Secure Image Retrieval over Untrusted Cloud Servers

Ayad Ibrahim Abdulsada, Aqeel N. Mohammad Ali, Zaid Ameen Abduljabbar, Haider Sh.Hashim

Abstract-Security issue represents the main barrier facing the wide adoption of cloud computing. Encryption is the best method to mitigate users' concerns. However, this method makes searching the encrypted data a challenging task. Accordingly, several approaches have been proposed to enable searching the encrypted, remotely stored data without decryption. Till now, almost all these approaches are limited to handle text search but not multimedia search.

In this paper, we propose an efficient scheme that provides content based search over encrypted image database. To do so, we utilize a locality sensitive hashing LSH method to build our searchable index. LSH index greatly enhances the system efficiency by returning the matching images in a ranked order with a minimum number of distance evaluations. For security purposes, we turn this index into a secure index to prevent the cloud server from learning any useful information from the contents of that index. Searchable index along with image collection are outsourced to the cloud server in their encrypted format. We provide several empirical experiments to illustrate the efficiency of our proposed scheme.

Key words: cloud computing, searchable encryption, LSH, image retrieval.

I. INTRODUCTION

We are living in a highly connected and data intensive world. The great advances in networking and information technologies have enabled users to collect and generate large amounts of data. However, maintaining and storing such amount of data require additional storage cost and computational power that may be not available to those users, especially in case of lightweight device (e.g. mobile and iPhone devices). Fortunately, cloud computing (utility computing) has been emerged as a new technology that offers to its user's attractive financial and technological advantages [1]. To exploit the benefits of this new paradigm, users have started moving their data and applications to cloud servers.

However moving sensitive data (such as health records, private photos, and secret documents) to the untrusted cloud servers poses a great challenge towards the privacy of user's data. To combat unsolicited access, users usually encrypt their sensitive data before outsourcing it to the cloud servers. However, traditional encryption schemes pose a significant barrier towards searching the encrypted data. Over the years, several *searchable encryption* (SE) approaches have been proposed [2]-[6] to provide the ability for selectively retrieving the encrypted documents. Typically, these systems build a secure index structure and outsource it along with the encrypted documents to the remote server.

Authorized users submit their requests as secret trapdoors that are integrated properly with the stored indexing information. The server uses the received trapdoor to search over the stored index, and retrieves the matching encrypted documents.

Traditionally, almost all the previous searchable encryption schemes are limited to handle keyword based search, where a user submits a secure keyword to search an encrypted text documents. In contrast, modern *information retrieval* (IR) [8] systems e.g. Google Goggles[1] introduce new technology that allow their clients to submit a photo as query and search a database of stored images, where images with similar visual content in the database are identified. Such new technology is termed as *content based image retrieval* (CBIR). Thus, it is highly recommended to develop a searchable encryption scheme that handle image based search in an accurate and efficient way.

In this paper, we bring together the developments of both CBIR systems and SE approaches to explore an image-based searchable encryption scheme over remote cloud servers. We build a searchable index from the image collection for speeding the search task. However, unless secured well, such index leaks important statistical information about the underlying stored data to the adversary server. Thus, the main issue here is how we can encrypt this index while preserving its ability to rank the relevant images.

The basic building block of our secure index is *the locality sensitive hashing* (LSH) [9]. LSH index allows answering efficiently near neighbor queries in high dimensional spaces of plain data [10]. In our scheme, we propose to utilize LSH in the context of the encrypted data. In such a context, it is critical to ensure the confidentiality of the sensitive data. We have conducted several empirical analyses on a real dataset to demonstrate the performance of our proposed scheme.

Our notable contributions can be summarized as follows. First, we utilize the appealing features of LSH index in the context of the encrypted data, and design an image-based searchable symmetric encryption scheme on top of this index. Second, our propose scheme indexes huge databases of images, in such a clever way, that reduce both storage requirements and run time, thus making one step closer towards practical deployment of privacy-preserving data hosting services in cloud computing.

The rest of this paper is organized as follows. Related works are reviewed and discussed in section II. Section III introduces the problem definition and the security requirements.

Manuscript published on 30 October 2013.

***** Correspondence Author (s)

Ayad Ibrahim Abdulsada, Computer Science Department, Basrah University/ Education College for Pure Science/ Basrah, Iraq.

