ISSN:0975-9646

Dynamic Multi-Keyword Ranked Searchable Security Algorithm Using CRSA and B-Tree

Prasanna B T[#1], C B Akki[*2]

[#]*Department of ISE, EPCET*
Associate Professor, Bengaluru, INDIA-560049

[*]*Department of ISE, SJBIT*
Professor, Bengaluru, INDIA-560060

***Abstract*— With the advantage of storage as a service many enterprises are moving their valuable data to the cloud, since it costs less, easily scalable and can be accessed from anywhere any time. The trust between cloud user and provider is paramount. We use security as a parameter to establish trust. Cryptography is one way of establishing trust. Searchable encryption is a cryptographic method to provide security. In literature many researchers have been working on developing efficient searchable encryption schemes. In this paper we explore some of the effective cryptographic techniques based on data structures like CRSA and B-Tree to enhance the level of security, hence trust. We tried to implement the search on encrypted data using Azure cloud platform.**

Keywords: Searchable Encryption, Multi keyword, CRSA, B tree, Azure

I INTRODUCTION

Cloud computing is one way of computing. Here the computing resources are shared by many users. The benefits of cloud can be extended from individual users to organizations. The data storage in cloud is one among them. The virtualization of hardware and software resources in cloud nullifies the financial investment for owning the data warehouse and its maintenance. Many cloud platforms like Google Drive, iCloud, SkyDrive, Amazon S3, Dropbox and Microsoft Azure provide storage services.

Security and privacy concerns have been the major challenges in cloud computing. The hardware and software security mechanisms like firewalls etc. have been used by cloud provider. These solutions are not sufficient to protect data in cloud from unauthorized users because of low degree of transparency [4]. Since the cloud user and the cloud provider are in the different trusted domain, the outsourced data may be exposed to the vulnerabilities [4] [14] [5]. Thus, before storing the valuable data in cloud, the data needs to be encrypted [2]. Data encryption assures the data confidentiality and integrity. To preserve the data privacy we need to design a searchable algorithm that works on encrypted data [13].

Many researchers have been contributing to searching on encrypted data. The search techniques may be single keyword search or multi keyword search [11]. In huge database the search may result in many documents to be matched with keywords. This causes difficulty for a cloud user to go through all documents and have most relevant documents. Search based on ranking is another solution, wherein the documents are ranked based on their relevancy to the keywords [3]. Economical searchable encryption techniques help the cloud users especially in pay-as-you use model. The researchers combined the rank of documents with multiple keyword search to come up with efficient economically viable searchable encryption techniques. In searchable encryption related literature, computation time and computation overhead are the two most frequently used parameters by the researchers in the domain for analysing the performance of their schemes. Computation time (also called "running time") is the length of time required to perform a computational process for example searching a keyword, generating trapdoor etc. Computation overhead is related to CPU utilization in terms of resource allocation measured in time.

In this research work, we analyse the security problems in cloud storage and propose a solution for the same. Our contribution can be summarized as follows:

1. For the first time, we define the problem of secure ranked keyword search over encrypted cloud data, and provide such an effective protocol, which fulfils the secure ranked search functionality with no relevance score information leakage against keyword privacy.
2. Thorough security analysis showed that our asymmetric based ranked searchable encryption scheme using CRSA and B-tree indeed enjoys “as-strong-as-possible” security guarantee compared to previous searchable symmetric encryption (SSE) schemes.
3. Extensive experimental results demonstrate the effectiveness and efficiency of the proposed solution.

In the remainder of this paper, the following information is presented: in Section II, literature review in related area is discussed. Section III describes problem formulation. Section IV presents our proposed search schemes. Security analysis and performance analysis are presented in Section V. Finally, in Section VI, the paper concludes with some suggestions for future work.

II LITERATURE SURVEY

The encryption on data is an effective way to protect the confidentiality of data in cloud. But when it comes to searching, efficiency gets low. In literature many research works are not efficient in searching specially for complex queries. This inefficiency may lead to leakage of valuable

information to unauthorized peoples. Song et al, for the first time proposed the practical symmetric searchable method based on cryptography. In this scheme the file is encrypted word by word. To search for a keyword user sends the keyword with same key to the cloud. The drawback of this scheme is that the word frequency will be revealed. Goh et al tried to overcome the drawback of Song's scheme by constructing secure index table using pseudorandom functions and unique document identifier randomized bloom filters. Bosch et al worked on the concept given by Goh et al. and introduced the concept of wild card searches. The drawback of this scheme is that bloom filters may introduce false positives. In Chang's et al proposed scheme, an index is built for each document. The scheme is more secured compared to Goh's scheme since number of words in a file is not disclosed. The limitation of this scheme is that it is less efficient and does not support arbitrary updates with new words. Golle et al scheme allows multiple keyword searches with one encrypted query. But this scheme is not practical. Curtmola et al for the first time proposed the concept of symmetric searchable encryption (SSE), later on Kamara et al proposed an extended version of SSE called dynamic SSE (DSSE), where addition and deletion of documents can be performed in index table. All these schemes are based on single keyword search [22] [23] [24].

