International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 5, Issue 1, January 2015)

A Review on Techniques for Searching and Indexing over Encrypted Cloud Data

Aakash Goplani[1], Jyoti Vaswani[2], Sneha Kukreja[3], Prof. Anjali Yeole[4]

[1, 2, 3]*Pursuing Graduation in Computer Engineering, V.E.S. Institute of Technology, Mumbai-74, India.*
[4]*Asst. Prof., Computer Dept., V.E.S. Institute of Technology, Mumbai-74, India.*

***Abstract*— Cloud computing becomes increasingly popular. To protect data privacy, sensitive data should be encrypted by the data owner before outsourcing, which makes the traditional and efficient plaintext keyword search technique useless. It is difficult to both maintain privacy in datasets while still providing adequate retrieval and searching procedures. This paper explores existing technique of searching over encrypted data.**

***Keywords*—cloud computing, multi-keyword, ranking, encrypted phrase, synonym query, fuzzy, privacy-preserving, order-preserving, semantic search.**

I. INTRODUCTION

Cloud computing is a new model of enterprise IT infrastructure that provides on-demand high quality applications and services from a shared pool of configuration computing resources. However, there may be existed unauthorized operation on the outsourced data on account of curiosity or profit. To protect the privacy of sensitive information and combat unauthorized accesses, sensitive data should be encrypted by the data owner before outsourcing. However, encrypted data make the traditional data utilization service based on plaintext keyword search useless. The simple and awkward method of downloading all the data and decrypting locally is obviously impractical, because the data owner and other authorized cloud customers must hope to search their interested data rather than all the data. What's more, taking the potentially huge number of outsourced data and great deal of cloud customers into consideration, it is also difficult to meet both the requirements of performance and system usability. Hence, it is an especially important thing to explore effective search service over encrypted outsourced data.

II. RELATED WORKS AND BACKGROUND

Searchable encryption is not a new concept but all current methods have failed in various aspects that keep them from becoming common or mainstream. Even ranked word proximity searches, a search that ranks results based on how close the query keywords are together, has not been fully implemented by previous research.

Song et al. [1] proposed a searchable encryption scheme based on a symmetric key. The scheme involved provably secure, query isolation for searches, controlled searching, and support hidden query. *Drawbacks*: Case insensitivity, regular expression and sub matches are not supported. Speed of searching and total space required is huge.

Eu-Jin Goh [2] proposed an index searching algorithm based on a bloom filter. Their scheme reduced the computational overhead of loading an index and searching for files. *Drawbacks*: Bloom filters result in false positives; updating procedure lacks security analysis, Security model not satisfactory for Boolean searches, unclear experimental evaluation.

PEKS: Public Encryption Keyword Search [3] makes use of public key cipher technique. This technique focuses on refreshing keywords, removing secure channel and processing multiple keywords. *Drawbacks:* List of keyword has to be determined carefully in order to keep length of message down. Public key algorithms require large prime numbers to be calculated in order to generate usable keys, so this process is potentially very time consuming.

PKIS: Practical Keyword Index Search on cloud datacenter [4] focuses on group search over encrypted database. It involves two schemes: PKIS-1 and PKIS-2. PKIS provide practical, realistic, and secure solutions over the encrypted DB. *Drawbacks:* The common keywords in different documents for certain group have the same index values, which leads to Brute Force attacks.

APKS: Authorized private keyword search over encrypted data in cloud computing [5] deals with multi-keyword search. Multi-dimensional query are converted to its CNF (Conjunctive Normal Form) formula and are arranged in a hierarchical way. *Drawbacks:* APKS does not prevent keyword attack.

Multi-keyword Ranked Search over Encrypted Cloud Data (MRSE) [6] 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.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 5, Issue 1, January 2015)

Drawbacks: Even though keywords are protected by trapdoors server can do some statistical analysis over search result. Server can generate trapdoor for subset of any multi keyword trapdoor request.

This paper presents the novel technique that provides sub-word matching, exact-matches, regular expressions, natural language searches, frequency ranking, and proximity-based queries i.e. all forms of searching that modern search engines employ and users expect to have.

III. Techniques To Perform Searching Over Encrypted Data

3.1 Encrypted Phrase Searching in the Cloud [7]

This technique allows both encrypted phrase searches and proximity ranked multi-keyword searches to encrypted datasets on untrusted cloud.

A. Overview

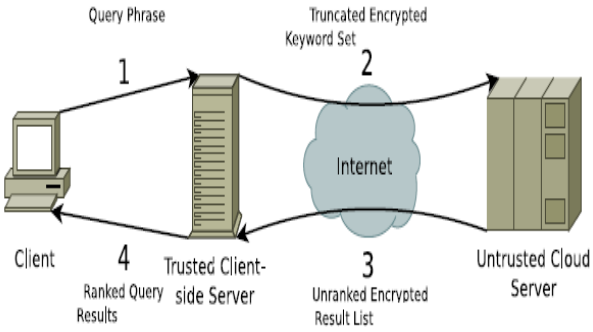

Fig. 1: Flowchart of the proposed encrypted phrase searching procedure.