Aqeel N. Mohammad Ali, Computer Science Department, Basrah University/ Education College for Pure Science / Basrah, Iraq,

Zaid Ameen Abduljabbar, Computer Science Department, Basrah University/ Education College for Pure Science / Basrah, Iraq,

Haider Sh.Hashim, , Computer Science Department, Basrah University/ Education College for Pure Science / Basrah, Iraq,

Section IV provides the proposed scheme. Performance investigations are provided in section V and conclusions and future works are drawn in section VI

II. RELATED WORKS

Text based searchable encryption. Prior searchable encryption systems have focused on retrieving encrypted text documents, where the data owner is allowed to outsource the storage of his data into another server and give it the ability to selectively retrieve the interested document through keyword-based search, without decryption. Some of these systems are designed to work in symmetric key setting [2, 3, 4], and the others are based on asymmetric key setting [5, 6]. However, all the above listed schemes are limited to perform an exact search but not a similarity search. Hence, such schemes could not be able to recover the typographical errors that exist frequently in the real world applications. To handle such problem, Li et al. in [7] used the wildcard technique for generating and storing a fuzzy set for each keyword, where fuzzy set incorporates the exact keyword and the slightly wrong strings that could happen due to the minor misspelling.

Image based searchable encryption. Shashank et al. [11] applied the private information retrieval PIR techniques to protect the privacy of the query image when searching over a public database, where the images in the database are not encrypted. However, since the database in PIR is always unencrypted, any scheme that tries to hide the access pattern (the IDs of the retrieved images) must touch all data items. Otherwise, the server learns information: namely, that the untouched item was not of interest to the user. Thus, PIR schemes require work which is linear in the database size. Secure image based search was also applied in the context of encrypted image databases. Lu et al. proposed in [12] to extract visual words from images and construct indexes according to them. They used some cryptographic techniques, such as order preserving encryption OPE [13] and randomized hash functions to provide both privacy protection and rank-ordered search capability. Actually, OPE primitive leaks to the untrusted server significant amount of information. Such leakage leads to conduct serious frequency attacks. In contrast, our proposed scheme does not leak anything to the untrusted server.

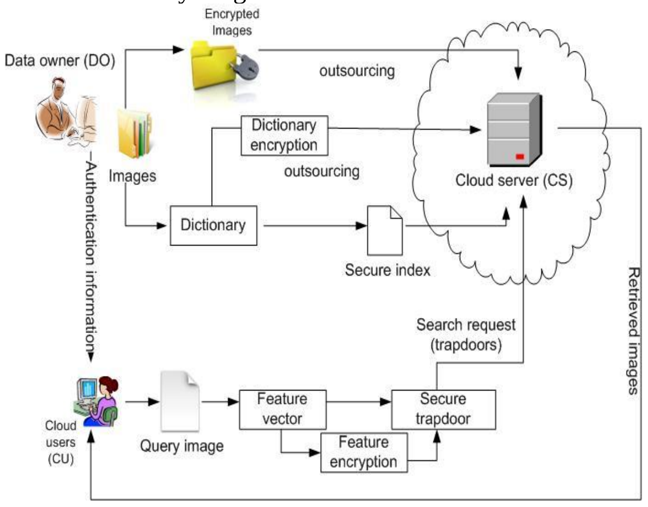

Figure 1: Basic architecture of our proposed scheme

III. SCHEME DESCRIPTION

A. *Problem Statement*

Suppose that the data owner DO has a database $D=\{Im_1, Im_2, ..., Im_n\}$ of n sensitive images that she wants to outsource to a cloud server CS. Data owner extracts a feature vector v_i from each image $Im_i \in D$ and constructs the dictionary $V=\{v_1, v_2, ..., v_n\}$ from the extracted features. Then she uses the dictionary to build the searchable index I. For security purposes, DO encrypts D collection, dictionary V, and index I before uploading them into the cloud server. Encryption prevents the cloud server from learning any useful information about the outsourced data except for the DO allows to leak. The encrypted index should enable the cloud server to search over encrypted data and return the items that are most similar to the user's request in a reasonable amount of time. To search the remotely stored image collection with an image-based query q, the authorized users generate the secret trapdoor Tq from the above query, and then send Tq to the cloud server. Once receiving Tq, the latter searches its secure index I with the presented trapdoor to retrieve the *candidate list* of images, this list identifies the set of the most similar images. Then, the server *refines* the candidate list by performing the Euclidean distance between the secure feature vector of the provided query and the dictionary subset corresponding to the candidate list items. Finally, cloud server selects the top-t image IDs, and sends back their corresponding encrypted images to the end user. Fig. 1 shows the basic structure of our proposed scheme.

B. *Security Definition*

To provide practical solutions, the security definition of all presented searchable encryption schemes allows revealing access pattern (the identifiers of the relevant documents), and search pattern (whether the query term has been searched before). We consider an honest-but-curious server in our model, which is consistent with most of the previous searchable encryption schemes. We assume the cloud server acts in an "honest" fashion and correctly follows the designated protocol specification, but is "curious" to infer and analyze the message flow received during the protocol so as to learn additional information. The formal definition of our scheme is illustrated below:

Definition 1 (Similarity searchable symmetric encryption). An index-based SE scheme is a collection of seven polynomial-time algorithms **SE** = *(Gen, Enc, Trpdr, IndBuild, Dict, Search, Dec)* such that,

1. $K \leftarrow Gen(1^\lambda)$: is a probabilistic key generation algorithm that is run by the data owner to setup the scheme. It takes λ as a security parameter, and outputs a secret key $K \in \{0,1\}^\lambda$.
2. $C \leftarrow Enc(K, D)$: is a probabilistic algorithm run by the user to encrypt the image collection. It takes as input a secret key K and an image collection $D=\{Im_1, Im_2, ..., Im_n\}$ and output the encrypting

collection $C = \{C_1, C_2, ..., C_n\}$.

