IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 3, 2020, accepted March 13, 2020, date of publication April 13, 2020, date of current version April 21, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2984916

Privacy-Preserving Image Retrieval and Sharing in Social Multimedia Applications

ZONGYE ZHANG[1], **FUCAI ZHOU**[1], **SHIYUE QIN**[2], **QIANG JIA**[1], **AND ZIFENG XU**[1]

[1]Software College, Northeastern University, Shenyang 100819, China
[2]School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China

Corresponding author: Fucai Zhou (fczhou@mail.neu.edu.cn).

This work was supported in part by the National Natural Science Foundation of China under Grant 61772127 and Grant 61872069, in part by the National Science and Technology Major Project under Grant 2013ZX03002006, in part by the Liaoning Province Science and Technology Projects under Grant 2013217004, and in part by the Fundamental Research Funds for the Central Universities under Grant N151704002.

ABSTRACT Every day social multimedia applications generate millions of images. To handle such huge amount of images, an optimal solution is using the public cloud, since it has powerful storage capability. Images usually contain a wealth of sensitive information, therefore social service providers need not only to provide services such as retrieval and sharing but also to protect the privacies of the images. In this paper, we propose a privacy-preserving scheme for content-based image retrieval and sharing in social multimedia applications. First, the users extract visual features from the images, and perform locality-sensitive hashing functions on visual features to generate image profile vectors. We then model the retrieval on the images as the equality search on the image profile vectors. To enable accurate and efficient retrieval, we design the secure index structure based on cuckoo hashing, which has constant lookup time. To meet the requirements of dynamic image updating, we enrich our service with image insertion and deletion. In order to reduce the key management overhead and the access control overhead in social applications, we process keys using secret sharing techniques to enable the users holding similar images to query and decrypt images independently. Finally we implement the prototype of the proposed scheme, and perform experiments over encrypted image databases.

INDEX TERMS Image retrieval, image sharing, multimedia, privacy-preserving.

I. INTRODUCTION

Due to the popularity of mobile devices with cameras, such as mobile phones, tablets, sensors, and etc., the amount of images has grown tremendously. Specifically, social multimedia applications, that provide platforms to post and share multimedia, generate massive amounts of images. According to Instagram, more than 100 million images are posted per day [1]. Due to the high storage costs, social service providers prefer to outsource such massive amount of images to public cloud platforms such as Amazon Cloud.

Images usually contain sensitive information that could reveal personal privacies, and encryption is an effective method to protect privacy. Based on cryptography, a number of privacy-preserving schemes [2]–[7] that support remote image retrieval and sharing have been proposed. Considering that social multimedia application is becoming one of the main platform for image retrieval and sharing, this paper proposes a privacy-preserving image retrieval and sharing scheme for social multimedia applications. The contributions of this paper can be summarized as follows:

- We propose a privacy-preserving content-based image retrieval and sharing scheme for social multimedia applications. The scheme allows the image owners to outsource images to the public cloud server, and outsource secure index to the social service provider.
- Our scheme is constructed under a stronger and more realistic threat model that the social service provider is not a completely trusted entity. The social service provider is allowed to acquire a part of the users' information, but it cannot be trusted to store the keys.
- Our scheme greatly simplifies access control and key management in multi-user social applications. We allow

The associate editor coordinating the review of this manuscript and approving it for publication was Sedat Akleylek.

the users holding similar images to submit the retrieval queries, and decrypt the query results on their own.

- We design a secure index that provides efficient retrieval service for large image set. The index requires constant lookup time and supports dynamic image updating.

II. RELATED WORK

Content-based image retrieval (CBIR) [8] using the content of query image to search for similar images is a main research branch in the area of image retrieval. In CBIR, one key work is to extract visual features from the images for image representation and similarity measurement. In previous works, high-dimensional local features such as SIFT [9], SURF [10], ORB [11], and etc. have been extensively explored as a routine image representation, and have been proved to be effective in CBIR.

Searchable encryption (SE) [12] was proposed to protect data privacy on the remote server side, while preserving search ability. And not limited to file and texture, the data types studied in SE have been extended to image. However, when image sources are diverse and authorized users are diverse, access control and key management are challenging problems in SE.

In terms of access control, it usually requires image owners to predefine access policies for their images. Xia *et al.* [2] and Xu *et al.* [3] required the users to request authentication from the image owner before performing query. Yuan *et al.* [4] distributed roles for the users and created search policies to decide who can search a particular image with these roles. In terms of key management, one solution is to allow image owners to manage their own keys. The schemes proposed by Zou *et al.* [5] and Wang *et al.* [6] required the image owners to store their own keys, and the users can not independently decrypt the encrypted images without additional interaction with the image owners. Another solution is to introduce third-party entities for key management. The scheme proposed by Li *et al.* [7] required non-colluding entities that perform cryptographic computations to handle key management. Moreover, in social multimedia application, which involve large number of users and huge amount of images, the common solution is to let the social service providers manage the keys. Under the assumption that the social service providers are trusted entities, Yuan *et al.* [13] and Ahmed *et al.* [14] designed privacy-preserving image-centric social discovery schemes, which give the secret keys to the social service providers for storage and delegate the decryption operations to the social service providers. However, such assumption is not always reasonable in real-world applications. Usually, social service providers cannot be completely trusted, therefore the outsourced private information is facing the risks of privacy leakage and abused. For example, on April 18, 2019, Facebook announced that millions of Instagram passwords were stored in a readable format and could be accessed by Facebook employees [15].

