A Survey on Searching and Indexing on Encrypted Data

Ankit Doshi
Computer Dept., V.E.S. Institute of Technology,Mumbai -74,India.

Kajal Thakkar
Computer Dept., V.E.S. Institute of Technology,Mumbai -74,India.

Sahil Gupte
Computer Dept., V.E.S. Institute of Technology,Mumbai -74,India.

Anjali Yeole
Computer Dept., V.E.S. Institute of Technology,Mumbai -74,India.

Abstract

As Cloud Computing is one of the biggest and trending technology, more and more Companies are outsourcing their databases on Cloud in order to lower the cost of maintaining hardware. But the biggest concern about Cloud is Security and Privacy. In order to solve this problem sensitive data is encrypted before outsourcing. Encryption solves the problem to some extent, however there is an overhead of searching on the Encrypted Data. This paper explores existing technique of searching over encrypted data.

1.Introduction

Cloud Computing is the an emerging technology, hence it has become the common practice to use cloud, since it not only provides cost efficiency but also on demand high quality service. The data on Cloud could be anything ranging from Emails, Personal Files, University records to Government data, etc. Also the users of this technology can range from Private sectors i.e. banks, universities, companies to Public Sector.

These users outsource their databases on Cloud to enjoy all the services. But as Cloud is hosted by a third party, data stored on cloud cannot be completely trusted for privacy and security. It follows that sensitive data usually should be encrypted prior to out-sourcing for data privacy and preventing unsolicited accesses. Database encryption introduces an additional layer to conventional network and application security solutions and prevents exposure of sensitive information even if the raw data is compromised[3]. Database encryption prevents unauthorized users from viewing sensitive data in the database and, it allows database administrators to perform their tasks without having access to sensitive information.Furthermore, it protects data integrity, as unauthorized modifications can easily be detected .

However as a result of encryption, effective data utilization becomes a challenging task. Moreover, in Cloud Computing, data owners may share their outsourced data with a large number of users. The individual users might want to only retrieve certain specific data files they are interested in during a given session, but encryption of data makes searching of documents required difficult.

Once the data is encrypted and outsourced to a far-off datacenter we should be able to access it. If we need to perform a search over the encrypted data, we have two options:

- We can download the entire data set, decrypt it and search it client-side.
- We can give the remote server the secret key we used to encrypt the data and let the server decrypt and search it.

Both of the 'solutions' described above have fairly obvious ramifications. If an individual decide to go with the first choice and download the data before decrypting and searching it client-side, they are going to experience an impossibly large amount of communication overhead.

Downloading of data every time you want to search for all certain data would result in wastage of resources and time.

If the individual opt to hand their secret key to the remote server and allow it to decrypt and search the data, the individual's required to trust the datacenter. Knowing the key, a rogue datacenter admin could perform a number of harmful acts, ranging from

simply decrypting the data and learning about their contents, to modifying or deleting them. The encryption used is rendered useless once the key is made available and merely acts to add overhead to the search and retrieval process. It would be far more efficient to encrypt the connection between the datacenter and the company if the latter is willing to fully trust the former.

Hence efficient search technique need to be implemented over encrypted data.

2.Techniques to perform searching over Encrypted Data

2.1.Practical Techniques for Searches on Encrypted Data by Song et al[4]

2.1.1.Encryption

For a set of documents, the following is repeated once for each document. This should be done by the client, before uploading it to a remote, untrusted server.[4]
The input document is tokenized into a set of words, W. This tokenization needs to still contain all of the input symbols, whilst separating words from punctuation. So, for example, a sentence such as "Something, something2!" would need to be transformed into the strings{'something', ', <space>', 'something2', '!'}
Once the document has been tokenized, the following process is performed on each word, Wi.

1. Three keys k', k'', k''' are generated using master private key in such a way that neither of keys can reveal information to derive other key. Thus allowing us to reveal one or two keys to an untrusted server. This gives enough information to perform a search but not enough to decrypt a document.
2. Word Wi is encrypted either using ECB or CBC mode with fixed Initialization Vector using key K''.

$X_i = E_{k''}(W_i)$

3. Using k''' stream cipher G is generated where x bits from G are taken which must be less than length of encrypted word. The value of x should be constant represented as S_i.
4. The encrypted X_i is then split into left and right halves (L_i and R_i) where length of L_i is x and length of R_i is length(X_i)-x.
5. Word specific key k_i is then created by combining the left half L_i with the key k' before hashing it.
6. S_i is then combine with the key k_i before being hashed to produce number of bits, equal in length to that of R_i.
7.The final step towards producing the cipher text is to perform a XOR between (Li,Ri) and (Si,F_{ki} (Si)).

