Protected Search Model for Encrypted Data in Cloud Computing

Juber Mirza[1], **Avdesh Kumar Sharma**[2]
[1, 2] Department of Computer Science & Engineering
[1, 2] SVITS- SVVV Indore

Abstract- *Cloud services have also become renowned because of reduction in the cost of storage and flexibility of use, but there is the risk of data loss, misuses and theft. Reliability and security of data stored in the cloud are a matter of concern, specifically for critical cloud applications and ones for which data security and privacy is important. Sensitive data commonly have to be encrypted before outsourcing. Efficient and secure search on encrypted data is a primary concern in computer science. In Existing Symmetric solution of searchable encryption, if disclosure of cryptographic key during the key exchange, can affect the security and privacy of data and data owner. Most of the search engines utilize the concept of indexing and the store the information on the local disk. This may increase the need of additional storage space a compression technique is required to reduce the storage overhead. The proposed work, constructs a new methodology of searching using Efficient and privacy preserving fuzzy keyword search algorithm over encrypted cloud data while maintaining keyword privacy. Use edit distance to quantify keywords similarity and cultivate an advanced technique on constructing fuzzy keyword sets. We use the hybrid crypto system for encrypting data file that utilizes the benefits of both symmetric key and public key cryptographic methods using AES and RSA. Construct the inverted index for fuzzy keyword set using hash tree methods and Applying Variable Byte Encoding (VBE) for index compression, so that we can reduce storage space on inverted index, and improve the search performance. We use bloom filters to perform simple as well advanced search in constant time. Which proves to be much faster and efficient than the traditional one. Combining both the symmetric-key and public-key algorithms provide greater security and Indistinguishability under Adaptive-Chosen Cipher text Attack. Through rigorous security analysis, we demonstrate that our proposed solution is secure and privacy-preserving, while correctly realizing the goal of the fuzzy keyword search. The implementation of the system is performed using the JAVA technology and for the deployment of the developed SaaS is performed using the public cloud (Open Shift). For obtaining the performance of the developed technique the performance of the system is also evaluated and reported in terms of precision, recall, f-measures, time complexity and space complexity. The performance outcomes demonstrate the improved performance of searchable encryption technique over the cloud environment.*

Keywords- Cloud security, Protect Outsource data, Privacy on Cloud, Trustworthy Cloud.

I. INTRODUCTION

The invention of the computer opens a number of rich domains for comforting the human beings and changing the life much rapidly. The need of human being in terms of computational ability is increases continuously and for supporting them a different kinds of efforts are made. Among the cloud computing is one of the most popular infrastructures for supporting the computational needs. This chapter provides the basic overview about the conduced study around the cloud computing.

Due to the growth of the technology and need of scalable computing and storage need to develop new techniques of storage and computation. The cloud computing provides a significant solution for the scalable computational need and the storage solution. The scalability of the storage or the computation is achieved by sharing of physical and logical resources. Due to these functional aspects the cloud server providers are collaborating with a number of data centers and keep their data on other servers for reducing the data management overheads. This concept of the data management is termed as the data outsourcing [1]. Most of the time due to security reasons and the privacy concerns the data stored on other servers in the cryptographic manner. Such kind of cloud storage is sometimes also known as the cryptographic cloud.

In this presented work the data retrieval process from the cryptographic cloud is studied in detail. In addition of that some key issues are also addressed by which the current security over the cryptographic cloud becomes suspected. Thus, in order to manage these issues a new technique of privacy preserving and secure technique of data retrieval is proposed in this work for development. The proposed concept of secure, searchable encryption technique improves the security and privacy of all the ends, i.e. data owner, storage and retrieval process.

This section provides the formal introduction of the proposed work and their domain aspects in the further sections the key addressed issues, motivation and objectives are reported to understand the proposed work and their need in the current generation data security and privacy management.

II. LITERATURE SURVEY

Cloud computing is an evidence of the growth of computational technology. Continuous and improved approaches are responsible for designing and development of their essential services. Therefore a number of technological experts and organizations are contributing for improving more and more the new generation technology. Clouds, by their nature, bring together many of the different technologies that have evolved over the last 30 years. Therefore, various lessons learned in other information technology sectors might have relevance to the future of cloud computing. Several clients needed to access information on separate terminals, but these technologies are expensive enough. To save money and the computational cost required to find a method by which multiple users share CPU according to time. In the 1950s, "time-sharing" is considered for corporations with computational power and resources. From there, to the stage of cloud development is a stepwise process [3].

The ideas of pioneers like J.C.R. Licklider were given an idea in the 1960s. They introduced the idea for an “Intergalactic computer network”. Licklider developed ARPANET (Advanced research Projects Agency Network) envisioning computation in the form of a global network and hoped one day everyone could access data and programs from anywhere. The term “Cloud Computing” was first used by professor Ramnath Chellappa in late 1997, that is a start of the development new technological era.

The explosive growth of data and once all of that data comes into being. Required a manner to store it all securely and allow end-users to access it efficiently. That demand is what’s putting a silver lining on the cloud. Within just a few years, companies began switching from hardware to cloud services because they were attracted to benefits like a reduction in capital costs as well as an easing in IT man power issues. In 1999, Salesforce.com became the first site to deliver applications and software over the Internet.

In 2002 AWS providing an advanced computational system using cloud services and in 2006, Amazon introduced the Elastic Compute Cloud (EC2) as a commercial web service. The developer-based platform is first developed by Salesforce.com in 2007, with Force.com. That is, able to build and run all of their business apps and website through the cloud. The Google App Engine brought low-cost computing and storage services, popularizing the concept in 2009. Google Apps allows people to store documents within the cloud. Microsoft followed with Windows Azure.

In 2010 with Database.com a platform again made for developers. Using this cloud allowed computing services to be used on any device or platform in any programming language. In order to allowing users to sync photos, apps, music and documents across a string of devices Apple put a concept by developing the iCloud.

This section provides the historical development and improvements on the computational cloud growth in the next section a detailed description is presented for understanding the cloud services and their distribution.

III. PROPOSED WORK

3.1 Domain Description

As the amount of data for storage and retrieval on the third party server increases the need of sensitivity and privacy management is necessary to improve. Therefore a number of efforts are placed recently to promote the concept of secure search techniques among them the searchable cryptography is a well technique for achieving both. In this presented work the main aim is to develop an enhanced searchable cryptographic technique by which not only the security and privacy is kept in track the search performance in terms of efficiency and search time is also improved. The proposed technique is also much helpful for reducing the storage overhead to maintain the index for search terms.