The first public key encryption with keyword search (PEKS) was proposed by Boneh et al. The scheme suffers from inference attack on trapdoor encryption method. Baek et al, Rhee et al improved hardness of security of Boneh's scheme. Baek's scheme introduces the concept of conjunction of keyword search. The public key encryption methods are computationally time consuming and complex that makes these algorithms inefficient. In Yang et al scheme the encrypted data is searched by individual users using a unique key allotted to them. The scheme suffers from key management. Boneh et al discussed functional encryption and related to conjunctional search, range queries and subset queries. Katz et al scheme is an updated version of Boneh's scheme and discussed predicate encryption for inner products and supports both conjunctions and disjunctions search on encrypted data [22] [23] [24].

There are many searching techniques implemented in the cloud. These techniques support only exact keyword search. Using fuzzy search the exact keywords are displayed along with similarity keywords and is analysed in [8]. This work concentrates on solving the problems of the user who searches the data with the help of fuzzy keyword on cloud.

Curtmola et al. [16], proposed a method where an inverted index (implemented using linked list) having document identifiers is maintained for each keyword. Every node in the list stores information about the position and the decryption key of the next node. The nodes from all inverted indexes are encrypted with random keys and are randomly inserted into an array. With this, by knowing position and decryption key of the first node of an inverted index, it is possible to find all documents which include the corresponding keyword. To improve the efficiency of the above scheme, top-k single keyword retrieval schemes are proposed in the literature [17].

Much work has been done in privacy preserving multi-keyword search on encrypted data for cloud computing sector. In [11], a model is proposed that solves the problem of effective secure ranked keyword search over encrypted cloud data. Here, it proposes an existing cryptographic primitive, order-preserving symmetric encryption (OPSE). The disadvantages of this technique are: does not support multi-keyword, does not include IDF (define) for the calculation of scores, does not use advanced crypto techniques.

S.Buyrukbilen et al [18], introduce the first method that provides ranked results from multi-keyword searches on public-key encrypted data. By avoiding a linear scan of the documents and by parallelizing the computations to the possible extent, this method reduces the computational complexity of public key cryptosystem. The scheme encrypts keyword information of each document in a bloom filter [19], and hierarchically aggregate (using homomorphic encryption) the individual indexes into a tree structure. Client will do the query processing, and traverse the tree in best-first manner. The query is hidden from the server or cloud provider by using an efficient private information retrieval (PIR) protocol [20]. In this method the indexes are split into multiple chunks, and use several CPUs in parallel to execute the user queries efficiently.

Wenhai Sun et al. [21], proposed a MRSE scheme that works on similarity based ranking. Here search index is created on the basis of term frequency and vector space. Search index is used for multi keyword search and ranking the search result. Search efficiency is improved by applying tree structure on index.

The future work being multi-keyword semantic search over the encrypted data has been represented in [6]. Considering the large number of data users and documents in the cloud, it is necessary to allow multiple keywords in the search request and return documents in the order of their relevance to these keywords. Here, privacy-preserving multi-keyword ranked search over encrypted data in cloud computing (MRSE) is proposed where among various multi-keyword semantics, it chooses the efficient similarity measure of coordinate matching and hence uses the cryptographic techniques. Therefore, it lacks integrity check of rank order in search result and privacy in stronger threat model. Synonym based multiple keywords ranked search over encrypted cloud data using balanced binary tree is proposed in [15]. Here author used symmetric encryption method for designing searchable encryption scheme and used b-tree for indexing.

Although many researchers across the globe have been investigating to identify a suitable privacy preserving technique for cloud domain, none of these solutions guarantee 100 percent privacy. There exists a wide range of research challenges. We therefore chose to work towards meeting this challenge.