1) The client sends a plaintext search query to a trusted client-side server.
2) The Client-side server encrypts all keywords in the search query individually using symmetric-key encryption; it then truncates the encrypted keywords to a set number of bits to improve security by allowing for collisions, and queries the untrusted cloud server for the documents containing the set of truncated encrypted keywords.
3) The cloud server does a database query of its encrypted index and returns to the client-side server encrypted data that corresponds to document paths, truncated encrypted keyword index offset, and encrypted keyword locations.
4) The client-side server decrypts this data first. From the newly decrypted keyword index offset it can then determine which returned results are actually for the keywords searched and which are simply collisions. It discards those collisions and filters and/or ranks the pertinent returned documents based on relevant keyword locations and frequency. Finally, it sends this ranked listing to the original client.

B. Indexing

Prior to searching for a document δ, an encrypted index of the corpus must be generated by the trusted client-side server. The index is then encrypted and sent to the untrusted cloud server. Each row in the encrypted index table corresponds to one document $\delta \in corpus$. Each row contains two columns: an arbitrarily assigned unique document id (ID) and a specialized data structure that contains truncated symmetric-key encrypted keywords associated with encrypted versions of the keyword's location in δ (Word Vectors). In addition, this data string contains an offset that is used to map the truncated encrypted keyword with its full version (stored on the trust client-side server). The cryptographic keys used for both encryptions are the same key $\mathcal{K}$ and that $\mathcal{K}$ can be used for decryption as well. Only the trusted client-side server has access to the value of $\mathcal{K}$. Once the encrypted index is transferred to the untrusted cloud server, an inverted index, based on the encrypted index, is generated by the cloud server to facilitate the searching speed of the index.

C. Keyword Truncation

To improve the security each encrypted keyword is truncated to a predefined number of bits β. The trusted client-side server creates a unique keyword truncation index value for each encrypted keyword. A table that is stored on the client-side server maps the truncated index value with the fully encrypted keyword. Each index number is stored along with the keyword locations in the encrypted index. This string is encrypted using AES and a randomized salt. As many multiple keywords now map to the same bits it makes any statistical frequency analysis attack far less useful.

D. Searching

When the search begins, the client sends the query phrase with multiple keywords, $k_1, \ldots, k_n$, to a client-side server, which concatenates the keywords to a list, K. The client-side server then encrypts each $k \in K$ using $\mathcal{K}$ in which the order of keywords is randomized. Each keyword in this list is truncated to β bits to create the encrypted keyword list, K'. The client-side server then transfers this encrypted query, K', to the untrusted cloud server.

The untrusted cloud server parses K' into individual encrypted keywords k' and, using the inverted index, determines the documents, δ, that contain a k'.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 5, Issue 1, January 2015)

The selected IDs, the keyword locations, and the truncated keyword index, are sent back to the trusted client-side server.

The client-side server parses the results that the cloud server returns, which include the document's IDs, paths, and associated encrypted keywords, index, and encrypted locations l'. Each l_i' is decrypted to the truncation index τ and l_i. A proximity ranking function R, hosted by the client-side server and is utilized to meaningfully rank the results.

Disadvantages:

1. Since it is impossible, prior to decryption, to determine which keywords in the collision set were actually being searched for, they must all be returned, decrypted, and then filtered.
2. Does not support fuzzy keyword search.
3. Does not support synonym query.
4. Not suitable for multi-user environment.

3.2. Programmable Order-Preserving Secure Index for Encrypted Database Query [8]

This technique proposes an order-preserving scheme for indexing encrypted data, which facilitates the range queries over encrypted databases. The scheme is secure since it randomizes each index with noises, such that the original data cannot be recovered from indexes. Moreover, scheme allows the programmability of basic indexing expressions and thus the distribution of the original data can be hidden from the indexes.

In this scheme, the i^{th} value in the plaintext domain is mapped to the i^{th} value in the cipher text domain, such that the order between plaintexts is preserved between cipher texts.

The scheme is built over the simple linear expressions of the form $a * x + b$. The form of the expressions is public, however the coefficients a and b are kept secret (not known by attackers). Based on the linear expressions, the indexing scheme maps an input value v to $a * v + b +$ noise, where noise is a random value. The noise is carefully selected, such that the order of input values is preserved. Indexing scheme allows the programmability of basic indexing expressions (i.e., the linear expressions). Users can make an indexing program that deals with different input values with different indexing expressions. On the one hand, the programmability improves the robustness of the scheme against brute-force attacks since there are more indexing expressions to attack. On the other hand, the programmability can help decouple the distributions of input values and indexes. When a single linear expression is used to index all input values, the distribution of indexes is identical to the distribution of input values.

This problem can be addressed by designing appropriate indexing programs.

3.2.1. Randomized Order-Preserving Indexing Over Integers:

Suppose v_1 and v_2 are two integers and $v_1 > v_2$. Then, the gap between them is at least 1, that is $v_1 - v_2 \geq 1$. We will use sensitivity to mean the least gap. To determine how much noise can be added into indexes, such that the indexes keep the order, we need to know the least gap; hence noise is selected in the range $[0, a * 1)$.

Indexes are generated as: $a * v_1 + b + \text{noise}_1 = i_1$, $a * v_2 + b + \text{noise}_2 = i_2$ and so on.

3.2.2. Programmability of Indexes:

This section describes how to compose basic indexing expressions (skindex or rindex) into indexing programs.

