CASHEIRS: Cloud Assisted Scalable Hierarchical Encrypted Based Image Retrieval System

Xin Li, Qinghan Xue, Mooi Choo Chuah
Department of Computer Science and Engineering, Lehigh University
{xil915, qix213}@lehigh.edu, chuah@cse.lehigh.edu

Abstract—**Image retrieval has become an important function in many emerging computer vision applications e.g. online shopping via images, medical health care systems. More and more images are being generated and stored in public clouds. However, recent photo leakage events raise concerns about privacy leaks for images stored in public clouds. In this paper, we present an efficient scalable hierarchical image retrieval system (CASHEIRS) which provides privacy-aware image retrieval feature. CASHEIRS employs transformed Convolutional Neural Network features to improve image retrieval accuracy and an encrypted hierarchical index tree to speed up the query process. Extensive evaluations using Caltech256 and INRIA Holiday datasets show that CASHEIRS is more effective than three existing schemes. We also demonstrate its practicality on a mobile device.**

Index Terms—**Hierarchical image retrieval, Convolutional Neural Network, image privacy**

I. Introduction

Image retrieval techniques have been used in many computer vision related applications. For example, Fischer et al. [1] employ image retrieval techniques for disease detection and diagnosis, Liu et al. [2] design a scheme to help customers search similar clothes online, etc. With the proliferation of powerful mobile devices, more images are being generated and large image datasets are often outsourced to the public cloud. To support image retrieval over large dataset, one needs to compare query image features with the features of all stored images in a database. Existing image retrieval schemes may yield low accuracy and incur large image retrieval response time if such schemes have to go through all stored images.

To combat lower image retrieval accuracy with larger image dataset, Deng et al. [3] propose a hierarchical-index-tree-based method to improve retrieval accuracy. They compute an attribute vector for each image that represents the probabilities that this image belongs to certain image categories. A similarity matrix that captures the relationship between different categories is used to compute similarity between the attribute vector of a query image and a stored image in the database. However, this approach is not scalable as the size of the attribute vector increases with increasing number of categories. In addition, it is expensive to do incremental learning as the number of categories changes since the attribute vectors of all training images need to be recomputed.

Apart from dealing with large image dataset, another issue that data owners face when outsourcing their datasets to the public cloud is that many images contain sensitive information e.g. MRI(Magnetic Resonance Imaging) images. Thus, directly outsourcing such images to pubic cloud raises privacy concern. In [4], Yuan et al. present a lightweight secure image search scheme over encrypted data. They employ K-Means clustering algorithm to build a hierarchical index tree. However, K-Means algorithm does not always yield satisfactory results. For example, K-Means algorithm may assign few images into one group, while assigning many images into another group. This will in turn results in an index tree with many levels, which yields large query response time.

In this paper, we propose an efficient and secure image retrieval system called CASHEIRS which addresses the limitations of existing privacy-aware large scale image retrieval schemes. Our CASHEIRS has four key features, namely: (i) Scalability: our CASHEIRS constructs a hierarchical index tree which groups similar clusters into a higher-level cluster and allows efficient search over subsets of categories rather than whole set during a query process. (ii) High accuracy: Convolutional Neural Network(CNN) features are used to improve image retrieval accuracy. (iii) Efficient storage and communication cost: High dimensional CNN image features are transformed into short binary codes to reduce storage and communication cost. (iv) Privacy-aware: an efficient encryption method is proposed to protect sensitive information of stored images and query privacy during a search process. We evaluate our design and demonstrate its practicality on a mobile device.

The rest of this paper is organized as follows. In Section II, we briefly discuss related work. Section III describes the system and threat models, and some important building blocks for our solution. In Section IV, we describe our proposed image retrieval system, followed by the security analysis in Section V. We report our experimental results in Section VI and describe the implementation details of our prototype system in Section VII. Finally, we conclude this paper in Section VIII.

II. Related Work

In recent years, much result has been done to efficiently and accurately recognize an object from a dataset. For example, Lowe et al. [5] propose Scale Invariant Feature Transform (SIFT) which provides strong discriminative features to describe several salient patches around keypoints in an image. In [6], Sivic and Zisserman propose a bag of words(BoW) scheme, which has been widely used in the task of object recognition [7] and image retrieval [8]. In Chechik et al. [9], the authors propose a fast online algorithm for scalable image similarity learning. However, all these papers focus on image feature representation or image retrieval without considering search privacy. In order to deal with the privacy issue, Lu et al. [10] and Hsu et al. [11] propose privacy preserving image search schemes over encrypted multimedia data set. The authors in [10] propose using order preserving encryption and Min-Hash to provide privacy-preserving feature but their scheme only works if visual words are used to represent images and researchers have shown that Fisher vector based image search algorithms provide 20% higher accuracy [4]. Similarly, the authors in [11] use encrypted SIFT features which may be larger than the compressed images and hence their method incurs large storage and communication cost.

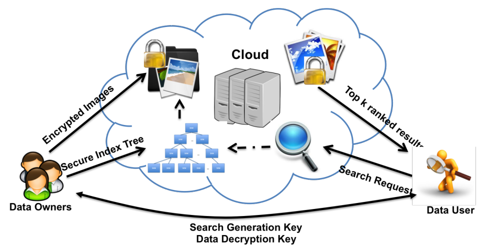


Fig. 1: System Model

III. Problem Formulation

A. *System Model*

Figure 1 illustrates a cloud-based image storage and retrieval system which allows data owners to securely outsource their images and authorized users to submit privacy-aware image-based queries. The system model consists of three entities: data owner, data user, and cloud server. A data owner first generates an encrypted searchable index tree for his images and encrypts all his images. Then, he outsources this large collection of encrypted images, together with the constructed encrypted searchable index tree to the cloud server. Next, he distributes query generation key and data decryption key to authorized users. Any authorized data user may create an encrypted query using the generation key, and then send it to the cloud server. After receiving the encrypted query, the cloud server will perform the search over the encrypted data and send back the top k ranked results to the data user. Finally, the data user uses the data decryption key to obtain the information.

B. *Threat Model*

In the system model, we assume that the cloud server is "honest-but-curious", i.e., the cloud server will honestly follow the protocol execution, but curiosity propels it to analyze the data and searchable index tree to gain more information. Depending on the available information to the cloud server, two threat models as described in [12] [13] are considered.

1. Known Ciphertext Model: The encrypted image set, searchable index tree and encrypted queries are all available to the cloud server.

2. Known Background Model: In addition to the available information assumed in the former model, the cloud server can also use statistical information to deduce specific contents in a query. It can even collude with other attackers to derive additional information from the encrypted data.

C. *Design Goals*

To address the security and threat models we have presented earlier, we design a secure hierarchical encrypted image retrieval system (CASHEIRS), which allows authorized users to conduct privacy-aware searches for similar images efficiently. CASHEIRS is designed with the following goals in mind:

1. Search Efficiency & Accuracy: The system should achieve high search efficiency and accuracy, i.e., it should return mostly correct answers with small search time.