III PROBLEM FORMULATION

Searchable Encryption (SE) schemes maintain the confidentiality and privacy of owner's data by facilitating

searching keywords directly on encrypted data. Users can upload their encrypted data to cloud. Later, the authorized users can perform private keyword search on encrypted data in cloud. Multiple domains like cryptography, indexing, storage etc. are involved in devising efficient, secure, SE algorithms over encrypted files. The participants of a secure search model in a cloud, typically involves data owner, data user and cloud server. Data owner encrypts the files and corresponding keywords based index files by using any known cryptographic algorithms. Both the encrypted files and index files are uploaded to the cloud server. The trapdoors (encrypted keywords) are used to search encrypted files by cloud server in cloud database.

A. System Model

Our system consists of 3 entities data owner, data user and the cloud server as shown in Figure 1.

1. Data owner encrypts the data files for securing the data in cloud using Commutative RSA (CRSA) before uploading into the cloud. They also define the access rights for the user who want to access those documents. The access right is a 2-state variable: permission granted or permission denied. Data owner creates an index tree based on B tree and encrypts the tree using CRSA.

2. Cloud server stores the encrypted data files and encrypted index tree. It accepts the encrypted keywords (trapdoor) and returns the matching data file based on their relevance.

3. Data user can search for encrypted data files in cloud with encrypted keywords (trapdoor). The purpose of using encrypted keywords is that even the cloud server must not be able to infer the contents of data files.

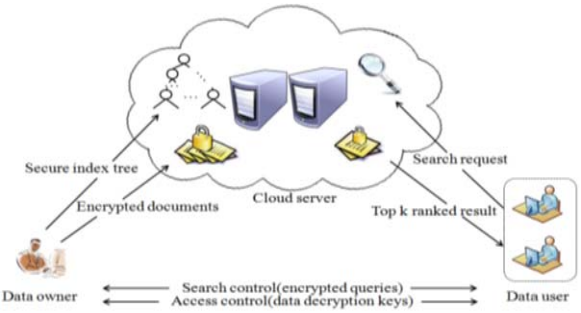

Figure 1: Searchable Encryption Architecture using CRSA

B. Threat Model

The threat model for our search scheme adopts "honest-but-curious" cloud server, that is the cloud server "honestly" follows the protocol specification, but it is "curious" to infer and analyze data (including indexes) in its storage and message flows received during the protocol in order to learn additional information.

C. Design Goals

The proposed solution addresses the following requirements

1. The search on encrypted document/file must be fully secure and cloud server must not be able to infer the contents of the documents in any way.

2. The search results must be ranked in order of relevance

To enable ranked searchable encryption for effective utilization of outsourced and encrypted cloud data under the aforementioned model, our system design should achieve the following security and performance guarantee. Specifically, we have the following goals: 1) Ranked keyword search: to explore different mechanisms for designing effective ranked search schemes based on the existing searchable encryption framework; 2) Security guarantee: to prevent 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; 3) Efficiency: above goals should be achieved with minimum communication and computation overhead.

Existing systems:

Existing searchable encryption schemes [6] [15] [38] allow a user to securely search over encrypted data through keywords. These techniques support multi keyword search. The similarity measure "coordinate matching" in MRSE [6] has some drawbacks when used to evaluate the document ranking order. First, it takes no account of term frequency such that any keyword appearing in a document will present in the index vector as binary value 1 for that document, irrespective of the number of its appearance. Obviously, it fails to reflect the importance of a frequently appeared keyword to the document. Second, it takes no account of term scarcity. Usually a keyword appearing in only one document is more important than a keyword appearing in several ones. In addition, long documents with many terms will be favoured by the ranking process because they are likely to contain more terms than short documents. Hence, due to these limitations, the heuristic ranking function, "coordinate matching", is not able to produce more accurate search results. More advanced similarity measure should be adopted from plaintext information retrieval community. On the other hand, the search complexity of MRSE is linear to the number of documents in the dataset, which becomes undesirable and inefficient when a huge amount of documents are present.

Proposed system:

For our system, we choose the B-tree as indexing data structure to identify the match between search query and data documents. Specially, we use inner data correspondence, i.e., the number of query keywords appearing in document, to evaluate the similarity of that document to the search query. Each document is converted to a balanced B-tree according to the keywords and encrypted using CRSA. Whenever user wants to search, he/she creates a trapdoor for the keywords. Our aim is to design and analyse the performance of multiple keywords ranked search scheme using Commutative RSA algorithm and B-tree data structure for searchable index tree.