Some works proposed using data as keys for encryption to simplify key management and access control. Pieters and Tang [16] proposed a (k, n) threshold data-based access control paradigm, where the access to information is based on the similarity between the information being accessed and the information provided to access it. Specifically, if k of n data-items are the same, the information being accessed can be recovered from the information provided to access it. Boldyreva *et al.* [17] came up with the similar idea and proposed the concept of keyless fuzzy search (KlFS) which can be applied to the images. KlFS masks database in such a way that the user can retrieve masked database and unmask the data if and only if it possesses some data that is "close to" the encrypted data. Inspired by the ideas of [16], [17], we use t-out-of-n secret sharing technique for access control and decryption to reduce the costs of key management and access control.

The rest of our paper is organized as follows: Section III reviews some preliminaries of this paper. Section IV presents the system overview. Section V proposes the detailed design of our scheme. Section VI gives security analysis and efficiency analysis. Section VII evaluates the performance of our scheme. Finally, Section VIII concludes the paper.

III. PRELIMINARIES

In this section, we introduce the basic primitives that are used in our privacy-preserving image retrieval and sharing scheme.

A. EXTENDED LOCALITY-SENSITIVE Hashing(eLSH)

Locality-Sensitive Hashing(LSH) [18] is a common algorithm used to solve the approximate nearest neighbor search in high-dimensional spaces. The high dimensionality of high-dimensional data results in high search time cost or high space overhead. LSH, which maps similar objects into the same hash buckets with high probability and reduces the dimensionality of data, has become a promising approach to similarity search in high-dimensional spaces [19]–[21]. In this paper, to amplify the accuracy of the parameters, an extension of LSH, i.e. eLSH, is adopted.

Definition 1 [Locality-Sensitive Hashing (LSH)]: Given threshold values δ^{C} and δ^{F}, where $\delta^{\mathrm{C}} < \delta^{\mathrm{F}}$, and probability values p_1 and p_2, where $p_1 < p_2$, the locality sensitive hash family $\mathcal{H} = \{h : \mathcal{D} \rightarrow \mathcal{U}\}$ is called $(\delta^{\mathrm{C}}, \delta^{\mathrm{F}}, p_1, p_2)-$sensitive if the following holds: for a distance function d, and for any $x,\ y \in \mathcal{D}$, if $d(x, y) \leqslant \delta^{\mathrm{C}}$ then $\Pr_{\mathcal{H}}[h(x) = h(y)] \geqslant p_1$; if $d(x, y) > \delta^{\mathrm{F}}$ then $\Pr_{\mathcal{H}}[h(x) = h(y)] \leqslant p_2$.

Definition 2 [Extended Locality-Sensitive Hashing (eLSH)]: Given $(\delta^{\mathrm{C}}, \delta^{\mathrm{F}}, p_1, p_2)-$sensitive hash family $\mathcal{H}$, and positive integers k and L. For $i \in [L]$ and $j \in [k]$, the hash function $g_i(x)$ is defined as $g_i(x) = (h_{i1}(x), h_{i2}(x), \ldots, h_{ik}(x))$, where $h_{ij} \in \mathcal{H}$. The set $\{g_1, g_2, \ldots, g_L\}$ is called the $(L, k) - eLSH$ [22] and satisfies the following. For a distance function d, and for any $x,\ y \in \mathcal{D}$, if $d(x, y) \leqslant \delta^{\mathrm{C}}$, then $\Pr[\exists i \in [L] : g_i(x) = g_i(y)] \geqslant 1 - (1 - p_1{}^k)^L = P_1$; if $d(x, y) > \delta^{\mathrm{F}}$, then $\Pr[\exists i \in [L] : g_i(x) = g_i(y)] \leqslant 1 - (1 - p_2{}^k)^L = P_2$.

In eLSH, the concatenation of k functions reduces the chance of collision between similar items and the set of L functions improves recall.

B. CUCKOO HASHING

Cuckoo hashing [23] is a dictionary scheme that uses two or more hash functions instead of only one to resolve hash collisions. It can balance the load and support efficient queries with constant lookup time in the worst case.

Definition 3 (Standard Cuckoo Hashing): Cuckoo Hashing has two hash tables, T_1 and T_2, which are associated with hash functions f_1 and f_2, respectively. In addition, each hash table has w buckets. An item x can be inserted into either the bucket $T_1[f_1(x)]$ of T_1 or the bucket $T_2[f_2(x)]$ of T_2.

Definition 4 (Improved Cuckoo Hashing): Improved Cuckoo Hashing [24] has l ($l > 2$) hash tables $\{T_1, T_2, \ldots, T_l\}$, where each table T_i has a hash function f_i and w buckets. An item x can be inserted into the bucket $T_i[f_i(x)]$ of T_i, where $i \in [l]$.

Give a simple example to explain cuckoo hashing. When inserting item c into the standard cuckoo hashing, if either of two available positions $T_1[f_1(c)]$ and $T_2[f_2(c)]$ is empty, c will be inserted. If the two available positions are occupied by a and b, c kicks away a or b and reinserts a or b.

C. SECRET SHARING

Secret sharing [25] is a cryptographic method that divides data into n pieces, and distributes pieces among n participants, so that the secret can be reconstructed by cooperation of no less than t participants. In terms of privacy, participants holding less than t pieces learns no more information about the secret than knowing no piece. A t-out-of-n secret sharing scheme is defined by algorithms (Share, $\mathsf{Reconst}$), where Share is the secret sharing algorithm and $\mathsf{Reconst}$ is the reconstruction algorithm.