Cipher text = (Li,Ri) xor (Si,F_{ki} (Si))

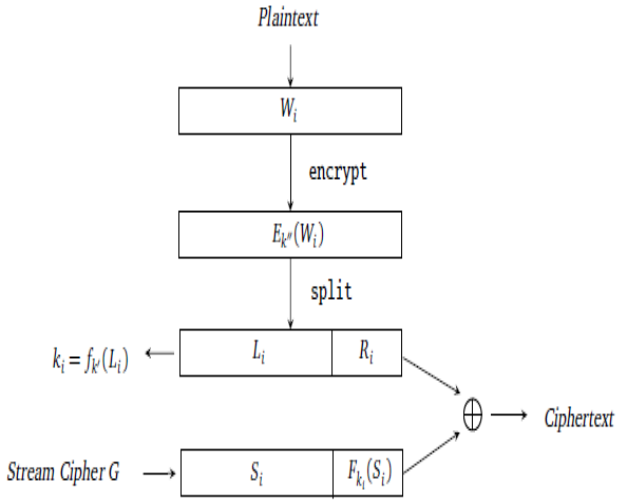

Fig. 1. Steps of Encryption proposed by Song et al

2.1.2.Searching

1.Same key k', k'', k''' are generated using master private key.
2.Word to be search is encrypted using same ECB or CBC mode using key k''.

$X = E_{k''}(W)$

3. x bits of encrypted word are extracted to generate word specific key k.

$k = f_{k'}(L)$

(X,k) are sent to the server. At server side following steps are performed.

1.For each encrypted word block *C* in the document, XOR it with the encrypted word *X*. This will result in the (Si , Fk(Si)) pair.
2. Using x number of bits from the front of the (Si,Fk (Si))pair, retrieve *Si* , the bits that were taken from the stream cipher during the encryption phase.
3. Since both *Si* and *k* (handed to the server by the client in order to search)are known, *Si* can be combined with *k* and hashed using the same process as encryption (step 6)and compared to the right part of the pair - *Fk*(*Si*). If this matches, then the word is found and the current document can be added to a list of documents, ready to be returned to the client after the entire document set is inspected.

Disadvantages:
1. Choice of x is important. From left part L of encrypted word right part R is produced. If length of L increases length of R will decrease. As a result available set of possible hash will also decrease.

Because of this hash collision will occur and document not containing the word will be returned. Such documents are known as false positive.
2. In this search technique we have used fixed length blocks capable of containing most possible input words. If word is shorter then block is padded. This will lead to same cipher text for plain text which will result in statistical analysis.
3. Document containing punctuations and spaces will take large space overhead due to fixed sized block used in this search technique.
4. Case insensitivity, regular expression and sub matches are not supported.

2.2.Secure Indexes by Eu-Jin Goh[5]

Goh's search scheme makes use of a data type known as a Bloom filter [4].
Bloom filter is an array of bits which is initially set to 0.
Before encrypting elements number of hashes are performed and the bit index returned by hash algorithm is set to 1.
Properties of Bloom Filter:
- History independent
- Once added, elements can't be removed

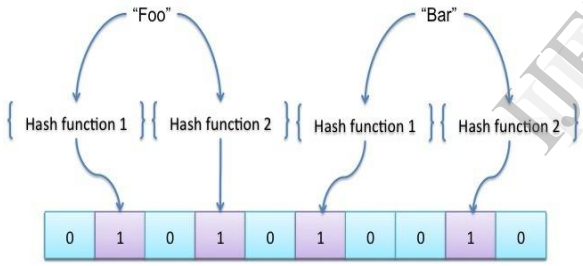

Fig. 2. Example of Bloom Filter

In this manner all keywords of the document are hashed and respective bit positions of the Bloom filter are set to 1.

Key Generation:
Keygen(s) takes security parameter and generates K_{priv} which is set of r keys.
Eg. SHA-512 can be used as $K_{priv}= (k_1, ...,k_r)$ r=16

Trapdoor:
In trapdoor a term is transformed before sending to untrusted server so that server can perform searching without gaining any knowledge. It is generated using private key and series of hash function.
$T_w = (f_{k1}(W), ..., f_{kr}(W))$

2.2.1.Encryption

BuildIndex(D,K_{priv}) takes private key and document D and returns Bloom Filter representing document index. Document identifier D_{id} is used to stop two identical documents from generating the same index.

