Secure Similarity Based Document Retrieval System in Cloud

Greeshma N. Gopal, Mahendra Pratap Singh

Department of Computer Science, National Institute of Technology,Surathkal,India

greeshmang@gmail.com

mahoo15@gmail.com

Abstract— **The introduction of Cloud computing concept has been instrumental in reducing resource unavailability in cyber world. Privacy of data stored in cloud resources though is a debatable subject and a matter of concern. As the number of documents stored on cloud resources increase, a search engine will have to be employed to search for information. Due to increased concern on these search engines itself misusing data, users are apprehensive about treating cloud resources as a data storage medium. Through this project we try to improve the security in cloud computing by introducing encryption of count list, as well as documents. The search is performed with the hashed keywords. Limitations in cloud computing are addressed by maximizing the utilization of cloud computing resources.**

***Keywords*— privacy, cloud, fully homomorphic encryption, similarity based retrieval, keyword search.**

I. Introduction

The technology that provides unlimited storage and openness on how to store data in cloud has brought good cheer to technicians across the world. But the unpredictability of where the data is stored and it being on a public network has made people to shy away from using cloud. Storing personal information and other documents, worries technicians, mainly due to the following reasons, 1)Uncertainty on whether the data is available on the remote location 2) whether the data is securely stored without any intrusion from unwanted external sources. These are stopping users from storing private and confidential documents in cloud.

Encrypting the document was the common method to hide the documents from a curious third party or server. But when the number of documents increases, it will be difficult for the owner to keep track of all the documents that have been uploaded. Otherwise owner should download the encrypted document, decrypt it and then check whether it is the indented one. If not, then owner must repeat the same procedures with another encrypted file. In such a situation user always wish for a search system that allows him for fast retrieval of the documents.

'Privacy-Preserving Multi-keyword Ranked Search over Encrypted Cloud Data' [3] is using Similarity calculation to find the match between a particular document and the given keywords. They are generating a trapdoor corresponding to the keywords given and this is used to find the match in encrypted keywords. The secret key which is known only to the owner cannot be shared to a person who wishes to access the document is a major problem in this technique. It is also clear that when the random factor *s* increases, in the trapdoor generation the accuracy of the search results decreases in this method.

In the method 'An Efficient Privacy Preserving Keyword Search Scheme in Cloud Computing' [4] it enable user to send a trapdoor for a certain keyword which is encrypted under his private key to Server, which will enable Server to find out all emails containing the keywords, but learns nothing else. Server then participates in the partial decipherment to calculate an intermediate result of the decipherment using its private key before returning the relevant encrypted emails back. Here they are considering the high computing capability of cloud environment, but it also lacks a ranking system that give burden to the user when the number of documents increases. Moreover the private key of the user is required to be shared if anyone other than owner needs to do a search on these documents.

Consider the following scenario in which hospitals are planning to keep electronic health records, since it provides several advantages especially to patients in terms of cost better care etc. It also provides an opportunity to the researchers all over in the medical field to access these documents to study about a particular disease, how the treatment is working on the patient. So the advantages of implementing electronic health records is pretty clear but people still worry about their privacy and security when the documents showing their ailment is outsourced to a public network which is a cloud, where we don't know exactly who is exactly providing the resources. The need for a secure system is so obvious that, we need to make things fast and efficient, while at the same time it should not add trouble to anyone's life.

Here we put forward the design for a secure, sharable, similarity based ranked document retrieval system that takes advantage of vast storage, immense computing power, and parallel computing facilities in a cloud environment.

II. PROBLEM STATEMENT

A. System Model

Here in this model we have four entities Document Owner O, Document User U, Cloud Server C and Document Manager DM. Owner has a set of documents D that is to be outsourced to cloud storage. The owner or user needs to securely retrieve these documents in future without any difficulty even if the number of documents increases in future. The Cloud server is running a search system in which for a set of keywords, the user or owner whoever be the retriever must get most relevant documents ranked according to their similarity score with the keywords. The system is modeled to utilize all the benefits of cloud parallelism by setting up a set of virtual machines VM, in the data hosts provided by the Cloud. The Data Center Broker distributes the task of finding match between the encrypted Query Words QK and the Keywords K.