2. Privacy Guarantee: The system should provide index confidentiality such that the cloud server cannot learn any useful information from stored data and encrypted index. Our system should also provide query privacy and unlinkability. Queries should be encrypted such that the same queries look different each time they are submitted so that the cloud server cannot deduce any sensitive information.

3. Scalability: The system should be scalable when dealing with large-scale datasets.

4. Extensibility: A desirable design should support efficient incremental update as the image dataset changes.

D. *Important Building Blocks*

Before we present the detailed description of our newly designed system, we first give a few definitions.

1. Image Feature Extraction Via CNN: Convolutional Neural Networks(CNNs) are biologically-inspired variants of multi-layer perceptrons(MLPs). Recently, with the availability of efficient GPU computing, researchers can train larger CNN-based networks, which allows CNNs to be widely used in solving several tasks such as image recognition [14]. In [15], Nagi et al. claim that CNNs are efficient at learning invariant features from images and can achieve higher accuracy by using SVM to train a classification model. In addition, Fischer et al. [16] have proved that the features from the last layer of a conventional deep network trained on ImageNet can outperform SIFT.

2. Iterative Quantization: To reduce the storage and communication cost, we use the Iterative Quantization (ITQ) scheme proposed in [17] to transform a high dimensional CNN feature into a short similarity-preserving binary code. The core idea is to find a rotation of zero-centered data so that the quantization error of mapping such data into the vertices of a zero-centered binary hypercube is minimized. Principal component analysis (PCA) is used to first reduce the dimensionality of data before using ITQ. The experimental results in [17] show that PCA-ITQ achieves a better performance than other state-of-the-art quantization methods, e.g., LSH [18], Spectral Hashing [19].

3. Secure Hamming Distance Computation

Here, we describe how the cloud server can compute the Hamming distance between two n-bit binary vectors (RV_1, RV_2) securely using their encrypted vectors:

(i) We first convert RV_1 and RV_2 to RV_1' and RV_2' by replacing all 0 values in these two vectors with -1.

(ii) Then, we select a n-bit binary vector S to split RV_1' and RV_2' into (RV_{11}', RV_{12}') and (RV_{21}', RV_{22}') as follows[13]:

- If the i^{th} bit of S is 0, then we split $RV_1'[i]$ into $RV_{11}'[i]$ and $RV_{12}'[i]$ such that $RV_{11}'[i] + RV_{12}'[i] = RV_1'[i]$; However, we set $RV_{21}'[i] = RV_{22}'[i] = RV_2'[i]$.
- If the i^{th} bit of S is 1, then we set $RV_{11}'[i] = RV_{12}'[i] = RV_1'[i]$; However, we split $RV_2'[i]$ into $RV_{21}'[i]$ and $RV_{22}'[i]$ such that $RV_{21}'[i] + RV_{22}'[i] = RV_2'[i]$.

(iii) Next, we generate two (n $\times$ n) invertible random matrices M_1 and M_2 and compute the encrypted vectors for RV_1' and RV_2': $Enc(RV_1')$ and $Enc(RV_2')$. Then, the Hamming distance between RV_1' and RV_2' is computed as the inner product of these two encrypted vectors:

$$\begin{aligned} &Enc(RV_1') \cdot Enc(RV_2') \\ &= \{M_1^T RV_{11}', M_2^T RV_{12}'\} \cdot \{M_1^{-1} RV_{21}', M_2^{-1} RV_{22}'\} \\ &= RV_{11}' \cdot RV_{21}' + RV_{12}' \cdot RV_{22}' = RV_1' \cdot RV_2' \end{aligned} \tag{1}$$

(iv) Finally, the Hamming distance of RV_1 and RV_2 is computed as $D(RV_1, RV_2) = \frac{n - RV_1' \cdot RV_2'}{2}$.

IV. Secure Image Retrieval Scheme

We first present the basic operations of CASHEIRS before we delve into its security related design details.

A. *Image feature extraction*

We use a pre-trained CNN model as an image feature extractor by removing the last output layer of a typical CNN. We extract a 4096-dimensional feature vector from each image using Caffe [20], a deep learning framework which implements CNN proposed by Krizhevsky in [21]. Since Caffe takes images of a fixed size as input, we convert each image from the datasets we use into one with 227*227 pixels since these images have higher resolution. Even if it does not, one can easily increase its resolution to 227*227 pixels. In addition, we normalize image intensities to the [0, 255] range. Next, we employ PCA-ITQ [17] to convert the extracted CNN image features into short similarity-preserving binary codes.

B. *Representative vector generation*

In CASHEIRS, we build a hierarchical index tree which groups images into different categories at various index-tree levels. To allow efficient computation of distance between two categories of images, we design a Representative Vector (RV) which captures the major features of a category of images. Each category RV has the same length as the binary codes of images and is computed using the binary codes extracted from M chosen images of a category using the PCA-ITQ method. The distance between two categories can be computed by performing an exclusive OR operation between two RVs.

Figure 2 shows how we compute a RV for a category of images. First, we randomly choose M=40 images from one category and generate the binary code vectors of these M images. Then, we compute the average value in each dimension of these binary code vectors to create a mean vector. If the average value of a particular bit position in this mean vector is larger than a pre-defined threshold, Th_{RV} (e.g. Th_{RV} = 0.8 means more than 80% image features of this category have 1s at the current position), then we assign the corresponding bit of the category RV to 1, otherwise 0. Our experimental results show that setting $Th_{RV} = 0.8$ yields the best results. Setting Th_{RV} to a lower or higher value results in a RV with either too many "1" (lower value) or "0" (higher value) bits which is not representative of the binary codes of images in that cluster.

In the next subsection, we discuss how to automatically build a hierarchical index tree for categories of images based on their representative vectors.

C. *Hierarchical index tree building*

First, we want to build a hierarchical index tree in which similar images are grouped into a cluster and similar clusters are grouped into a higher level cluster, so that we can efficiently search only subsets of clusters within the index tree given any query image. Hierarchical Clustering Algorithm proposed by Johnson in [22] can be used to build a hierarchical tree quickly. This algorithm starts by assigning each item to its own cluster. Then, it iteratively chooses and merges the closest pair of clusters into one cluster until all clusters are merged into one cluster. However, such method only generates a binary tree with many levels. In addition, we may want to group several similar image categories into one cluster and hence a binary tree is not a good choice.

To overcome this problem, we adapt the original Hierarchical Clustering Algorithm. Before describing our modified algorithm, let us first introduce our notation. We use $C_{h,i} = \{RV_{h,i}, E_{h,i}, IND_{h,i}\}$ to denote a cluster, where h is the level that the cluster belongs to, and i is its index at level h. Each cluster $C_{h,i}$ contains three attribute values: (1) a representative vector of the current cluster, denoted as $RV_{h,i}$, (2) a set of child nodes of the current cluster, denoted as $E_{h,i} = \{e_1, e_2, \cdots, e_{|E_{h,i}|}\}$, and (3) $IND_{h,i}$ is the maximum distance among pairs of clusters within $E_{h,i}$. We define the distance between two clusters $C_{x,y}$ and $C_{s,t}$ as $D(C_{x,y}, C_{s,t}) = \sum_{i=1}^{n\_bits} (RV_{x,y} \bigoplus RV_{s,t})_i$, where n_bits is the number of bits in a representative vector. $IND_{0,i}$ for every leaf node i is set to 0.