Share: It takes as input a secret s, and outputs a set of pieces $\{p_1, p_2, \ldots, p_n\}$.

$\mathsf{Reconst}$: It takes as input a set of pieces $\{p_1, p_2, \ldots, p_m\}$, and outputs s if $m \geqslant t$, or $\perp$ if $m < t$.

D. CLOSENESS DOMAINS

$\mathcal{D}$ is defined as either a finite or an infinite domain, and Cl is defined as the symmetric (partial) closeness function that takes any $x, y \in \mathcal{D}$ as input and outputs a member of $\{close, far\}$. For a distance function d, closeness parameters are defined as δ^{C} and δ^{F}, where $\delta^{\mathrm{C}} < \delta^{\mathrm{F}}$. The closeness domain ($\mathcal{D}$, Cl) [26] is defined as follows. For any $x, y \in \mathcal{D}$,

$$Cl(x,y) = \begin{cases} close, & \text{if } d(x,y) \leqslant \delta^{\mathrm{C}} \\ far, & \text{if } d(x,y) > \delta^{\mathrm{F}} \end{cases} \tag{1}$$

IV. SYSTEM OVERVIEW

To get closer to reality, we treat the social service providers as incompletely trusted entities. And under the such assumption, we design privacy-preserving image retrieval and sharing scheme for social multimedia applications. To support efficient retrieval on large-scale images, we use cuckoo hashing to design efficient index. To reduce the costs of key management and access control caused by multi users in social applications, we use the idea of KIFS to process the images. The intuition of our scheme is using image content to search for similar images and share images. In social multimedia applications, if two users post images with similar content, such as the same restaurant they went to, the same poster they like, or the same movie they saw, they are very likely to have similar interests and they can be potential friends. For example, Alice posts the images about her mountain climbing, then she may want to search for mountaineering enthusiasts through the posted images, and further discuss climbing skills or meet up with a mountain trip. So our scheme can be extended to friend recommendation and other applications.

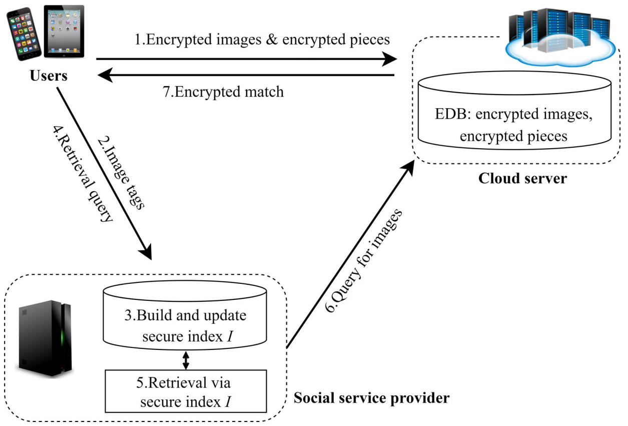

Users
1.Encrypted images & encrypted pieces
7.Encrypted match
EDB: encrypted images, encrypted pieces
Cloud server
4.Retrieval query
2.Image tags
6.Query for images
3.Build and update secure index I
5.Retrieval via secure index I
Social service provider

FIGURE 1. System architecture.

A. ARCHITECTURE AND ENTITIES

Fig.1. illustrates the architecture of our scheme in a high-level. The scheme involves three entities: the users, the social multimedia application server (SS), and the cloud server (CS).

The users are mobile terminals with certain level of computing power, such as laptop, smartphone, tablet, and etc. The SS is the internal server of social service provider that offers social multimedia services to the users, such as Instagram, Facebook, Flickr, and etc. The CS is a third-party cloud server such as Amazon cloud.

B. SERVICE FLOW

As shown in Fig.1, the service flow of our scheme involves 4 phases: 1) *Image preprocessing (steps 1&2)*: A user first encrypts his images. Then the user extracts visual features from the images and calculates the profile vectors from the visual features. To simplify the key management, the user uses a secret sharing scheme to share the keys that used to encrypt the images, and encrypts all the pieces using the profile vectors. Then the user uploads both the encrypted images and the encrypted pieces to the CS. Meantime, the user derives the tags that used for similarity retrieval from the

TABLE 1. **Notations.**

Notations	Descriptions
I	The image
I^*	The encrypted image
m	The threshold value for secret sharing
K	The symmetric secret key
$\mathcal{I}$	The secure index
n	The pieces number, the extracted features number
N	The size of the image set
L, k	The parameters of *eLSH*
$g_i()$=$(h_{i1}()$,$h_{i2}()$,. . . ,$h_{ik}())$	The i-th hash function of *eLSH*, $i \in[L]$
V=$\{v_1, v_2, ..., v_n\}$	The image feature vector set
$Piece$=$\{p_1, p_2, ..., p_n\}$	The secret piece set of K
M_i=$\{m_{i1}, m_{i2}, ..., m_{iL}\}$	The profile vector set of v_i, i∈$[n]$
sk_{ij}	The secret key derived from m_{ij}, i∈$[n]$, $j \in[L]$
p^*_{ij}	The encrypted piece calculated by p_i and sk_{ij}, $i \in[n]$, $j \in[L]$
$Piece^*$	The encrypted piece set of K
tag_{ij}	One tag of v_i, derived from m_{ij}, i∈$[n]$, j∈$[L]$
Tag	The tag set of the image
T=$\{T_1, T_2, ..., T_l\}$	The cuckoo hashing with l hash tables
d	The probe value of the cuckoo hashing
H, G	Two secure pseudo-random functions

profile vectors, and transfers the tags to the SS. 2) *Index Update (step 3)*: Upon receiving the tags from the user, the SS inserts the tags into the index. 3) *Image retrieval (steps 4&5)*: For the retrieval query from the user, the SS performs similarity retrieval via the index. 4) *Image sharing (steps 6&7)*: After getting retrieval results, the SS asks the CS to return the corresponding encrypted images and the encrypted pieces to the user. The user reconstructs the keys and decrypts the images.