Suppose v is an input value. Then, $I(v)$ means the application of I to v, generating v's index. If I is $\text{rindex}^{sens}_{[a,b]}$, then $I(v) = \text{rindex}^{sens}_{[a,b]}(v)$. If I is S; rindexsens [a,b], then $I(v) = \text{rindex}^{sens}_{[a,b]}(i)$, where $i = S(v)$. The semantics of indexing steps S is defined inductively. If S is $\text{skindex}^{sens}_{[a,b]}$, then $S(v) = \text{skindex}^{sens}_{[a,b]}(v)$. If S is the conditional indexing step, then $S(v) = S_1(v)$ if v makes the condition C true; otherwise, $S(v) = S_2(v)$. The condition C is gt(c) or ge(c). The condition gt(c) is true if $v > c$, and ge(c) is true if $v \geq c$. If S is a sequential composition of steps, then $S(v) = S_2(i)$, where $i = S_1(v)$.

An indexing program is said well-formed if it is order preserving. Since in an indexing program the basic indexing expressions skindex and rindex are already order preserving, it is order-preserving if all conditional indexing expressions are also order-preserving indexes generated by S_1 and S_2.

In an indexing program that consists of a sequence of expressions, all intermediate indexes are calculated by skindex, which does not change the sensitivity of input values. Hence, programmers can use the sensitivity of input values in the whole program, easing the burden of programming.

3.2.3. Query of Encrypted Databases:

3.2.3.1 Creation of Encrypted Databases and Tables

To create a database and a table, the database application can issue the following two statements.

create database *dbname*

create table *tblname* (*colnm Type,...*)

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 5, Issue 1, January 2015)

In the statement above, *Type* is the data type for the column *colnm*. The statements are translated into the following statements by the proxy. In addition, the proxy records the schema of the created table in its metadata.

create database *Hash* (*k, dbname*)
create table *Hash* (*k, tblname*)
(*Hash* (*k, colnm* + *"EqIdx"*) String,
Hash (*k, colnm* + *"RngIdx"*) Num,
Hash (*k, colnm* + *"Enc"*) String,...)

That is, three columns are created for the column *colnm*. The column *colnm* + "*EqIdx*" have the type String, since its values are always hexadecimal strings generated by secure hash functions. The values of column *colnm* + "*RngIdx*" are generated by our indexing mechanism and have the numerical type. The column *colnm* + "*Enc*" for cipher text also have the type String.

3.2.3.2. Insertion of Values into Tables:

After a table is created, the database application can put a new record into the table by using the following statement.

insert into *tblname* (*colnm,...*)
values (*v,...*)

In the new statement, the value v is hashed, indexed and encrypted for storing into different columns.

insert into *Hash* (*k, tblname*)
(*Hash* (*k, colnm* + "*EqIdx*"),
Hash (*k, colnm* + "*RngIdx*"),
Hash (*k, colnm* + "*Enc*"),...)
values (*Hash* (*k, v*), *Index*(*v, sens*), *Enc*(*k, v*),...)

3.2.3.3. Queries:

A query from the database application can take the following basic form.

select *colnm*,... from *tblname* where *cond*

The condition *colnm* < *c* is translated into *Hash* (*k, colnm* + "*RngIdx*") < *Index*(*c, 0*). Recall that *Index*(*c, 0*) is the minimum index of *c*. The condition *colnm* = *c* is simply translated into *Hash* (*k, colnm +"EqIdx"*) = *Hash (k, c).* Assume the sensitivity of values in the colnm column is sens. Then, *c* + *sens* is the next value of *c*, and *colnm* > *c* is equivalent to the new condition *colnm* ≥ *c* + *sens*, which is translated into *Hash*(*k, colnm* + "*RngIdx*") ≥ *Index*(*c*+*sens*,0).

Note that *Index*(*c*+*sens*,0) is the minimum index of *c* + *sens*. The keywords order by colnm and group by colnm are frequently used in queries.

They are translated into order by *Hash* (*k, colnm* + "*RngIdx*") and group by *Hash* (*k, colnm +"EqIdx"*), respectively.

3.3 Privacy- Preserving Keyword-based Semantic Search over Encrypted Cloud Data [9]

The Semantic search technique reinforces the system usability by returning the exactly matched files and the files including the terms semantically similar to the query keyword. The co-occurrence of terms is used as the metric to evaluate the semantic distance between terms in semantic relationship library (SRL). The SRL is constructed as a weighted graph structure.

A. Overview

1. Data owner has a collection of text files the owner then constructs a metadata for each file and outsources the encrypted metadata set to the private cloud server. The text files are encrypted by using traditional symmetric encryption algorithm and uploaded to the public cloud server.
2. Private cloud server constructs the inverted index and semantic relationship library using metadata set provided by data user. Then the Inverted index is outsourced to the public cloud server for retrieval.

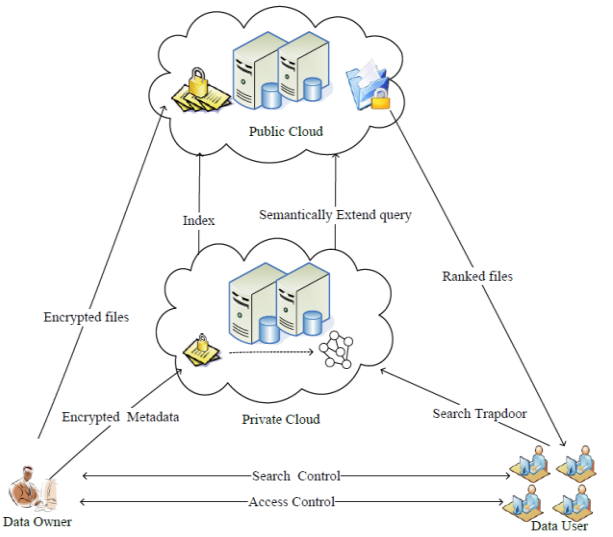

Fig. 2: Architecture of the Semantic Search Scheme.