As in the original Hierarchical Clustering Algorithm, we also build our index tree iteratively. The pseudo-code for building the index tree is presented in Algorithm 1. DM in Algorithm 1 captures the dissimilarity between the distance of two RVs and the maximum pairwise distance between images of these two clusters. Th_t is a predefined threshold used to determine if we want to merge two clusters together. Experimental results show that setting $Th_t = 0.5$ yields the best results. Setting it too high prevents similar clusters from being merged and hence reduces the query accuracy rate.

Figure 3 shows an example of the index tree construction. First, we cluster all images into k=6 groups using the K-Means algorithm. Each group corresponds to a leaf node at level 0 in Figure 3(a). Thus, $C\_set = \{C_{0,1}, C_{0,2}, \ldots, C_{0,6}\}$.

Then, as shown in Figure 3(a) level 1, we compute the distances of all pairs of these 6 clusters and choose the closest pair, e.g. cluster $C_{0,1}$ and $C_{0,2}$. Because $IND_{0,1} = IND_{0,2} = 0$, so we merge cluster $C_{0,1}$ and $C_{0,2}$ into a new cluster $C_{1,1}$. Similarly, we merge cluster $C_{0,4}$ and $C_{0,5}$ into a new cluster $C_{1,2}$. Next, let us assume that the closest cluster pair is $C_{1,1}$ and $C_{0,3}$, and $D(C_{1,1}, C_{0,3}) = 9$. Thus, $DM = \frac{|D(C_{1,1},C_{0,3}) - max(IND_{1,1}, IND_{0,3})|}{max(IND_{1,1}, IND_{0,3})} = \frac{9-2}{2} = 3.5 > 0.5(Th_t)$, so we merge cluster $C_{1,1}$ and $C_{0,3}$ into a new cluster $C_{2,1}$ (Figure 3(b) level 2). After that, let us assume that the closest two clusters are cluster $C_{1,2}$ and $C_{0,6}$ and their distance is 12. Thus, $DM = \frac{|D(C_{1,2},C_{0,6}) - max(IND_{1,2}, IND_{0,6})|}{max(IND_{1,2}, IND_{0,6})} = \frac{12-10}{10} = 0.2 < Th_t$, so we merge cluster $C_{0,6}$ into cluster $C_{1,2}$ (Figure 3(b) level 1). We repeat this process to merge clusters until all clusters are merged into one cluster (Figure 3(b) level 3).

D. *Search Process Without Encryption*

To find similar images using the hierarchical index tree, the cloud server, upon receiving a search request, needs to determine which nodes to visit next as it traverses from the

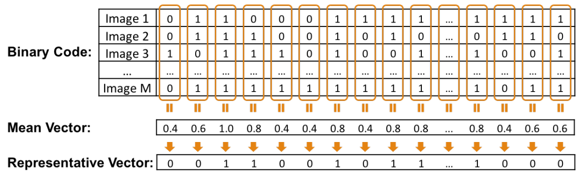
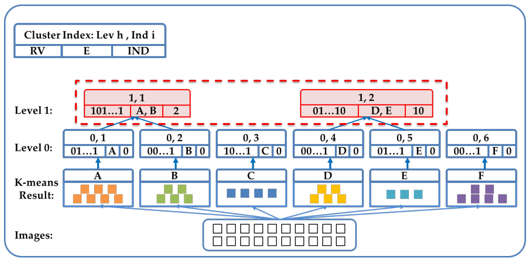
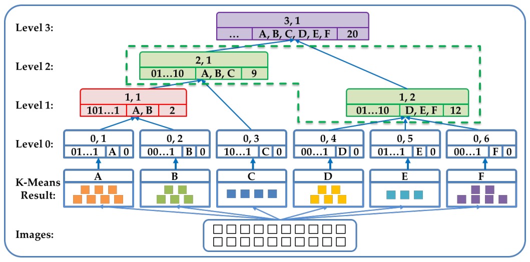


Image 1	0	1	1	0	0	0	1	1	1	1	...	1	1	1	1
Image 2	0	1	1	1	0	0	1	0	1	0	...	0	1	1	0
Image 3	1	0	1	1	1	0	1	0	0	1	...	1	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Image M	0	1	1	1	1	1	1	1	1	1	...	1	0	1	1
Mean Vector:	0.4	0.6	1.0	0.8	0.4	0.4	0.8	0.4	0.8	0.8	...	0.8	0.4	0.6	0.6
Representative Vector:	0	0	1	1	0	0	1	0	1	1	...	1	0	0	0

Fig. 2: Category Representative Vector Generation



(a)



(b)

Fig. 3: Hiererchical Index Tree Building

Algorithm 1: Building Index Tree

```
Input : A set of images: $IMG\_set = \{i_1, i_2, \cdots, i_N\}$, number of
        centers for K-Means: $k$, predesigned threshold: $Th_t$
Output: Index Tree $TR$. Each node $j$ within $TR$ has a node identifier
        $NI(j) = (h, i)$ where $h$ represents its height, and $i$ is its
        index at height $h$ within $TR$.
1  begin
2      $TR \leftarrow \emptyset$
3      $CNN\_set \leftarrow Extract\_CNN\_feature(IMG\_set)$
4      $BinFeatures \leftarrow PCA\_ITQ(CNN\_set)$
5      $C\_set \leftarrow KMeans(BinFeatures, k)$ /*$C\_set = \{c_{0,1} \cdots c_{0,k}\}$*/
6      $Insert(TR, C\_set)$ /* insert leaf nodes into $TR$*/
7      while $len(C\_set) > 1$ do
8          /*$find\_nearest(C\_set)$ returns 2 clusters ($C_u$, $C_v$) that have the smallest cluster
           distance, where $C_u = C_{h1,i1}$, $C_v = C_{h2,i2}$ and either ($h1 > h2$) or
           ($h1 == h2$, $i1 > i2$)*/
9          $C_u, C_v \leftarrow find\_nearest(C\_set)$
10         if $C_u.IND == 0$ and $C_v.IND == 0$ then
11             $C\_set \leftarrow merge(TR, C\_set, C_u, C_v, True)$
12         else
13             $max\_value \leftarrow max(C_u.IND, C_v.IND)$
14             $Dis \leftarrow D(C_u, C_v)$
15             $DM \leftarrow (Dis - max\_value)/max\_value$
16             if $DM > Th_t$ then
17                 $C\_set \leftarrow merge(TR, C\_set, C_u, C_v, True)$
18             else
19                 $C\_set \leftarrow merge(TR, C\_set, C_u, C_v, False)$
20     return $TR$
21 function merge($TR, C\_set, C\_x, C\_y, flag$)
22 begin
23     if $flag$ then
24         $C_{new_c} = (RV_{new_c}, E_{new_c}, IND_{new_c}) =$
           $CreateNode(C_x, C_y)$ /* this function computes node identifier
           ($h_{new}, i_{new}$), and $RV$ of $new_c$, set $E_{new_c} = x, y$*/
25         $Insert(TR, C_{new_c})$ /* insert $new_c$ into $TR$*/
26         $C\_set.append(C_{new_c})$
27         remove $C\_x$, $C\_y$ from $C\_set$
28     else
29         $C\_x.append(C\_y)$ /*updates $C\_x = [RV_x, E_x, IND_x]$ after including
           $C_y$ as an additional child nodes */
30         remove $C\_y$ from $C\_set$
31     return $C\_set$
```

