International Journal of Science and Research (IJSR)
ISSN: 2319-7064
ResearchGate Impact Factor (2018): 0.28 | SJIF (2018): 7.426

Multikeyword Rank Search Scheme for Unindexed Encrypted Cloud Data

Vaishali Bambode

Master of Engineering, Department of Computer Science and Engineering, SIPNA COET, Amravati, India

Abstract: ***The increasing popularity of cloud computing leads to more and more data owners to outsource their data to cloud servers for great convenience and reduced cost in data management. However, sensitive data should be encrypted before outsourcing For privacy requirements, which no longer support data utilization like keyword –based document retrieval. In this review paper, we present Asecure multi keyword ranked search scheme over encrypted cloud data, which simultaneously supports dynamic update operation like deletion and insertion of documents. Specifically, the vector space model and the widely-used TF-IDF model are combined in the index construction and query generation. We construct a special tree-based index structure and propose a "Greedy Depth-first Search" algorithm to provide efficient multi keyword ranked search. The secure kNN algorithm is utilized to encrypt the index and query vectors, and meanwhile ensure accurate relevance score calculation between encrypted index and query vectors. In order to resist statistical attacks, phantom means illusory terms are added to the index vector for blinding search results. Due to the use of special tree-based index structure, the proposed scheme can achieve sub-linear search time and deal with the deletion and insertion of documents flexibly.***

Keywords: Cloud Computing, Searchable schemes, Multi keyword Rank Search, Encrypted Data, Dynamic Update

1. Introduction

1.1 Overview

Cloud computing has been considered as a new model of enterprise IT infrastructure, which can organize huge resource of computing, storage and applications, and enable users to enjoy ubiquitous, convenient and on demand network access to a shared pool of configurable computing resources with great efficiency and minimal economic overhead . The cloud service providers (CSPs) that keep the data for users may access users sensitive information without authorization. A general approach to protect the data confidentiality is to encrypt the data before outsourcing. However, this will cause a huge cost in terms of data usability. Downloading all the data from the cloud and decrypt locally is obviously impractical. In order to address the above problem, researchers have designed some general-purpose solutions with fully-homomorphism encryption means conversion of data into cipher text that can be analyzed and worked with as if it were still in its original form. However, these methods are not practical due to their high computational overhead for both the cloud sever and user.

1.2 Searchable Encryption Scheme

On the contrary, more practical special-purpose solutions, such as searchable encryption (SE) schemes have made specific contributions in terms of efficiency, functionality and security. Searchable encryption schemes enable the client to store the encrypted data to the cloud and execute keyword search over cipher text domain. So far, abundant work has been proposed under to achieve various search functionality, such as single keyword search, similarity search, multi-keyword Boolean search, ranked search, multi-keyword ranked search, etc. Among them, multi-keyword ranked search achieves more and more attention for its practical applicability. Recently, some dynamic schemes have been proposed to support inserting and deleting operations on document collection. These are significant works as it is highly possible that the data owners need to update their data on the cloud server. But few of the dynamic schemes support efficient multi-keyword ranked search.

2. Proposed Work

This project proposes a secure tree-based search scheme over the encrypted cloud data, which supports multi-keyword ranked search and dynamic operation on the document collection. In order to obtain high search efficiency, we construct a tree-based index structure and propose a "Greedy Depth-first Search (GDFS)" algorithm based on this index tree. In existing system, The techniques of data updation are utilizes effectively but there is big problem in working with sharing keys and decrypted data with other users which may disturb the security as well in this a unencrypted index key is used for ranking which may break security as well. So that we proposed a mechanism in which the encrypted index term key will get generated and perform the evaluation for the multi keyword searching in all encrypted cloud storage. This project helps to implement metadata based keyword search. Metadata based keyword search means search engine that powers a portal search based on a specific metadata schema. It encourages to implement usability based ranking optimization i.e. to check how many times a particular file has been accessed. It supports privacy preserving over shared data to cloud by means of encrypting data. As well to implement efficient ranking system based on term frequency generation. The TF-IDF module is used for page ranking.