3. $V \leftarrow Dict(D)$: is deterministic algorithm run by data owner to generate the dictionary *V*, feature vector set, from the given image collection D.
4. $Tq \leftarrow Trpdr(K, q)$: is a deterministic algorithm run by the user to generate a trapdoor for a given image. It takes as input a secret key K and an query image q, and outputs a trapdoor Tq.
5. $(I, SV) \leftarrow IndBuild(K, D, V)$: is a deterministic algorithm run by data owner to generate the secure index *I* and the secure dictionary *SV*. It takes as input a secret key *K*, the image collection *D*, the dictionary *V*.
6. $Relevant \leftarrow Search(I, V, Tq)$: is a deterministic algorithm run by the cloud server to search the index *I*. It takes as input an encrypted index *I*, the dictionary *V*, and the trapdoor Tq and outputs a set *Relevant* of encrypted images, such that $dist(V(Relevant), v(Tq)) \le r_1$, where r_1 is a user defined threshold, and *dist* is a metric distance function.
7. $\text{Im}_i \leftarrow Dec(K, C_i)$: is a deterministic algorithm run by the client to recover the plaintext image. It takes as input a secret key K and the encrypted image C_i, and outputs an image Im_i.

The index-based SE scheme is correct if for all K output by $Gen(1^\lambda)$, for all *(I, C)* output by *Enc(K, D)*,

$$Search(I, V, Trpdr(K, q)) = \{\text{Im}_j \ \forall \text{Im}_j \in D \mid dist(V(\text{Im}_j), v(Tq)) \le r_1 \wedge \text{Im}_j = Dec(C_j, K) \wedge C_j \in C\}$$

IV. THE PROPOSED SCHEME

In this section we explain the details of our proposed scheme. We have two phases, the setup phase and the retrieving phase.

A. *Setup Phase*

The data owner initiates the scheme by calling $Gen(1^\lambda)$, generates random keys $k_1, k_2, kcoll \xleftarrow{R} \{0,1\}^\lambda$. Furthermore, *Do* also generate randomly the matrix M_1 and its inverse M_2, both of size $(d + 2, d + 2)$, where d is the size of the feature vector. *Do* outputs $K = \{k_1, k_2, kcoll, M_1, M_2\}$.

1. Feature Vectors Generation

In order to deal efficiently with a huge amount of images, while still being able to keep a sizeable portion of the data in main memory, we need to generate an extremely compressed feature vector for each image. Such vectors represent images and potentially enabling fast and scalable search. Data oner runs the algorithm $V \leftarrow Dict(D)$ to generate the dictionary *V* for the entire image collection. To find the similar images, it is assumed that Euclidean distance between their feature vectors is a meaningful measure of similarity.

2. Features Indexing

Once generating image dictionary, Data owner runs the algorithm $(I, SV) \leftarrow IndBuild(K, D, V)$ to build a searchable index *I* on top of this entity. A simple method to index the vectors of dictionary is to store each vector directly in the index, and then use the brute force search to determine which images in the database are similar to the image query. However such naive search method is completely inefficient due to the high dimensionality and size of the data. Furthermore, such method is insecure. This is because feature vectors may leak information about image content. Thus, it is preferable to use an efficient data structure to quickly search the huge number of database features and identify candidate images, while protecting the security of the underlying data.

To meet the above listed requirements, namely: efficiency and security, we utilize the building block of *locality sensitive hashing* LSH to construct our searchable index. In what follows we explain the LSH index and then provide our methods to turn such efficient index into the context of encrypted data.

LSH Index

LSH is an efficient algorithm for near neighbor search in high dimensional spaces [9], [10]. The key idea of LSH is to “hash” items several times, in such a way that similar items are more likely to be hashed into the same bucket than dissimilar items are. To achieve this goal, LSH uses a set of hash functions to map items into several buckets, such that similar items will share a common bucket in high probability.

Given the metric space M of real *d*-dimensional points, a distance metric *dist*, a threshold r_1, two probabilities P_1 and P_2, and the approximation factor $c>1$. We define the LSH family *H* of functions $h: R^d \leftarrow N$ to be a (r_1, cr_1, P_1, P_2)-sensitive family if it satisfies the following conditions for any two points $p, q \in M$ and a function h chosen uniformly at random from *H*:

- if $dist(p, q) \le r_1$, then $\Pr[h(p) = h(q)] \ge P_1$.
- if $dist(p, q) \ge cr_1$, then $\Pr[h(p) = h(q)] \le P_2$.