C. DEFINITION AND NOTATION

Before presenting the definition of our scheme, we first define some notations, as listed in Table 1, which will be used in our following definition and constructions. Then we formally define our notion of Privacy-preserving Image Retrieval and Sharing scheme in Social Multimedia applications.

Definition 5 (Privacy-Preserving Image Retrieval and Sharing): A privacy-preserving image retrieval and sharing scheme in social multimedia applications consists of the following algorithms or protocols.

- Setup$(1^\lambda, params) \rightarrow (PFuncs, T)$: Run by the SS. It takes as input a security parameter λ and function initialization parameters *params*. It outputs public functions *PFuncs* and a cuckoo hashing T.
- ImageProcess$(1^\lambda, PFuncs, I) \rightarrow (I^*, id_I, Piece^*, Tag)$: Run by the user. It takes as input a security parameter λ, public functions *PFuncs* and the image I. It outputs the encrypted image I^*, the image identifier id_I, the encrypted piece set *Piece**, and the tag set *Tag*.
- IndexGen$(T, Tag) \rightarrow \mathcal{I}$: Run by the SS. It takes as input the cuckoo hashing T and the tag set *Tag*. It outputs the secure index $\mathcal{I}$.
- EDBGen$(id_I, I^*, Piece^*) \rightarrow EDB$: Run by the CS. It takes as input the image identifier id_I, the encrypted image I^*, and the encrypted piece set *Piece**. It outputs the encrypted database *EDB*.
- QueryGen$(PFuncs, I) \rightarrow Tag$: Run by the user. It takes as input the public functions *PFuncs* and the query image I. It outputs the query token *Tag*.
- Retrieval$(Tag, \mathcal{I}) \rightarrow (\{I^*\}, \{Piece^*\}, AUX)$: Run by the SS and CS. It takes as input the query token *Tag* and the secure index $\mathcal{I}$. It outputs the encrypted similar images $\{I^*\}$, the corresponding encrypted piece sets $\{Piece^*\}$, and the auxiliary information set *AUX*.
- ImageRecover$(I^*_o, Piece^*, I, AUX, PFuncs) \rightarrow I_o$: Run by the user. It takes as input the encrypted image I^*_o, the corresponding encrypted piece set *Piece**, the query image I, the auxiliary information set *AUX*, and public functions *PFuncs*. It outputs the the image I_o.

We also formally define our notion of Image Similarity.

Definition 6 (Image Similarity): For two images I_1 and I_2, $V_1 = \{v_1, v_2, \ldots, v_n\}$ and $V_2 = \{v'_1, v'_2, \ldots, v'_n\}$ are feature vector sets extracted from I_1 and I_2, respectively. For two feature vectors v and v', if they have the same $(L, k) - eLSH$ value, $Cl(v, v') = close$ and v and v' are considered similar. If I_1 and I_2 have no less than m similar feature vectors, $Cl(I_1, I_2) = close$ and I_1 and I_2 are considered similar.

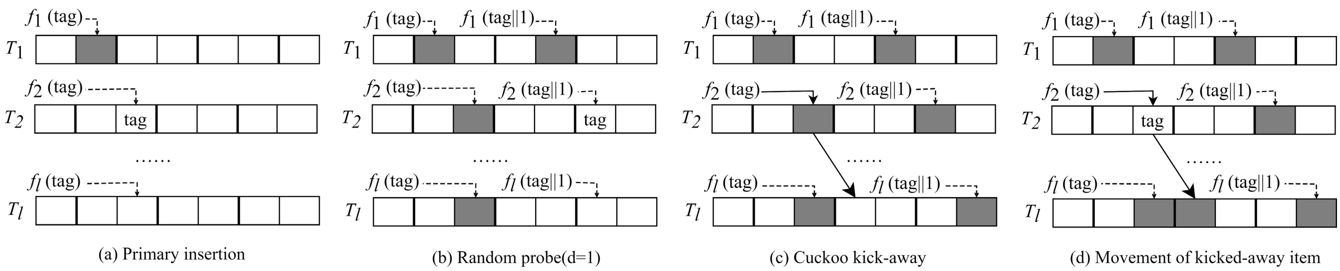
D. THREAT MODEL

In our scheme, we suppose the CS and the SS are two non-colluding "honest-but-curious" adversaries. We first consider the primary security threat from the CS. The CS is assumed to follow the specified service flow faithfully, but it intends to learn the content of the images, the profile vectors of the visual features, and the keys for the images. We focus on preserving the confidentiality of the images and the key pieces outsourced in the CS. The SS is also assumed to follow the scheme faithfully, but it also intends to infer the keys and the visual features. We should ensure that the SS cannot acquire the keys, and the tags do not reveal the content of the visual features. For the users, we assume that they are trustworthy and can preserve their secrets, including the keys, the visual features, and the profile vectors.

V. CONSTRUCTION

We propose the detailed construction of our privacy-preserving image retrieval and sharing scheme in social multimedia applications.