3. The authorized data users provide the search trapdoor to the private cloud server. Here, the authorization between the data owner and users is appropriately done.
4. Upon receiving the request, the private cloud server extends the query keyword upon SRL and uploads the extended query keywords set to the public cloud.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 5, Issue 1, January 2015)

5. Upon receiving the search request, the public cloud retrieves the index, and returns the matching files to the user in order.
6. Finally, the access control mechanism is employed to manage the capability of the user to decrypt the received files.

B. Notations

- F - The plaintext file collection denoted as a set of n data files $F = (F_1, F_2, \ldots, F_n)$
- M - The encrypted metadata set, denoted as $\{M_1, M_2, \ldots, M_n\}$, where M_i is built for F_i.
- I - The inverted index built from the metadata set by the server, including a set of posting lists.
- w - The distinct keywords set extracted from F_i.
- T_w - the trapdoor generated for a query keyword w by a user.

C. Algorithm

- *KeyGen*(k, l, $\hat{l}$, p): In this algorithm the data owner takes k, l, $1^{\hat{l}}$, p as inputs and generates the random keys. It generates the random keys in this way: $x \leftarrow \{0, 1\}^k$, $y \leftarrow \{0, 1\}^l$ and outputs $Key = \{x, y, 1^l, 1^{\hat{l}}, 1^p\}$.
- *BuildMD*(*key*, F): The data owner builds the secure metadata for each file in file collection F.
- *BuildIndex*(M): On receiving the secure metadata, the private cloud builds the inverted index.
- *BuildSRL*(M): This algorithm is also run by the private cloud server to construct the semantic relationship library. It takes the metadata set as inputs, and exploits data mining algorithm *e.g.*, Apriori algorithm, to discover the co-occurrence probabilities of keywords.
- *TrapdoorGen(Key*, w): In this process, for a search input, the user computes a trapdoor and sends it to the private cloud. The second step is for the private cloud to extend the query *trapdoor* and obtain the extensional query keywords set.
- *Search*(T_w, I): upon receiving the request, the search can be divided into two steps : The first step is for the private cloud to extend the query trapdoor and sends the extensional query keywords set to the public cloud. The second step is for the public cloud to locate the matching entries of the index which include the file identifiers and the associated order-preserved encrypted scores. The public cloud server then computes the total relevance score of each file to the query. In the end, the public server sends back the matched files in a ranked sequence, or sends top-k most relevant files.

D. Security Analysis

For one – to – many Order Preserving Encryption (OPE)

The one-to-many order-preserving encryption introduce the file ID as the additional seed in the cipher text chosen process, so the same plaintext will not be deterministically mapped to the same cipher text , but a random value in the allocated bucket form range by sampling, which help flatten the score distribution, and protect the keyword privacy form statistical attack.

For Ranked Semantic keyword search

1. *File Confidentiality:* The file is encrypted with traditional symmetric encryption algorithm. The encryption cipher is preserved by the data owner only, so the file confidentiality depends on the inherently security strength of the symmetric encryption scheme. Thus the file content is protected.
2. *Keyword Privacy:* If the data owner properly enlarges the range R, the relevance score will be randomly mapped to only a sequence of order-preserved numeric values with very low duplicates. The flattened encrypted relevance score distribution makes it difficult for the adversary to predict the plaintext score distribution, let alone predict the keywords.

Search Result analysis:

The overall recall rate is improved, and the query results are more in line with the user's actual intentions. For *e.g.*, a user inputs a keyword 'protocol', the files which contain related words like 'internet', 'network' ,'authentication' will also be returned, in addition, the files which include most of the words will also be ranked forward.

Disadvantages:

1. Does not support multi-user environment.
2. The size of range cannot be unboundedly large. So the range size | R | should be properly tradeoff between randomness and efficiency.
3. Does not support synonym query.

3.4 Verifiable Attribute-based Keyword Search with Fine-grained Owner-enforced Search Authorization in the Cloud [10]

The outsourced dataset can be contributed from multiple owners and are searchable by multiple users, i.e. multi-user multi-contributor case. The attribute-based keyword search scheme with efficient user revocation (ABKS-UR) enables scalable fine-grained (i.e. file-level) search authorization (*fine-grained owner-enforced search authorization*) using attribute based (CP-ABE) technique.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 5, Issue 1, January 2015)

In the CP-ABE technique specifically, for each file, the data owner generates an access-policy-protected secure index, where the access structure is expressed as a series of AND gates. Only authorized users with attributes satisfying the access policies can obtain matching result. Users can generate their own search capabilities without relying on an always online trusted authority. The scheme enables authenticity check over the returned search results. The index is encrypted with an access structure rather than public or secret keys based on the attributes (properties of users) of authorized users, which makes the proposed scheme more scalable and suitable for the large scale file sharing system. In this key policy attribute-based encryption (KP-ABE) scheme, cipher text can be decrypted only if the attributes that are used for encryption satisfy the access structure on the user private key.

A. Overview

1. The data owner generates the secure indexes with attribute-based access policies before outsourcing them along with the encrypted data into the CS (cloud server).
2. To search the datasets contributed from various data owners, a data user generates a trapdoor of keyword of interest using his private key and submits it to the CS. So as to accelerate the entire search process, enforce the coarse-grained *dataset search authorization* with the *per-dataset* user list such that search does not need to go to a particular dataset if the user is not on the corresponding user list.

