is also encrypted using symmetric encryption

cryptographic scheme. This searchable encrypted index *I'* and encrypted file collection *F'* are outsourced to the cloud server.

CreateReq (REQ, SK, x) – the authorized user when generates a multi-keyword query, REQ, the algorithm processes the query, filters out stop words, encrypts the remaining tokens using SK and then re-encrypts them using equation (2), to *REQ'*. This encrypted search request is then sent to the cloud server to make a search over the encrypted index.

As soon as this search request is generated and sent to the cloud server the randomly generated *x* value for encryption in equation (2) is automatically regenerated and updated.

Scoreclaculate(I',REQ')- On receiving the search request cloud server computes the score of each file in *I'* with respect to *REQ'* based on equation (1).

Search (REQ',I') – cloud server calls *Scorecalculate(I',REQ')* and sorts the files based on their relevance scores and sends back encrypted files to the user.

Decrypt(SK, F') - the data user receives the encrypted files. The user then decrypts the files. Hence files are securely ranked searched over encrypted data.

Hence, the entire scenario is considered to be divided into three phases or modules:

❖ *Data Owner Module (Initialisation phase)*

1) Data owner calls *GenKey(r)* algorithm to Generate the symmetric encryption key *SK* using the security parameter r. This symmetric key *SK* is shared between the data owner and the authorized users to access the outsourced files $f_i \in F$ $(1 \leq i \leq m)$.
2) The data owner calls *Preprocessing(F,SK,x)* algorithm to extract tokens T from each $f_i \in F$, and then filters out the stop words to create a set of n tokens $T=\{t_i \mid 1 \leq i \leq n\}$. The data owner then calculates the relevance scores based on *tf-idf* rule for each $t_i \in T$. It then creates an (n * 3) searchable inverted index I containing (t_i | *FID* | *Rel. Score*) and a file record table containing (*FID*| *Fname* | *update status*).
3) Each token t_i is then encrypted using equation (2). Entire file collection F is encrypted using a symmetric cryptographic scheme with SK and then the doubly encrypted index, and encrypted file collection are outsourced to the cloud server.

❖ *Data User Module (Retrieval Phase)*

An authorized data user can call *CreateReq (REQ, SK, x)* algorithm to generate a multi-keyword search request $REQ = (t_1, t_2, \ldots, t_s)$. REQ is encrypted twice to generate *REQ'*, and then user sends *REQ'* to the cloud server.

As soon as the user initiates the search by generating an encrypted query *REQ',* our algorithm automatically updates the value of parameter *x* in equation (1).

❖ *Cloud server Module (processing and Ranking Phase)*

1) Cloud server receives *REQ*', which contains encrypted tokens and calls *Scorecalculate(I',REQ')* algorithm to compute the relevance scores for the files.
2) After calculating the relevance scores the files are sorted in descending order according to the scores.
3) Top k ranked files are picked up from the sorted list and sent to the user who initiated the search request.

V. Performance Analysis

We conducted a complete experimental evaluation of the proposed scheme. Our experiment environment includes a user and a server. The user acts as a data owner and a data user, and the server acts as a cloud server. We used c# .Net platform on a windows 7 machine with core i5 processor. The overall performance of our proposed scheme is evaluated on real data set: National Science Foundation Research Awards Abstracts 1990-2003 [22].We evaluated efficiency of the proposed scheme based on following parameters:

A. Index Generation time

Index generation time refers to the time taken to populate the index whenever a new file is added. It includes the time to extract the tokens, remove stop words and finally creating the records in the index table together with their relevance scores.

Figure 2 shows the index generation time taken by our scheme with respect to the number of tokens in the index. For 500 tokens from 3 files to be uploaded in the index our scheme has time complexity O(N x3), i.e., for 500 x 3 =1500 elements it takes 196 milliseconds (approx) and for 4000 tokens to be uploaded, i.e. , for 4000 x 3 =12000 elements to be uploaded it takes 2748 milliseconds.