1) Setup$(1^\lambda, params) \rightarrow (PFuncs, T)$: Given a security parameter λ, the SS initializes Pseudo-Random Functions (PRF) H and G. Given function initialization parameters $params = \{L, k, m, n, l, w, u\}$, SS initializes an extended locality-sensitive hashing $(L, k) - eLSH$, a m-out-of-n secret sharing scheme (Share, Reconst), and a cuckoo hashing $T = \{T_1, T_2, \ldots, T_l\}$ with l hash tables. Each table T_i has a hash function f_i and w buckets, where each bucket is $u-$bit in length. Then the SS sets $PFuncs = \{H, G, (L, k) - eLSH, (\text{Share}, \text{Reconst})\}$ as the public functions and makes *PFuncs* public.



(a) Primary insertion (b) Random probe(d=1) (c) Cuckoo kick-away (d) Movement of kicked-away item

FIGURE 2. **Index.**

2) ImageProcess$(1^{\lambda}, PFuncs, I) \rightarrow (I^*, id_I, Piece^*, Tag)$: Before uploading images to the social multimedia application, the user takes the following steps to process the image.
   - Image encryption: The user first generates identifier id_I for the image I. Then given a security parameter λ, the user chooses a secret key K for the image I, and invokes semantic secure symmetric encryption algorithm to encrypt I and obtains the encrypted image I^*.
   - Secret key sharing: The user performs $Piece \leftarrow \mathsf{Share}(K)$, where $Piece = \{p_1, p_2, \ldots, p_n\}$ is the secret piece set.
   - Profile generation: From the image I, the user extracts the feature vector set $V = \{v_1, v_2, \ldots, v_n\}$, e.g. ORB, SURF, and etc. Note that, the number of the extracted feature vectors is equal to the number of K's pieces. For each v_i in V, the user performs $(L, k) - eLSH$ to generate the profile vector set $M_i = \{m_{i1}, m_{i2}, \ldots, m_{iL}\}$, where $m_{ij} = g_j(v_i) = \{h_{j1}(v_i), h_{j2}(v_i), \ldots, h_{jk}(v_i)\}$, $i \in [n]$ and $j \in [L]$.
   - Tags generation: For each m_{ij} in M_i, the user performs $tag_{ij} \leftarrow H(m_{ij})$ and appends $\{tag_{ij}, id_I, i, j\}$ to the tag set Tag of the image I, where tag_{ij} is used for equality matching in retrieval, id_I is the image identifier, i is the feature identifier, and j is the $eLSH$ identifier which indicates that the j-th hash function $g_j()$ of $eLSH$ is used for generating m_{ij}. After performing the above operations on all profile vectors of I, the user gets the tag set Tag of the image I.
   - Pieces encryption: For each m_{ij} in M_i, the user performs $sk_{ij} \leftarrow G(m_{ij})$ and gets the secret key sk_{ij}. Then the user invokes symmetric encryption algorithm $p^*_{ij} \leftarrow \mathsf{Enc}(sk_{ij}, p_i)$ to encrypt the secret piece p_i of K. The user appends the encryption result p^*_{ij} to $Piece^*$, i.e. the encrypted piece set of I. After performing the above operations on all profile vectors, the user gets the encrypted piece set $Piece^*$ of the image I.

   Finally, the user uploads the encrypted image I^* and the encrypted piece set $Piece^*$ to the CS, and uploads Tag to the SS.
3) IndexGen$(T, Tag) \rightarrow \mathcal{I}$: Given the cuckoo hashing T, the SS takes the following steps to insert Tag into it and construct the secure index $\mathcal{I}$. The specific algorithm is described in Algorithm 1 and the index design is illustrated in Fig. 2.

   For each $item = \{tag, id_{image}, id_{feature}, id_{eLSH}\}$ in Tag:
   - Primary insertion: the SS tries to insert $item$ into one of the l hash tables. The SS first initializes two sets, $Occupied = \varnothing$ and $Whole = \{1, 2, \ldots, l\}$. Then it randomly picks an $i \in Whole$. If the bucket $T_i[f_i(tag)]$ is empty, the SS inserts $item$ into $T_i[f_i(tag)]$ and the insertion is finished; otherwise, if the bucket $T_i[f_i(tag)]$ is occupied, the SS appends i to $Occupied$ and update $Whole = Whole - Occupied$. The SS repeats the above operations iteratively until $item$ is inserted into $\mathcal{I}$ or $Whole = \varnothing$.
   - Random probe: If none of the above l buckets is empty, then the SS expands the number of target buckets to $d+1$ for each hash table. The SS initializes $Occupied = \varnothing$ and $Whole = \{1, 2, \ldots, l\}$. The SS first randomly picks an $i \in Whole$. Then for each $\delta \in [1, d]$, if the bucket $T_i[f_i(tag \parallel \delta)]$ is empty, the SS inserts $item$ into $T_i[f_i(tag \parallel \delta)]$ and the insertion is finished; otherwise, if all of d buckets are occupied, the SS appends i to $Occupied$ and update $Whole = Whole - Occupied$. SS repeats the above operation iteratively until $item$ is inserted into $\mathcal{I}$ or $Whole = \varnothing$.
   - Cuckoo kick-away: If all of the above $l + l \times d$ buckets are occupied, the SS randomly selects an $i \in [l]$ and kicks away the item $Kicked_{out}$ stored in $T_i[f_i(tag)]$. Then the SS inserts $item$ into the bucket $T_i[f_i(tag)]$ and re-inserts $Kicked_{out}$ back via the above steps iteratively.