Interestingly, LSH is useful for similarity search if $P_1 > P_2$. This is because we want to retrieve all images that are close to the query point with a reasonable amount of dissimilar images. The hash function $h_{a,b}$ $h_{a,b} \in H$ is defined as :

$$h_{a,b}(p) = \left\lfloor \frac{<a.p> + b}{w} \right\rfloor$$

where <.> is the dot product, p is a vector in *d*-dimensional space, a is a vector with components that are selected at random from a Gaussian distribution, b is a real number chosen uniformly from the range $[0, w]$, where w is the bin width. Thus, hash function *h(p)* projects the feature vector p onto a random direction and then returns the bin number where the projection lies. The intuition behind this method is that nearby items in the original space will fall into the same bin. Note that each a and b are chosen randomly for each hash function $h_{a,b}(.)$.

```
Algorithm 1 Bucket identifier generation.
Input: $h_1(p)$, …, $h_k(p)$: $k$ independent hash functions.
Output: $g(p)$: the bucket identifier.
Define Prime to be a set of prime numbers of length PL.
Min = minimum(PL, k);
For i=1 to Min
  If (mod(i,2)=1)
    hpos(i) = (i+1)/2;
  Else
    hpos(i) = Min-(i/2)+1;
  End if
End for
 - Compute:
    $g(p) = \sum_{i=1}^{Min} (h_{hpos(i)}(p).\Pr ime(i) + 1$
```

Furthermore, we can use the hash family H to build a new hash family G of hash functions in the following form: $g(p) = mix(h_1(p), h_2(p), ..., h_k(p))$, such that the function $g(.)$ is constructed by mixing k randomly chosen hash functions from H. The hash function $g(.)$ represents the bucket identifier of the point p in the hash table. Algorithm 1 shows our proposed method to generate the bucket identifier, g(p), from a set of k independent hash functions hj(p).

Moreover, we can choose L different hash functions $g(.)$ to index the point p into L hash tables. In this case, each vector (point) of the dictionary V is indexed into a set of L hash tables. Each table has a set of hash functions h1, h2, …, hk, which are then mixed by the function g(.) to get the index of the bucket within the table where the feature should go. All features with the same hash value go to the same bucket. We refer to the bucket identifier as BI and bucket content as BC.

The above construction amplify P1 and P2 into $P_1' = 1-(1-P_{1k})^L$ and $P'_2 = 1-(1-P_2^k)^L$, respectively. Hence, our scheme succeeds in finding a point within distance cr_1 from the query q with probability at least P'1. The values of k and L can be fixed as the follows to push P'1 closer to 1 and P'2 closer to 0:

$$k = \frac{\log n}{\log(1/P_2)},$$

and

$$L = n^{\rho},$$

where n is the number of indexed points, and. $\rho = \frac{\log P_1}{\log P_2}$

LSH Index Protection

LSH index reveals the number of buckets and the contents of each bucket to the cloud server. Such leakage may be employed from the cloud server to infer the entire image collection. Thus, we have to encrypt LSH index before moving it to the cloud server. In this case, only the data owner who generates the secret key or the authorized users who know the secret key can generate a valid query and searches the encrypted image database. Our method to protect LSH index includes the following steps:

1. (Bucket identifier protection): suppose that $\pi_{k_1}(.)$ is a collision resistant hash function with the following parameters: π : {0, 1}λ × {0, 1}∗ → {0, 1}p where p > logn. In practice, π(.) will be instantiated by off-the-self hash function like SHA-1, in which case p is 160 bits. We replace the bucket identifier BI of each hash table with $\pi_{k_1}(.)$ before outsourcing it to the cloud server. Without knowing the secret key k1, it seems imposable for cloud server to generate valid bucket identifiers.
2. (Bucket content protection): Let Max_b be the length of the maximum bucket in LSH. We encrypt the elements of each bucket under the secret key $f_{k_2}(BI)$, where BI is the bucket identifier corresponding to the bucket that want to be encrypted and $f(.)$ is a pseudo-random function in the following form: f : {0, 1}λ × {0, 1}∗ → {0, 1}$^{\ell}$. To hide the length of each bucket, we pad each bucket with Max_b –Nj random values of the same size of the encrypted data, where Nj is the current length of the bucket j.
3. (Hash table protection): Given Max_t to be the length of the longest hash table. We unify the length of all the L hash tables by padding Max_t – Lti fake records, where Lti is current length of the hash table i.