top to the leaf level of the tree. To help make this decision, we define a weight metric for each node i (denoted as NoI)(assume that there are N_h nodes in level h of the index tree):

$$NoI(D(RV_{h,i}), Q) = 1 - \frac{D(RV_{h,i}, Q)}{\sum_{j=1}^{N_h} D(RV_{h,j}, Q))} \tag{2}$$

The cloud server orders the weight metrics of all children nodes (from the largest to the smallest) and selects a sufficient number of nodes such that the sum of their weight metrics exceeds Th_s. Most of the time, only one node with the largest weight metric (DMax(h)) will be selected but sometimes two or more nodes are selected. All the selected nodes are considered matched nodes at level h. Experimental evaluations show that the mean average precision (mAP) becomes stable after $Th_s = 0.8$ but the search latency increases with increasing Th_s since higher Th_s yields more matching categories. Thus, we set $Th_s = 0.8$ since it achieves the best mAP and decent search latency (6ms for 100 categories). After finding the matched nodes, the cloud server will iteratively go through those nodes until it finds the matched leaf node. The cloud server then ranks the stored images in this matched leaf node based on their Hamming distances and returns the top k images (smallest distance first) to the data user.

E. Security Design in CASHEIRS

To keep his information private, a data owner encrypts all his images and the image features stored in the hierarchical index tree before he outsources this information to the cloud. Furthermore, CASHEIRS provides query unlinkability to prevent the cloud server from tracking queries from a data user. The security design for the query in CASHEIRs (similar to the one in [4]) mainly splits an image vector into two random vectors and encrypts them such that a query with the same image looks different every time it is submitted. The representative vectors stored in the hierarchical index tree are also stored as split encrypted representative vectors such that the Hamming distance between the encrypted image vector of a query and any stored encrypted representative vector can be securely computed. Next, we present a detailed description of the security design we have in CASHEIRS.

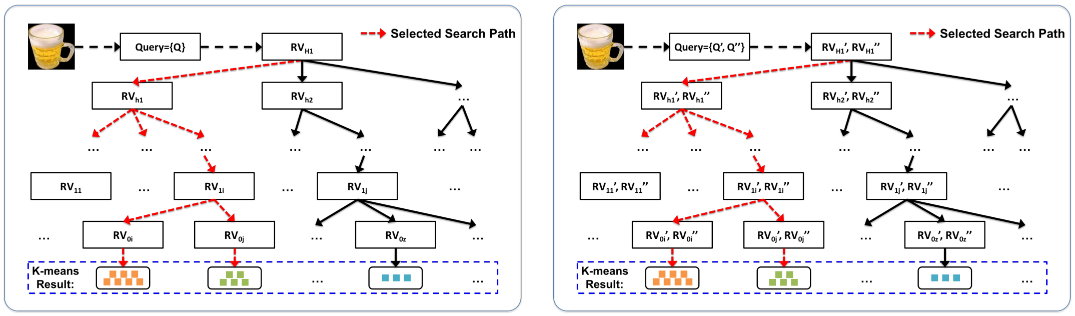
1. KeyGen: In the initialization phase, a secret key SK is produced by the data owner which consists of two components: (i) a n-bit randomly generated splitting vector S; and (ii) two (n $\times$ n) vectors $\{M_1, M_2\}$, Thus, SK can be denoted as a 3-tuple $\{S, M_1, M_2\}$. The data owner later sends the secret key, SK, to the authorized users.

2. GenIndex(I,SK): To speed up the image search process, the data owner will build an index tree for all images.

(i) For each image f stored in the dataset, the data owner first generates a n-dimensional image vector V_f based on the pre-trained CNN model and PCA-ITQ.

(ii) Then, the data owner maps all images into N selected clusters and computes the n-dimensional representative vector for each cluster based on the image vectors.

(iii) Next, the data owner builds the hierarchical tree using the procedures describe earlier. The owner first sets each cluster as one leaf node of the index tree and then associates both the n-dimensional representative vector $RV_{h,i}$ and image IDs to every leaf node, where h is the node's height and i is its index at height-h.



(a) Query Process Without Encryption (b) Query Process With Encryption

Fig. 4: Query Process

(iv) In addition, the owner also assigns a n-dimensional representation vector to each non-leaf node.

(v) In order to achieve privacy, the data owner later transforms each representative vector $RV_{h,i}$ into an encrypted value using the following substeps: first, the data owner simply converts $RV_{h,i}$ to $\widetilde{RV}_{h,i}$, such that all 0s in $RV_{h,i}$ are converted into -1s in $\widetilde{RV}_{h,i}$. In addition, the data owner splits $\widetilde{RV}_{h,i}$ into two random vectors as $\{\widetilde{RV}_{h,i}{}'$,$\widetilde{RV}_{h,i}{}''\}$ based on the splitting vector S, such that if the j^{th} bit of S is 1, $\widetilde{RV}_{h,i}{}'[j]$ and $\widetilde{RV}_{h,i}{}''[j]$ are set as the same as $\widetilde{RV}_{h,i}[j]$; but if the j^{th} bit of S is 0, $\widetilde{RV}_{h,i}{}'[j]$ and $\widetilde{RV}_{h,i}{}''[j]$ are set to two random numbers so that their sum is equal to $\widetilde{RV}_{h,i}[j]$. Thus, the original $RV_{h,i}$ is stored as two encrypted values.

(vi) Finally, the data owner constructs the encrypted index tree $Enc_{SK}(I)$ with each encrypted representative vector stored as $\{M_1^T\widetilde{RV}_{h,i}{}'$,$M_2^T\widetilde{RV}_{h,i}{}''\}$.

3. GenQuery(Q,SK): To provide query unlinkability, we need to generate a different search request even with the same query content.

(i) To perform an encrypted image search, a data user first extracts the CNN feature from a query image, and converts it into a binary code using PCA-ITQ as Q =$\{q_1, q_2, \cdots\}$. Then, he generates $\widetilde{Q}$ by converting all the 0s in Q to -1s.