Therefore the proposed technique involves the different pre-defined concepts and techniques that are helps to improve the cryptographic search processes. As usual the cryptographic process involves additional effort for process and converting the data therefore the additional time and storage is need to compensate with some improvement techniques. Therefore the given section involved the reporting, indexing technique, compression technique, cryptographic technique and the fuzzy concept for improving the search. Additionally a combined data model is also reported in this chapter that arranged in a specific manner to improve the security, privacy and the search relevancy.

This section provides the basic overview of the proposed technique and the involved components for improving the cryptographic data search. In addition of that the next section provides the detailed understanding about the different aspects involved with the proposed searchable cryptographic data model.

3.2 Proposed Technique

This section provides the detailed description of the proposed work model for improved cryptographic search processes. The figure 3.1 shows the entire concept of the proposed model that involved data file holder, input shopper and cloud server. Given a compilation of N encrypted data file C = (DF_1, DF_2, . . . , DF_N) reserved in the cloud server, a predefined collection of isolated keywords W = {w_1,w_2, ...w_N}, the cloud server provides the search service for the authentic users by the unscripted data C. Assume the approval among the data possessor and data users is properly done. An authentic user enters in an appeal to retrieve selected data files of individual passion. The cloud server is liable for calculating the seeking request to a group of input files, where every file is inverted indexes by a file ID and associated to a group of access. The fuzzy keyword search pattern recovers the search outcome, as per the following rules:

A. If the user's searching input appropriately resembles the pre-set keyword, the server is expected to return the files consists of the relevant keyword.
B. If there exist format inconsistencies and/or typos in the searching files, the server will return the convenient probable outcome relied upon pre specified similarity semantics.

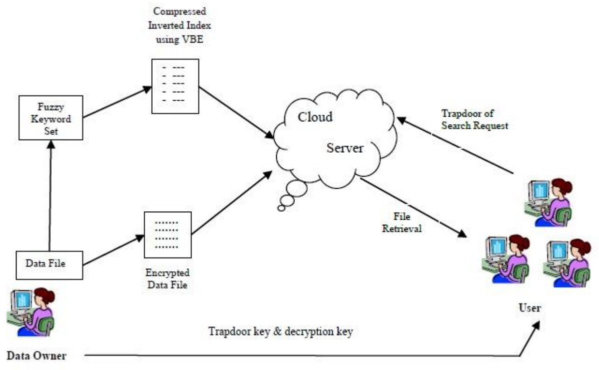
Compressed Inverted Index using VBE
Fuzzy Keyword Set
Cloud Server
Trapdoor of Search Request
File Retrieval
Data File
Encrypted Data File
User
Trapdoor key & decryption key
Data Owner

Figure 3.1: Proposed working model

Before describing the working of the cryptographic search process need to understand the used terminology for entire process model.

A. KeyGen(s): Takes a security parameter, s, and generates a Recipient public/private key pair Rpub, Rpri.

B. $Encrypt_{SK}$(DF): Encrypt data file using shared secret key to get encrypted data file.

C. Encrypt (R_{pub}, W): For a public key Rpub and a word W, produces a searchable encryption of W.

D. Trapdoor(R_{pri}, W): Given Recipient's private key and a word W produces a trapdoor TW.

E. Test(R_{pub}, S, TW): Given Recipient's public key, a searchable encryption S = Encrypt(R_{pub}, W'), and a trapdoor TW = Trapdoor(R_{pri}, W), outputs 'yes' if W = W' and 'no' otherwise.

F. Decrypt (R_{pri}, SK): Decrypt EPUB(SK) using Receiver private key PRI to retrieve SK.

G. $Decrypt_{SK}$ (DF):Use the retrieved SK as the decryption key to decrypt E_{SK} (DF) to get original message.

3.2.1 The Efficient and Privacy Preserving Fuzzy Keyword Search Algorithm

A. Takes a security parameter, s, and generates KeyGen(s) a shared secrete key SK for symmetric encryption AES-128 and Recipient public/private key pair Rpub, Rpri, for asymmetric encryption RSA.

B. Collect the Data File DF_i to be indexed, Parse and tokenize the text, Pre-process the tokens, Remove the stop word, Normalize the message keyword Wi.

C. To construct an index for wi with edit space d, the data owner first constructs a fuzzy keyword set SWi,d using the wildcard based technique. Encrypt the keyword Encrypt(R_{pub},W) using RSA algorithm, and construct an inverted index using the HASH TREE method.

D. Apply Variable Byte Encoding for compressed the inverted index.

E. The data owner encrypts the Data File DF_i as $Encrypt_{SK}$(DF) with a shared secret key using AES-128 bit algorithm.

F. The index table $\{(\{T_{w'i}\}_{w'i \in S_{wi,d}}, Encrypt_{SK}(DF))\}_{wi \in W}$ and encrypted data files are outsourced to the cloud server for storage.

G. Send Authentication Key to Recipient.

H. To search with (w, k), the authenticate user figure out the trapdoor set $\{T_{w'}\}_{w' \in S_{w,k}}$, Trapdoor($R_{pri}$,W) , where Sw,k is also borrowed from the wildcard-based fuzzy set development. He after that delivers $\{T_{w}'\}_{w'} \in S_{w,k}$ Trapdoor(R_{pri}, W) to the server.

I. On getting the search appeal $\{T_{w}'\}_{w' \in S_{w,k}}$, Test($R_{pub}$, S, TW)the server correlates them with the index table using Bloom Filter and returns all the desirable encrypted Data File file identifiers { $Encrypt_{SK}$(DF)} according to the fuzzy keyword definition.

J. The user decrypts the shares key using his/her private key Decrypt(R_{pri}, SK) and get the shared key and then decrypt the Data File using secrete key $Decrypt_{SK}$(DF). Returned results and retrieves relevant files of interest.

3.3 Concept Study In this section the different techniques and concepts are studied by which the proposed searchable cryptographic is becomes feasible for implementation.

3.3.1 Edit Distance: There are various methods to quantitatively quantify the string similarity. The edit distance ed(w1,w2) between two keywords w1 and w2 is the total number of operations needed to transform one of them into the other. The three primitive operations are 1) Insertion:- inserting a single character into a word. 2) Deletion:- deleting one character from a word; 3) Substitution:- changing one character to another in a word; Given a keyword w, we let Skw, d denotes the set of words w′ propitiate ed(w, w′) ≤ d for a certain integer d.