3. Dictionary Encryption

Beside the secure index, we outsource the dictionary V of the feature vectors in order to prune the candidate list. Note that the additional bytes of the stored dictionary are not a main issue due to the cheap storage cost on nowadays cloud servers. However, moving the feature vectors, in its plaintext format, to the cloud server may reveal important information. Encryption is considered the best method to preserve data privacy. However, all the traditional encryption schemes do not provide the ability to evaluate distance function over the encrypted data. To cumbersome this problem we utilized the solution of [14] to perform the distance between the query and the stored dictionary vectors without know neither the query nor the dictionary. This method allows performing the approximate Euclidean distance function among the encrypted vectors without decryption. We describe it briefly as follows:

Suppose that v and q are the data point, and the query point, respectively. Both points are represented as d-dimentional vectors. The basic idea behind this method is based on the fact that $v.q^T = (vM^{-1}).(Mv^T)$, where M is an invertible matrix. Data owner can store Mq^T instead of q^T at cloud server site, and keeps M secret from the cloud server. DO will send vM^{-1} to the server each time she wants to send a query v; therefore cloud server can compute $v.q^T$ without even knowing v and q. If we can use $v.q^T$ to represent the $\sum_{j=1}^{d}(v_j - q_j)^2$, we can make it possible for the cloud sever to conduct an approximate distance. We suggest augmenting the original representations of these points to be of d+3 dimensions as follows:

$$v = (\sum_{j=1}^{d} y_j^2 + R - R_i, v_1, v_2, ..., v_d, 1, R_i), \quad …(1)$$

$$q = (1, -2q_1, -2q_2, ..., -2q_d, Ra, 1), \quad …(2)$$

We will have

$$v.q^T = \sum_{j=1}^{d} v_j^{\,2} - 2\sum_{j=1}^{d} v_j q_j + R + Ra, \quad \ldots (3)$$

and thus the Euclidean distance will equal to $v.q^T + (\sum_{j=1}^{d} q_j^2 - R - Ra)$. However, since $(\sum_{j=1}^{d} q_j^2 - R - Ra)$ is a constant, we can delete it because it does not affect the final result; therefore the cloud server can use v.q to compute the closest match.

These points are encrypted as follows:

v'= Ev (v,M-1) = vM-1 …(4)

q' = EQ(q,M) = MqT, …(5)

Notice that we have introduced random numbers R, Ra Ri for i=1,…,n. The purpose of R is to prevent the cloud server from knowing the actual distance between q and the items in the database; the purpose of Ra is to prevent the server from knowing the relationship between two different queries; the purpose of Ri is to prevent the server from knowing the relationship among items in the database. Recall that feature vector encryption is performed just one time in the data owner side, and thus it does have any effect on the search performance.

4. Database Encryption

Data owner runs the algorithm $C \leftarrow Enc(K, D)$ to protect the privacy of his images contents. DO uses Enckcoll(.) to encrypt the image collection D before uploading it to the cloud server, where kcoll is a secret key. Given the image Imi and its identifier Id(Imi), the encrypted collection will be C = {(Id(Imi), Enckcoll(Imi)} ∀ Imi ∈ D. Image encryption can be done using state-of-the-art ciphers such as AES or RSA directly by treating images as ordinary data, or using image specific techniques such as selective and format-compliant encryption [15], [16], [17] to enable post-processing such as transcoding on encrypted images. Particularly, we use AES with counter mode as an instance of Enckcoll(.), with a key of 128-bit length. Algorithm 2 shows our method to generate the secure LSH index.

B. Retrieval phase

Once the encrypted database C, secure LSH index I, and secure dictionary SD are outsourced to the cloud server, authorized users are able to selectively retrieve images from the remote server. To do so, DO shares the following information with data users:

1. *Kcoll* : secret key of data collection encryption
2. k_1, k_2: secret keys of index construction.
3. M_1: secret key for encrypting the feature vector of the query.
4. A set of L locality sensitive hash functions *g(.)*.

```
Algorithm 2 Secure LSH index building.
Input: $\lambda$: the security parameter, D: image collection, L: the number of resulting hash tables, $g_1, g_2, \ldots, g_L$: L hash functions, MAX_b: the maximum bucket, Max_t: the length of the longest hash table.
Output: the secure index I, in the form of L hash tables and the secure dictionary SV.
{Key generation}
- Use $\lambda$ to generate $k_1$, $k_2$ and Kcoll secret keys.
{Index construction}
For each $Im_i \in D$
    - Let $Id(Im_i)$ be the identifier of the image $Im_i$.
    – Generate the feature vector $v_i$ from the image $Im_i$.
    – For each hash table j, $1 \le j \le L$
          - Use Algorithm 1 to compute the bucket identifier: $BI_j = g_j(v_i)$.
          - Store $Id(Im_i)$ in the bucket $BI_j$ of table j.
      End for
End for
{Index protection}
  - For each hash table $i \in I$, $1 \le i \le L$
      - For each bucket $BI_j$, $1 \le j \le BL_j$
          - Encrypt the $N_j$ elements of the bucket $BI_j$ with the key $f_{k2}(BI_j)$.
              - Pad the remaining ($Max\_b - N_j$) entries, if any, with fake random values of the same size of the existing $N_j$ entries.
            - replace $BI_j$ with $\pi_{k1}(BI_j)$.
      End for
      - Pad the hash table i with ($Max\_t - Lb_i$) fake records.
End for
{Dictionary encryption}
For each vector $v_i$, $1 \le i \le n$
      - Expand and encrypt the vector $v_i$ by using (1) and (4), respectively, to generate the refining vector $Sv_i$.
End for
- Set SV={Svi} $\forall i = 1, \ldots, n$
```