B. Privacy and security issues.

Adversary knowing keywords and search results

Adversary can study the behavior of a user by knowing his search pattern. The adversary can obtain satisfactory results about the investigation about a person if he could get the users search results.

Cloud Server learning keywords and search results.

An honest but curious server may create headache to the users if he could get to know the search pattern of a fellow, by studying the keywords, search results or document itself.

Threat through a Known Cipher text

The server or an adversary can get only encrypted keywords from the search query or in the index list.

Threat due to known background

The cloud server can study the keywords in a document by using statistical information like keyword frequency or document frequency.

C. Notations

D – The plaintext document collection, denoted as a set of m data documents D = (D_1,D_2,….., D_m).

C – The encrypted document collection stored in cloud server, denoted as C = (C_1,C_2,…..,C_m).

K – The distinct keywords extracted from document collection D, denoted as K= (K_1,K_2,….,K_n).

QK – The subset of K, representing the keywords in a search request, denoted as QK= (QK_1,QK_2,…..,QK_t).

S – the Similarity score of the query keyword with the keywords in the list , denoted as S = (S_1,S_2,…..,S_n).

SP – top *p* Similarity scores, denoted as SP=(S_1,S_2,…..,S_p).

D. Preliminaries

Fully Homomorphic Encryption

The system is designed in such a way that it can use Gentry's Fully Homomorphic encryption [1], which allows computation with encrypted data. A homomorphic encryption technique ensures that if we are decrypting that computed data which is again in the encrypted format, will give us the result which is same as computing with the plain text.

Key Gen(): The secret key and public key is generated in this phase. We have to generate a set of rational numbers $(r_1.....r_l)$ in the range **[0,2)** such that there is a sparse subset ***S*** that will sum up to ***1/p***

$$\sum_{i\in S} r_i = \frac{1}{p}$$

This sparse subset is kept as secret key and is only known to the owner. The subset is represented as a bit vector .If ith bit of that bit vector is 0 then r_i is not an element of the sparse subset. If that bit is 1 the element r_i is a part of the sparse subset. Publishing this bit vector will not negotiate with the security of the system, since the sparse subset sum is known to be a hard problem.

Enc(): For encryption we have to sample ***p***. Here we are generating distributions say some integers $(x_0.....x_l)$. These integers are of length ρ bit. When we form the cipher text we select a random subset of integer say $s \in (x_0.....x_l)$ and also a random integer ***r*** which is of ρ bits .Then

$$\boxed{c = [\, m \;+\; 2r \;+\; 2\sum xi \,]}$$

After encryption we have to find the mod ***N*** of that integer, to keep value small .Then we to multiply the cipher text with the rational numbers$(r_1.....r_l)$.Before doing this operation the binary of the integer ***c*** and also of the rational values are taken. We have to perform binary multiplication here. This is the post processing after each encryption that helps the decryption faster.

$$\boxed{\varphi_i \;= c \times r_i \;\; for\ all\ \ i \in T}$$

So along with the cipher text we need to store Ψ values.

Dec():During the decryption , the owner must use the secret key which is in the form of bit vector. We have to multiply each Ψ values with the corresponding bit vector. This will finally give the product of the relevant rational numbers that sum up to ***1/p*** and the cipher text. Summing up these values will give $\left\lfloor \frac{c}{p} \right\rceil$ which is required.