We designed a scheme based on secured ranked multiple keyword search over encrypted cloud data using CRSA. Further, we analysed its performance over B-tree based searchable index tree. In [6] [38], authors have studied the performance of RSA algorithm on B tree. We have used Microsoft's Azure platform to emulate the proposed system and to study its performance.

D. Preliminaries

Commutative Encryption (CRSA): The RSA cryptosystem is one of the optimum public key cryptography approaches. However, its overall robustness gets limited due to one way encryption and majority of existing RSA schemes suffer from reorder issues. Therefore, in order to make this system least complicated and more efficient, an approach called Commutative RSA has been proposed. In this scheme, the order in which encryption has been done would not affect the decryption if it is done in the same order. Encryption is the standard method for making a communication private. With the many cryptographic approaches, our system follows the commutative RSA algorithm. The mathematical scheme for performing this encryption is described by a pseudo algorithm presented below.

Let us consider two prime numbers $Prime\_P_p^{CRSA}$ and $Prime\_Q_q^{CRSA}$ initialized amongst all the group members. Let G_A and G_B represent the group members required to communicate over the documents. To compute the encryption keys and decryption key pairs of the commutative RSA algorithm the parameters $Param\_N^{CRSA}$ and $Param\_\phi^{CRSA}$ are computed using the following

$$Param\_N^{CRSA} = [(Prime\_P_p^{CRSA}) \times (Prime\_Q_q^{CRSA})]$$

$$Param\_\phi^{CRSA} = [(Prime\_P_p^{CRSA} - 1) \times (Prime\_Q_q^{CRSA} - 1)]$$

From the above equations it is clear that

$Param\_N_X^{CRSA} = Param\_N_Y^{CRSA}$ and $Param\_\phi_X^{CRSA} = Param\_\phi_Y^{CRSA}$ for X and Y.

The encryption key pair of X and Y are represented as $(Param\_N_X^{CRSA}, Param\_E_X^{CRSA})$ and $(Param\_N_Y^{CRSA}, Param\_E_Y^{CRSA})$ is to be obtained.

The $Param\_E^{CRSA}$ is obtained by randomly selecting numbers such that it is a co-prime of $Param\_\phi^{CRSA}$ or in other terms

$$Fn_{GCD}(Param\_E^{CRSA}, Param\_\phi^{CRSA}) = 1$$

Where $Fn_{GCD}(u, v)$ represents the greatest common divisor function between two variables u and v.

The decryption key pair of X and Y is represented by $(Param\_N_X^{CRSA}, Param\_D_X^{CRSA})$ and $(Param\_N_Y^{CRSA}, Param\_D_Y^{CRSA})$ and the parameter $Param\_D^{CRSA}$ is computed based on the following equation

$$Param\_D^{CRSA} = (Param\_E^{CRSA})^{-1} Mod(Param\_N^{CRSA})$$

Let Enc_U represent the encrypted data U. The encryption operation is defined as follows

$$Enc_U = U^{Param\_E^{CRSA}} Mod(Param\_N^{CRSA})$$

The commutative RSA decryption operation on the encrypted data V is defined

$$Dec_V = V^{Param\_D^{CRSA}} Mod(Param\_N^{CRSA})$$

B- Tree: A B-tree is a data structure as shown in Figure 2. The tree contains index nodes and leaf nodes. All leaf nodes are at the same level (same depth). Each index nodes contain keywords and pointers. Each node except root node in a B-tree with order n must contain keys between n to 2n keys. Each node also contains (number of keys + 1) pointers to its child nodes. If the root node is an index node then it must have at least 2 children. The insertion, deletion, search operations takes only logarithmic time.

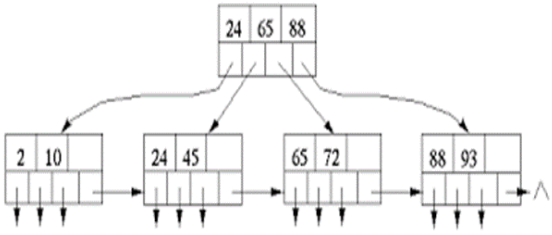

Figure 2: B tree data structure

IV SEARCHABLE ENCRYPTION SCHEME

To design an efficient multi-keyword searchable encryption scheme based on public key cryptography, we included the following modules.

Encryption Module: By using CRSA, data in a file can be updated dynamically without affecting the overall performance of searching on B-tree. If the encrypted indexed data is modified, re-indexing for the whole data is not needed. Similarly there is no need of re-encrypting the files in the database whenever the file is modified. This is a desirable feature as it reduces the computation time.