3.3.2 Wildcard-based Fuzzy Set Construction: The wildcard-based fuzzy set of wi with edit distance d is denoted as Swi,d = $\{S'_{wi,0}, S'_{wi,1}, \cdots, S'_{wi,d}\}$, where $S'_{wi,\tau}$ notify the set of words $w_{'i}$ with τ wildcards. Note that every wildcard represents an edit operation on wi. For instance, the keyword AZURE with the pre-set edit distance 1, its wildcard-based fuzzy keyword set can be created as SAZURE,1 = {AZURE, *AZURE, *ZURE, A*ZURE, A*URE, · · · , AZUR*E, AZUR*, AZURE*}. The total number of variants on AZURE constructed is only 13 + 1, instead of 13 × 26 + 1 as in the exhaustive enumeration approach when the edit distance is set to be 1.

The size of Swi,1 will be only 2l + 1 + 1 , for a given keyword wi with length l. As compared to (2l + 1) × 26 + 1 obtained in the straightforward approach. The larger the pre-set edit distance, the more storage overhead can be reduced: with the same setting of the example in the straightforward approach, the proposed method can help reduce the amount of storage of the index from 30GB to approximately 41MB. In case the edit distance is set to be 2 and 3, the size of $S_{wi,2}$ and $S_{wi,3}$ will be C^1 l+1+C^1 l · C^1 l +2C^2 l+2 and C^1 l + C^3 l + 2C^2 l + 2C^2 l · C^1 l . In other words, the number is only O(l d) for the keyword with length l and edit distance d.

3.3.3 Hybrid Crypto System: The proposed method is a method to combine strengths of public-key algorithms with symmetric key cryptographic algorithms. The design of the proposed hybrid crypto system is based on performing encryption and decryption using symmetric key algorithm, but uses a public key algorithm to encrypt the symmetric key before performing key transfer. In this hybrid crypto system, a data block is encrypted using any of the standard symmetric-key cryptographic algorithms using AES. The symmetric encryption key used to encrypt the block is then encrypted using a public key crypto algorithm (like RSA) using that destination's public key. The encrypted symmetric key is then concatenated with the encrypted data block and the whole data block is sent to the destination system. The encrypted data block sent to the destination contains the encrypted data and the corresponding publicly encrypted symmetric key to decrypt that data. The block sent to the destination. The destination uses its private key to decrypt the symmetric key first and then uses that symmetric key as the decryption key to decrypt the received data block.

Table 3.1 Encryption process

Encryption Process
Input**:** Data File (DF), Symmetric Key (SK), Destination Public Key (PUK) Output: Encrypted Message. Encrypted Shared Key. **Process:** A. Encrypt data file using shared secret key to get encrypted data file ESK (DF). B. Encrypt shared Key using destination public key to get encrypted shared key E_{PUB}(SK). C. Encrypted data file send to the cloud server. D. Encrypted shared key send to receiver.

Table 3.2 Decryption process

Decryption Process
Input: Encrypted Data File. Encrypted Shared Key. Output: Original Data File. Shared Secret Key. **Process**: A. Decrypt E_{PUB}(SK) using private key PRI to retrieve SK (Note: This should be done using the same public key algorithm which is used at source) B. Use the retrieved SK as decryption key to decrypt ESK (DF) to get

original message. (Note: This should be done using the same symmetric-key crypto algorithm which is used at the source).

3.3.4 Inverted Index: Inverted lists are usually used to index underlying documents to retrieve documents according to a set of keywords efficiently. Since inverted lists are usually large, many compression techniques have been proposed to reduce the storage space and disk I/O time. An efficient index structure, using the hash tree method is reported here that reduces search complexity to O(logw) and after constructing index structure applies index compression technique.

Table 3.3 Forward index

Document ID	Keywords
1	w_2, w_5, w_7
2	w_1, w_2, w_4, w_6, w_8
…	……
N	w_2, w_5, w_6

Table 3.4 Inverted index

Document ID	Keywords
w_1	2,3,9
W_2	1,2,6,7
…	……
W_m	1,3,8

A hash tree is a tree of hashes in which the leaves are hashes of data blocks in, for instance, a file or set of files. Nodes further up in the tree are the hashes of their respective children. For example, in the picture hash 0 is the result of hashing the result of concatenating hash 0-0 and hash 0-1. That is, hash 0 = hash(hash 0-0 + hash 0-1) where + denotes concatenation. Most hash tree implementations are binary (two child nodes under each node) but they can just as well use many more child nodes under each node

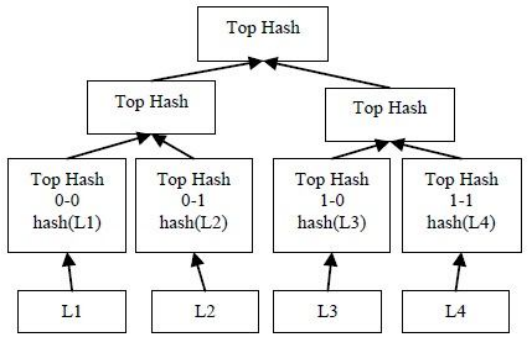

Figure 3.2 Hash tree

Usually, a cryptographic hash function such as SHA-2 is used for the hashing. If the hash tree only needs to protect against unintentional damage, much less secure checksums such as CRCs can be used. In the top of a hash tree there is a top hash (or root hash or master hash). Before downloading a file on a p2p network, in most cases the top hash is acquired from a trusted source, for instance a friend or a web site that is known to have good recommendations of files to download. When the top hash is available, the hash tree can be received from any non-trusted source, like any peer in the p2p network. Then, the received hash tree is checked against the trusted top hash, and if the hash tree is damaged or fake, another hash tree from another source will be tried until the program finds one that matches the top hash. The main difference from a hash list is that one branch of the hash tree can be downloaded at a time and the integrity of each branch can be checked immediately, even though the whole tree is not available yet. For example, in the picture, the integrity of data block 2 can be verified immediately if the tree already contains hash 0-0 and hash 1 by hashing the data block and iteratively combining the result with hash 0-0 and then hash 1 and finally comparing the result with the top hash. Similarly, the integrity of data block 3 can be verified if the tree already has hash 1-1 and hash 0. This can be an advantage since it is efficient to split files up in very small data blocks so that only small blocks have to be re-downloaded if they get damaged. If the hashed file is very big, such a hash tree or hash list becomes fairly big. But if it is a tree, one small branch can be downloaded quickly, the integrity of the branch can be checked, and then the downloading of data blocks can start.