3. Literature Survey

Searchable encryption schemes [1] enable the clients to store the encrypted data to the cloud and execute keyword search over cipher text domain. Due to different cryptography primitives, searchable encryption schemes can be constructed using public key based cryptography[2][3] or

symmetric key based cryptography. Multi-keyword Boolean search[4] allows the users to input multiple query keywords to request suitable documents. Among these works, conjunctive keyword search scheme only return the documents that contain all of the query keywords. Disjunctive keyword search schemes return all of the documents that contain a subset of the query keywords. Predicate search are proposed to support both conjunctive and disjunctive search. All these multi-keyword search schemes retrieve search results based on the existence of keywords, which cannot provide acceptable result ranking functionality. Practically, the data owner may need to update the document collection after he upload the collection to the cloud server. Thus, the SE schemes are expected to support the insertion and deletion of the documents. There are also several dynamic searchable encryption schemes. Goh [7] proposed a scheme to generate a sub-index (Bloom filter) for every document based on keywords. Then the dynamic operations can be easily realized through updating of a Bloom filter along with the corresponding document. However, Goh's scheme has linear search time and suffers from false positives. In 2012, Kamara and Papamanthou[8] Constructed an encrypted inverted index that can handle dynamic data efficiently. But, this scheme is very complex to implement. Subsequently, as an improvement, Kamara and Papamanthou proposed a new search scheme based on tree-based index, which can handle dynamic update on document data stored in leaf nodes. However, their scheme is designed only for single-keyword Boolean search.

Multi-keyword Boolean search allows the users to input multiple query keywords to request suitable documents. Among these works, conjunctive keyword search schemes, only return the documents that contain all of the query keywords. Disjunctive keyword search schemes return all of the documents that contain a subset of the query keywords. Predicate search schemes are proposed to support both conjunctive and disjunctive search. All these multi keyword search schemes retrieve search results based on the existence of keywords, which cannot provide acceptable result ranking functionality. Ranked search can enable quick search of the most relevant data. Sending back only the top-k most relevant document scan effectively decrease network traffic .Some early works [5] have realized the ranked search using order-preserving techniques, but they are designed only for single keyword search. Cao et al. [9] realized the first privacy-preserving multi-keyword ranked search scheme, in which documents and queries are represented as vectors of dictionary size. With the "coordinate matching", the documents are ranked according to the number of matched query keywords. However, Cao et al.'s scheme does not consider the importance of the different keywords, and thus is not accurate enough. In addition, the search efficiency of the scheme is linear with the cardinality of document collection. In the work of Song et.al[6] ,the each document is considered as a sequence of fixed length words and is individually indexed. In the work of Sun et al. [10] presented a secure multi-keyword search scheme that supports similarity-of document collection. In the work of Song et.al[6] ,the each document is considered as a sequence of fixed length words and is individually indexed. In the work of Sun et al. [10] presented a secure multi-keyword search scheme that supports similarity-

4. Problem Definition

In existing system the challenge is symmetric searchable schemes. It requires huge cost in terms of data usability. For example, the existing techniques on keyword-based information retrieval, which are widely used on the plaintext data, cannot be directly applied on the encrypted data. Downloading all the data from the cloud and decrypt locally is obviously impractical. Existing System methods not practical due to their high computational overhead for both the cloud sever and user. In the proposed scheme, the data owner is responsible for updating information and sending them to the cloud server. Thus, the data owner needs to store the unencrypted index tree and the information that are necessary to recalculate the IDF values. IDF is inverse document frequency ,is a numerical statistic that is intended to reflect how important a word is to a document in data retrieval. Such an active data owner may not be very suitable for the cloud computing model. It could be a meaningful but difficult future work to design a dynamic searchable encryption scheme whose updating operation can be completed by cloud server only, meanwhile reserving the ability to support multi-keyword ranked search. In addition, as the most of works about searchable encryption, our scheme mainly considers the challenge from the cloud server.