For each word, W_i , the following algorithm is then performed
1. A trapdoor is constructed from the word Wi and the private key K_{priv} using the Trapdoor (K_{priv} , W) algorithm described previously.
$T_w = (f_{k1} (W_i), ..., f_{kr} (W_i))$

2. A codeword is then constructed based on the trapdoor T_w . This simply takes each element of T_w and hashes it with the document ID.
$C_w = (f_{Did} (T_w \downarrow_1), ..., f_{Did} (Tw \downarrow_r))$

3. This codeword can now be added to the Bloom filter (I_{Did}) that represents the document index.

Once the index is constructed, the plaintext document is encrypted using a standard block cipher and the private key K_{priv}. The tuple containing this encrypted document, the document identifier D_{id} and the index can then be uploaded to the untrusted server.
Cipher text = (Did,I_{Did},E_{Kpriv}(D))

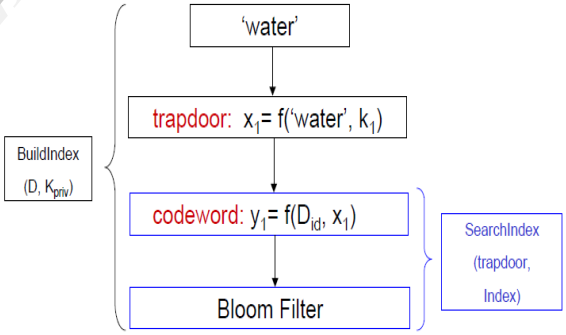

Fig. 3. Encryption Steps proposed by En-Jin Goh

2.2.2.Searching

When user wants to search a word (W) it will pass the word to Trapdoor (K_{priv},W),Trapdoor generated is passed to untrusted server.
At server side following steps are performed:
1. Generation of code word from Trapdoor.
2. Document's Bloom Filter is checked to see if this code word is a member.
3. If bloom filter replies positively, the document is added to the set of documents to be returned to the user.

Drawbacks
1. Bloom filters result in false positives.
2. Updating procedure lacks security analysis.
3.Security model not satisfactory for Boolean searches.
4. Unclear experimental evaluation.
5.It cannot handle regular expression, case insensitivity and sub matches.

2.3 Public Encryption With Keyword Search

This scheme make use of public key cipher technique. Encrypted message is appended with information required to perform searching over data.
For a keyword, a PEKS (Public-key Encryption with Keyword Search) value can be generated which will allow the server to perform a search using a trapdoor.[6]
Consider message M with keywords $\Sigma = W_1, ..., W_i$, the cipher text will be
(E_{Apub} (M), PEKS(Apub, W1), … , PEKS(Apub, Wi))
where Apub is public key of intended recipient.
PEKS Scheme:

- KeyGen(Σ): A pair of keys, A_{pub} and A_{priv} (the public and private keys, respectively) is generated for every keyword in Σ. The final private key A_{priv} is set of all private key and the final public key A_{pub} is set of all public key.
- PEKS(A_{pub}, W):Firstly, a random number, M, is generated. The algorithm then returns the tuple containing this M and its value, encrypted with the public key associated with W (which is in the set $Apub$).
  (M, E^{W}_{k}pub (M))
- Trapdoor(Apriv, W): The trapdoor for a given (Tw) word W is simply its private key, K^{W}_{priv} as defined in the set Apriv.
- Test(Apub,S,Tw): The test algorithm allows an untrusted server, given a trapdoor Tw, the set of public keys Apub and an output from PEKS algorithm (S) to test to see if the word represented by the trapdoor matches the word used to generate the PEKS. This is done by taking the PEKS, S and attempting to decrypt the second element of the tuple with the trapdoor (the private key for the word). If the output of this decryption matches the first element of the tuple, the algorithm returns true, if not, false is returned.
  D_{Tw} (S $\downarrow_2$) $\equiv$S $\downarrow_1$

Disadvantages
1.List of keyword has to determined carefully in order to keep length of message down.
2. Public key algorithms require large prime numbers to be calculated in order to generate usable keys, so this process is potentially very time consuming.