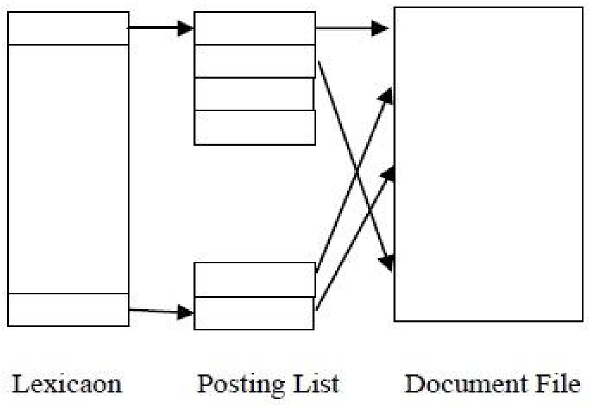

Figure 3.3 Structure of invIndex

Inverted file indices are probably the most common method used for indexing documents. Figure 3.3 shows the structure of an inverted file index. It consists first of a lexicon with one entry for every term that appears in any document. For each item in the lexicon the inverted file index has an inverted file entry (or posting list) that stores a list of pointers (also called postings) to all occurrences of the term in the main text. Thus to find the documents with a given term need only look for the term in the lexicon and then grab its posting list. Boolean queries involving more than one term can be answered by taking the intersection (conjunction) or union (disjunction) of the corresponding posting lists. Required to consider the following important issues in implementing inverted file indices A. How to minimize the space taken by the posting lists? B. How to access the lexicon efficiently and allow for prefix and wildcard queries? C. How to take the union and intersection of posting lists efficiently?

3.3.5 Inverted File Compression: The total size of the posting lists can be as large as the document data itself. In fact, if the granularity of the posting lists is such that each pointer points to the exact location of the term in the document, then in effect recreate the original documents from the lexicon and posting lists (i.e., it contains the same information). By compressing the posting lists both reduce the total storage required by the index, and at the same time potentially reduces access time since fewer disk accesses will be required and/or the compressed lists can fit in faster memory. This has to be balanced with the fact that any compression of the lists is going to require on-the-fly un-compression, which might increase access times. The key to compression is the observation that each posting list is an ascending sequence of integers (assume each document is indexed by an integer). The list can therefore be represented by an initial position followed by a list of gaps or deltas between adjacent locations. For example: Original posting list: elephant: [3, 5, 20, 21, 23, 76, 77, 78]

Posting list with deltas: elephant: [3, 2, 15, 1, 2, 53, 1, 1]

The advantage of using the deltas is that they can usually be compressed much better than indices themselves since their entropy is lower. To implement the compression on the deltas only needs some model describing the probabilities of the deltas. Based on these probabilities we can use a standard Huffman or Arithmetic coding to code the deltas in each posting list. Models for the probabilities can be divided into global or local models (whether the same probabilities are given to all lists or not) and into fixed or dynamic (whether the probabilities are fixed independent of the data or whether they change based on the data).

```
VBENCODENUMBER(n)

bytes ← {}
while true
    do PREPEND(bytes, n mod 128)
       if n < 128
          then BREAK
       n ← n div 128
bytes[LENGTH(bytes)] += 128
return bytes
```

```
VBENCODE(numbers)

bytestream ← {}
for each n ∈ numbers
    do bytes ← VBENCODENUMBER(n)
       bytestream ← EXTEND(bytestream, bytes)
return bytestream
```

```
VBDECODE (bytestream)

numbers ← {}
n ← 0
for i ← 1 to LENGTH(bytestream)
    do if bytestream[i] < 128
          then n ← 128× n + bytestream[i]
          else n ← 128× n + (bytestream[i] − 128)
               APPEND(numbers, n)
               n ← 0
return numbers
```

3.3.6 Bloom Filter: Bloom Filter provides an answer in the Constant time and this constant time is the time to hash. The method is explained along with implementation details as how bloom filter is working on the inverted index to give better performance. Bloom filter is a space-efficient probabilistic data structure which is used to test whether an element is a member of a set. Bloom filter is used to perform membership queries. A Bloom Filter can be used to represent a set of elements. Bloom Filter makes use of a bit vector and a set of hash functions to represent the data set, so as to query a given data effectively Bloom filter allow false positive but the space saving when the probability of an error is controlled. One can first add elements of a set to the structure (Bloom Filter). Later on, the structure (Bloom Filter) can be queried for the membership of elements. The elements themselves are not stored in the Bloom Filter, only their membership may be queried by an application. A Bloom Filter consists of an array of m-bits, which are all initially set to 0.

	0	0	0	0	0	0	0	0	

Figure 3.4 Bloom filter initialized with 0

There must also be K different hash functions each of which maps or hashes some set element to one of the m array positions with a uniform random distribution. The hash functions should be independent, uniformly distributed and they should be as fast as possible. There must also be K different hash functions each of which maps or hashes some set element to one of the m array positions with a uniform random distribution. The hash functions should be independent, uniformly distributed and they should be as fast as possible.

A. Inserting a word in Bloom Filter

A.1. Let the word to be inserted be "hello".

A.2. Find the indexes using K different hash functions one by one.

A.3. Set 1 at these K indexes.

B. Searching a word in Bloom Filter

B.1. Let the word to be searched is "hello".

B.2. Find the indexes using K different hash functions.

B.3. If value at all of these indexes is 1 then "hello" is present, otherwise "hello" is not present.

Here, for the word "hello" we get the same indexes by all K hash functions. At these K indexes, the Bloom Filter contains 1indicating that "hello" is present.

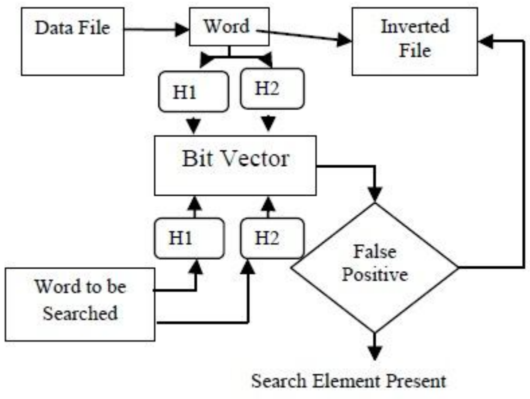

Figure 3.5 Bloom filter construction

C. Advantages of Bloom Filter

A Bloom Filter does not store the elements, but only the membership of elements.