5. Architecture

Figure 1: The architecture of ranked search over encrypted cloud data

This scheme, different data owners use different secret keys to encrypt their documents and keywords while authorized data

users can query without knowing keys of these different data owners. The authors proposed an "Additive Order Preserving Function" to retrieve the most relevant search results. However, these works don't support dynamic operations. Practically, the data owner may need to update the document collection after he upload the collection to the cloud server. Thus, the SE schemes are expected to support the insertion and deletion of the documents. Xia etc.[4] constructed a tree-based index structure and proposed a greedy depth-first search [GDFS] algorithm that achieved higher search efficiency. Wang etc[4] raised a secure ranked search which returned the top-k relevant files and was only designed for single keyword search. based ranking. The authors constructed a searchable index tree based on vector space model and adopted cosine measure together with TF×IDF to provide ranking results. Sun et al.'s search algorithm achieves better-than-linear search efficiency but results in precision loss. ¨Orencik et al. [11] proposed a

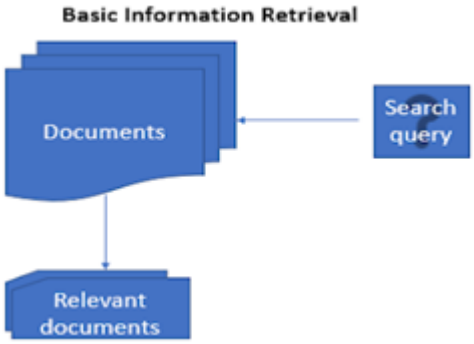
secure multi-keyword search method which utilized local sensitive hash only designed for single keyword. D.Bonehetc a [5]],have discussed the public key encryption for keyword search.

The system model in this paper involves three different entities: data owner, data user and cloud server, as illustrated in Fig.1Data owner has a collection of documents that he wants to outsource to the cloud server in encrypted form while still keeping the capability to search on them for effective utilization. In our scheme, the data owner first builds a secure searchable tree index from document collection , and then generates an encrypted document collection . Afterwards, the data owner outsources the encrypted collection and the secure index to the cloud server, and securely distributes the key information of trapdoor generation (including keyword IDF values) and document decryption to the authorized data users. Besides, the data owner is responsible for the update operation of his documents stored in the cloud server. While updating, the data owner generates the update information locally and sends it to the server. Data users are authorized ones to access the documents of data owner. With query keywords, the authorized user can generate a trapdoor TD according to search control mechanisms to fetch k encrypted documents from cloud server. Then, the data user can decrypt the documents with the shared secret key. Cloud server stores the encrypted document collection C and the encrypted searchable tree index for data owner. Upon receiving the trapdoor TD from the data user, the cloud server executes search over the index tree, and finally returns the corresponding collection of top-k ranked encrypted documents. Besides, upon receiving the update information from the data owner, the server needs to update the index and document collection according to the received information.

6. Methodology/Approach

In this section, we review various methodologies that are used in this project work.

Vector Space Model with TF * IDF value
Vector space model along with TF-IDF rule is widely used in plaintext information retrieval, which efficiently supports ranked multi-keyword search . Here, the term frequency is the number of times a given term (keyword) appears within a document, and the containing the keyword. In the vector space model, each document is denoted by a vector, whose elements are the normalized TF values of keywords in this document. Each query is also denoted as a vector Q, whose elements are the normalized TF values of keywords in this document. Each query is also denoted as a vector Q, whose elements are the normalized IDF values of query keywords in the document collection. Naturally, the lengths of both the TF vector and the IDF vector are equal to the total number of keywords. TF denotes the frequency of a given keyword appearing in the file and IDF is the logarithm of the total number of files divided by the number of files containing the keyword and get value obtained the logarithm. In the proposed work, we learn about building a basic search engine or document retrieval system using Vector space model. This use case is widely used in information retrieval systems. This use case is widely used in information retrieval systems. Given a set of documents and search term(s)/query we need to retrieve relevant documents that are similar to the search query. The problem statement explained above is represented as in below image.