Data owner first generates secret and public key pair (EK, DK) using a standard public-key encryption scheme ie CRSA. Then owner makes the public key DK public and keeps the secret keys EK private. Documents {D | D1, D2,…, Dn} are encrypted using EK resulting in a ciphertexts {C | C1,C2,….Cn}. The generated C is stored in cloud database.

The constructed index based on B tree is also encrypted using CRSA, i.e each derived keywords {W| w1,w2,….wn}from a document is indexed in a tree and encrypted using CRSA. This results in a set of encryptions {e| e1,e2,..en} where each ej (for $1 \leq j \leq m$) is defined as E_wj = CRSA_Enc (EK, wj), where E_wj denotes encrypted keyword.

Index Module: Index structures for huge datasets cannot be stored in main memory. Disk is a possible alternative. Storing it on disk requires different approach. The solution is to use more branches to reduce the height of the tree. For this we used B-tree data structure for each document. B-tree is a data structure of order n. The nodes are filled from n to 2n keys. Nodes are always at least half full of keys. The keys are within each node. A list of pointers is inserted between keys. These pointers help to navigate through tree. In general, a node with k keys has (k+1) pointers.

The design for creating and querying the index tree can be given by ALGORITHM-1, ALGORITHM-2 and ALGORITHM-3. ALGORITHM-1 and ALGORITHM-2 are used to create an index tree and ALGORITHM-3 describes how search can be performed on index tree.

ALGORITHM-1

Btree_insert (root, Key, Object_value)

Input: root pageID of a B-tree, the key and the value of an object.

//Inserts when Object_value doesn't exist in a B-tree

1. NODE = Disk_Read (root).
2. if NODE_x is full

(a) y = Allocate_Page(), z = Allocate_Page().

(b) Locate the middle object o stored in NODE_x.

- Move the objects to the left of object o into NODE_y.
- Move the objects to the right of o into NODE_z.
- If NODE_x is an index page,
- Then move the child pointers of NODE_x accordingly.

(c) NODE_ x: child [1] = NODE_y, NODE_x: child [2] = NODE_z.

(d) Disk_Write (NODE_x); Disk_Write (NODE_y); Disk_Write (NODE_z).

3. end if
4. Insert_Not_Full (NODE_x; Key; Object_value).

ALGORITHM-2

Insert_Not_Full (NODE_x, key, Object_value)

Input: an in-memory page NODE_x of a B-tree, the key and the value Object_value of a new object.

// This algorithm inserts when page of NODE_ x is not full.

// Insert the new Object_value into the sub-tree rooted by NODE_x.

1. if NODE_ x is a leaf page

(a) Insert the new Object_value into NODE_x, keeping Object_values in sorted order.

(b) Disk_Write (NODE_x).

2. else

(a) Find the child pointer NODE_x: child[i] whose key range contains Key.

(b) NODE_w = Disk_Read (NODE_x: child [i]).

(c) if NODE_w is full

- NODE_y = Allocate_Page ().
- Locate the middle object o stored in NODE_w. Move the objects to the right of o into NODE_y.
- If NODE_w is an index page, move the child pointers accordingly.
- Move o into NODE_x. Add a right child pointer in NODE_x pointing to NODE_y
- Disk_Write (NODE_x); Disk_Write (NODE_y); Disk_Write (NODE_w).
- If (Key < o. key), call Insert_Not_Full(NODE_w; KEY; Object_value);

else, call Insert_Not_Full(NODE_y; Key; Object_value).

(d) else Insert_Not_Full(NODE_w; Key; Object_value).

(e) end if

3. end if

The Disk_Read in ALGORITHM-1 reads the corresponding page from disk to memory and returns the location in memory that gets stored in node NODE_x. If the node NODE_x is full, allocate memory for 2 nodes and store the corresponding addresses in NODE_y and NODE_z. Find the middle object stored in NODE_x. Split the node NODE_x by moving the values to the left of middle object o in to NODE_y and right values of middle object o to NODE_z. If NODE_x is index page then move the pointers accordingly i.e. NODE_x: child [1] = NODE_y, NODE_x: child [2]=NODE_z. The NODE_x is promoted to higher level. This increases the height of the tree. Write all the values back to disk from memory by using Disk_Write operation. Else if NODE_x is not full then call Insert_Not_Full function. Insert_Not_Full function finds the path from root to leaf, and inserts the Object_value in to the leaf. Using the key range of the child pointer where the key of new object exists, the algorithm follows the pointer. The algorithm loops recursively on each of those nodes which are not full along the path till leaf level. The Object is inserted at the leaf level.