2.4Fuzzy Keyword Search

Fuzzy keyword search greatly enhances system usability by returning the matching files when user's searching inputs exactly match the predefined keywords or the closest possible matching files based on key word similarity semantics, when *exact* match fails.[1].
Fuzzy keyword search technique starts by using following algorithms and concepts:
Keygen (λ): This algorithm is run by the data owner to setup the scheme. It takes a security parameter λ as input, and outputs the trapdoor generation key sk and secret key k.

Buildindex (sk, W): This algorithm is run by the data owner for generation of index. It takes a secret sk and the distinct keyword set W of the documents as inputs, and outputs a symbol-tree G_w.

Trapdoor (sk,Sw,d): This algorithm is run by the user to generate trapdoors for all fuzzy keywords of the user input keyword w. It takes a secret key sk and a fuzzy keyword set Sw,d as inputs, and outputs a trapdoor set {Tw'}w'ε$S_{w,d}$.

Edit Distance: Edit distance is a measure of similarity between two strings. The edit distance
ed (w1,w2) between two words w1 and w2 is the minimum number of operations required to transform one to the other. There are three primitive operations.

- **Substitution**: changing one character to another in a word
- **Deletion**: deleting one character from a word;
- **Insertion**: inserting a single character into a word. Given a keyword w, we let Sw,d denote the set of keywords w' satisfying ed(w;w') <d for a certain integer d.[1]

Using edit distance, the definition of fuzzy keyword search can be formulated as follows:
Input:
1.Collection of encrypted data files, C.
2.Set of distinct keywords W with edit distance d.
3.Search input, (w,k) edit distance is k(k<=d)
Output:
Set of file IDs,FID_w whose files may contain the word w to be searched.
Condition:
If $w = w_i \in W$, then return FID_{wi} ; otherwise, if $w \in W$, then return $\{FID_{wi}\}$, where

ed(w,wi) ≤ k.

2.4.1.Fuzzy Keyword Generation

1.Simple Exhaustive Method

Construction of fuzzy keyword set $S_{wi,d}$ for each keyword $w_i \in W$ with edit distance d is done by listing all possible words w_i that satisfy the similarity criteria ed(w_i, w_i) ≤ d, that is, all the words with edit distance d from w_i are listed. For example, the following is the listing variants after a substitution operation on the first character of keyword CASTLE: {AASTLE, BASTLE,DASTLE, …. , YASTLE, ZASTLE}.

2.Wildcard-based Fuzzy Set Construction

In the above straightforward approach, all the variants of the keywords have to be listed even if an operation is performed at the same position.

The wildcard-based fuzzy set of w_i with edit distance d is denoted as $S_{wi,d}$={$S_{wi,0}$, $S_{wi,1}$, … , $S_{wi,d}$}, where $S_{wi,\tau}$ denotes the set of words w_i with τ wildcards. Note each wildcard represents an edit operation on w_i.[1]

To build a storage-efficient fuzzy keyword set, we utilize the wildcard technique. The idea is to consider the positions of the three primitive edit operations. Namely, we use a wildcard " * " to denote all edit operations at the same position.

For example, for the keyword cat with the pre-set edit distance 1, its wildcard-based fuzzy keyword set can be constructed as

Scat,1 = {cat,*cat,*at, c*at, c*t, ca*t, ca*,cat*}.

The total number of variants on CAT constructed in this way is only 7 + 1, instead of 7 × 26 + 1 as in the above exhaustive enumeration approach when the edit distance is set to be 1.

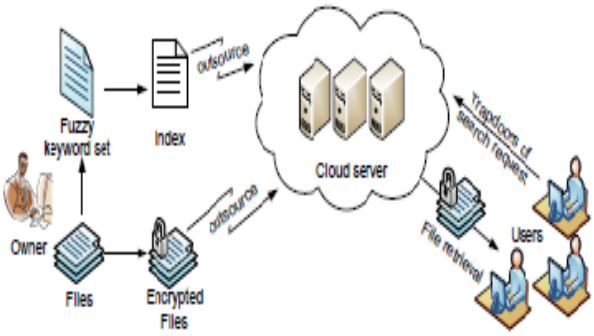

Fig. 4. Architecture of Fuzzy Keyword Search

Fuzzy Keyword Searching

Search(G_w,{Tw'}): This algorithm is run by the server in order to search for the files in C(set of encrypted files) that contain keyword w. It takes the symbol-tree G_w of the file collection C and a trapdoor set {Tw'} of the fuzzy keyword set Sw,d as inputs, and if search is successful outputs IDw and the proof, otherwise outputs the proof .[2]

Verify (k, proof): This algorithm is run by the user to test whether the server is honest. It takes a secret k and proof as inputs, and outputs True if pass, otherwise outputs False.