Document retrieval system

Vector Space Model:
A vector space model is an algebraic model, involving two steps, in first step we represent the text documents into vector of words and in second step we transform to numerical format so that we can apply any text mining techniques such as information retrieval, extraction ,information filtering etc. Let us understand with an example. consider below statements and a query term. The statements are referred as documents hereafter.
Document1: Cat runs behind rat
Document2: Dog runs behind cat
Query: rat

Document vectors representation:
In this step includes breaking each document into words, applying preprocessing steps such as removing stop words, punctuations, special characters etc. After preprocessing the documents we represent them as vectors of words. Below is a sample representation of the document vectors.
Document1: (cat, runs, behind, rat)
Document2: (Dog, runs, behind, cat)
Query: (rat)
the relevant document to Query = greater of (similarity score between (Document1, Query), similarity score between (Document2, Query)
Next step is to represent the above created vectors of terms to numerical format known as term document matrix.

Term Document Matrix:
A term document matrix is a way of representing documents vectors in a matrix format in which each row represents term vectors across all the documents and columns represent document vectors across all the terms. The cell values frequency counts of each term in corresponding document. If a term is present in a document, then the corresponding cell value contains 1 else if the term is not present in the document then the cell value contains 0. After creating the term document matrix, we will calculate term weights for all the terms in the matrix across all the documents. It is also important to calculate the term weightings because we need to find out terms which uniquely define a document.

We should note that a word which occurs in most of the documents might not contribute to represent the document

relevance whereas less frequently occurred terms might define document relevance. This can be achieved using a method known as term frequency - inverse document frequency (tf-idf), which gives higher weights to the terms which occurs more in a document but rarely occurs in all other documents, lower weights to the terms which commonly occurs within and across all the documents.

Tf-idf = tf X idf

tf=term frequency is the number of times a term occurs in a document
idf= inverse of the document frequency, given as below
idf = log(N/df), where df is the document frequency-number of documents containing an inverse document frequency
Note: idf is calculated using logarithm of inverse fraction between document count and document frequency

Term document matrix with tf-idf			
words\documents	Document1	document2	query term
cat	0	0	0
runs	0	0	0
behind	0	0	0
rat	0.30103	0	0.30103
dog	0	0.30103	0

tf-idf calculation
Note: Tf-idf weightage is calculated using tf X idf
Note, there are many variations in the way we calculate the term-frequency(tf) and inverse document frequency (idf)

Variants of term frequency (TF) weight

weighting scheme	TF weight
binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} / \sum_{t' \in d} f_{t',d}$
log normalization	$1 + \log(f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K)\frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

term frequency variation

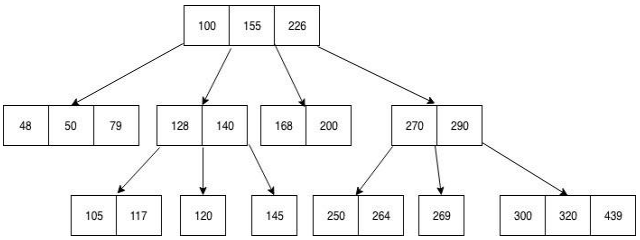
Variants of inverse document frequency (IDF) weight

weighting scheme	IDF weight ($n_t = \lvert\{d \in D : t \in d\}\rvert$)
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log\left(1 + \frac{N}{n_t}\right)$
inverse document frequency max	$\log\left(\frac{\max_{\{t' \in d\}} n_{t'}}{1 + n_t}\right)$
probabilistic inverse document frequency	$\log \frac{N - n_t}{n_t}$

inverse document frequency variation

Tree Based Indexed Structure
To improve the efficiency of the search, Xia et.al[11] first proposed the keyword balanced binary tree. In our scheme, data owner builds A secure keyword balanced binary tree and outsource them to the cloud server. The cloud server merges those index trees and performs the efficiently multi keyword ranked search. First we must know about index and unindexed files..The index stores a list of all words, each with a list of documents that contain it. There is no need to search every document for the keyword each time the user wants to search for a word. The keyword is simply located in the index (if it exists), and a list of documents that contain it is immediately available. An unindexed search is one that cannot be processed using the set of index defined in the server. It will necessitate iterating through most or all of the entries in the database. Unindexed encrypted data simply means not provided with an index or indexes.

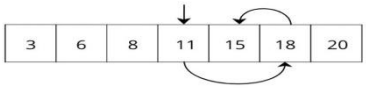
When we think about the performance of a database, indexing is the first thing that comes to the mind. Here, we are going to look into how database indexing works on a database in the proposed scheme .B-tree is a data structure that store data in its node in sorted order. We can represent sample B-tree as follows.