Search Module: Searching a B-tree is like searching a binary tree. Here instead of making a binary branching decision at each node, we make a multiway branching decision according to the number of the node's children.

Let's suppose cloud server has received n encrypted documents of this form, so that it now holds a set of encrypted documents {C|c1,C2,…,Cn}. Now, if user wants to retrieve the documents with keyword w, he just needs to generate a secret trapdoor encrypted using CRSA i.e Enc_CRSA (w1, w2, ..). The trapdoor containing the encrypted keywords is sent as token to the server. The server then uses this trapdoor to match the encrypted keywords in index tree node. If match found stores the pointer to that document in encrypted database. The search continues for other encrypted keywords. The following ALGORITHM-3 gives the stepwise information about how search will be done on B-Tree.

ALGORITHM-3

Search_Query (root, trapdoor)

Input: root, trapdoor containing keyword to be searched.

Output: pointer to the documents containing the keywords; NULL if non-exist.

1. NODE_x = Disk_Read (root).
2. if NODE_x is an index node

(a) If there is an object o in NODE_x such that o: key = keyword, return o: value.

(b) Find the child pointer x: child [i] whose key range contains key.

(c) Return Search_Query(NODE_x:child[i], key).

3. else If there is an object o in NODE_x such that o:key = keyword, return o:value.

Otherwise, return NULL.

4.end if.

The ALGORITHM-3 takes trapdoor and root as input and searches for the keywords match in cloud database. The Disk_Read reads the corresponding root page from disk to memory and returns the location in memory that gets stored in node NODE_x. If NODE_x is index node then trapdoor is checked to see for keyword match. If found returns the

corresponding document pointed by the node. Otherwise based on keyword, search will move to the child of NODE_x using pointers. The search continues recursively. Otherwise if NODE_x represents leaf then return the pointer to document if search succeeds otherwise NULL.

Ranking Module: In large databases, it is quite likely that the keyword might be matching with more number of documents. It is cumbersome for a user to decrypt and go through all the documents. Therefore there is a need for ranking the documents based on their relevance to the keywords. In our scheme we used (TF * IDF) to rank the documents. TF is the term frequency i.e. occurrence of keywords in a document and IDF is inverse document frequency i.e. total number of documents divided by number of documents containing the keyword. Similarity measure is used to find the rank based on relevance. For this, we maintain two vectors one for storing TF weight and other to store IDF weight.

Platform Used: Microsoft Azure is a cloud service provider. It provides storage as a service to the customers. Azure architecture contains roles, i.e. the worker role and the web role as shown in Figure 3. The web role is used for designing UI, whereas worker role is used to run background asynchronous applications. The workers in the B-tree provide search encryption services which support the multi-keyword search application. The workers are defined as $WR = \{Wr_1, Wr_2, Wr_3, \ldots\ldots, Wr_n\}$ where Wr_n is n^{th} search provided by the worker. The encrypted index tree is created by tree builder function using encrypted keyword contents (worker A). Cloud users (web role) enter the keywords for search. The B-tree based tree search algorithm i.e. searches for the encrypted keywords in index tree. The search results are obtained using query on index tree and using tree search algorithm. Relevance score for ranking the search results is calculated using search algorithm, the index tree and database (worker B) as explained above in rank module. The system architecture of the azure cloud search over an encrypted data by the worker and web role is shown in Figure 3.

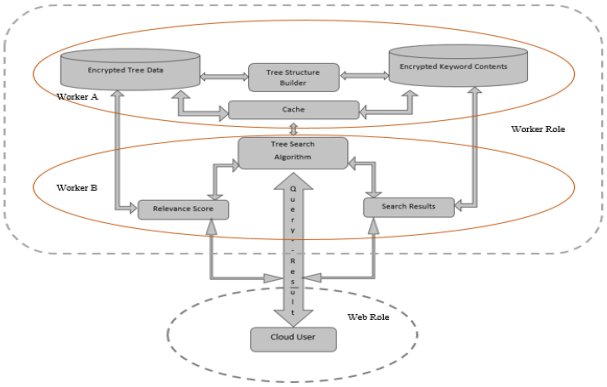

Figure 3: Architecture of searchable encryption scheme in Azure

V PERFORMANCE ANALYSIS

The security of the designed system is provided by using CRSA. As long as private key (encrypted) is kept secret the cloud provider cannot deduce index tree or documents set. Since trapdoor is also encrypted using CRSA, the provider cannot make out the keywords inside the trapdoor maintaining the confidentiality at index and query level. The documents in cloud storage are also protected, since documents are encrypted using CRSA. Without having the decryption key it is highly hard to decrypt the documents thus provides security at storage level.