Given the image query *q* and the secret keys, authorized users first generate the feature vector *vq* from *q*, and then extend and encrypt the resulting vector by (2) and (4), respectively, to get the refining vector *QSV*. Finally she runs the trapdoor algorithm *Trpdr(K, q)* to hash the query *q* into *L* bucket identifiers with the same settings of the index setup. The secure trapdoor *Tq* is constructed as $Tq=\{\pi_{k1}(BI_1), f_{k2}(BI_1), \pi_{k1}(BI_2), f_{k2}(BI_2), \ldots, \pi_{k1}(BI_L), f_{k2}(BI_L), QSV, t\}$, where *t* is a user defined parameter

Table I: Symbols used in the experiments

Symbol	Meaning
n	Number of indexed images
m	Number of queries
d	Size of feature vector
k	Number of hash functions to generate the bucket identifier
t	Number of top retrieved images
L	Number of hash tables

```
Algorithm 3 Secure image retrieval.
Input: q: query image, t: the number of retrieved images,
and g1, g2, …, gL: L hash functions.
Output: The set of top-t encrypted images.

{User side}
−   Generate the feature vector v for the query image q.
−   Extend and encrypt the resulting vector by (2) and (5), respectively, to get the refining vector QSV.
−   For each hash table j, $1 \le j \le L$
      - Use Algorithm 1 to compute the bucket identifier: BIi=gj(v).
    End for
−   Set Tq={πk1(BI1), fk2(BI1), πk1(BI2), fk2(BI2),…, πk1(BIL), fk2(BIL), QSV, t}.
−   Send Tq to cloud server.

{Cloud server side}
  - Candidate= $\phi$;
  - For all πk1(BIi) $\in$ Tq, $\forall i = 1,\ldots,L$
      - Search (πk1(BIi), BCi) in hash table i.
      - Decrypt BCi with the secret key fk2(BIi)
      - Remove the random values from BCi.
      - Candidate= $\cup BC_i$
    End for
  - Remove the duplicate elements in Candidate list.
  - For each Id(Imj) $\in$ Candidate
      - Calculate the Euclidean distance as in (3) between QSV and SV(Id(Imj)).
    End for
  - Set HS vector to capture the image IDs of the minimum t distances, HS = {hs1, hs2, …, hst}
  - Set Retrieve = {Retimage(hs1), Retimage(hs2), …, Retimage(hst)} be the set of the top-t encrypted images.
  - Send Retrieve to the end user.
```

controlling the number of retrieved images. The trapdoor *Tq* is sent to the cloud server.

1. Searching and Retrieving

Cloud server accepts the trapdoor *Tq* from the authorized users and use the secure element $\pi_{k1}(BI_i)$ to scan the hash table *i* for each $1 \le i \le L$, and then retrieve the bucket content BC_i corresponding to the bucket BI_i. The cloud server uses the key $f_{k2}(BI_j)$ to decrypt the bucket *j*, and then removes the fake random values. All the image identifiers Id(Im) resulting from decryption of the buckets' contents are merged together into a single list which is called as the *candidate list*. The cloud server removes the duplicate elements in the candidate list. The next step is refining the candidate list. To do so, cloud server measures the Euclidean distance between the provided refining vector *QSV* and the refining vectors *Svi* belonging to the image identifiers that are reside in the candidate list. Such process is conducted securely by using Theorem 1. Finally, the cloud server returns the encrypted images of the *t* lower distance to the end user. Algorithm 3 shows our proposed protocol for retrieving the top-*t* images from the remote cloud server.

2. Image Decryption

Once the encrypted images corresponding to the search request are retrieved, user decrypts them with the key *Kcoll* to obtain their plain versions.

V. EXPERIMENTAL EVALUATION

In this section, we present the experimental evaluation of the proposed scheme. To perform our evaluation, we used a database of 59500 motorcycle images. These images are all of grey level format. Since this paper is focusing on security issue, we do not pay more attention on generating the feature vectors. Instead, we resize each image into 20*20 dimensions and then reshape the result into 400-dimensional vector to be the feature vector. We built a secure LSH index on the dictionary of feature vectors. Our experiments have been conducted on a 2.61GHz Pentium processor, Windows 7 operating system, with a RAM of 1GB. We use MATLAB R2008a to implement our experiments. Table I shows the set of symbols that are used in our experiments.