Similarity based document retrieval

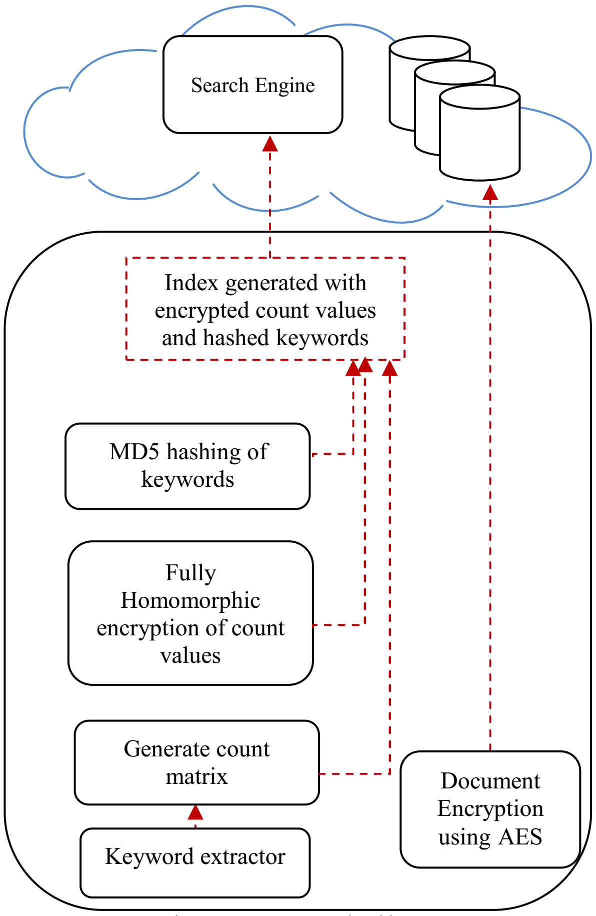
Similarity based document retrieval [6] has an added advantage that the documents are ranked during when a match is found with a given query and the document. The similarity score is calculated by finding the cosine angle between the query vector and the TFIDF matrix. TF means term frequency and IDF is Inverse document frequency. This matrix is generated based on the document frequency and the keyword frequency in that document. Query vector contains as much rows as of the number of keywords and a single column. The weight in TFIDF matrix uses t_{max} as maximum term frequency in a document, N as number of documents in a collection and n as number of documents containing a query term, to calculate the weights in TFIDF as

$$w = tf * IDF = tf * \log(N - n)\,/n$$

III. Secure Similarity Based Document Retrieval System Architecture in Cloud

A. Document Upload by Owner

Fig.1 shows how Document upload process takes place. The owner O of the documents has to upload the encrypted documents, as well as the count list to the Cloud server. The owner first extracts the keywords in each document and makes a keyword list to generate the count list. Before outsourcing this index to the cloud, the Owner generates the hash of the keywords which represent the row identifier of the matrix.



Search Engine
Index generated with encrypted count values and hashed keywords
MD5 hashing of keywords
Fully Homomorphic encryption of count values
Generate count matrix
Keyword extractor
Document Encryption using AES

Fig. 1 Documents upload by owner

This encryption of count list is done using Fully Homomorphic Encryption. Random values that are generated by owner form the column identifier. The encrypted count list is then outsourced to the search engine module of the cloud. The documents is encrypted using a Symmetric key encryption technique AES and is outsourced to the cloud Server to be stored in the cloud storage.

B. Document Search by Owner/User

During a search either, by user or by owner, the keywords in the Query ($QW_1,\ldots\ldots QW_t$) is first hashed before sending it to server. The Data Center Broker in the Cloud takes these hashed keywords and find whether it is matching with existing hashed keywords $K_1,K_2,\ldots.,K_n$.

The count list is kept in the form of a linked list as shown in the figure Fig. 2. which has two entities in each node. The first entity is Document Id and the second entity is the count of the keyword in that document. The keywords form the header of the linked list.

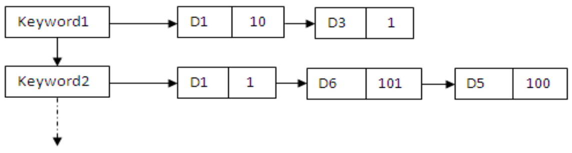


Fig. 2. Structure of count list

In this scheme only the relevant document vectors are kept. But this scheme will not help to provide the complete privacy at the server side. Privacy to the keywords is already given during the query as well as at the server side with hashing technique. But we could not guarantee a situation that server itself enacts as user and perform a search for a known keyword. In this case server is curious to know the contents of the documents uploaded. If the given keyword is matched with the existing hashed keyword list then the count list will show all the relevant documents connected to that keyword. So to avoid this privacy breach we have to make count list that holds the value of all the documents in the server or a set of n documents in the server, so that it is difficult for the server to deduce the mapping of a keyword to the document. Here irrelevant documents are included to the list which has document value equal to zero, but server could not know that it is irrelevant because the values are encrypted. The modified count list is shown in Fig. 3.