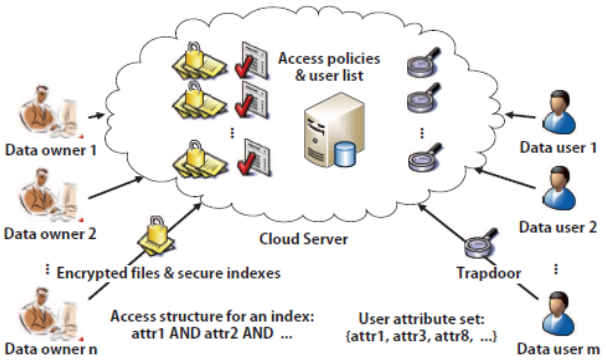

Fig. 3: Framework of authorized keyword search over encrypted cloud data.

3. The fine grained *file-level* search authorization is applied on the authorized dataset in the sense that only users, who are granted to access a particular file, can search this file for the intended keyword. The data owner defines an access policy for each uploaded file.
4. The CS will search the corresponding datasets and return the valid search result to the user if and only if the attributes of the user on the trapdoor satisfy the access policies of the secure indexes of the returned files, and the intended keyword is found in these files.

B. Search phase

In the search phase, the CS returns the search result along with the auxiliary information for result authenticity check later by the data user. The auxiliary information includes all the user list bloom filters BF_{UL} of the datasets stored on the server , the keyword bloom filters BF*w* of the datasets that the user is authorized to access, the file list L'_w for the intended keyword *w*. If the search result contains files from this dataset, the tuple in each related UL (user list) and all the corresponding signatures. If the search result does not contain files from this dataset, it is not necessary to return the corresponding file list. Otherwise, the CS generates L'_w as follows. For the file li in L_w but not in the search result, the CS merely computes its hash value $h(l_i)$ and puts the tuple $< D_i, w, h(l_i) >$ in L'_w.

C. Result Authentication

The result may contain errors that may come from the possible storage corruption, software malfunction, and intention to save computational resources by the server, etc. assure data user of the authenticity of the returned search result by checking its *correctness* (the returned search result indeed exist in the dataset and remain intact), *completeness* (no qualified files are omitted from the search result), and *freshness* (the returned result is obtained from the latest version of the dataset).

The user first computes tuple hash values h_{l1}, h_{l2} and h_{l3} respectively. User then generates the hash chain to obtain the file list hash value h_{Lw}, and verifies $\sigma(h_{Lw})$. Next, user can search this list with his trapdoor and corresponding $\overline{D}_f$ from the CS to check if all the matching files have been returned. Thus, the data user can ensure the authenticity of the returned search result.

D. User Revocation

The aim is to efficiently revoke users from the current system while minimizing the impact on the remaining legitimate users. To revoke a user from current system, we re-encrypt the secure indexes stored on the server and update the remaining legitimate users' secret keys. These tasks can be delegated to the CS using proxy re-encryption technique so that user revocation is very efficient. In particular, the TA (Trusted authority) adopts the re-encryption key generation algorithm to generate the re-encryption key set.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 5, Issue 1, January 2015)

Disadvantages:

1. Does not support synonym query.
2. Does not support semantics based keyword search.

3.5 Privacy-Preserving Multi-Keyword Fuzzy Search over Encrypted Data in the Cloud [11]

The multi-keyword fuzzy search scheme exploits the locality-sensitive hashing technique. This scheme eliminates the requirement of a predefined keyword dictionary.

A. Overview

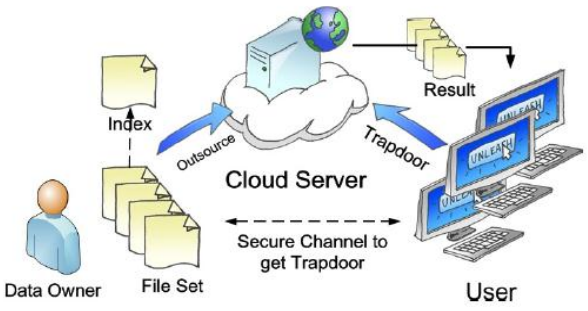

Fig. 4: Architecture of the Multi-keyword fuzzy Search Scheme

1. To outsource a set of files to the cloud, the *data owner* builds a secure searchable index for the file set and then uploads the encrypted files, together with the secure index, to the *cloud server*.
2. To search over the encrypted files, an authorized *user* first obtains the trapdoor, i.e., the "encrypted" version of search keyword(s), from the data owner, and then submits the trapdoor to the cloud server.
3. Upon receiving the trapdoor, the cloud server executes the search algorithm over the secure indexes and returns the matched files to the user as the search result.

B. Locality-Sensitive Hashing

A Locality-Sensitive Hashing (LSH) function hashes close items to the same hash value with higher probability than the items that are far apart. To support fuzzy and multiple keyword search, we first convert each keyword into a bigram vector and then use LSH functions instead of standard hash functions to insert the keywords into the Bloom filter *ID*.

C. Main Idea

1) Bigram vector representation of keyword: Bigram vector is a 26^2-bit long vector that represents a bigram set. Each element in the vector represents one of the 26^2 possible bigrams.