(ii) Next, the data user splits $\widetilde{Q}$ into two random vectors as $\{\widetilde{Q}', \widetilde{Q}''\}$ using the splitting vector S and a similar splitting procedure described in $GenIndex$. The difference is that if the i^{th} bit of S is 1, $\widetilde{Q}'[i]$ and $\widetilde{Q}''[i]$ are set to two random numbers so that their sum is equal to $\widetilde{Q}[i]$; if the i^{th} bit of S is 0, $\widetilde{Q}'[i]$ and $\widetilde{Q}''[i]$ are set as the same as $\widetilde{Q}[i]$.

(iii) In addition, the data user computes M_1^{-1} and M_2^{-1}. Then, he generates the encrypted search request $\{M_1^{-1}\widetilde{Q}'$, $M_2^{-1}\widetilde{Q}''\}$ and submits it to the cloud server.

4. Search($Enc_{SK}(I)$,$Enc_{SK}(Q)$): The search process with encryption is shown in Figure 4(b). It can be compared to the one without encryption (shown in Figure4(a)).

(i) After receiving a search request, the cloud server searches through the index tree from the top to the leaf level.

(ii) At each level h, the cloud server finds the nodes of the next level to traverse by comparing the Hamming distance of the stored encrypted vector in node i at level h with that of the query, $D(Enc_{SK}(I_{h,i}), Enc_{SK}(Q))$.

(iii) Then, the cloud server computes a weight metric for each node i (denoted as NoI)(assume there are N_h nodes in level h of the index tree):

$$NoI(D(Enc_{SK}(I_{h,i}), Enc_{SK}(Q))) \\ = 1 - \frac{D(Enc_{SK}(I_{h,i}),Enc_{SK}(Q))}{\sum_{j=1}^{N_h} D(Enc_{SK}(I_{h,j}),Enc_{SK}(Q)))} \quad (3)$$

Next, it selects enough nodes such that the sum of their weight metrics exceeds a predefined threshold, Th_s. All these selected nodes are then marked as the matched nodes at level h.

(iv) After finding the matched nodes, the cloud server will iteratively go through those nodes until it finds the matched leaf node. Finally, the cloud server returns the ranked images stored at the matched leaf node to the data user.

F. Index Tree Update

In CASHEIRS, we consider two updating operation: adding and deleting an image. When the data owner needs to add an image, he first performs a Searching operation to find the corresponding leaf node and inserts the feature of the new image into this leaf node. Then, Category Representative Vectors (RVs) of all non-leaf nodes on the path from this leaf node to the root node will be updated. Similarly, to remove an image, the data owner first performs a Searching operation to find the corresponding leaf node and delete its feature followed by updating all related non-leaf nodes.

V. Security Analysis

In this section, we will show how our security design satisfies several search privacy requirements:

1. Index and Query confidentiality in both the known ciphertext model and the known background model. More details are provided in subsequent subsections.

2. Query unlinkability: the adopted vector encryption method provides non-deterministic encryption via the randomly vector splitting procedure. Thus, our schemes can create different search requests even with the same query and hence provide query unlinkability to a certain extent.

A. Security Analysis Under the Known Ciphertext Model

Here, we adapt the simulation-based security model in Sun et al. [13] and Wang et al.[23] to prove that our basic scheme can be secure under the known ciphertext attack. Before proving, we will introduce some notations that will be used in the proving process.

• History: it is an image collection F, an index set I and a set of queries Q = $\{Q_1$,Q_2,...$\}$ submitted by users, denoted as H=(F, I, Q).

• View: the cloud server can only see the encrypted form of a H, denoted as $VI(H)$, including the secure indexes

$Enc_{SK}(I)$, the encrypted images $Enc_{SK}(F)$ and the encrypted search request for each query Q_i, $Enc_{SK}(Q_i)$ = $\{Enc_{SK}(Q_1), Enc_{SK}(Q_2), \cdots\}$.

• Trace: A history trace is a trace of a set of queries, denoted as $Tr(H) = \{Tr(Q_1), Tr(Q_2), ...\}$. It captures the information to be learned by the cloud server for each query Q_i including the length of the query vectors, the search pattern PA_{Q_i} and the outcome of the search, e.g., $Tr(Q_i) = \delta_i$ where δ_i is the set of returned image identifiers matching the query Q_i.

As in Sun et al. [13] and Wang et al.[23], our proof is based on the following argument: given two histories that produce the same trace, if the cloud server cannot distinguish which history is produced by the simulator, then, the cloud server cannot learn additional knowledge beyond the information that the system is willing to leak.

We adopt a simulator which can simulate a view, VI' indistinguishable from cloud server's view, VI. It works as follows: The simulator randomly picks two vectors U_1, U_2, one split vector $S' \in \{0,1\}^n$ and sets SK'=$\{U_1, U_2, S'\}$.

1. For an encrypted query $Enc_{SK}(Q_i)$, the simulator generates $Enc_{SK'}(Q'_i)$ as follows:

(i) The simulator selects a random string $s \in \{0,1\}^n$. Ensure that the number of 1s in s is the same as the number of 1s in Q_i but their positions are different and sets Q'_i=s.

(ii) Generate the encrypted search request for each Q'_i and sets $Enc(Q')$=$\{Enc_{SK'}(Q'_1), Enc_{SK'}(Q'_2), \cdots\}$

2. Based on the search pattern PA_{Q_i}, the simulator can generate $Enc(I')$ as follows:

(i) Assume PA_{Q_i} goes through node j at level h of the index tree. Let $V'_{h,j}$ be the stored vector at the corresponding node in the simulated index tree. $V'_{h,j}$ is initially a null vector.

(ii) The simulator sets $V'_{h,j} = V'_{h,j} + Q'_i$.

(iii) After all queries are processed, the simulator converts the $V'_{h,j}$ into a vector $\in \{0,1\}^n$ by replacing the elements bigger than 1 with 1 and generates $Enc_{SK'}(V'_{h,j})$.

3. The simulator outputs the view $VI(H')$=($Enc_{SK'}(F')$, $Enc_{SK'}(I')$, $Enc_{SK'}(Q')$).

In summary, the $Enc_{SK'}(I')$ and $Enc_{SK'}(Q')$ are generated such that the lengths of individual components are the same as those in $Enc_{SK}(I)$ and $Enc_{SK}(Q)$. They also generate the same trace as the one that the cloud server has. Thus, we claim that no probabilistic polynomial-time (P.P.T) adversary can distinguish between the view $VI(H')$ and $VI(H)$.

B. *Security Analysis of Our Scheme Under the Known Background Model*

Under the known background model, we assume the cloud server can analyze past queries and likely link frequently used search contents in encrypted queries with their matched results. Thus, in this subsection, we analyze the security of our basic and enhanced schemes under the known background attack model. For each query Q_i we generate the encrypted search request as $Enc_{SK}(Q_i)$. Since the adopted vector encryption method provides non-deterministic encryption, in light of the random vector splitting procedure. Thus, the same search request (e.g., same search contents) will be encrypted to different query vectors. Therefore, our schemes can achieve query unlinkability such that it is hard for the cloud server to link one transformed query request to another even if both contain the same search content.