A. Retrieval Evaluation

To evaluate the retrieval success, we initially selected 1000 random images from the original database. Trapdoors are generated with the same settings that are used to generate the secure LSH index. Retrieval performance is evaluated using precision-recall curves, where precision and recall of the image query *q* are defined as:

Table II: LSH bucket distribution

k	10			15		
Hash table	No. buckets	Max. bucket	Avg. bucket	No. buckets	Max. bucket	Avg. bucket
1	1137	6939	2976	4435	6176	2204
2	1245	10935	3097	5273	6278	1566
3	1133	9989	4407	5780	6198	1184
4	1693	9359	2659	4705	4934	1058
5	1313	8272	4244	4350	7579	2185
6	1337	6364	2194	4357	7541	2063
7	1220	9362	3575	4720	8650	2758
8	1454	6171	1851	4250	5275	1378
9	1258	14142	5980	5001	5377	1498
10	1440	5795	2039	4623	3675	1121

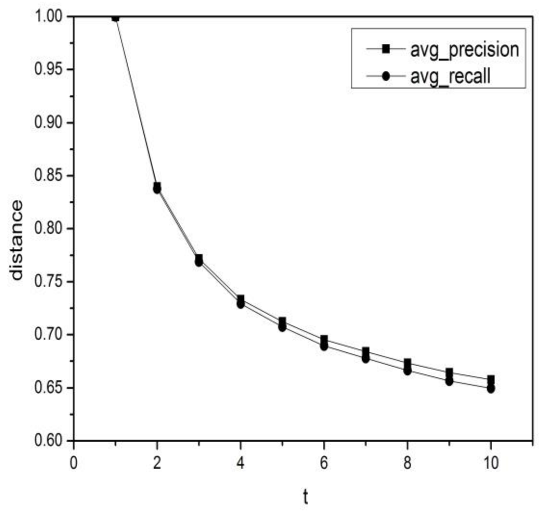

Figure 2: Image retrieval evaluation

$$\Pr ecision(q) = \frac{|R \cap A|}{|A|}, \ recall(q) = \frac{|R \cap A|}{|R|},$$

where *A* is the set of retrieved images during our proposed protocol, *R* is the set of relevant images. Given the set *Q ={q*₁*, q*₂*, …, q*ₘ*}* of *m* queries, we can compute the average of precision and recall as:

$$avg\_press = \frac{\sum_{i=1}^{m} precision(q_i)}{m}, \ avg\_recall = \frac{\sum_{i=1}^{m} recall(q_i)}{m}$$

Once a query is issued, retrieved images are ranked according to their distances. Then, images with top *t* lower distances are requested from the cloud server. Average precision and recall for issued 1000 queries with changing *t* is demonstrated in Fig. 2. As expected, less similar items are retrieved with increasing *t*.

B. LSH Index Efficiency

In this part, we test the efficiency of the LSH index to enhance the search time. We compare LSH index against the *scan index,* which use brute force search to get the nearest neighbor for each query image. Fig. 3 shows that our proposed index is much faster than the scan index. This is because our scheme measures the distance of only the candidate list, which is much smaller than the total number of the stored images.

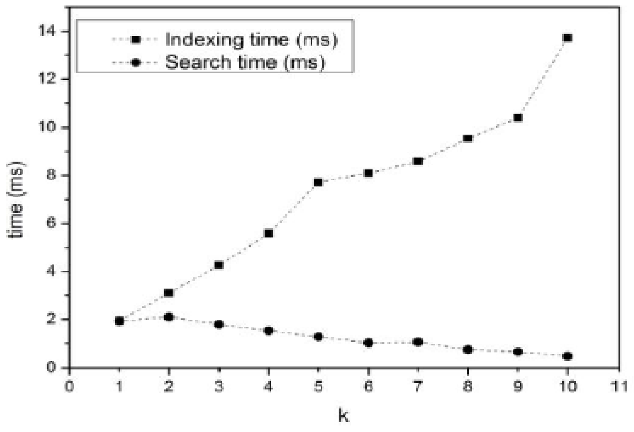

Figure 4: Effect of *k* on search and index times

C. Performance Evaluation

In this part, we evaluated the performance of our proposed index. We evaluate the effect of LSH index parameters (*k* and *L*), number of indexed images *n* on the indexing and search times. To do so, we measure the average indexing time, average search time for 1000 queries with distinct settings. Search time is simply the time between the search request and the identification of the image identifiers. To observe the effect of distinct settings, we modify a single parameter at a time and used the default values for the others. We use *k* = 10, *L* = 10, *w*=4, *n* = 59500 and *d* = 400 as default values. Figure 4 shows the effect of changing *k* value on both indexing and search times. Unsparingly, increasing the *k* value increases the indexing time. This is because more costly hashing operations are conducted in case of using large *k* values. Search time, on contrast, decrease as *k* values increase.