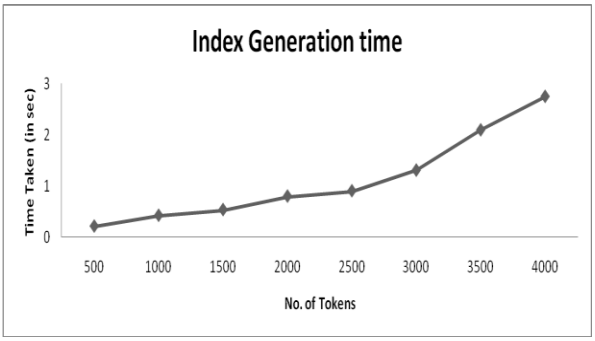

Fig.2. Time Taken to Generate Index on the Scale of Number of Tokens.

B. Update Efficiency

The update efficiency refers to the time it takes to update the relevance scores, as well as updating the tokens in the index, for every addition, modification, or deletion of a file. For every update operation the entire index has to be refreshed to update the relevance score of a token as it depends on inverse document frequency values.

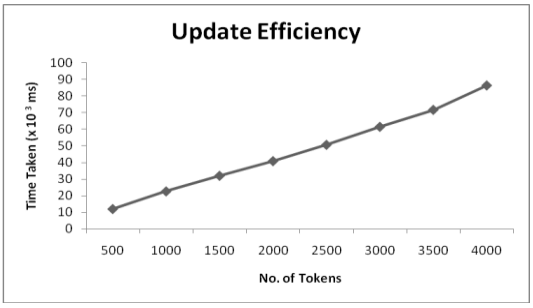

Fig.3. Time Taken to Update Index on the Scale of Number of Tokens.

Hence, for an index having 500 tokens requires approximately 11 seconds to update the index while an index with 4000 tokens can be updated in 86 seconds.

C. *Search Efficiency*

During the search process, after the search request has been converted in an encrypted trapdoor and sent to the cloud server, the database uses a binary search tree to find out the specified matches. The time complexity for searching using binary search trees leads to O(m) where m is the number of tokens in the index. We have analyzed the search efficiency in two different scenarios.

Figures 4 shows the search time required to search the documents when number of tokens in the index is varied from 500 to 4000 and number of tokens in search query is fixed to 10. The figure shows that in our proposed for a fixed number of tokens in search query there is a slight increase as the length of index increases as the search algorithm has more number of tokens to search for. But as our algorithm uses an optimized and efficient binary search tree method for searching the increase is not remarkable.

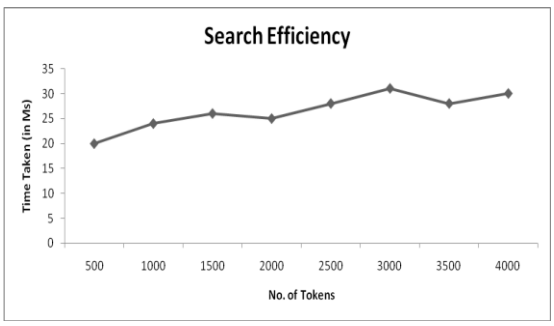

Fig.4. Time Taken to search Index on the Scale of Number of Tokens When Number of Tokens in the Search Query Are Fixed (t=10)

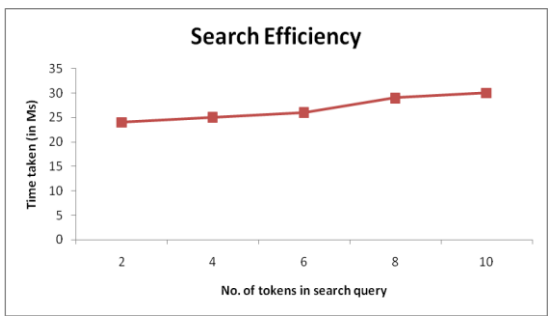

Fig.5. Time Taken to Search Index on the Scale of Number of Tokens in the Search Query with a Fixed Number of Tokens in the Dictionary (m=4000).

Figures 5 shows the search time required to search the documents when number of tokens in the search query is varied and number of tokens in the dictionary is fixed to 4000. Varying the number of tokens in the search query doesn't has any remarkable effect but there is a slight increase in time with the increase in number of tokens in the search query.