To be useful and usable, databases must support operations, such as search, deletion and insertion of data. For large organizations the databases are huge in size and cannot be maintained entirely in memory. By using balanced B-trees to construct the index for the data we can improve the search efficiency. B-tree minimizes the disk I/O (disk read and disk write) by copying a block of data (page) containing many records at a time into memory. This in turn improves the search efficiency. Asymptotically, Searching an unsorted database without indexing will have a worst case running time of O(n), where n represents the number of keywords. If the same data is indexed with a B-Tree, the same search operation will run in logarithmic time i.e O(log n).

Result Analysis: The privacy preserved multi-keyword search based on the encrypted cloud data has been designed. The system model presented has been developed on Visual Studio 2010 framework 4.0 with C#. The overall system has been developed and implemented with Microsoft Azure cloud platform.

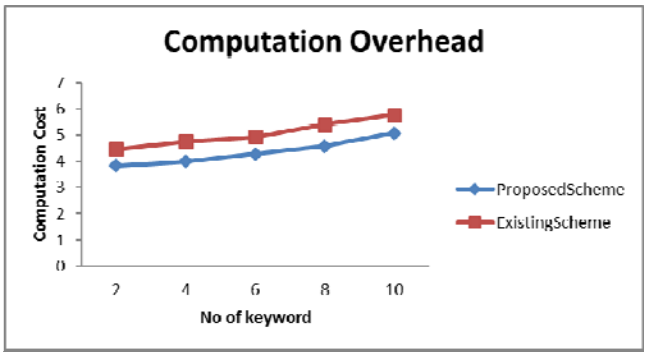

Figure 4: Computation Overhead

Figure 4 depicts the computation overhead in seconds based on the number of keywords. In this study, we compared the performance of our proposed system with the RSA based system proposed in [15]. Results clearly show that even for 10 keywords, the overhead computation using CRSA is low as compared to the RSA based system [15]. For example, RSA based system takes approximately 4.5 seconds for searching 2 keywords, whereas our proposed CRSA based scheme takes only 4 seconds. The computation cost for search increases linearly in both schemes. But from Figure 4 it is evident that our proposed CRSA based scheme performs better even under increased number of keywords.

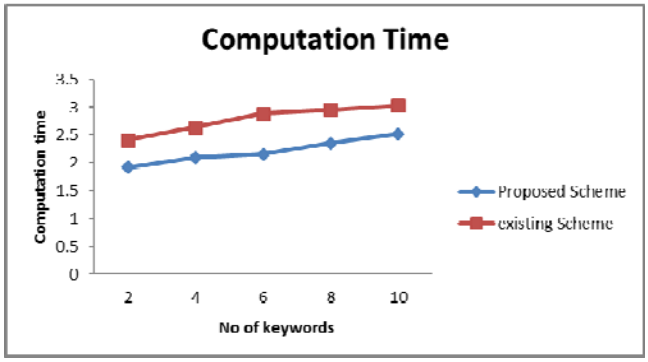

Figure 5: Time Comparison

The graph in Figure 5 plotted above makes the comparison of the search computation time in seconds of our proposed system against the RSA based system. For two keywords search, the time taken by the RSA based scheme is approximately 2.5 seconds, whereas our proposed system takes approximately 0.5 seconds less. As the number of keywords increased for search, the computation time for search also increases linearly in both schemes. But CRSA based scheme is found to perform better.

Thus it is evident that encryption algorithm CRSA with B Tree as index tree performs better than RSA and B tree *Combination.*

VI Conclusion and Future Work

This work uses CRSA asymmetric algorithm for encrypting data files and index tree based on B-tree. CRSA increases the data security and improves privacy of data by its commutative nature. Using CRSA, data in a file can be updated dynamically without affecting the overall performance of searching on B-tree. In our proposed system, if encrypted data is modified, re-encrypting for the whole data is not needed. This is a desirable feature as it reduces the computation time.

The future work would concentrate on using Elliptic Curve Cryptography (ECC) encryption technique for better performance. Further, we intend to analyze the behavior of our proposed system(s) for multiuser environment.

References

[1] M. Armbrust et al., 'Above the Clouds: A Berkeley View of Cloud Computing,' Feb 2009.