For example, the bigram set of keyword "network" is {ne, et, tw, wo, or, rk}. The element is set to 1 if the corresponding bigram exists in the bigram set of a given keyword. It is not sensitive to the position of misspelling, nor is it sensitive to which letter it was misspelled to. "n*w*twork", "n*v*twork", or "netwo*y*k" will all be mapped to a vector with two-element difference from the original vector.

2) Bloom filter representation of index/query: Bloom filter has been used to build per document index for single keyword exact search scenario. With those hash functions, two similar inputs, even if they are only off by one bit, will be hashed to two totally different random values. Therefore, they can only be used for exact keyword search. LSH functions will hash inputs with similarity within certain threshold into the same output with high probability.

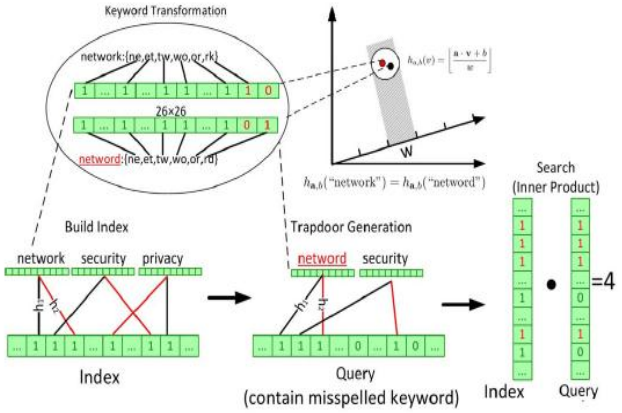

Fig. 5: i).Transform a keyword into a vector. ii).Use two LSH functions h_1, h_2 from the same hash family to generate the index and the query. The word "network" has the same hash value with the misspelled work "netword" under LSH function $h_{a,b}$ because the Euclidean distance between their vector representations is within the pre-defined threshold. iii).The misspelled query matches exactly with the index contains the keywords "network" and "security".

Fig. 5 shows the idea that a misspelled keyword "netword" in the user query is hashed into the same bucket as the correctly spelled keyword "network" so that a match can be found during the search process. The use of LSH functions in building the per-file Bloom filter based index is the key to implementing fuzzy search.

3) Inner product based matching algorithm: The query can be generated in the by inserting multiple keywords to be searched into a query Bloom filter. The search can then be done by qualifying the relevance of the query to each file, which is done through a simple inner product of the index vector and the query vector.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 5, Issue 1, January 2015)

If a document contains the keyword(s) in the query, the corresponding bits in both vectors will be 1 thus the inner product will return a high value. This simple inner product result thus is a good measure of the number of matching keywords.

- *Known Ciphertext Model:* The cloud server can only access the encrypted files, the secure indexes and the submitted trapdoors. The cloud server can also know and record the search results. The semantic meaning of this threat scenario is captured by the non-adaptive attack model.
- *Known Background Model:* The cloud server knows additional background information in this model. The background refers to the information which can be learned from a comparable dataset, for example, the keywords and their statistical information, such as the frequency.

This technique is based on under *known background model*, the adversary potentially can recover the encrypted indexes through linear analysis and further infer the keywords in the index. To secure the linkage between the keywords and the Bloom filter, we introduce an extra security layer, i.e., a pseudo-random function *f*.

D. Algorithm

- *KeyGen*(*m, s*): This algorithm Given a parameter *m*, generates the secret key $SK(M_1,M_2, S)$, where $M_1,M_2 \in \mathbb{R}^{m\times m}$ are invertible matrices while $S \in \{0, 1\}_m$ is a vector. Given another parameter *s*, generate the hash key pool (HK).
- *BuildIndex*(*D, SK, l*) : Choose *l* independent LSH functions from the *p*-stable LSH family *H* and one pseudorandom function $f : \{0, 1\}^* \times \{0, 1\}_s \rightarrow \{0, 1\}^*$.For each file *D*, ,

1) Extract the keywords set $W = \{w_1, w_2, \ldots\}$ from *D*.
2) Generate a *m*-bit Bloom filter I_D. Insert *W* into I_D using the hash functions $\{g_i \mid g_i = f_{ki} \circ h_i, h_i \in H, 1 \le i \le l\}$.
3) Encrypt the I_D with *SK* and return $Enc_{SK}(I_D)$ as the index.

- *Trapdoor*(*Q, SK*): Generate a *m*-bit long Bloom filter. Insert the *Q* using the same hash functions g_i, i.e., $g_i = f_{ki} \circ h_i, h_i \in H, 1 \le i \le l$ into the Bloom filter. Encrypt the *Q* with *SK* and return the $Enc_{SK}(Q)$ as the trapdoor.
- *Search*($Enc_{SK}(Q)$,$Enc_{SK}(I_D)$): Output the inner product $< Enc_{SK}(Q), Enc_{SK}(I_D) >$ as the search result for the query *Q* and the document *D*.

Disadvantages:

1. The technique is vulnerable to predictive attacks under known cipher text model.
2. Does not support synonym query.
3. Not suitable for multi-user environment.
4. Does not allow semantics based search.

3.6 Multi-keyword Ranked Search over Encrypted Cloud Data Supporting Synonym Query [12]

The technique proposes a semantics-based multi-keyword ranked search scheme over encrypted cloud data which supports synonym query. The search results can be achieved when authorized cloud customers input the synonyms of the predefined keywords, not the exact or fuzzy matching keywords, due to the possible synonym substitution and/or her lack of exact knowledge about the data.