Disadvantages:

The above all techniques based on searchable encryption supports only Boolean search which has two major drawbacks. They are,

a) User wants to decrypt every file that contains the keyword to match their file when retrieving the file is based on keyword

b) Retrieving all files leads to network traffic.

2.5 Secure Indexing over Encrypted Data

To speed up the execution of queries in databases is to use a pre-computed index. However, once the data is encrypted, the use of standard indexes is not trivial and depends on the encryption function used. And if several users share the database i.e. users with different access rights search on the same database, there are chances of index getting overlapped. Hence the need of developing indexes which can solve this problem is required.

Discretionary Access Control (DAC)

In a multi-user (discretionary) database environment each user only needs access to the database objects (e.g., group of cells, rows and columns) needed to perform his job. Encrypting the whole database using the same key, even if access control mechanisms are used, is not enough. For example, an insider who has the encryption key and bypasses the access control mechanism can access data that are beyond his security group. Encrypting objects from different security groups using different keys ensures that a user who owns a specific key can decrypt only those objects within his security group. Following this approach, different portions of the same database column might be encrypted using different keys. However, a fundamental problem arises when an index is used for that column as illustrated in Fig. 5.

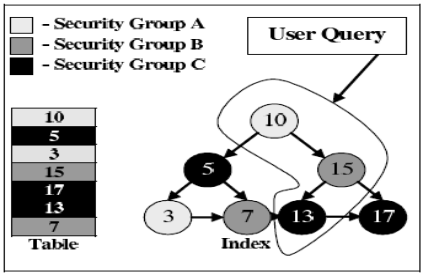

Fig. 5. An Indexed Column Encrypted using Different Keys

Figure 5 illustrates an index that is queried by users who belong to different security groups. Each one of them needs access to the entire index, possibly to indexed elements, which are beyond their access rights. The same problem arises when the index is updated.

Granularity Level

Solution to this problem as proposed is to use finer encryption granularity which affords more flexibility in allowing the server to choose what data to encrypt or decrypt, whereas *Whole Index* level encryption requires the whole index to be decrypted, even if a small number of index nodes are involved in the query. Hence *Single values* level encryption of the index is most suitable which enables decryption of values of interest only and also provides security of access rights.

Comparison of various granularity levels in terms of Information level, unauthorized modifications, structure perseverance and Performance is given in table 1. According to this comparison the best suited method of creating indexes is Single Values level of granularity

Table. 1.Comparing Different Levels of Encryption Granularity

	Information Leakage	Unauthorized Modifications	Structures Perseverance	Performance
Single values	Worst	Worst	Best	Best
Nodes	Low	Low	Medium	Medium
Pages	Medium	Medium	Low	Low
Whole Index	Best	Best	Worst	Worst

In this model, the indexes are encrypted using the Encryption Keys which are present at the client side. The decryption of indexes are done using the keys provided by the user which are transmitted to the server by secure means. These keys are not stored at the server, but are kept only for that particular session. Hence access rights are preserved.
In order to solve the problem of colliding indexes for more than one users, we split the index into several sub-indexes where each sub-index relates to values in the column encrypted using the same key.[8]

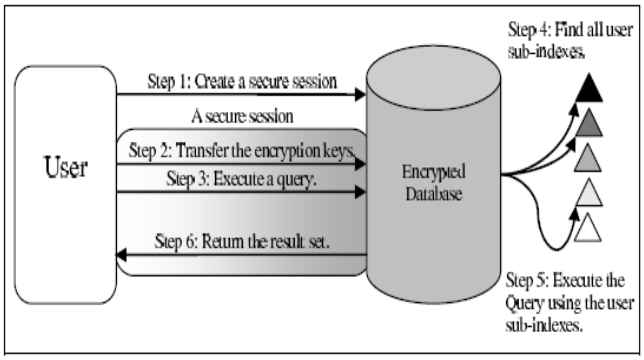

Fig. 6 Illustration of how a query is executed using subindexes.

A secure session between the user and the database server is created (step 1). The user supplies his encryption keys(step 2). During the secure session, the user submits queries to the server (step 3). The server uses the encryption keys in order to find the set of indexes that the current user is entitled to access(step 4). The query is executed on the set of indexes found (step 5). The result set is returned to the user (step 6).