VI. System Evaluation

To evaluate the performance of our scheme, we implement CASHEIRS using Python programming language and evaluate it using a laptop running OS X with 2.5GHz Intel Core i7 CPU and 16GB Memory. We use Precision at top k(P@k) and Mean Average Precision(mAP) as the performance metrics.

- **P@k:** Precision at top k (P@k) is defined as follows:
$$P@k = \frac{num\_correct}{k} \tag{4}$$
where $num\_correct$ is the number of relevant images in the top-k positions in the query result list.
- **mAP:** Mean average precision (mAP) is the overall average of the average precision (AP) computed for each query image. The average precision can be computed as follows:
$$AP(q) = \frac{\sum_{k=1}^{n}(P@k * rel(k))}{N} \tag{5}$$
where k is the position in the ranked result; n is the number of returned images for a query image q; $rel(k)$ is 1 if the item at rank k in the result list is a ground truth image, 0 otherwise; N is the number of ground truth images.

We conduct the experimental evaluation using two well-known datasets: Caltech256 [24] and INRIA Holidays dataset [25]. The Caltech256 dataset has 30,608 images which can be classified into 256 distinct object categories and an "other" category. Each category contains at least 80 images. The INRIA Holiday dataset contains 1491 images, which can be classified into 500 image groups, each representing a distinct scene or object. In addition, for CNN feature extraction, we use Caffe [20] implementation and evaluate the performance of CASHEIRS by varying PCA-d, $d = \{2048, 1204, 512, 128, 64\}$, used in the PCA-ITQ step.

A. *Construction Time & Memory Cost For Index Tree*

To set up our system, five operations are needed: (i) loading a pre-trained CNN model, (ii) extracting CNN features from training images, (iii) encoding CNN features to similarity-preserving binary codes, (iv) encrypting features of training images, and (v) building a Hierarchical Index Tree. In our scheme, we randomly choose 50 images from each category of Caltech256 [24] and store them in the cloud after encryption. Then, as in [17], we only use 40 images per category for training. We use 50 iterations to learn a Rotation Matrix which is used to convert a 4096 dimension CNN feature into a 128 dimensional binary code in the PCA-ITQ approach. Next, we apply our adapted Hierarchical Clustering method to group images into different clusters (leaf nodes) by using K-Means clustering algorithm, and merge clusters (with $Th_t = 0.5$) at different levels of a hierarchical index tree. We also compute the Representative Vectors that need to be stored at intermediate nodes of the hierarchical index tree. Finally, all the image binary codes and RVs are encrypted. The memory cost and computation time incurred by a data owner for this

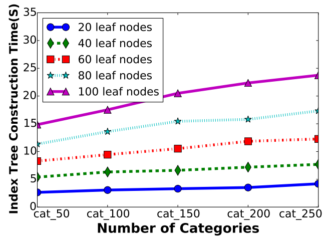


Fig. 5: Index Tree Construction Cost

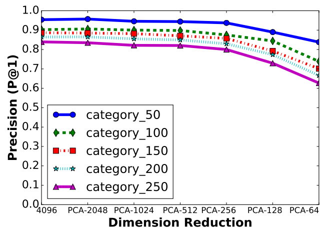


Fig. 6: Precision of Candidate Categories

index tree construction process are shown in Table I assuming PCA-128 is used, where all data are averaged over 5 runs. In general, a fixed memory cost of 306.25MB is incurred for storing a pre-trained CNN model, PCA matrix, Rotation Matrix, Encryption keys, and the system and program code. As for the memory cost of constructing hierarchical tree, a linear cost of 4MB+0.0166*total images is incurred which is about 45.5MB for Cat-50, and hence total memory cost for Cat-50 is about 351.8MB.

Figure 5 shows the computation time of building a hierarchical index tree using different numbers of categories and leaf nodes. All categories are from the Caltech256 dataset [24]. For each category, we randomly choose 50 images to build the index tree. From the results, we see that the construction time increases almost linearly with the number of categories but it is $O(n^2)$ with the number of leaf nodes, n.

B. Storage Cost

CASHEIRS needs to store six files: (i) the pre-trained CNN model which is used to extract CNN feature from images; (ii) PCA and rotation matrices which are used to encode a CNN feature into a binary code; (iii) the encrypted hierarchical index tree; (iv) encrypted image features; (v) encrypted images in databases; and (v) encryption keys. Here, we compare the storage cost of CASHEIRS with the following 3 schemes:

1. Hierarchical Semantic Indexing for Large Scale Image Retrieval Scheme (B-Hie) [3]: B-Hie trains a 1-VS-all SVM classifier for each category and fits a sigmoid function to convert the output of a classifier into a probability value. Then, B-Hie feeds each training image to all trained classifiers to get the predicted probability values and regards the probability values as an attribute vector of that image. B-Hie uses locality sensitive hashing (LSH) to assign all images into different groups based on their attribute vectors. In the query process, B-Hie first generates an attribute vector for a query image using all classifiers. Then, B-Hie uses this attribute vector to find a matching group. All images in this group are then regarded as candidate images. Next, B-Hie computes a similarity score between the query image and each candidate image, ranks the candidate images based on their similarity scores, and return the related results.

TABLE I: System Setup Cost

CASHEIRS Set Up	**Cat_50**	**Cat_100**	**Cat_150**	**Cat_200**	**Cat_250**
Memory Cost (MB)	351.80	393.09	432.08	487.73	542.96
Computation Time (s)	163.93	323.42	484.51	646.60	817.46
Cloud Storage Cost (MB)	**Cat_50**	**Cat_100**	**Cat_150**	**Cat_200**	**Cat_250**
CASHEIRS	40.6	81.0	121.5	161.4	201.7
B-Hie	59.8	120.4	181.0	241.7	302.4
OASIS	109.3	197.4	285.5	373.6	461.7
SEISA (PCA-128)	40.3	80.8	122.3	167.1	211.8
SEISA (PCA-512)	55.7	113.9	175.9	239.6	303.3

2. Online Algorithm for Scalable Image Similarity Learning (OASIS) [9]: OASIS first extracts a feature vector p_i from each image, i. Then, OASIS learns a weight metric W which is used in a relevance metric $Sw(p_i, p_j) = p_i^T W p_j$ where Sw produces a large value for a pair of similar images but a small value for a pair of dissimilar images. Thus, this relevance metric can be used to check if two images are similar or share the same class label.

3. Secure and Efficient Encrypted Image Search With Access Control Scheme (SEISA) [4]: SEISA extracts Fisher vectors from all images and employs K-Means algorithm to group them into different clusters. If any cluster contains more than T (a predefined threshold) images, then all images in that cluster will be reclustered again. This procedure is repeated until no cluster contains more than T images. In the query process, SEISA first uses a Fisher vector extracted from the query image to search through the index tree. At each level, SEISA chooses the node with the shortest distance between its mean vector and the feature vector of a query image, and searches through the child nodes of this selected node. After finding the matched leaf node, SEISA returns the sorted list of all images in this node and their associated similarity scores.