B-tree stores data such that each node contains keys in ascending order. Each of these keys has two references to another two child.nodes. The left side child node keys are less than the current keys and the right side child node keys are more than the current keys. If a single node has “n” number of keys, then it can have maximum “n+1” child nodes.

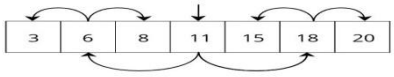
Why Is Indexing Used in the Database?

Imagine you need to store a list of numbers in a file and search a given number on that list. The simplest solution is to store data in an array and append values when new values come. But if you need to check if a given value exists in the array, then you need to search through all of the array elements one by one and check whether the given value exists. If you are lucky enough, you can find the given value in the first element. In the worst case, the value can be the last element in the array. We can denote this worst case as O(n) in asymptotic notation. This means if your array size is “n,” at most, you need to do “n” number of searches to find a given value in an array.How could you improve this time? The easiest solution is to sort the array and use binary search to find the value. Whenever you insert a value into the array, it should maintain order. Searching start by selecting a value from the middle of the array. Then compare the selected value with the search value. If the selected value is greater than search value, ignore the left side of the array and search the value on the right side and vice versa.



Binary search

Here, we try to search key 15 from the array 3,6,8,11,15, and 18, which is already in sorted order. If you do a normal search, then it will take five units of time to search since the element is in the fifth position. But in the binary search, it will take only three searches.If we apply this binary search to all of the elements in the array, then it would be as follows



Binary search to all element

It is a Binary tree. This is the simplest form of the B-tree. For Binary tree, we can use pointers instead of keeping data in a sorted array. Mathematically, we can prove that the worst case search time for a binary tree is O(log(n)). The concept of Binary tree can be extended into a more generalized form, which is known as B-tree. Instead of having a single entry for a single node, B-tree uses an array of entries for a single node and having reference to child node for each of these entries. Below are some time complexity comparisons of each pre-described method.

Type	Insertion	Deletion	Lookup
Unsorted Array	O(1)	O(n)	O(n)
Sorted Array	O(n)	O(n)	O(log(n))
B-tree	O(log(n))	O(log(n))	O(log(n))

B+treeis another data structure that used to store data, which looks almost the same as the B-tree. The only difference of B+tree is that it stores data on the leaf nodes. This means that all non-leaf node values are duplicated in leaf nodes again. Below is a sample B+tree.

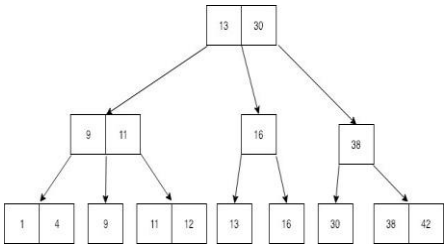
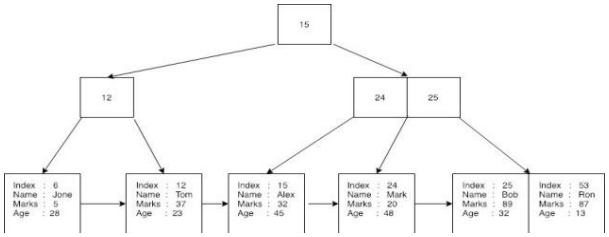


Figure: B+Tree

13, 30, 9, 11, 16, and 38 non-leaf values are again repeated in leaf nodes. Can you see the specialty in this tree at leaf nodes? Yeah, leaf node includes all values and all of the records are in sorted order. In specialty in B+tree is, you can do the same search as B-tree, and additionally, you can travel through all the values in leaf node if we put a pointer to each leaf nodes.