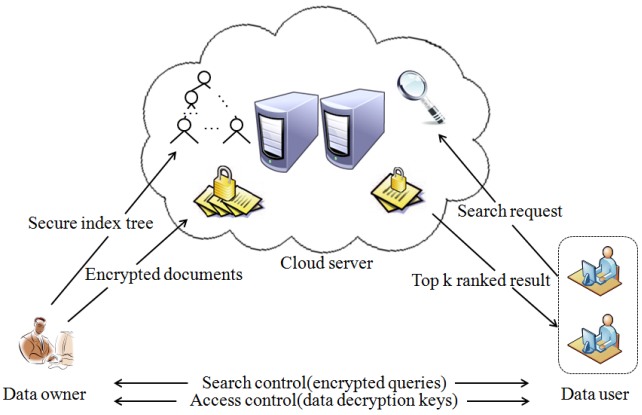

Fig. 6: Architecture of the Multi-keyword ranked search supporting synonym query

Notations

- *DC* – the plaintext document collection, expressed as a set of m documents $DC = \{d \mid d_1, d_2, d_3..., d_m\}$.
- *C* – the encrypted form of *DC* stored in the cloud server, expressed as $C = \{c \mid c_1, c_2... ,c_m\}$.
- *W* – the keyword dictionary, including n keywords, expressed as $W = \{w \mid w_1, w_2... ,w_n\}$.
- *I* – the searchable index tree generated from the whole document set *DC*. (Each leaf node in the index tree is associated with a document in *DC*.)
- D_d – the index vector of document *d* for all the keywords in .
- *Q* – the query vector for the keyword set $\bar{W}$.
- $\bar{D}_d$ – the encrypted form of D_d.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 5, Issue 1, January 2015)

Rank function: In information retrieval, a ranking function is usually used to evaluate relevant scores of matching files to a request. The ranking function used here is "*TF*×*IDF*" where *TF* (term frequency) denotes the occurrence of the term appearing in the document, and *IDF* (inverse document frequency) is often obtained by dividing the total number of documents by the number of files containing the term. That means, *TF* represents the importance of the term in the document and *IDF* indicates the importance or degree of distinction in the whole document collection. Each document is corresponding to an index vector D_d that stores normalized *TF* weight, and the query vector *Q* stores normalized *IDF* weight. Each dimension of D_d or *Q* is related to a keyword in *W*, and the order is same with that in *W*, that is, $D_d[i]$ is corresponding to keyword w_i in *W*. The notations used in similarity evaluation function are showed as follows:

- $f_{d,j}$, the TF of keyword w_j within the document *d;*
- f_j , the number of documents containing the keyword w_j ;
- *M*, the total number of documents in the document collection*;*
- *N*, the total number of keywords in the keyword dictionary*;*
- $w_{d,j}$, the TF weight computed from $f_{d,j}$*;*
- $w_{q,j}$, the IDF weight computed from *N* and f_j*;*

The definition of the similarity function is as follows:

$$SC(Q, D_d) = \frac{\sum_{j=1}^{N} w_{d,j} \cdot w_{q,j}}{\sqrt{\sum_{j=1}^{N}(w_{d,j})^2}\sqrt{\sum_{j=1}^{N}(w_{q,j})^2}}$$

Where $w_{q,j}$ = 1+ ln $f_{d,j}$. The normalized *TF* and *IDF* weight

are $\frac{w_{d,j}}{\sqrt{\sum_{j=1}^{N}(w_{d,j})^2}}$ and $\frac{w_{q,j}}{\sqrt{\sum_{j=1}^{N}(w_{q,j})^2}}$ respectively and the vectors *Q* and D_d are unit vectors.

Construction of keyword set extended by synonym:

Let *N* be the total number of texts in corpus, let *n* be the number of texts containing the term *i* in corpus, let E_1 be the number of texts in the largest category containing the term *i*, let E_2 be the number of texts in the second largest category containing the term *i*. The new weighting factor C_d is added to the formula of TFIDF, the improved formula is as follows:

$$W_{ik} = TF*IDF*C_d = TF* \frac{1}{IDF} *C_d = f_{ik} * \log\frac{N}{n_k} * \frac{E_1 - E_2}{n}$$

So the keywords are extracted from each outsourced text document by using our improved method. All keywords extracted from the same one text form one keyword subset, and all subsets form the keyword set at last. All the outsourced text documents can be expressed as follows:

$$\begin{cases} \text{file 1: } k_{f_1}^1, k_{f_1}^2\ k_{f_1}^3 \ldots, k_{f_1}^{n-1}, k_{f_1}^n \\ \text{file 2: } k_{f_2}^1, k_2^2\ k_{f_2}^3 \ldots, k_{f_2}^{n-1}, k_{f_2}^n \\ \ldots\ldots \\ \ldots\ldots \\ \text{file m: } k_{f_m}^1, k_{f_m}^2\ k_{f_m}^3 \ldots, k_{f_m}^{n-1}, k_{f_m}^n \end{cases}$$

We build a common synonym thesaurus on the foundation of the New American Roget's College Thesaurus (NARCT). NARCT is decreased in quantity by us according to the following two principles:

(1) Selecting the common words;
(2) Selecting the words which can be semantically substituted completely.