We report the storage cost for all 4 schemes using the Caltech256 dataset in Tables I. In general, both the data owner and the data user need to store 248.1MB of CNN related information and encryption keys: a pre-trained CNN model (244.7MB), PCA matrix (3.3MB), Rotation Matrix (105KB) and Encryption Keys (1.7KB). In addition, the data owner also needs to store encrypted images (about 14.28KB/image for the Caltech 256 dataset) and information related to the index tree. For 50 categories, we need a total of 40.6MB: 440KB for the encrypted hierarchical index tree, encrypted image features of 5.3 MB, and 2500 (50 images/categories) encrypted images. The cloud only needs to store encrypted images, encrypted image features and the index tree information.

C. Search Evaluation

In this sub-section, we compare the search performance of CASHEIRS with the three schemes described earlier.

1) Query Latency & Search Time Performance: For each query, the latency mainly consists of three components: (i) encrypted query generation time, (ii) the search time for the cloud to traverse through the index tree, (iii) internet round trip latency between a client and the server. In this paper, we only report the first two latency components. An additional 300-400ms can be added for the 3rd component assuming the client and the server are in east/west coast respectively and no internet congestion occurs during the query process.

On the average, it takes a total of 52ms to generate an encrypted query: 48.8ms in extracting CNN feature (4096 dimension) from a query image, 2.9ms in binary encoding (using PCA-128), and 0.3ms in encrypting. After receiving an encrypted query (PCA-128), the cloud takes on the average 25ms (compared to 6ms without encryption) to search for candidate images through the index tree built for 100 categories. Table II shows the search time comparison of the 3 schemes we considered. For a fairer comparison, we assume that the 10,000 dimension image feature vector used in OASIS is compressed via LSH into a 8192 bit vector for more efficient similarity score computation.

*2) **Search Accuracy**:* We conducted 3 experiments (using $Th_s = 0.8$) to compare CASHEIRS with three schemes.

(A) Exp 1: This experiment is designed to evaluate the effectiveness of using RV in our index tree to identify the correct category of a query image. Here, we use the Caltech256 dataset [24] which has category labels. We randomly choose $50, 100, 150, 200, 250$ categories from the Caltech256 dataset. For each category, we split its images into two sets (50 images for building the index tree and 25 images for query). Here, we simply use the category labels of the images for index tree construction instead of employing K-Means clustering. In the query process, after receiving a query, CASHEIRS goes through the hierarchical index tree to identify a category for the query image. Then, we compare the identified category with the ground truth to compute the P@1 (precision at 1). Figure 6 shows that RV is quite effective in representing image features of images belonging to each intermediate node since the achieved precision is high. In addition, one can observe that more categories lead to lower accuracy because more categories result in a higher probability that two categories have similar Category Representative Vector (RV), e.g. beer-mug and coffee-mug. Overall, CASHEIRS (without the K-means clustering step) achieves better than 70% precision, even if the dimension of the feature vector is reduced to PCA-128. The precision reduces sharply when the dimension is reduced to PCA-64. Thus, we choose PCA-128 as a default setting in this paper, as this setting improves the storage and communication cost while maintaining high query accuracy.

(B) Exp 2: In this experiment we compare CASHEIRS with two existing methods: B-Hie [3] and OASIS [9]. We first choose 10, 20, and 50 classes (the same classes as Deng et al. [3] and Chechik et al. [9]) from the Caltech256 dataset [24] (For more details about the chosen classes, please refer to [9]). Then, we randomly split all images into two sets (25 images as queries and 50 images as the correct retrieval results per class). Next, for each category, we randomly choose 40 images (the same number of training images as B-Hie [3] and OASIS [9]) as the input to ITQ to learn the rotation matrix. During the process of index tree construction, we first assume all images are without labels and employ K-Means algorithm to cluster images into 100 groups before building the index tree. We report the average precision at top k of 5 runs for 10, 20 and 50 classes from the Caltech 256 dataset in Figure 7 as we vary k. From the results, we can see that CASHEIRS is more accurate than the B-Hie [3] and OASIS schemes [9]. We also observe that comparing the achievable result for 50 categories in Fig. 7 with that in Fig. 6, using K-means clustering results in lower accuracy than using category labels but K-means clustering is useful for many image datasets do not have category labels.

TABLE II: Comparison Results

Caltech256			
Schemes	**Search Time (ms)**		
	Category-10	Category-20	Category-50
CASHEIRS	4.1	9.6	12.9
B-Hie	13.3	23.2	52.2
OASIS	131.9	132.6	133.9

INRIA Holiday (10 million images)		
Scheme	Search Time (ms)	mAP
CASHEIRS	95.2	0.64
SEISA	87.5	0.55

(C) Exp 3: Finally, we also use the INRIA Holidays Dataset to compare CASHEIRS with SEISA [4] where Yuan et al. propose a hierarchical index tree for encrypted image retrieval. Both SEISA [4] and CASHEIRS use hierarchical index tree, but their methods for building index tree and generating queries are totally different. SEISA [4] builds the index tree from top down, while CASHEIRS (ours) builds an index tree from bottom up. In addition, during query processing, SEISA [4] only chooses one node from the next level, while CASHEIRS may choose one or multiple nodes at each level.

We follow the same experimental set-up as Yuan et al. [4]. Specifically, we generate 10 Million image features (PCA-128) randomly, and then mix them with the features extracted from INRIA Holiday dataset [25]. Before building the index tree, we employ K-Means algorithm to cluster all images into 100 groups. In the query process, the first image of each image group is used as a query image while the remaining images within the same group are considered the correct retrieval results. We report the comparison results in Table II.

Communication Cost: Communication cost in CASHEIRS consists of four parts: (i) the encrypted data which the data owner sends to the cloud; (ii) the information that the data owner sends to users; (iii) an encrypted search query sent from a user to the cloud; and (iv) the query results returned to a querying user by the cloud server.

The data owner needs to send encrypted images, encrypted image features and encrypted index tree to the cloud, with the same size as the storage in the cloud(11.2MB + 14.28KB/encrypted image). A user receives CNN related information(248.1MB) from the data owner. For each query, a user sends 2KB encrypted query information to the cloud and receives 14.28KB/encrypted image from the cloud.

VII. Prototype Evaluation

Our initial prototype system consists of a cloud server (a laptop described in Section VI) and a Samsung S5 phone as a data user's mobile device. Samsung S5 has a Snapdragon 801 chip with Quad-core CPU@2.5GHz and 2GB RAM. The smartphone communicates with the server via a WiFi router. In the Android client software, Caffe APIs are called to extract CNN feature from a query image. Then, this CNN feature is converted into 128bit binary code and later encrypted. To handle complex image transformations and minimize memory usage, Picasso, an image downloading library for Android, is used to download query results.