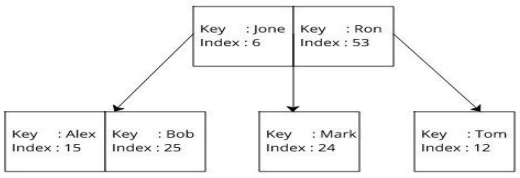
Index construction & Query Generation

In the proposed scheme..B-tree comes to the database indexing, this data structure gets a little complicated by not just having a key, but also having a value assigned with the key. This value is a reference to the actual data record. The key and value together are called a payload.First, the database creates a unique random index (or primary key) for each of the given records and converts the relevant rows into a byte stream. Then, it stores each of the keys and record byte streams on a B+tree. For example,



B+tree on database pages

Here you can see that all records are stored in the leaf nodes of the B+tree and index used as the key to creating a B+tree. No records are stored on non-leaf nodes. Each of the leaf nodes has reference to the next record in the tree. A database can perform a binary search by using the index or sequential search by searching through every element by only traveling through the leaf nodes.If no indexing is used, then the database reads each of these records to find the given record. When indexing is enabled, the database creates three B-trees for each of the columns in the table as follows. Here the key is the B-tree key used to indexing. The index is the reference to the actual data record. For Example,



When indexing is used first, the database searches a given key in correspondence to B-tree and gets the index in O(log(n)) time. Then, it performs another search in B+tree by using the already found index in O(log(n)) time and gets the record. Each of these nodes in B-tree and B+tree is stored inside the Pages. Pages are fixed in size. Pages have a unique number starting from one. A page can be a reference to another page by using page number. At the beginning of the page, page meta details such as the rightmost child page number, first free cell offset, and first cell offset stored. Databases should have an efficient way to store, read, and modify data. B-tree provides an efficient way to insert and read data. In actual Database implementation, the database uses both B-tree and B+tree together to store data. B-tree used for indexing and B+tree used to store the actual records. B+tree provides sequential search capabilities in addition to the binary search, which gives the database more control to search non-index values in a database.

Greedy DFS Algorithm

The search process that is used in this paper is a recursive procedure upon the tree, named as "Greedy Depth-First Search" algorithm. We construct a result list which stores the k accessed documents with the largest relevance scores to the query.The elements of the result list are ranked in descending order and will be updated timely during the search process. A greedy algorithm is an algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a

global optimum. There are essentially two graph traversal algorithms, known as Breadth-first search and Depth-search first. In general, Given a graph G=(V,E) determine all nodes that are connected from a given node v via a(directed) path. BFS-from node v, visit each of its neighboring nodes in sequence then visit their neighbors etc. while avoiding repeated visits. DFS-from node v, visit its first neighboring node and all its neighbors using recursion, then visit node v's second neighbor applying the same procedure, until all v's neighbors are visited while avoiding repeated visits. General procedure to follow here is,
[1] create a Boolean array visited [1…….n],initialize all values to false except for visited[v] to true
[2] call DFS(v) to visit all nodes reachable via a path DFS(v) for each neighboring nodes w of v do if visited [w] is false then set visited[w] to true; call DFS(w).

Knn Algorithm
A secure knn algorithm is utilized to encrypt the index and query vector ,and meanwhile ensure accurate relevance score calculation between encrypted index and query vectors in this paper. It also helps to protect the security of the scheme. **K nearest neighbor algorithm** is very simple. It **works** based on minimum distance from the query instance to the training samples to determine the K-nearest neighbors. ... The data for **KNN algorithm** consist of several multivariate attributes name that will be used to classify

Encryption & Descryption Algorithm

Encryption is the process of converting a plaintext message into ciphertext which can be decoded back into the original message. An **encryption algorithm (AES) & (DES)** along with a key is used in the **encryption and decryption** of data. After creating an index, to ensure the privacy of index and files the data owner encrypts both index and key. After encrypting data, the data owner sends encrypted file collection and index to the cloud service provider. While retrieving data from the csp through the ranked keyword search it consist of three phases: Trapdoor generation, Ranked keyword search and data decryption.

1) Trapdoor Generation—After sorting data in cloud, whenever the authorized user wants retrieve the file containing keywords, computes the trapdoor for keywords and sends to the CSP as search request.
2) Ranked Keyword Search—In this method, the cloud server searches for the matching files after receiving the trapdoor from the user as follows: The cloud server first finds the matching entries of the file via trapdoor,if server finds matching files identifies along with their associated relevance scores. Then, the server ranks the matched files according to relevance scores and sends top-k most relevant files to the user.
3) Data Decryption: After receiving the matched files from CSP for corresponding search request, the authorized user decrypts them with the private key and obtains their plain text.