2.6 Practical keyword index search on cloud datacenter

This technique focuses on group search over encrypted database. Search over encrypted data is not restricted between single user and server but it is extended to multiple user and server. As we know in cloud data is shared between multiple user like government organization, or private companies share information with their employee and with period of time new employee join organization and some employee leave organization. Newly joined employee should be able to access all documents and those who left organization should not able to access those documents anymore. Previously discussed techniques does not deal with this situation. Hyun-A Park, Jae Hyun Park and Dong Hoon Lee designed PKIS technique to overcome that defeat.[9]
This scheme is made up of three parties:
(1) users of group members,
(2) a group manager GM, and
(3) a datacenter server DS.
Users of group members are the owners of documents, and they are registered in their organization. GM plays a similar role of a client server, and it is a trusted party. GM manages
the group session keys and the search keys of all groups, for secure communication and secure keyword index search.

Notations

- TG: a huge hierarchical group
- g_i: ith small group of G
- g_i^j: a small group g_i at jth session
- Dn: nth documents
- Wn: keywords list of Dn
- w_n^i: ith keyword of Wn
- dn: identifier of Dn
- gk_i: group session key of a small group gi
- ik_i: index generation key of a small group gi
- dk_i: documents encryption key of a small group gi
- gk_i^j: group session key of gi at jth session
- ik_i^j: index generation key of g_i at jth session
- dk_i^j: documents encryption key of g_i at jth session
- kc: GM's secret key
- f (·): pseudorandom function (PRF)
- h(·): one-way hash function

Algorithm

- SysPara(1^k). It takes an input as a security parameter k and outputs a system parameter **λ**. **λ** determines elements in order to set the encrypted database system such as the size of database, encryption/ decryption algorithm, functions, the size of parameters, and so on.
- KeyGen(**λ**). Taking **λ** as an input, this algorithm generates users' group session key set {gk}, index generation key set {ik}, and document encryption key set {dk}.
- IndGen(ik, W). Inputs of algorithm IndGen are an index generation key ik and a keyword set W. Output is index list table.
- DocEnc(dk, D). Given a document encryption key dk and a document D, this algorithm outputs an encrypted document.
- TrapGen(w, ik). This algorithm takes a keyword w and index generation key ik. It encrypts the keyword w with index generation key ik and returns the encryption value, which is the trapdoor Tw for the keyword w.
- Retrieval(Tw). This algorithm takes input as trapdoor Tw. If there exist matching values to the trapdoor Tw in an index list, then it outputs the encrypted documents that are mapped to the identifiers of the matching values in the index list table.
- Dec(E(D), dk). Given a document encryption key dk and encrypted document E(D), it outputs a plaintext document D.

Practical Keyword Index Search-I (PKIS-I)

If an event of a session-change happens for a subgroup g1, the first session is changed into the second session and then the group session key, a document encryption key, and an index generation key are changed like this: $gk_1^1 \rightarrow gk_1^2$, $dk_1^1 \rightarrow dk_1^2$, $ik_1^1 \rightarrow ik_1^2$

One way hash property prevents leaving members to access documents. Also newly added member will be able to get all keys through applying the current key to hash function h repeatedly.

With every change of session new search key is created. $dk_1^2$and ik_1^2 are encrypted with gk_1^2 and distributed to all members of group g1. While uploading user encrypt document with session search key and send it to GM, GM further encrypt it with its own secret key and then upload it to data server. While searching keyword w, it may be included at any session, so word w is encrypted with all session's search key and passed to GM. GM now encrypt it with it's own secret key and send it to data server. Data server have two tables 'index list' and 'encrypted document'. Index list table contains indexes and document identifier. Encrypted document table stores identifier and encrypted document. If trapdoor for word matches any identifier present in index list then corresponding document is returned.

Practical Keyword Index Search–II (PKIS-II)

Main difference between PKIS-I and PKIS-II is that search key is not changed irrespective of session key. GM keeps the key matching information for groups, which consists of all of the group session keys and group search keys for each group. All users of group members do not know their group search keys. The only thing they know is a group session key. Instead, GM takes users' places for search processes. Rest all functions works in identical manner as that of PKIS-I. When any member leaves the group new session key is created. Session key is distributed to all it's members. As leaving member does not get session key, he will not be able to generate valid trapdoor to search a word because GM decrypts a trapdoor with the group's newly updated session key. When new member get added to group he gets session key. He get access to documents as group search key is unchanged. If new joiner is authenticated as valid user then GM generates valid trapdoor for him for search.