[2] S. Kamara and K. Lauter, 'Cryptographic cloud storage,' in RLCPS, January 2010, LNCS. Springer, Heidelberg.

[3] A. Singhal, 'Modern information retrieval: A brief overview,' IEEE Data Engineering Bulletin, vol. 24, no. 4, pp. 35–43, 2001.

[4] Cloud Security Alliance, 'Security Guidance for Critical Areas of Focus in Cloud Computing,' http://www.cloudsecurityalliance.org, 2009.

[5] R. Brinkman, 'Searching in encrypted data,' in University of Twente, PhD thesis, 2007.

[6] Ning Cao; Cong Wang; Ming Li; Kui Ren; Wenjing Lou, 'Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data,' Parallel and Distributed Systems, IEEE Transactions on , vol.25, no.1, pp.222,233, Jan. 2014

[7] Dawn Xiaoding Song; Wagner, D.; Perrig, A., 'Practical techniques for searches on encrypted data,' Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on ,doi: 10.1109/SECPRI.2000.848445 vol., no., pp.44,55, 2000

[8] J. Li et al., 'Fuzzy Keyword Search Over Encrypted Data in Cloud Computing,' Proc. IEEE INFOCOM '10 Mini-Conf., San Diego, CA, Mar. 2010.

[9] M. Li et al., 'Authorized Private Keyword Search over Encrypted Data in Cloud Computing,' 31st Int'l. Conf. Distributed Computing Systems, 2011, pp. 383–92.

[10] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, 'Public key encryption with keyword search,' in Proc. of EUROCRYPT, 2004.

[11] C. Wang et al., 'Secure Ranked Keyword Search Over Encrypted Cloud Data,' Proc. ICDCS '10, 2010

[12] Wenjun Lu; Varna, A.L.; Min Wu, 'Confidentiality-Preserving Image Search: A Comparative Study Between Homomorphic Encryption and Distance-Preserving Randomization,' Access, IEEE, vol.2, no., pp.125,141, 2014

[13] W. K. Wong, D. W. Cheung, B. Kao, and N. Mamoulis, 'Secure knn computation on encrypted databases,' in Proc. of SIGMOD, 2009.

[14] K. Ren, C. Wang, and Q. Wang, 'Security Challenges for the Public Cloud,' IEEE Internet Computing, vol. 16, no. 1, pp. 69-73, 2012.

[15] Zhangjie Fu et al, 'Multikeyword Ranked Search Supporting Synonym Query over Encrypted Data in Cloud Computing', IEEE Conference, 2013.

[16] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, 'Searchable symmetric encryption: improved definitions and efficient constructions,' in ACM CCS, 2006.

[17] P. Naresh, K. Pavan kumar, and D. K. Shareef, 'Implementation of Secure Ranked Keyword Search by Using RSSE,' International Journal of Engineering Research & Technology (IJERT) ISSN: 2278-0181 Vol. 2 Issue 3, March – 2013.

[18] S.Buyrukbilen and S.Bairas, 'Privacy preserving ranked search on public key encrypted data,' in Proc. IEEE International Conference on High Performance Computing and Communications (HPCC), November 2013.

[19] B. H. Bloom, 'Space/time trade-offs in hash coding with allowable errors,' Communications of the ACM, vol. 13, no. 7, 1970, pp. 422–426.

[20] C. Gentry and Z. Ramzan, 'Single-database private information retrieval with constant communication rate,' in ICALP, pp. 803–815.2005.

[21] Sun, W., Wang, B., Cao, N., Li, M., Lou, W., Hou, Y.T., Li, H., 'Privacy-preserving multikeyword text search in the cloud supporting similarity-based ranking,' Proceedings of the 8th ACMSIGSAC symposium on Information, computer and communications security, ACM, pp. 71–82.2013.

[22] Prasanna B.T, C.B. Akki, 'A Survey on Homomorphic and Searchable Encryption Security Algorithms for Cloud Computing,' Communicated to International Journal of Information Technology and Computer Science, November, 2014.

[23]Prasanna B.T, C.B. Akki, 'A Comparative Study of Homomorphic and Searchable Encryption Schemes for Cloud Computing,' Communicated to International Journal of Communication Networks and Distributed Systems, November, 2014.

[24] Prasanna B.T, C.B. Akki, 'A Survey on Challenges and Security Issues in Cloud,' Presented in conference presented in Conference on Evolutionary Trends in Information Technology, May 20-22 2011, at Visvesvaraya Technological University, Belgaum, Karnataka.