7. Discussion on System Modules

Following modules are tentatively used in this paper work:

1) Login Module
In this we are signing in to the application, If the credentials are correct then it will open homepage otherwise it will alert to enter correct details. There are two parts in this module Data owner login and data user login. In case of data user login, data owner register with his login details. This helps the owner to upload his file with encryption using RSA algorithm. This ensures the files to be protected from unauthorized user. Data owner has the collection of files that he wants to outsourced to the cloud server in encrypted form while still keeping the capability to search on them for effective utilization. Data owner firstly builds a secure searchable tree index then generates an encrypted document collection. Afterword's, the data owner outsources the encrypted collection and secure index to the cloud server. And securely distributes the key information of trapdoor generation and document decryption to the authorized data users. Data owner is responsible for the update operations f his documents stored in the cloud server. While updating the data owner generates the update information localy and sends it to the server. In case of data user login, login details of user are given. This module used to help the client to search the file using the multiple keywords concept and get the accurate result list based on the user query. Data users are authorized ones to access the documents of data owner.

2) Keyword Encryption Module
The input text we are providing is encrypted by using several algorithmic techniques. This module is used to encrypt the document with an activation code and then the activation code is send to user to download the document. Cloud server stores the encrypted document and the encrypted searchable tree index for data owner. Cloud server executes search over the index tree, it returns the corresponding collection of top k ranked encrypted documents .While updating information from the data owner, the server needs to update the index and document collection according to the received information.

3) Data Integration Module
We are combining technical and business processes used to combine **data** from desparate sources into meaningful and valuable information. A complete **data integration** solution delivers trusted **data** from various sources.

4) Encryption Module
Here we are actually doing the encryption using cryptography. The sole purpose of encryption is to protect the confidentiality of data stored on computer system or some other network. Encryption does not guarantee the protection of data but it does add a layer of security that makes it more difficult for hackers or dishonest users to misuse data. In case of symmetric encryption readable message or plaintext is encrypted to make it unreadable by means of a secret key.At the receiver end , encrypted data is decrypted using the same key. Algorithms used for symmetric encryption and decryption are AES and DES.

5) Index Encryption Module
Single values level **encryption** of the **index** reveals sensitive information, such as frequencies of the **index** values. Whole **Index** level **encryption** ensures that information

about the indexed data cannot be leaked, since the **index** is **encrypted** as one unit.

8. Objective Analysis

To enable secure, efficient, accurate and dynamic multi-keyword ranked search over outsourced encrypted cloud data under the above models, our system has the following design goals. The proposed scheme is designed to provide not only multi-keyword query and accurate result ranking, but also dynamic update on document collections. Search efficiency. The scheme aims to achieve sublinear search efficiency by exploring a special tree- and an efficient search algorithm. Privacy-preserving. The scheme is designed to prevent the cloud server from learning additional information about the document collection, the index tree, and the query. The specific privacy requirements are summarized as follows,

1) Index confidentiality and query confidentiality. The underlying plaintext information, including keywords in the index and query, TF values of keywords stored in the index, and IDF values of query keywords, should be protected from cloud server;
2) Trapdoor unlink ability. The cloud server should not be able to determine whether two encrypted queries (trapdoors) are generated from the same search request.
3) Keyword privacy. The cloud server could not identify the specific keyword in query, index or document collection by analyzing the statistical information like term frequency. Note that our proposed scheme is not designed to protect access pattern, i.e., the sequence of returned documents.
4) Index Encryption: In this to achieve the level of security proposed methodology finds the required data index and perform the index encryption.

9. Acknowledgement

I wish to acknowledge Dr A ABARDEKAR (Department of Computer & Science Engineering, SIPNA COET) for his contribution in my work of documentation and development of this review paper.