Disadvantage

The common keywords in different documents for certain group have the same index values. Even if an adversary does not know what the keywords mean, the adversary can know that the keywords have something in common. An adversary might guess that two documents have something correlated. This is because we use only deterministic symmetric functions that have the same encryption value under the same data and the same key.

2.7Authorized Private keyword Search (APKS)

Authorized Private Keyword Search (APKS) deals with multi-keyword search, while above techniques conducts with single keyword which misses query flexibility and efficiency. In fine-grained authorization framework, every user obtains searching capabilities authorization from Local Trusted Authority (LTA).Hierarchical Predicate Encryption (HPE), a cryptographic primitive uses attribute hierarchy for simple range queries.

In HPE, given a ciphertext $\mathcal{C}$ for plaintext vector $\vec{x}$ and a secret key $sk_{\vec{v}}$ for predicate vector $\vec{v}$, the decryption will succeed if $\vec{x} \cdot \vec{v} = 0$.[7]

APKS based on HPE

The following steps are required while searching takes place in encrypted data using multi-keyword

a. Multi-dimensional query are converted to its CNF (Conjunctive Normal Form) formula.

b. Attributes are defined in a hierarchical way. i.e., attribute hierarchy.

c. Indexes and capabilities are generated by *GenIndex* and *GenCap* algorithm respectively.

When a user wants to retrieval a file using a keyword from LTA, LTA checks whether the user has an attribute value set and if it matches then user can retrieval the file from the server.

Disadvantage

1.APKS does not prevent keyword attack.

2.APKS+ adds a secret key while encryption and decryption takes place which hides the data from the attackers. Therefore it prevents dictionary keyword attack and accomplishes index privacy and query privacy.

2.8 Privacy-Preserving Multi-keyword Ranked Search over Encrypted Cloud Data

All the technique we saw before this searches single keyword in document but in real world we need to search document containing multiple keywords and it returns all document containing keyword which may include false positive documents as well which in turn increases network overhead of transmission as user may be interested only in top k results of search query. The problem can be solved by Multi-keyword Ranked Search over Encrypted Cloud Data(MRSE) technique.[10]

MRSE uses "co-ordinate matching" principle i.e. as many matches as possible, it is an efficient principle among multi-keyword semantics to refine the result relevance. Specifically, "inner product similarity" , i.e., the number of query keywords appearing in a document is used. During index construction binary vector is associated with every document. Each bit in binary index represent whether corresponding keyword is present in document or not. The search query is also described as a binary vector where each bit means whether corresponding keyword appears in this search request, so the similarity could be exactly measured by inner product of query vector with data vector.

Notations

- F – the plaintext document collection, denoted as a set of m data documents F = (F1, F2,…..,Fm).
- C – the encrypted document collection stored in cloud server, denoted as C = (C1,C2, …,Cm).
- W – the distinct keywords extracted from document collection F, denoted as W = (W1,W2, …,Wn).
- I – the searchable index associated with C, denoted as (I1, I2,…, Im) where each sub index Ii is built for Fi.
- $\hat{W}$ – the subset of W, representing the keywords in a search request, denoted as $\hat{W}$= (W_j1 ,W_j2 ,….,W_jt).
- $T_{\hat{W}}$ – the trapdoor for the search request $\hat{W}$.
- $F_{\hat{W}}$ – the ranked id list of all documents according to their similarity with $\hat{W}$.

MRSE consists of four algorithms as follows.

- Setup(1^l) Taking a security parameter l as input, data owner outputs a symmetric key as SK.
- BuildIndex(F, SK) Based on the dataset F, data owner builds a searchable index I which is encrypted by the symmetric key SK and then outsourced to cloud server.
  After the index construction, the document collection can be independently encrypted and outsourced.
- Trapdoor($\hat{W}$) With t keywords of interest in $\hat{W}$ as input, this algorithm generates a corresponding trapdoor $T_{\hat{W}}$.
- Query($T_{\hat{W}}$, k, I) When cloud server receives a query request as ($T_{\hat{W}}$, k), it performs the ranked search on the index I with the help of trapdoor $T_{\hat{W}}$, and finally returns $F_{\hat{W}}$, the ranked id list of top-k documents sorted by their similarity with $\hat{W}$.

Disadvantage

1.Even though keywords are protected by trapdoors server can do some statistical analysis over search result.