   After performing the above operations on all tags in Tag, the index $\mathcal{I}$ is updated.
4) EDBGen$(id_I, I^*, Piece^*) \rightarrow EDB$: Given the image identifier id_I, the CS maps it to an address Add_I in the cloud storage. Then the CS stores I^* and $Piece^*$ at Add_I.
5) QueryGen$(PFuncs, I) \rightarrow Tag$: The user first extracts the feature vector set $V = \{v_1, v_2, \ldots, v_n\}$ from the query image I. Then for each v_i in V, the user performs $(L, k) - eLSH$ to generate the profile vector set $M_i = \{m_{i1}, m_{i2}, \ldots, m_{iL}\}$. Finally for each m_{ij} in M_i, the user performs $tag_{ij} \leftarrow H(m_{ij})$ and generates the tag set Tag. The user sends Tag to the SS for retrieval.

6) $\mathsf{Retrieval}(Tag, \mathcal{I}) \rightarrow (\{I^*\}, \{Piece^*\}, AUX)$: After receiving *Tag* of the query image I, the SS takes the following steps to retrieve similar images and asks the CS to return the user the encrypted images. The specific protocol is described in Algorithm 2.
   - Trapdoor generation: For each $item = \{tag_{ij}, id_I, i, j\}$ in *Tag* of the query image I, where $i \in [n]$ and $j \in [L]$, the SS transforms it into a trapdoor $t = \{(t_1^0, \ldots, t_1^d), \ldots (t_l^0, \ldots, t_l^d)\}$, where $t_\eta^\delta = f_\eta(tag_{ij} \parallel \varphi)$, $\varphi = \varnothing$ if $\delta = 0$, $\varphi = \delta$ if $\delta \in [d]$. After performing the above operations on all tags in *Tag*, the SS gets $\mathbf{t}$, which consists of $n \times L$ trapdoors t.
   - Bucket location: For t_η^δ in t, where $\eta \in [l]$ and $\delta \in [d]$, the SS locates the bucket $T_\eta[t_\eta^\delta]$. If $item = \{tag_{ij}, id_I, i, j\}$, that is used to generate t, and $\{tag, id_{image}, id_{feature}, id_{eLSH}\}$, that is the content of $T_\eta[t_\eta^\delta]$, have the same *eLSH* identifier, i.e. $id_{eLSH} = j$, then SS appends $\{id_{image}, id_{feature}, i, j\}$ to the intermediate result set *IR*. The SS repeats the above operations on all t of $\mathbf{t}$ and updates *IR*.
   - Threshold filtering: For each image id_{image} that exists in the *IR*, the SS counts the number of similar feature vectors between id_I and id_{image}. If the number of similar features is greater than or equal to the threshold value m, the SS adds the image identifier id_{image} to the query result set *QR*, and appends all items $\{id_{image}, id_{feature}, i, j\}$ associated with id_{image} to the auxiliary information set *AUX*. Finally, the SS sends *QR* to the CS for requesting images, and returns *AUX* to the user for speeding up the decryption operation.
   - Query for images: After receiving query result set *QR* from the SS, the CS retrieves encrypted database *EDB*. For each image identifier id_{image} in *QR*, the CS maps it to the address Add_{image} and obtains I^*_{image} and $Piece^*_{image}$ stored at Add_{image}. After performing the above operations on all identifiers in *QR*, the CS gets all encrypted similar images $\{I^*\}$ and the corresponding encrypted piece set$\{Piece^*\}$. Finally the CS returns them to the user.

7) $\mathsf{ImageRecover}(I_o^*, Piece^*, I, AUX, PFuncs) \rightarrow I_o$: For the encrypted image I_o^* and the corresponding encrypted piece set $Piece^*$, the user takes the following steps to recover the image.
   - Pieces recovery: For each item $\{id_{I_o}, id_{feature}, i, j\}$ in *AUX* corresponds to the encrypted image I_o^*, the user first performs $sk_{ij} \leftarrow G(m_{ij})$, where m_{ij} of the query image I has been calculated in the $\mathsf{QueryGen}$ step. Then the user invokes symmetric decryption algorithm $p_{id_{feature}} \leftarrow \mathsf{Dec}(sk_{ij}, p^*_{id_{feature},j})$ and obtains the secret piece $p_{id_{feature}}$. After performing the above operations on all of the items correlative to I_o^* in *AUX*, the user recovers at least m pieces, i.e. $\{p_1, p_2, \ldots, p_{m^*}\}$, where $m^* \geqslant m$.
   - Secret key reconstruction: The user performs the algorithm $K \leftarrow \mathsf{Reconst}(p_1, p_2, \ldots, p_{m^*})$.

Algorithm 1 Index Generation

Input: $Tag = \{\{tag_{11}, \ldots, tag_{1L}\}, \ldots, \{tag_{n1}, \ldots, tag_{nL}\}\}$: the tag set of the inserted image, T: the cuckoo hashing