D. *Trapdoor generation till completion*

Fig 6 shows trapdoor generation till completion time which includes the total time taken after the user enters the queried keywords till he gets back the top-k encrypted results. This entire trapdoor time includes the encrypting the queried keywords, searching the encrypted index for queried keywords, and finally sending the top-k files to the user in the encrypted manner.

For 500 tokens in the dictionary it requires 85 milliseconds to search the given keywords in the search query.while for 4000 tokens in the dictionary it takes 360 ms to to exexute the query and provide the users with required encrypted set of files.

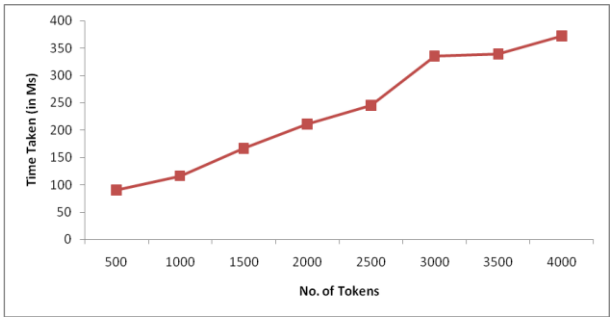

Fig.6. Time Taken to Create Trapdoor Till Encrypted Top-K Files Sent to the Users on the Scale of Number of Tokens When Number of Tokens in the Search Query Are Fixed (t=10)

VI. Conclusion

We have proposed a very simple and efficient multi-keyword symmetric searchable encryption scheme that could be used to create a searchable index and is able to make an efficient search over encrypted cloud data. The proposed scheme is more efficient than TRSE scheme [4]. It used a doubly encryption method which makes it comparatively more secure and efficient yet simple and easy to implement. Our technique also considers automatic updates as and when a new file is added, deleted or modified in the given file set. Finally we implanted the scheme on a real data set that shows reduction in time required to generate, update and search in an encrypted index.

References

[1] D. Boneh, G. Crescenzo, R. Ostrovsky, G. Persiano, " Public Key Encryption with Keyword Search", Proc. International Conference on Theory and Applications of Cryptographic Techniques (Eurocrypt), 2004.

[2] D. Song, D. Wagner, and A. Perrig, "Practical Techniques for Searches on Encrypted Data," Proc. IEEE Symp.