C.1. Space Efficient: The space required to represent a set using a bloom filter is sufficiently less compared to other data structures like linked lists, hash tables, arrays, etc. The amount of space needed to store bloom file is very small as compared to the data. [11]

C.2. Time Efficient: The time needed to check whether an element is present or not is independent of the number of elements present in the set. We just need to find the K indexes using K hash functions.

D. False Positive Rate of Bloom Filter False negatives are not possible in bloom filters. False positives are possible, but their frequencies can be controlled. If m is the number of bits in the array, k is the number of hash functions and n is the number of elements inserted the False Positive Rate (FPR) can be approximated as:

FPR= {1-(e-K_n)/m}K

Hence false positive scan reduce by increasing size of bloom filter or by increasing the number of hash functions. However, this may slow the retrieval process. Hence an appropriate combination of file size and hash functions are required, which vary from application to application. Other techniques also exist to reduce the false positive rate of bloom filters. Suppose Word1 and Word2 are the two words from the documents to be stored in the Bloom Filter. For ease and enhanced accepting only 2 words in this graphic illustration. H1 and H2 are the two hash functions used to store the cost in the bit vector. Now when the word is investigated its hash value is also considered using the two hash functions. If the value equivalent to both hash values is zero in the bit vector, formerly the portion is not created. This is the case of false negative, which is the basic idea behind the implementation of IIBF. As mentioned earlier main concern is the probability of false negative. In sequence to compute the possibility of false negative, let us first compute the possibility of false positive. Consider that a hash function selects each cluster position with equal probability. If m is the number of bits in the pattern, and k is the number of hash functions, then the possibility that a assertive bit is not set to 1 by a certain hash function during the insertion of an element is then (1 – 1/m)K. The probability that it is not set to 1 by any of the hash functions is (1 – 1/m)K. If n elements are inserted, the probability that a certain bit is still 0 is (1 – 1/m)K_n. The probability that it is 1 is therefore1 − (1 – 1/m)K_n. Now test membership of an element that is not available or present in the group. Each of the k pattern location calculated by the hash functions is 1 with a probability as above. The probability of all of them is 1, which would cause the algorithm to erroneously claim that the element is in the set, is often given as $(1 - 1/m)k_n \approx [1 - (e-k_n)/m\ ]K$. This is not strictly correct as it assumes independence for the probabilities of each bit being set. However, assuming it is a close approximation we have the probability of false positives decreases as m increases, and increases as n increases, where m is the number of bits in the array, and n is the number of inserted elements. So the probability of false negative would be (1- probability of false positive) i.e. (1-[1-1/m]K_n)K . A bit vector is array data structure that compactly stores bits. A bit array is effective at exploiting bit-level parallelism in hardware to perform operations quickly. A typical bit array stores k_w bits, where w is the number of bits in the unit of storage, such as a byte or word and k is some non-negative integer. If w does not divide the number of bits to be stored, some space is wasted due to internal fragmentation. A bit array is a mapping from some domain (almost always a range of integers) to values in the set 0, 1. The values can be interpreted as absent/present, valid/invalid, etc. The point is that there are only two possible values, so they can be stored in one bit. The array can be viewed as a subset of the domain (e.g. 0, 1, 2 . . .,n_1), where a 1 bit indicates a number in the set and a 0 bit a number not in the set.

IV. RESULTS ANALYSIS

4.1. Precision

In any data retrieval or search applications the precision is a fraction of search results which is most relevant to the input query. The provided precision of the proposed cryptographic search process is given using figure 4.1. This can be evaluated using the user feedback basis and also can be evaluated by the following formula.

$$\text{Precision}=\frac{\text{relevant document} \cap \text{retrieved documents}}{\text{retrieved documents}}$$

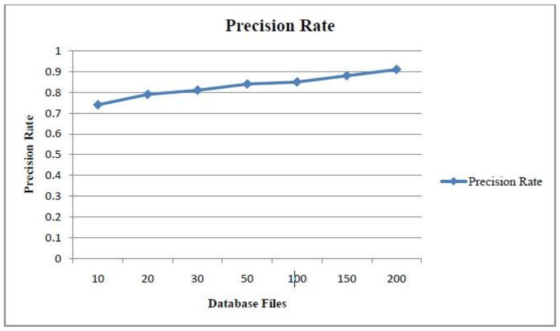

Figure 4.1 Precision rate

The obtained precision of the proposed system is given using figure 4.1 and the given table 4.1. In the given diagram the X axis shows the numbers of files are stored in the server system during making the search request for file and the Y axis contains the obtained precision rate according to the search outcomes. According to the given results the search accuracy of the proposed cryptographic system is improved as the amount of data in the database is increases. Therefore the proposed cryptographic cloud data model is adaptable and efficient for data hosting and secure data search and retrieval process.

Table 4.1 Precision rate

Amount of File in Storage	Precision Rate
10	0.74
20	0.79
30	0.81
50	0.84
100	0.85
150	0.88
200	0.91

4.2 Recall

In data retrieval application, or the search application recall values are measured for accuracy measurement in terms of relevant document retrieved or relevant data obtained according to the input user query. That defines the degree of accurate document extraction during the search process. This can be evaluated using the following formula

$$\text{Recall}=\frac{\text{relevant document} \cap \text{retrieved documents}}{\text{relevant document documents}}$$

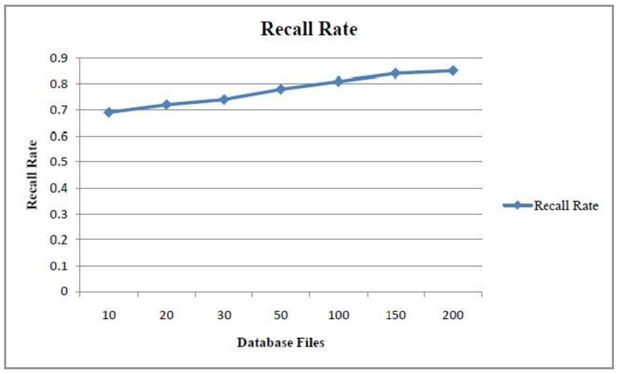

Figure 4.2 Recall rate

The recall rate of the proposed implemented system is demonstrated using the figure 4.2 and the table 4.2. In this diagram the X axis contains the amount of files stored in the cloud server and the Y axis shows the corresponding recall rate of the system. According to the obtained results the proposed technique improves the cryptographic search accuracy as the amount of data is increases over the data repository. Therefore the model is adoptable and accurate for use with the real world system for cryptographic data search processes.