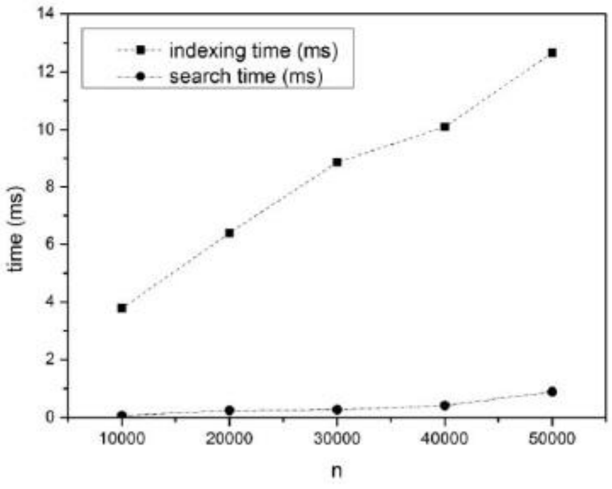

Figure 6: Effect of *n* on search and index times

The main reason for this situation is that using small *k* value yields a low number of buckets in each table and this would mean that there are, necessarily, some huge buckets, and at search time those would cancel the efficiency effect of LSH index. See experiment 4.5 for more details about the relationship between the value of *k* and the distribution of items among the buckets. The effect of *L* and *n* values is demonstrated in Fig. 5 and figure 6, respectively. As expected, the indexing time increases as the values of *L* and *n* increases. This is due to the increased number of costly hash functions evaluations. It is also easy to see that search time increases as L increased. Moreover, increasing L has an additional cost of larger trapdoor. Fig. 6 shows that large number of indexed images leads to large search time. This can be interpreted as follows: increasing the indexed images n results in more items in the candidate list that satisfy the search request.

D. LSH Distribution

In this experiment, we show the effect of *k* value on the distribution of stored items among the buckets of the individual hash tables. Table II shows that using larger *k* will increase the number of buckets in each hash table. This situation leads to distribute the stored items among a larger number of buckets. This is amazing feature to improve the search time. This is because small buckets lead to small candidate list.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)
© Copyright: All rights reserved.

VI. CONCLUSIONS

In this paper, as an initial attempt, we address and solve the problem of supporting efficient content-based image retrieval over encrypted data in cloud computing. We utilized locality sensitive hashing which is widely used for fast similarity search in high dimensional spaces for plain data. We proposed LSH based secure index and a search scheme to improve the search time in the context of encrypted data. In such a context, it is very important to preserve the privacy of the outsourcing data without sacrificing functionality. We conduct eexperimentaresults on a real data to demonstrate the efficiency of our solution.

Following the current research, we propose several possible ideas for future work on ranked image search over encrypted data. The most promising one is the support for searching color images. In this case, advanced techniques are used to extract the feature vector for each image like local feature descriptors (SIFT) [18]. Another idea is to use a bag of words approach to build visual words from the feature vectors and use inverted index [19] or min hash [20] data structures to perform faster search through the visual features.

REFERENCES

[1] Reese, G. (2009) Cloud Application Architectures: Building Applications and Infrastructure in the Cloud. O'Reilly.

[2] Song, D. X., Wagner, D., and Perrig, A. (2000) Practical techniques for searches on encrypted data. Proceedings of the 2000 IEEE Symposium on Security and Privacy, Berkeley, CA, 14-17 May, pp. 44-55. IEEE Computer Society, Washington, DC, USA.

[3] Goh, E.-J. (2003). Secure indexes. Cryptology ePrint Archive, Report 2003/216.

[4] Curtmola, R., Garay, J., Kamara, S., and Ostrovsky, R. (2006) Searchable symmetric encryption: improved definitions and efficient constructions. Proceedings of the 13th ACM conference on Computer and communications security (CCS '06), Alexandria, Virginia, USA, 30 October, pp. 79-88. ACM, New York, NY, USA.

[5] Boneh, D., Crescenzo, G. D., Ostrovsky, R., and Persiano, G. (2004) Public key encryption with keyword search. Proceedings of the 24th international conference on cryptology (CRYPTO'04), Santa Barbara, California, pp. 506-522. LNCS 3027.

[6] Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P., and Shi, H. (2008) Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. Cryptology, 21, 350-391.

[7] Li, J., Wang, Q., Wang, C., Cao, N., Ren, K., and Lou, W. (2010) Fuzzy keyword search over encrypted data in cloud computing. Proceedings of the 29th conference on Information communications (INFOCOM'10), San Diego, California, 15-19 March, pp. 441-445. IEEE Press, Piscataway, NJ, USA.

[8] Manning, C. D., Raghavan, P., and Schutze, H.(2008), Introduction to Information Retrieval. Cambridge UP.

[9] P. Indyk and R. Motwani, “Approximate nearest neighbors: towards removing the curse of dimensionality,” in 30th STOC, 1998, pp. 604–613.