The constructed synonym set contains a total of 6353 synonym groups after the reduction. The keyword set is extended by using our constructed synonym thesaurus. The new keyword set containing synonym is shown as follows:

$$\begin{cases} \text{file 1: } k_{f_1}^1 \text{ or } s_1, k_{f_1}^2 \text{or } s_2, k_{f_1}^3 \text{or } s_3 \ldots, k_{f_1}^{n-1} \text{ or } s_{n-1}, k_{f_1}^n \text{or } s_n \\ \text{file 2: } k_{f_2}^1 \text{ or } s_1, k_{f_2}^2 \text{or } s_2, k_{f_2}^3 \text{or } s_3 \ldots, k_{f_2}^{n-1} \text{ or } s_{n-1}, k_{f_2}^n \text{or } s_n \\ \ldots\ldots \\ \ldots\ldots \\ \text{file m: } k_{f_m}^1 \text{ or } s_1, k_{f_m}^2 \text{or } s_2, k_{f_m}^3 \text{or } s_3 \ldots, k_{f_m}^{n-1} \text{ or } s_{n-1}, k_{f_m}^n \text{or } s_n \end{cases}$$

Where s_1 represents the synonym of k_{fi}. If a keyword has two or more synonyms, then all synonyms are added into the keyword set. The repetitive keywords are deleted to reduce the burden of storage. At last, a simplified keyword set and corresponding keyword scoring table are constructed and used.

Disadvantage:

1. Does not support syntactic transformation, anaphora resolution and other natural language processing technology.

IV. Conclusion

In this paper various techniques to find documents with required keywords are discussed. To make the search of required documents more accurate these techniques can be integrated that will provide the user with ease of supporting multi keyword search, fuzzy keyword search, search based on synonym words in the query, semantics based search.

The efficient keyword search would be provided using Ranked Search Algorithm based on proximity and similarity based ranking techniques. Also technique to deal with indexing and searching on encrypted databases and data shared between multi-users, multi-contributors scenarios is discussed in this paper. To provide confidentiality user authentication, user revocation and authentication of returned results is done using Fine-grained Owner-enforced Search Authorization. To provide better security at server technique that supports keyword truncation is used.

REFERENCES

[1] Dawn Xiaodong, Song David Wagner and Adrian Perrig, "Practical Techniques for Searches on Encrypted Data" in Proc. of IEEE Symposium on Security and Privacy'00, 2000.

[2] Eu–Jin Goh, "Secure indexes", in the Cryptology ePrint Archive, Report 2003/216, March 2004.

[3] Joonsang Baek, Reihaneh Safiavi-Naini, Willy Susilo, "Public Key Encryption with Keyword Search Revisited", in Cryptology ePrint Archive, Report 2005/191, 2005.

[4] Hyun-A Park, Jae Hyun Park and Dong Hoon Lee, "PKIS: practical keyword index search on cloud datacenter", in EURASIP Journal on Wireless Communications and Networking 2011.

[5] Li, M., S. Yu, N. Cao and W. Lou, "Authorized private keyword search over encrypted data in cloud computing." in Proceedings of the 31st International Conference on Distributed Computing Systems, June 20-24, 2011, Minneapolis, MN, USA, pp: 383-392.

[6] Ning Cao, Cong Wang, Ming Li, Kui Ren, and Wenjing Lou, "Preserving Multi-keyword Ranked Search over Encrypted Cloud Data".

[7] Steven Zittrower and Cliff C. Zou, "Encrypted Phrase Searching in the Cloud" in Globecom - Communication and Information System Security Symposium, 2012.

[8] Dongxi Liu Shenlu Wang, "Programmable Order-Preserving Secure Index for Encrypted Database Query", in IEEE Fifth International Conference on Cloud Computing, 2012.

[9] Xingming Sun, Yanling Zhu, Zhihua Xia and Liahong Chang, "Privacy- Preserving Keyword-based Semantic Search over Encrypted Cloud Data", in International Journal of Security and Its Applications Vol.8, No.3, pp.9-20, 2014.

[10] Wenhai Sun, Ahucheng Yu, Wenjing Lou and Y. Thomas Hou, "Verifiable Attribute-based Keyword Search with Fine-grained Owner-enforced Search Authorization in the Cloud", in IEEE Journal, 2013.

[11] Bing Wang, Shucheng Yu, Wenjing Lou, Y. Thomas Hou, "Privacy-Preserving Multi-Keyword Fuzzy Search over Encrypted Data in the Cloud", in IEEE INFOCOM 2014 - IEEE Conference on Computer Communications.

[12] Zhangjie Fu, Xingming Sun, Nigel Linge and Lu Zhou, "Multi-keyword Ranked Search over Encrypted Cloud Data Supporting Synonym Query", in IEEE Transactions on Consumer Electronics, Vol. 60, No. 1, February 2014.

[13] Ankit Doshi, Kajal Thakkar, Sahil Gupte and Anjali Yeole, "A Survey on Searching and Indexing on Encrypted Data", in International Journal of Engineering Research & Technology (IJERT), ISSN: 2278-0181, Vol. 2 Issue 10, October – 2013.

[14] Xingming Sun, Lu Zhou, Zhangjie Fu and Jin Wang, "Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Data in the Cloud supporting Dynamic Update", in International Journal of Security and its Application (IJSIA), Vol.8, No.6, pp.1-16, 2014.

[15] Zhihua Xia, Yanling Zhu, Xingming Sun, and Liahong Chen, "Secure Semantic expansion based search over encrypted cloud data supporting similarity ranking", in Journal of Cloud Computing, 2014.