Table 4.2 Recall rate

Amount of Data in Database	Recall Rate
10	0.69
20	0.72

30	0.74
50	0.78
100	0.81
150	0.84
200	0.85

4.3 F-measures

The f-measures of the system demonstrate the fluctuation in the computed performance in terms of precision and recall rates. The f-measures of the system can be approximated using the following formula

F - Measures =2. (Precision * recall)/ (precision + recall)

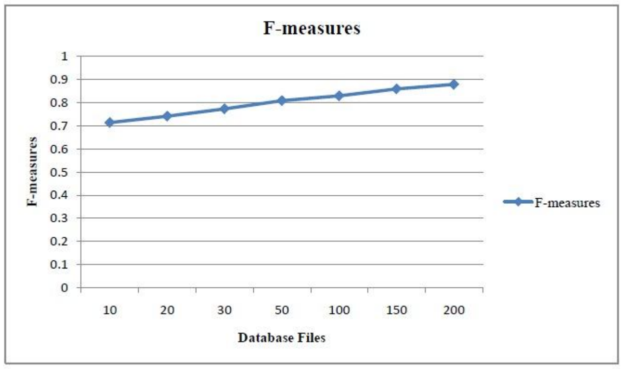

Figure 4.3 F-measures

The given figure 4.3 and the table 4.3 show the f-measures of the proposed cryptographic secure search process. That shows the consistency of the resultant from the system. The given figure contains the different amount of data size in the search repository and the corresponding search consistency is given in terms of f-measures in Y axis. According to the given results the performance of the search system is consistent and improving in the similar ratio as the amount of data to being search is increases. Therefore the proposed model is adaptable and efficient in terms of data retrieval relevancy.

Table 4.3 F-measures

Amount of Data in Database	F-measures
10	0.7141
20	0.7417
30	0.7734
50	0.8088
100	0.8295
150	0.8595
200	0.8789

4.6 Memory Usages

The amount of main memory required to successfully process the user request by the implemented system is considered here as the memory consumption of the system. The figure 5.4 and table 5.4 shows the memory consumption of the implemented system. According to the given outcomes the X axis contains the number of files in the search database and the Y axis contains the corresponding memory consumption of the system. According to the obtained results the memory consumption of the proposed system is increases as the amount of data is increases for processing. Therefore the significant amount of data search needs a significant amount of memory space to process accurately user requests.

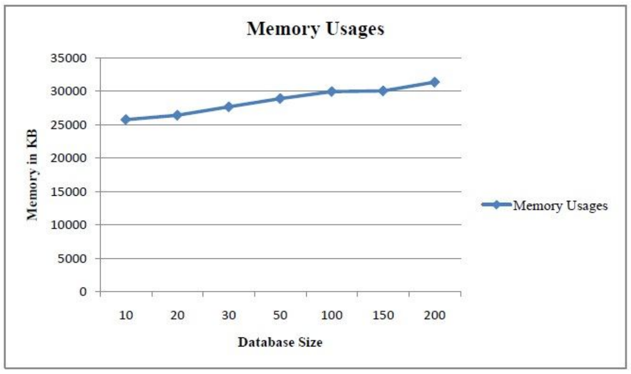

Figure 4.4 Memory usages

Table 4.4 Memory usages

Amount of Data in Database	Memory Usages
10	25719
20	26374
30	27636
50	28873
100	29910
150	30018
200	31331

4.5 Response Time

The amount of time required to process the user query request during searching of the encrypted files from the data base is termed here as the time consumption or the time complexity. The estimated response time of the system is demonstrated using the table 5.5 and the figure 5.5. In this diagram the X axis include the amount of files in the database for search and the Y axis shows the amount of time required for processing the user requests. The results show the amount of time for generating the response is increased as the amount of data during the search is increasing. The estimated reported here is given in terms of milliseconds and adoptable according to the preserving privacy during the search process and data recovery.

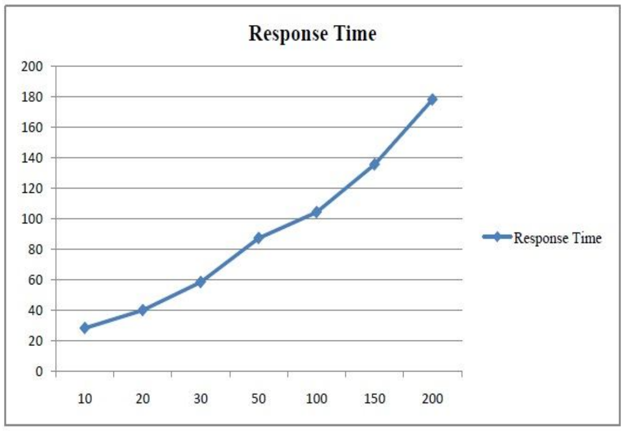

Figure 4.5 Response time

Table 4.5 Response time

Amount of Data in Database	Response Time
10	28.51
20	40.28
30	58.67
50	87.5
100	104.46
150	135.74
200	178.36

6. Performance of InvIndex

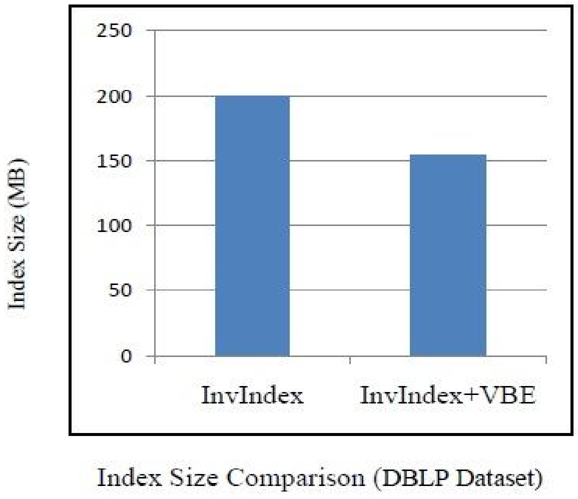

Figure 4.6 Size of index, uncompressed (64-bit integer) and compressed (VBE).