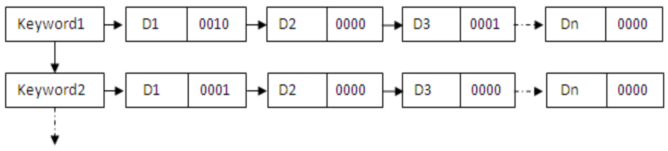


Fig. 3. Modified count list

To keep the privacy with the server the document ids are also encrypted using a commutative encryption key of the Document Manager, DM. The count list is finally kept as shown in figure Fig. 4.

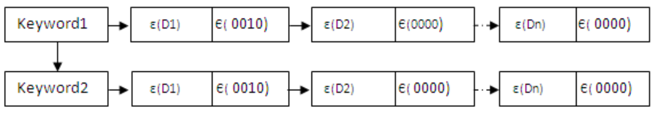
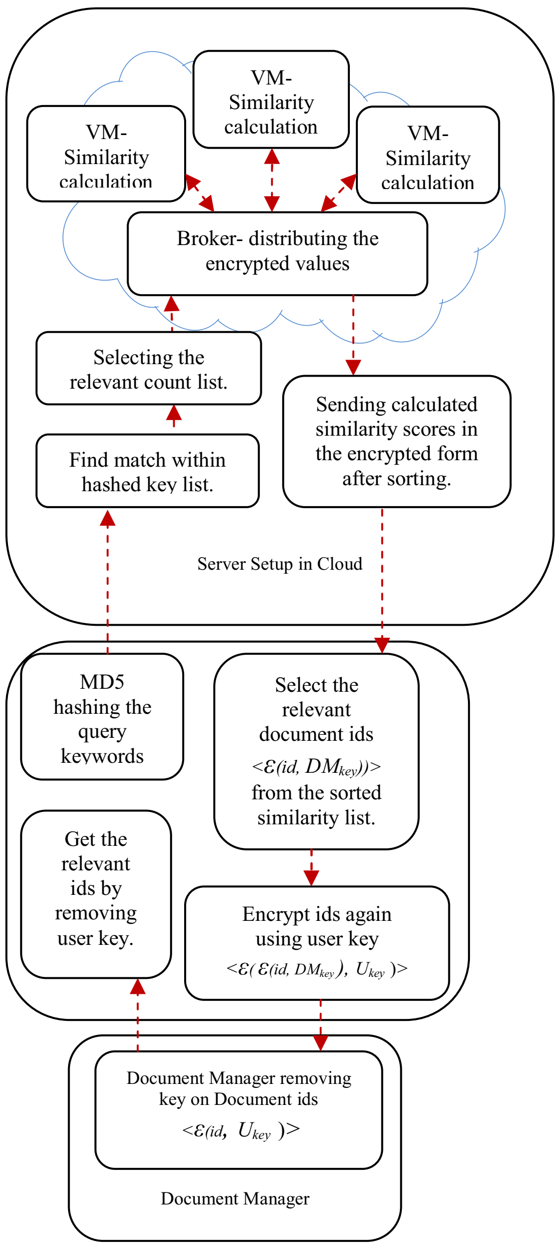


Fig. 4. Encrypted count list in serve

During the search if a match is found with the query keywords and the existing keywords then that count list corresponding to a keyword is selected for the similarity calculation. The selected count list is provided to the Virtual ($VM_1\ldots\ldots VM_c$).