10. Conclusion & Future work

In this paper, a secure, efficient and dynamic search scheme is proposed, which supports not only the accurate multi keyword ranked search but also the dynamic deletion and insertion of documents. The scheme proposes to construct a special keyword balanced binary tree as the index, and propose a "Greedy Depth-first Search" algorithm to obtain better efficiency than linear search. In addition, the parallel search process can be carried out to further reduce the time cost. To protect the security of the scheme a secure kNN algorithm is suggested. There are still many challenge problems in symmetric SE schemes. In the proposed scheme, the data owner is responsible for generating updating information and sending them to the cloud server. Thus, the data owner needs to store the unencrypted index tree and the information that are necessary to recalculate the IDF values. Such an active data owner may not be very suitable for the cloud computing model. It could be a meaningful but difficult future work to design a dynamic searchable encryption scheme whose updating operation can be completed by cloud server only, meanwhile reserving the ability to support multi-keyword ranked search. In addition, as the most of work about searchable encryption, our scheme mainly considers the challenge from the cloud server. Actually, there are many secure challenges in a multi-user scheme. First, all the users usually keep the same secure key for trapdoor generation in a symmetric SE scheme. In this case, the revocation of the user is big challenge. If it is needed to revoke a user in this scheme, we need to rebuild the index and distribute the new secure keys to all the authorized users. Second, symmetric SE schemes usually assume that all the data users are trustworthy. It is not practical and a dishonest data user will lead to many secure problems. For example, a dishonest data user may search the documents and distribute the decrypted documents to the unauthorized ones. Even more, a dishonest data user may distribute his/her secure keys to the unauthorized ones. In the future works, we will try to improve the SE scheme to handle the SE challenge problems.

References

[1] TIANYUE PENG , (Student Member, IEEE), YAPING LIN, (Member, IEEE), XIN YAO , (Student Member, IEEE), AND WEI ZHANG "An Efficient Ranked Multi-Keyword Search for Multiple Data Owners Over Encrypted Cloud Data" Received March 12, 2018, accepted April 11, 2018, date of publication April 20, 2018, date of current version May 9, 2018.
[2] C. Liu, L. Zhu, and J. Chen, ''Efficient searchable symmetric encryption for storing multiple source data on cloud ,''J .Netw. Comput. Appl., vol. 86, pp. 3–14, May 2017.
[3] S. K. Pasupuleti, S. Ramalingam, and R. Buyya, ''An efficient and secure privacy-preserving approach for outsourced data of resource constrained mobile devices in cloud computing,'' J. Netw. Comput. Appl., vol. 64, pp. 12–22, Apr. 2016.
[4] Zhihua Xia, Xinhui Wang, Xingming Sun and Qian Wang," A Secure and dynamic multikeyword ranked search scheme over encrypted cloud data"IEEE transactions on parallel and distributed systems,vol 27, no 2 february 2016.
[5] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in Proc. Adv. Cryptol.-Eurocrypt, 2014, pp. 506–522
[6] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in Proc. IEEE Symp. Secur. Privacy, 2007, pp. 44–55.
[7] E.-J. Goh, "Secure indexes," IACR Cryptol.ePrint Archive, vol. 2003, p. 216, 2003.
[8] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in Proc. ACM Conf. Computation 2012, pp. 965–976.
[9] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in Proc. IEEE INFOCOM, Apr. 2011, pp. 829–837.
[10] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based

ranking," in Proc. 8th ACM SIGSAC Symp. Inf., Comput.Commun.secur., 2013, pp. 71–82.
[11] C. Orencik, M. Kantarcioglu, and E. Savas, "A practical and secure multi-keyword search method over encrypted cloud data," in Proc. IEEE 6th Int. Conf. Cloud Comput., 2013, pp. 390–397.
[12] W. Zhang, S. Xiao, Y. Lin, T. Zhou, and S. Zhou, "Secure ranked multi-keyword search for multiple data owners in cloud computing," in Dependable Syst. Networks (DSN), IEEE 44th Annu. IEEE/IFIP Int. Conf., 2014, pp. 276–286.

Author Profile

Vaishali Bambode received the BE degree in Computer Engineering , MIT Pune University . She is pursuing her Master's in engineering from SIPNA Amravati University. Her research study interest include cloud computing security and machine learning.