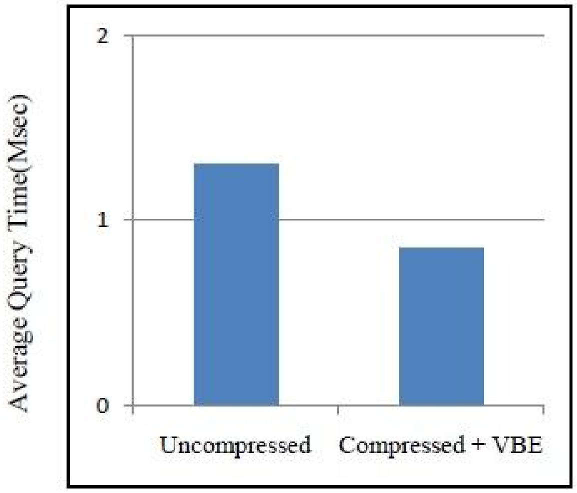

Figure 4.7 Search engine querying speed: compressed versus uncompressed

Data Set	Index Size	
	Uncompressed	Compressed (VBE)
Shakespe are	10.5 MB	2.7 MB (26%)
TREC45	2333.1MB	533.0 MB (22.85%)
GOV2	328.3 GB	62.1GB (19%)

Figure 4.8 Index Comparison Uncompressed Vs Compressed

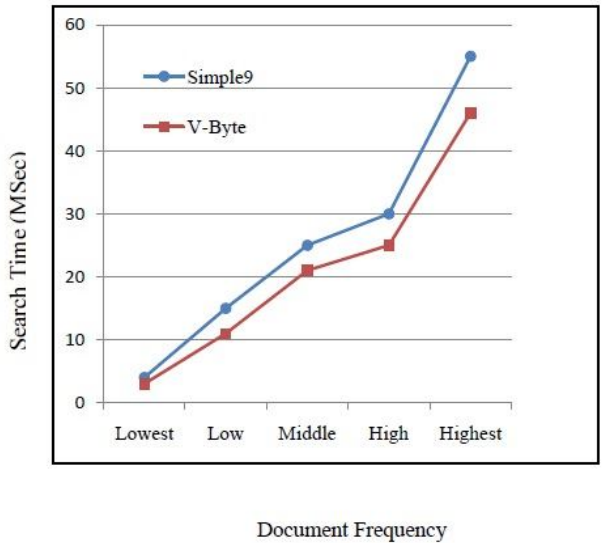

Figure 4.9 Simple9 Vs V-Byte

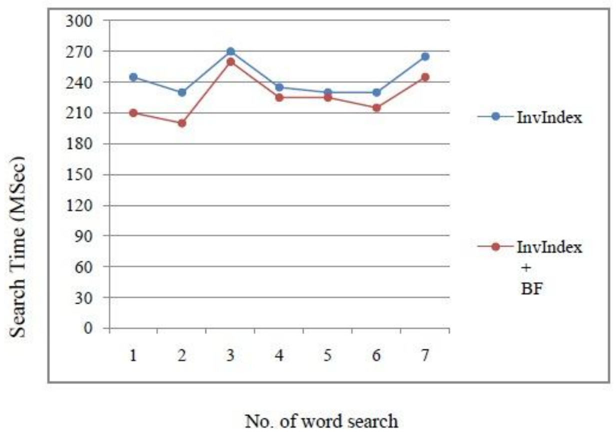

Figure 4.10 InvIndex Vs invIndex+ BF

4.7 Performance of Hybrid Encryption.

Table 4.6 Encryption technique comparison

FACTORS	**AES**	**RSA**	**HYBRID**
NATURE			
Type of algorithm	Symmetric	Asymmetric	Hybrid
Data size	Big	Small	Big
Block size	128,192, 256 bits	Minimum 512 bits	128 bits
Power consumption	Low	High	Low
Rounds	10,12,14	1	AES-14, RSA-1
Memory space	Less	More	More
SECURITY			
Confidentiality	Moderate	High	High
Integrity	None	High	High
Authentication	None	Moderate	High
Non-repudiation	None	None	High
Digital signatures	None	Possible	Available
Hash function	None	Possible	Available
TIME			
Encryption/Decryption	Fast	Slow	Fast
Speed of computation	Fast	Slow	Fast
Software Implementation	Fast	Slow	Fast
Hardware Implementation	Fast	Slow	Fast
KEY			
Key generation	Yes	Yes	Yes
Key storage	No	Yes	Yes
Multiple keys	No	Yes	Yes
Secret key	Yes	Yes	Yes
Public key	No	Yes	Yes
Shared key	Yes	No	No
Distribution public key	No	Yes	Yes
Distribution of secret key	Yes	No	No
Pair keys	No	Yes	Yes
Inverse keys	Yes	No	Yes

4.8. Performance of Bloom Filter

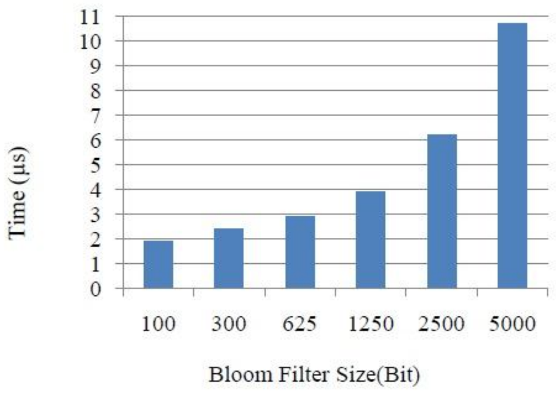

Figure 4.11 Time (microseconds) for computing the bitwise and of bloom filters for different size

V. CONCLUSION AND FUTURE WORK

5.1 Conclusion

The need of new generation computation and storage is fulfilled using the cloud servers. The cloud servers are providing the scalable techniques for extending their computational ability and storage capabilities. But the storage of the data on the third party server is suspected to the end user and the data owners. The users are always worried about the privacy and sensitivity of the stored data on the cloud servers. Therefore the cloud service providers are providing the cryptographic cloud storage for improving the security of data in third party servers.