2.Server can generate trapdoor for subset of any multi

keyword trapdoor request.

MRSE_I :Basic Scheme
1)Secure KNN Computation
Euclidean distance is used to find k nearest neighbors. The secret key is composed of one (d+1)-bit vector as S and two (d + 1)× (d + 1) invertible matrices as (M1,M2), where d is the number of fields for each record p. To find k nearest neighbor first data vector and query vector are extended to (d+1)dimension vector where (d+1)bit is set to $-0.5\|p^2\|$ and 1 respectively. Query vector is scaled by random vector r>0 as (rq,r). p and q are split into two vectors and encrypted as $\{M_1^T p', M_2^T p''\}$ and $\{M_1^{-1} q', M_2^{-1} q''\}$ respectively. When user enters query product of data vector and query vector is calculated i.e. $-0.5r(\|p\|^2-2p.q)$. This distance is used to select k nearest neighbors.
2)MRSE_I Scheme

- Setup: After extracting the distinct keywords set W from the document collection F, data owner randomly generates a (n+1)-bit vector as S and two (n+1)×(n+1) invertible matrices (M1,M2). The secret key SK is (S,M1,M2).
- BuildIndex(F,SK): Data owner generates binary vector D_i for every document F_i where each binary bit represent $D_i[j]$ corresponding keyword W[j] in document. Subsequently, every sub index Ii is generated by applying dimension extending, splitting and encrypting procedures on Di.
- Trapdoor($\hat{W}$): With t keywords of interest in $\hat{W}$ as input, Query vector Q is generated which is scaled by random number r≠0 and extended to (n+1) dimension vector(rQ,t) where t is another random number. After applying splitting and encryption process trapdoor $T_{\hat{W}}$ is generated.
- Query($T_{\hat{W}}$, k, I):With trapdoor $T_{\hat{W}}$ cloud server calculates similarity score for each document and top k results are returned.

Advantage

- Random variable t in query vector Q generates different trapdoor for same query so search pattern is well protected.

Disadvantage

- As trapdoor generation process is deterministic trapdoor privacy is violated.

3.Conclusion

In this paper various techniques to find document with required keyword are discussed.Single keyword search technique as well as multi keyword search technique have been discussed. Also technique to deal with when data is shared between multiple user scenario is also discussed in this paper where if few member leave and new member join, new member should be able to access encrypted document and leaving member should not be able to access it anymore.A complete analysis of various techniques was reviewed with each technique's advantages and disadvantages.
From this study we conclude that for optimized searching we would need to generate indexes on database as done in Secure Indexing technique and efficient keyword search would be provided using Ranked Search Algorithm while user authentication to prevent access rights would be done using Secure Indexing.Furthermore this combination can be extended for a scenario of multiple user shared data using Practical Keyword Index Search.

4.References

[1] "Fuzzy Keyword Search over Encrypted Data in Cloud Computing" by Jin Li , Qian Wang, Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou,INFOCOM, 2010 Proceedings IEEE
[2] "A New Efficient Verifiable Fuzzy Keyword Search Scheme" by Jianfeng Wang, Hua Ma, Qiang Tang, Jin Li, Hui Zhu, Siqi Ma, Xiaofeng Chen.
[3] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
[4] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. of IEEE Symposium on Security and Privacy'00*, 2000.
[5] Eu–Jin Goh. "Secure indexes". In the Cryptology ePrint Archive, Report 2003/216, March 2004.
[6] Joonsang Baek and Reihaneh Safiavi-naini and Willy Susilo. Public Key Encryption with Keyword Search Revisited. Cryptology ePrint Archive, Report 2005/191, 2005.
[7] Li, M., S. Yu, N. Cao and W. Lou, 2011. "Authorized private keyword search over encrypted data in cloud computing." Proceedings of the 31st International Conference on Distributed Computing Systems, June 20-24, 2011, Minneapolis, MN., USA.,pp: 383-392.
[8] Designing Secure Indexes for Encrypted Databases by Erez Shmueli, Ronen Waisenberg, Yuval Elovici, and Ehud Gudes.
[9] PKIS: practical keyword index search on cloud datacenter by Hyun-A Park, Jae Hyun Park and Dong Hoon Lee
[10] Preserving Multi-keyword Ranked Search over Encrypted Cloud Data by Ning Caoy, Cong Wangz, Ming Liy, Kui Renz, and Wenjing Louy