Output: Updated secure index $\mathcal{I}$

```
function INSERTIDX($Tag$, $T$)
    for $i \leftarrow 1$ to $n$ do
        for $j \leftarrow 1$ to $L$ do
            Insert:
            $Occupied \leftarrow \varnothing$;
            $Whole \leftarrow \{1, 2, \ldots, l\}$;
            while $Whole \neq \varnothing$ do
                Randomly pick $k \in Whole$;
                if $T_k[f_k(tag_{ij})] = NULL$ then
                    $T_k[f_k(tag_{ij})] \leftarrow tag_{ij}$
                    goto:Next;
                end if
                Append $k$ to $Occupied$;
                $Whole \leftarrow Whole - Occupied$;
            end while
            $Occupied \leftarrow \varnothing$;
            $Whole \leftarrow \{1, 2, \ldots, l\}$;
            while $Whole \neq \varnothing$ do
                Randomly pick $k \in Whole$;
                for $\delta \leftarrow 1$ to $d$ do
                    if $T_k[f_k(tag_{ij}||\delta)] = NULL$ then
                        $T_k[f_k(tag_{ij}||\delta)] \leftarrow tag_{ij}$;
                        goto:Next;
                    end if
                end for
                Append $k$ to $Occupied$;
                $Whole \leftarrow Whole - Occupied$;
            end while
            Randomly pick $k \in [l]$;
            $Kicked_{out} \leftarrow T_k[f_k(tag_{ij})]$;
            $T_k[f_k(tag_{ij})] \leftarrow tag_{ij}$;
            goto:Insert; // insert $Kicked_{out}$
            Next:
        end for
    end for
end function
```

   - Image decryption: The user invokes symmetric decryption algorithm to decrypt I_o^* with K, and obtains the image I_o.

Image Update: As the user adds or deletes the image, not only the encrypted database stored in the CS should be updated, but also the secure index stored in the SS is updated. For image addition, *Tag* of the added image should be inserted into $\mathcal{I}$. The SS inserts *Tag* into the cuckoo hashing following the steps in $\mathsf{IndexGen}$. For image deletion, *Tag* of the deleted image should be removed from $\mathcal{I}$. First the SS performs trapdoor generation to generate $\mathbf{t}$ and locates $n \times L \times l \times (d+1)$ buckets according to the steps in $\mathsf{Retrieval}$. Then the SS checks whether the items stored in these buckets belong

Algorithm 2 Image Retrieval

Input: $Tag = \{\{tag_{11}, \ldots, tag_{1L}\}, \ldots, \{tag_{n1}, \ldots, tag_{nL}\}\}$: the query token, $\mathcal{I}$: the secure index

Output: $\{I^*\}$: the encrypted similar images, $\{Piece^*\}$: the corresponding encrypted piece set; AUX: auxiliary information set.

```
function RETRIEVAL($Tag$, $\mathcal{I}$)
    Retrieval on SS:
    $IR \leftarrow \varnothing$
    for $tag_{ij}$ in $Tag$ do
        for $k \leftarrow 1$ to $l$ do
            for $\delta \leftarrow 0$ to $d$ do
                if $\delta = 0$ then
                    $\varphi = \varnothing$;
                else
                    $\varphi = \delta$;
                end if
                $t_k^\delta = f_k(tag_{ij} \parallel \delta)$;
                $(tag, id_{image}, id_{feature}, id_{eLSH}) \leftarrow T_k[t_k^\delta]$;
                if $id_{eLSH} = j$ then
                    $R \leftarrow \{id_{image}, id_{feature}, i, j\}$;
                    Append $R$ to $IR$;
                end if
            end for
        end for
    end for
    $count_{dict} \leftarrow$ dict();
    $aux \leftarrow$ invertedindex();
    for $R$ in $IR$ do
        if $R.id_{image} \in count_{dict}$ then
            $count_{dict}[R.id_{image}]$+=1;
            Append $R$ to aux[$R.id_{image}$];
        else
            $count_{dict}[R.id_{image}]$=1;
            Append $R$ to aux[$R.id_{image}$];
        end if
    end for
    $QR \leftarrow \varnothing$;
    $AUX \leftarrow \varnothing$;
    for $id_{image}$ in $count_{dict}$ do
        if $count_{dict}[id_{image}] \geqslant m$ then
            Append $id_{image}$ to $QR$;
            Append aux[$id_{image}$] to $AUX$;
        end if
    end for
    Retrieval on CS:
    $\{I^*\} \leftarrow \varnothing$;
    $\{Piece^*\} \leftarrow \varnothing$;
    for $id_{image}$ in $QR$ do
        $Add_{image} \leftarrow map(id_{image})$;
        Read $(I^*_{image}, Piece^*_{image})$ at $Add_{image}$;
        Append $I^*_{image}, Piece^*_{image}$ to $\{I^*\}$& $\{Piece^*\}$;
    end for
end function
```

to *Tag*. For the items belong to *Tag*, the SS deletes the items and empties the corresponding buckets.

VI. ANALYSIS OF THE PROPOSED SCHEME

A. SECURITY ANALYSIS

In our scheme, there are two non-colluding and ''honest-but-curious'' servers. We first quantify the information leakage in the view of the SS and the CS respectively, and then demonstrate that neither the SS nor the CS can infer any extra information from their views.

First, we quantify the information leakage in the view of the SS. The SS stores the secure index $\mathcal{I}$, and handles the image retrieval. The leakage patterns obtained by the SS include the closeness pattern which is inferred from the index $\mathcal{I}$, the search pattern which is inferred from the retrieval queries, and the access pattern which is inferred from the retrieval operations. The formal definitions are as follows.

Definition 7 [Closeness Pattern (CP)]: For all images $\{I_1, I_2, \ldots, I_N\}$ inserted into the index $\mathcal{I}$, **CP** is defined as a symmetric matrix $CP_{N\times N}$, where the element $CP_{N\times N}[i][j]$ is a list that records the identifiers of the similar features between two images and the identifier of eLSH that results in the same tag of these two similar features. For $1 \leqslant i, j \leqslant N$, if I_i and I_j have no similar feature, $CP_{N\times N}[i][j] = \varnothing$, otherwise, for $tag_{k\eta} \in I_i$ and $tag_{l\eta} \in I_j$, if $tag_{k\eta} = tag_{l\eta}$, (k, l, η) is an element of $CP_{N\times N}[i][j]$.