Due to the storage placed on the cryptographic cloud then the normal search or data retrieval techniques are not suitable for relevant data search. Therefore the searchable encryption schemes are developed for making a search on the cryptographic data storage. In this presented work a detailed investigation about the cryptographic storage and searchable encryption techniques are performed. Additionally, using the different available techniques of searchable cryptography a new technique is proposed. That incorporates the technique of fuzzy keyword search for improving the search relevancy. Additionally a hybrid cryptographic process is also involved to secure the data during storage. But the search efficiency is also a major concern in data retrieval processes. Therefore the inverted index process is used with the hash tree function to reduce the time consumption and storage overhead for the developed indexing technique. The storage overhead is reduced due to compression of the inverted index to store the index information.

The implementation of the proposed technique is performed using the JAVA technology and for demonstrating the entire functional aspects the proposed system is demonstrated using the public cloud namely Open Shift. After implementation of the system the performance of the system is evaluated. The evaluated performance of the proposed cryptographic search process is summarized using the given table 5.1

Table 5.1: Performance of cryptographic search process

S. No.	Parameters	Remark
1	Precision	Improving the search precision rate as the amount of data is increases in storage
2	Recall	Recall rate shows the relevancy of the document during search which is also enhanced as the data increases
3	F-measures	The f-measures shows the consistency of search outcomes which is stable and not fluctuating with increasing size of data in storage
4	Memory usage	The memory consumption of the system is increases as the amount of data in search space is increases
5	Response time	The time consumption of system is depends on the amount of data in storage space

According to the obtained performance of the system the proposed model is working efficiently and can be used with various cryptographic search processes. Therefore that is adoptable, secure and efficient for different data retrieval systems.

5.2 Future work

In this presented work the issues of the searchable encryption techniques are investigated and a new process model is developed that are used to keep in track the privacy and security of the keyword based search process. In addition of that the approach is also promising for reducing the search time, storage overhead and search relevancy. Therefore the key aim of developing the enhanced search outcomes in cryptographic cloud is developed and successfully implemented. That can be further extendable with the authentication and access control during data retrieval. Therefore, in the near future the proposed data model is extended for integrating the access control and authentication for full proof security over the cloud storage.

REFERENCES

[1] Richard Chow, Philippe Golle, Markus Jakobsson, Ryusuke Masuoka, Jesus Molina, Elaine Shi, Jessica Staddon, "Controlling Data in the Cloud:Outsourcing Computation without Outsourcing Control", CCSW'09, November 13, 2009, Chicago, Illinois, USA.Copyright 2009 ACM.

[2] Christoph Bosch, Pieter Hartel, Willem Jonker, and Andreas Perer, "A Survey of Provably Secure Searchable Encryption", ACM Computing Surveys, Vol. 47, No. 2, Article 18, Publication date: August 2014.

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz,Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing", UC Berkeley Reliable Adaptive Distributed Systems Laboratory, http://radlab.cs.berkeley.edu/, February 10, 2009.

[4] Alexa Huth and James Cebula, "The Basics of Cloud Computing", © 2011 Carnegie Mellon University, Produced for US-CERT.

[5] Nariman Mirzaei, Cloud Computing, Fall 2008,Community Grids Lab, Indiana University Pervasive Technology Institute.

[6] Mike Ricciuti, "Stallman: Cloud computing is 'stupidity'", http://news.cnet.com/8301-1001_3-10054253-92.html.

[7] Cloud Storage: Non profit Technology Collaboration, Last Updated: 3/05/2013, http: //www.baylor.edu/ business/mis/nonprofits /doc.php/197132.pdf.

[8] [8] Seny Kamara, Kristin Lauter, "Cryptographic Cloud Storage", Microsoft Research Cryptographic Group, http://research.microsoft.com/en-us/people/klauter/ cryptostoragerlcps.pdf.

[9] Rajnish Noonia, ".Net Cryptography (Encryption / Decryption)", http://www.pixytech.com/rajnish/2013/04/net-cryptography-encryption-decryption/

[10] Dan Boneh, Giovanni Di Crescenzo, "Public Key Encryption with keyword Search", Supported by NSF and the Packard foundation.

[11] Majid Bakhtiari, Majid Nateghizad, Anazida Zainal, "Secure Search Over Encrypted Data in Cloud Computing", International Conference on Advanced Computer Science Applications and Technologies, 2013 IEEE.

[12] M.M. Tajiki, M.A.Akhaee, "Secure and privacy preserving keyword searching cryptography", 2014 11th International ISC Conference on Information Security and Cryptology (ISCISC),IEEE.

[13] Jin Li, Qian Wang, Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou, "Fuzzy Keyword Search over Encrypted Data in Cloud Computing", 2010 Proceedings IEEE,INFOCOM.

[14] Qiuxiang Dong, Zhi Guan, Liang Wu, and Zhong Chen, "Fuzzy Keyword Search over Encrypted Data in the Public Key Setting", WAIM 2013, LNCS 7923, pp. 729–740, 2013. Springer-Verlag Berlin Heidelberg 2013.

[15] Dr. narendra shekokar, kunjita sampat, chandni chandawalla, jahnavi shah, "implementation of fuzzy keyword search over encrypted data in cloud computing", international conference on advance computing technologies and applications 2015.

[16] Adnan Abdul-Aziz Gutub, Farhan Abdul-Aziz Khan, "Hybrid Crypto Hardware Utilizing Symmetric-Key & Public-Key Cryptosystems", 2012 International Conference on Advanced Computer Science Applications and Technologies.

[17] Saibal K. Pal, Puneet Sardana and Ankita Sardana, "Efficient Search on Encrypted Data Using Bloom Filter", 978-93-80544-12-0/14/$31.00 c 2014 IEEE.

[18] Abha Sachdev, Mohit Bhansali, "Enhancing Cloud Computing Security using AES Algorithm", International Journal of Computer Applications (0975 – 8887) Volume 67– No.9, April 2013.

[19] Razvan Rughinis, "Enhancing Performance of Searchable Encryption in Cloud Computing", CODASPY'13, February 18–20, 2013, San Antonio, Texas, USA. ACM 978-1-4503-1890-7/13/02.

[20] Qin Liu, Guojun Wang, Jie Wu, "Secure and privacy preserving keyword searching for cloud storage services", Journal of Network and Computer Applications, 2011 Elsevier Ltd. All rights reserved.

[21] Jian Wan, Shengyi Pan, “Performance Evaluation of Compressed Inverted Index in Lucene”, 2009 International Conference on Research Challenges in Computer Science, 978-0-7695-3927-0/09 $26.00 © 2009 IEEE.