This job division allows parallel execution in these Virtual Machines located in Data Hosts. The distribution of their job depends on the factor $f= qk*n/c$, where n is the total number of files in the index list, qk is the number of query keywords and c is the number of virtual machines available. This work distribution can effectively reduce the time taken for the computation over Fully Homomorphically encrypted data. The Similarity Scores of the relevant document $S=(S_1\ldots\ldots\ldots S_n)$ is calculated from the encrypted count values in the count list. The documents are ranked according to this relevance score to a subset $SP= (S_1\ldots\ldots\ldots S_p)$ as the first p relevant document ids is given to the user. Fig.5. illustrates the design of the complete search system.



VM- Similarity calculation
VM- Similarity calculation
VM- Similarity calculation
Broker- distributing the encrypted values
Selecting the relevant count list.
Find match within hashed key list.
Sending calculated similarity scores in the encrypted form after sorting.
Server Setup in Cloud
MD5 hashing the query keywords
Select the relevant document ids <ℰ(id, DMkey)> from the sorted similarity list.
Get the relevant ids by removing user key.
Encrypt ids again using user key <ℰ(ℰ(id, DMkey), Ukey)>
Document Manager removing key on Document ids <ℰ(id, Ukey)>
Document Manager

Fig. 5. Documents search by owner/user

IV. SIMULATION OF THE MODEL IN CLOUDSIM

A) *CloudSim Toolkit*

Evaluating the performance of Cloud provisioning policies, application workload models, and resources performance models in a repeatable manner under varying system and user configurations and requirements is difficult to achieve. To overcome this challenge, Rajkumar Buyya (Rajkumar Buyya et al. 2009) [7] proposes CloudSim: an extensible simulation toolkit that enables modeling and simulation of Cloud computing systems and application provisioning environments. The CloudSim toolkit supports both system and behavior modeling of Cloud system components such as datacenters, virtual machines (VMs) and resource provisioning policies. It implements generic application provisioning techniques that can be extended with ease and limited efforts.

B) *CloudSim Extension*

The existing CloudSim toolkit have been extended to support the user task as part of Cloudlet execution by extending DataCenter class. The user task is parallelly executing in the given working environment by using Executor class in Java.

C) *Implementation details*

The simulation environment was setup with Intel Dual Core processor with a speed of 2 GHz, and a 2 GB RAM. A single Datacenter with 20 Virtual machines, each executing one cloudlet was generated. The virtual machines are running on two processors as the Datahosts.

V. RESULTS

A) *Meeting privacy and security issues*

1) *Query keywords privacy and security*

The keywords provided from the user as the query is always hashed and matching is done against the hashed keywords in the server side. Here a curious server or a third party is unable to find the keywords provided by the user, since obtaining plain text from the hashed keyword is difficult. In an application which demands high security of these keywords can be kept encrypted and each time different instance of hashed keywords can be generated.

2) *Privacy and security of the result set*

The count list is kept in the encrypted form. The similarity computation is done with this encrypted values and the decryption of the result is done only at the user side. So the adversary or server itself is unable to deduce the set of documents queried by the user.

3) Privacy and security during the update operation

During the update operation all the nodes of count list corresponding to a particular keyword is updated. That means server has to add a set of n values to the existing list which are in the homomorphically encrypted form. This value can be either zero or greater than zero depending upon the document. But server could not know which node has been updated, from the encrypted data. It is to be noted that even though a comparison can be performed by the server on existing encrypted list to check whether it is zero or not the plain text result can be obtained only when secret key is there with the server.

The encryption key of FHE contains a hint about the secret key *p*. But this key is semantically secure, unless either it is easy to break the semantic security of encryption, or the following sparse subset sum problem (SSSP) is easy: given α set of β numbers Y and another number *s*, find the sparse (α-element) subset of Y whose sum is *s*. The SSSP has been studied before in connection with server-aided cryptosystems. If α and β are set appropriately, the SSSP is a hard problem, as far as we know. In particular, if we set a to be about λ(bit length of the key), it is hard to find the sparse subset by "brute force," since there are $\binom{\beta}{\alpha}$ possibilities.

B) Performance improvement obtained

It is found that even though encrypted keyword matching based on Fully Homomorphic Encryption is taking too much time to produce the result, the parallel processing facilities available in the cloud environment can improve the performance of this search system ρ times where ρ is the number of Datahosts available.