Here, we focus our evaluation on the client device. We measure the computation time, memory cost, and the energy cost of performing a search request. In our prototype, the Android client takes on the average 5725ms to load the pre-trained CNN model and 602ms to generate the encrypted image feature. In contrast, it takes 144ms and 52ms correspondingly on the 2.5GHz Intel Core i7 CPU laptop with

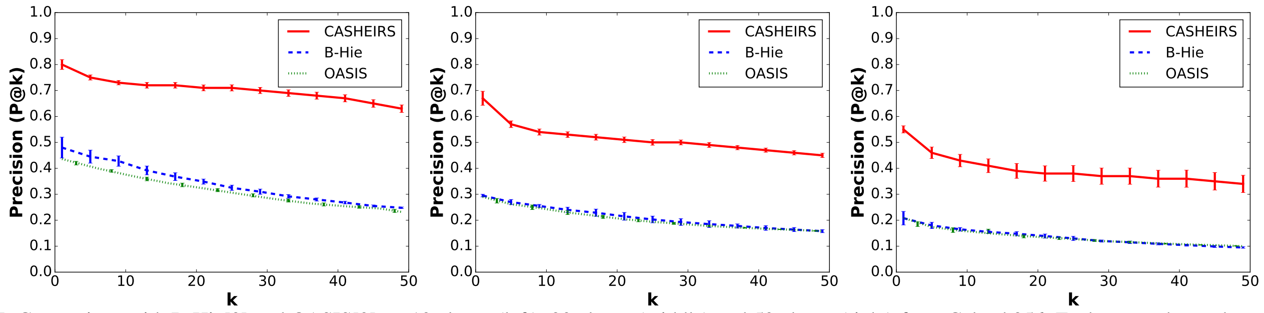


Fig. 7: Comparison with B-Hie[3] and OASIS[9] on 10 classes(left), 20 classes(middle) and 50 classes(right) from Caltech256. Each curve shows the precision at top k as a function of k neighbors. All curves except ours are extracted from [3] and [9].

16GB memory. The phone takes 161ms, 257ms and 473ms to process top-5, top-10 and top-15 returned images with an index tree for 50 categories. For 100 categories, it takes 182ms, 327ms and 564ms correspondingly. The Android client consumes the same memory cost of 306.3MBytes (same as the laptop) for storing the pre-trained CNN model, PCA matrix, rotation matrix, encryption keys and client code. It costs 6.30J to load the CNN model, and 2.26J to generate the encrypted feature for a query image. During a query process, it costs 0.12J, 0.28J and 0.47J to download and process the top-5, top-10 and top-15 returned images respectively.

VIII. Conclusion and Future Research

In this paper, we propose an efficient and secure image retrieval scheme called CASHEIRS which employs CNN model to extract discriminative features from images to improve retrieval accuracy. For storage and communication efficiency, CASHEIRS converts CNN features into binary codes. For search efficiency, a hierarchical encrypted index tree with encrypted representative vectors of categories of images is constructed. Our experimental results show that CASHEIRS achieves higher accuracy in finding matching images while incurring smaller storage and communication cost when compared to several existing schemes. In the near future, we intend to investigate the performance of CASHEIRS using larger image datasets, e.g., ImageNet. Furthermore, compared to the non-secure version, CASHEIRS with the security feature incurs 64 times larger storage and communication cost. Thus, we also hope to explore other more efficient security solution.

Acknowledgement: This work is partially supported by a NSF CSR grant 1217379.

References

[1] B. Fischer, A. Brosig, P. Welter, C. Grouls, R. W. Günther, and T. M. Deserno, "Content-based image retrieval applied to bone age assessment," in *SPIE Medical Imaging*. International Society for Optics and Photonics, 2010.

[2] S. Liu, Z. Song, G. Liu, C. Xu, H. Lu, and S. Yan, "Street-to-shop: Cross-scenario clothing retrieval via parts alignment and auxiliary set," in *Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2012.

[3] J. Deng, A. C. Berg, and L. Fei-Fei, "Hierarchical semantic indexing for large scale image retrieval," in *Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2011, pp. 785–792.

[4] J. Yuan, S. Yu, and L. Guo, "Seisa: Secure and efficient encrypted image search with access control," in *International Conference on Computer Communication (INFOCOM)*. IEEE, 2015, pp. 2083–2091.

[5] D. G. Lowe, "Object recognition from local scale-invariant features," in *International Conference on Computer Vision*. IEEE, 1999.

[6] J. Sivic and A. Zisserman, "Video google: Efficient visual search of videos," in *Toward Category-Level Object Recognition*. Springer, 2006.

[7] P. M. Roth and M. Winter, "Survey of appearance-based methods for object recognition," *Technical Report, Institute for Computer Graphics and Vision, Graz University of Technology*, 2008.

[8] Z. Wu, Q. Ke, J. Sun, and H.-Y. Shum, "A multi-sample, multi-tree approach to bag-of-words image representation for image retrieval," in *International Conference on Computer Vision*. IEEE, 2009.

[9] G. Chechik, V. Sharma, U. Shalit, and S. Bengio, "Large scale online learning of image similarity through ranking," *The Journal of Machine Learning Research*, pp. 1109–1135, 2010.

[10] W. Lu, A. Swaminathan, A. L. Varna, and M. Wu, "Enabling search over encrypted multimedia databases," in *IS&T/SPIE Electronic Imaging*. International Society for Optics and Photonics, 2009.

[11] C.-Y. Hsu, C.-S. Lu, and S.-C. Pei, "Image feature extraction in encrypted domain with privacy-preserving sift," *IEEE Trans. Image Processing*, pp. 4593–4607, 2012.

[12] C. Wang, K. Ren, S. Yu, and K. M. R. Urs, "Achieving usable and privacy-assured similarity search over outsourced cloud data," in *International Conference on Computer Communication (INFOCOM)*. IEEE, 2012, pp. 451–459.

[13] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *8th ACM Symposium on Information, Computer and Communications Security*, 2013, pp. 71–82.

[14] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE Transactions on Neural Networks*, pp. 98–113, 1997.

[15] J. Nagi, G. Di Caro, A. Giusti, F. Nagi, L. M. Gambardella *et al.*, "Convolutional neural support vector machines: Hybrid visual pattern classifiers for multi-robot systems," in *2012 11th International Conference on Machine Learning and Applications (ICMLA)*. IEEE.

[16] P. Fischer, A. Dosovitskiy, and T. Brox, "Descriptor matching with convolutional neural networks: a comparison to sift," *arXiv preprint arXiv:1405.5769*, 2014.

[17] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2011, pp. 817–824.

[18] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the twentieth annual symposium on Computational geometry*. ACM, 2004.

[19] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Advances in neural information processing systems*, 2009, pp. 1753–1760.

[20] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[22] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, pp. 241–254, 1967.

[23] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *International Conference on Computer Communication (INFOCOM)*. IEEE, 2014.

[24] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.

[25] H. Jégou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *European Conference on Computer Vision (ECCV)*. Springer, 2008, pp. 304–317.