Security and Privacy, 2000.
[3] R. Curtmola, J.A. Garay, S. Kamara, and R. Ostrovsky, "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions," Proc. ACM 13th Conf. Computer and Comm. Security (CCS), 2006.
[4] C. Wang, N. Cao, K. Ren, W. Lou, " Enabling Secure and Efficient Ranked Keyword Search over Outsourced Cloud Data", IEEE Transactions On Parallel and Distributed Systems, Vol. 23, No. 8, August 2012.
[5] Swaminathan, Y. Mao, G.-M. Su, H. Gou, A.L. Varna, S. He, M. Wu, and D.W. Oard, "Confidentiality-Preserving Rank-Ordered Search," Proc. Workshop Storage Security and Survivability, 2007.
[6] P. Golle, J. Staddon, and B. Waters, "Secure Conjunctive Keyword Search over Encrypted Data," Proc. Second Int'l Conf. Applied Cryptography and Network Security (ACNS), pp. 31-45, 2004.
[7] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-Preserving Multikeyword Ranked Search over Encrypted Cloud Data," Proc. IEEE INFOCOM, 2011.
[8] H. Hu, J. Xu, C. Ren, and B. Choi, "Processing Private Queries over Untrusted Data Cloud through Privacy Homomorphism," Proc. IEEE 27th Int'l Conf. Data Eng. (ICDE), 2011.
[9] E. Christal Joy, K. Indira, " Multi keyword Ranked Search over Encrypted Cloud Data", International Journal of Applied Engineering Research, Vol. 9, No. 20, pp 7149-7176, 2014.
[10] J. Yu, P. Lu, Y. Zhu, G. Xue, M. Li, "Toward Secure Multikeyword Top-k Retrieval over Encrypted Cloud Data", IEEE Transactions on Dependable and Secure Computing, vol. 10, No. 4, July 2013.
[11] Raghavendra S, Geeta C M, Rajkumar Buyya, Venugopal K R, S S Iyengar, and L M Patnaik, MSIGT: Most Significant Index Generation Technique for Cloud Environment, Proceedings of the 2015 Annual IEEE India Conference (INDICON 2015), Delhi, India, Dec. 17-20, 2015.
[12] V. Arora, S.S. Tyagi, " Analysis of Symmetric Searchable Encryption Over Encrypted Cloud Data", International Journal of Computer Applications (0975-8887), vol. 127, no. 12, pp 46-51, October 2015.
[13] A. Singhal, "Modern Information Retrieval: A Brief Overview," IEEE Data Eng. Bull., vol. 24, no. 4, pp. 35-43, 2001.
[14] C.D. Manning, P. Raghavan, H. Schutze," An Introduction to Information Retrieval", Cambridge University Press, Online edition ©, 2009 Cambridge UP.
[15] S. L. Pallickara, S. Pallickara, and M. Zupanski, "Towards Efficient Data Search and Subsetting of Large-Scale Atmospheric Datasets," Future Generation Computer Systems, vol. 28, no. 1, pp. 112–118, 2012.
[16] R. Curtmola, J.A. Garay, S. Kamara, and R. Ostrovsky, "Searchable Symmetric Encryption: Improved Definitions and Efficient constructions, "Proc. 13th conf. computer and communication security (CCS), 2006.
[17] A. Boldyreva, N. Chenette, Y. Lee, A. Neill, " Order Preserving Symmetric Encryption", Advances in Cryptology- CRYPTO 2011, 31st Annual International Cryptography Conference, LNCS, Springer, 2011.
[18] L. Xiao, I. Yen, "Security Analysis for Order Preserving Encryption Schemes", IEEE, 2012.
[19] D. Dubin, "The Most Influential Paper Gerard Salton Never Wrote," Library Trends, vol. 52, no. 4, pp. 748-764, 2004.
[20] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully Homomorphic Encryption over the Integers," Proc. 29th Ann. Int'l Conf. Theory and Applications of Cryptographic Techniques, H. Gilbert, pp. 24-43, 2010.
[21] Z. Xia, X. Wang, X. Sun, Q. Wang, "A Secure and Dynamic Multi-Keyword Ranked Search Scheme Over Encrypted Cloud Data", IEEE transaction on Parallel and Distributed Systems Vol: pp No: 99, 2015.
[22] "NSF Research Awards Abstracts 1990-2003," http://kdd.ics.uci. edu/databases/nsfabs/nsfawards.html, 2013.

Authors' Profiles

Vasudha Arora is a Ph. D. Scholar and currently working as an assistant professor in Department of Computer Science & Engineering in Manav Rachna International University, Faridabad, India. She did her Masters of Engineering in Computer Science & Engineering from Panjab University, Chandigarh, and Bachelors from M.D. University, Haryana, India. Her research interests include cloud computing, ad-hoc networks, information security.

Dr. S S Tyagi is having 24 years of experience and currently working as Professor and Head, Department of Computer Sc & Engg. in Manav Rachna international University, Faridabad, India. He did his Ph.D. (Computer Engg.) from Kurukshetra University, M. E from BITS Pilani and B. Tech from Nagpur University in Computer Sc. & Engg.

He is a senior member of IEEE and other professional societies like CSI, QCI, and ASQ. He is a consultant to software/ IT companies and academic institutes/universities and guiding in the area of research of cloud computing, Big Data and Internet of things."

How to cite this paper: Vasudha Arora, S.S. Tyagi,"An Efficient Multi-keyword Symmetric Searchable Encryption Scheme for Secure Data Outsourcing", International Journal of Computer Network and Information Security(IJCNIS), Vol.8, No.11, pp.65-71, 2016.DOI: 10.5815/ijcnis.2016.11.08