TABLE I
PERFORMANCE OBTAINED

No: of bits in count list	No: of files	With Cloud	Without Cloud
48	7	2256	24820
96	14	3323	36910
192	28	9017	42619
384	56	16287	67797

The search system is selecting only the matching count list for similarity calculation. The performance of the system directly depends on the number of bits involved during search process. When number of query keywords increases number of bits also increases. The application is run without cloud and found that the execution in cloud environment can give better performance and the high computational complexity of fully homomorphic encryption have been considerably reduced.

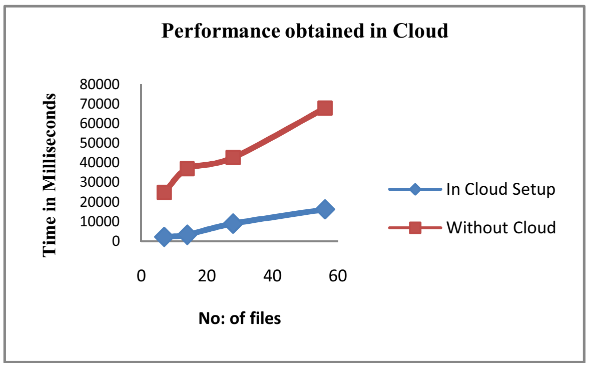


Fig.6.Performance Graph

In addition to that here care is taken to provide only a limited set of values for the computation, without compensating with the privacy. We have considered 7 to 56 files to evaluate the performance of the system. The performance improvement graph obtained from the table TABLE 1 is shown in Fig.6.

Finally the sorted document list is provided to the user. Not that user or even server could not see the similarity score in plain text unless they have the secret key of the fully homomorphic system. The user selects first *p* document ids of top p similarity scores SP=(S_1,S_2,…..,S_p) which is kept encrypted by Document Manager's key DM_{key} .This is again encrypted by User with the User Key U_{key} and send to the Document Manger. Document Manager removes his key and send the ids back to user. User upon removing U_{key} will get the relevant ids in plain text.

Document	Similarity
D3	e(1.9459101490553132)
D1	e(0.5596157879354227)
D2	e(0.5596157879354227)
D0	e(0.0000000000000000)
D4	e(0.0000000000000000)
D5	e(0.0000000000000000)
D6	e(0.0000000000000000)

VI. CONCLUSIONS

In this paper we introduced a secure storage system suitable for a cloud computing environment. The keyword search is performed by doing computation on index keywords that is kept encrypted using Fully Homomorphic Encryption. The search keywords and search results are in encrypted format during the whole network communication. The index list keywords are also encrypted to keep the data secure from honest but curious cloud. The system utilizes the immense resources of cloud computing to overcome the performance issue addressed by fully homomorphic encryption.

REFERENCES

[1] Marten van Dijk, Craig Gentry, Shai Halevi and Vinod Vaikuntanathan,"Fully Homomorphic Encryption over the Integers.". In Advances in Cryptography - EUROCRYPT'10, LNCS vol. 6110, pages 24-43, Springer, 2010.

[2] Craig Gentry, “Computing arbitrary functions of encrypted data.” Commun. ACM 53(3): 97-105 (2010)

[3] Ning Cao, Cong Wang, Ming Li, Kui Ren, Wenjing Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," INFOCOM, 2011 Proceedings IEEE , vol., no., pp.829-837, 10-15

[4] April 2011

[5] Qin Liu,Guojun Wang, Jie Wu, "An Efficient Privacy Preserving Keyword Search Scheme in Cloud Computing," Computational Science and Engineering, 2009. CSE '09. International Conference on , vol.2, no., pp.715-720, 29-31

[6] http://www.afn.org/~afn21533/keyexchg.htm

[7] Hweehwa Pang, Jialie Shen, and Ramayya Krishnan.”Privacy-preserving similarity-based text retrieval. “ACM Trans. Internet Technol. 10, 1, Article 4 (February 2010), 39 pages

[8] Buyya, R., Ranjan, R., Calheiros, R.N,’Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities. “ University of Melbourne, Australia (July 2009)