Definition 8 [Search Pattern (SP)]: For a set of q queries $\{Tag_1, Tag_2, \ldots, Tag_q\}$, **SP** is defined as a symmetric matrix $SP_{q\times q}$, where the element $SP_{q\times q}[i][j]$ is a list that records the identifiers of the similar features between two queries and the identifier of the eLSH that results in the same tag of these two similar features. For $1 \leqslant i, j \leqslant q$, if Tag_i and Tag_j have no similar feature, $SP_{q\times q}[i][j] = \varnothing$, otherwise, for $tag_{k\eta} \in Tag_i$ and $tag_{l\eta} \in Tag_j$, if $tag_{k\eta} = tag_{l\eta}$, (k, l, η) is an element of $SP_{q\times q}[i][j]$.

Definition 9 [Access Pattern (AP)]: For a set of q queries $\{Tag_1, Tag_2, \ldots, Tag_q\}$, **AP** *is defined as* $(\mathbf{B}, AP_{(q+\omega)\times(q+\omega)})$. $\mathbf{B} = (\{B_1\}, \{B_2\}, \ldots, \{B_q\})$, where $\{B_\iota\}$ is the accessed buckets of Tag_ι. $AP_{(q+\omega)\times(q+\omega)}$ is a symmetric matrix, where ω is the number of image identifiers appeared in $\mathbf{B}$ and the element $AP_{(q+\omega)\times(q+\omega)}[i][j]$ is a list that records the identifiers of similar features between two images and the identifier of the eLSH that results in the same tag of these two similar features. For $1 \leqslant i, j \leqslant q + \omega$, $SubTag_i$ and $SubTag_j$ are the subsets of Tag_i and Tag_j, which are inferred from the queries and $\mathbf{B}$. If $SubTag_i$ and $SubTag_j$ have no similar feature, $AP_{(q+\omega)\times(q+\omega)}[i][j] = \varnothing$, otherwise, for $tag_{k\eta} \in SubTag_i$ and $tag_{l\eta} \in SubTag_j$, if $tag_{k\eta} = tag_{l\eta}$, (k, l, η) is an element of $AP_{(q+\omega)\times(q+\omega)}[i][j]$.

Definition 10 [View of SS ($\mathcal{V}_{SS}$)]: $\mathcal{V}_{SS}$ is the information leakage obtained by SS for a set of q queries, and defined as $\mathcal{V}_{SS} = (\mathcal{I}, \mathbf{CP}, \mathbf{SP}, \mathbf{AP})$.

Theorem 1 (Indistinguishability for SS): Let $\mathcal{A}$ be a probabilistic polynomial-time (P.P.T.) adversary, Sim_{SS} be a P.P.T. simulator, and $\widetilde{\mathcal{V}}_{SS}$ is the simulated view. For a set of q

queries, we have $|\Pr[\mathcal{A}(\mathcal{V}_{SS}) = 1] - \Pr[\mathcal{A}(\widetilde{\mathcal{V}}_{SS}) = 1]| < \epsilon$, where ϵ is negligible, i.e., $\mathcal{V}_{SS}$ and $\widetilde{\mathcal{V}}_{SS}$ are computationally indistinguishable.

Proof: We first give the process of the simulating view $\widetilde{\mathcal{V}}_{SS}$:

- Simulation of index $\widetilde{\mathcal{I}}$. Sim_{SS} generates l hash tables, and each table contains w buckets. i.e., $\widetilde{\mathcal{I}}$ has the same size as $\mathcal{I}$. Recall that the items stored in the buckets of $\mathcal{I}$ contain *tag*, the identifier of the image, the identifier of the feature, and the identifier of the *eLSH*. Therefore, for each occupied bucket of $\mathcal{I}$, according to the closeness pattern **CP** of $\mathcal{I}$, Sim_{SS} first generates a random string $\widetilde{tag}$ that has the same length with *tag*, and then generates three random strings as the identifiers that has the same lengths with the real ones.
- Simulation of trapdoors and accessed buckets for a set of q queries. For each item in the query q_1, Sim_{SS} simulates a trapdoor $\widetilde{t_1} = \{(\widetilde{t_1^0}, \ldots, \widetilde{t_1^d}), (\widetilde{t_2^0}, \ldots, \widetilde{t_2^d}), \ldots (\widetilde{t_l^0}, \ldots, \widetilde{t_l^d})\}$, where $\widetilde{\mathrm{t}_\eta^\delta}$ is a random string that has the same length as the real one. According to the access pattern **AP**, for $\widetilde{t_1}$, Sim_{SS} simulates the accessed buckets $\{\widetilde{B_1}\}$, where the number of simulated buckets is equal to the number of real ones. For queries q_{i+1} $(1 \leq \mathrm{i} \leq q\text{-}1)$, Sim_{SS} adaptively simulates trapdoors $\widetilde{t_{i+1}}$ according to the search pattern **SP** obtained from the previous i queries. For the subcomponent of $\widetilde{t_{i+1}}$ that has not appeared in the previous queries, Sim_{SS} adopts the above approach to generate a random string as the subcomponent and then Sim_{SS} adopts the same approach to simulate accessed buckets via **AP**. For subcomponent of $\widetilde{t_{i+1}}$ that appeared before, Sim_{SS} uses the same string used before and the same bucket for simulation.

Due to the pseudo-randomness of the secure PRFs, $\widetilde{tag}$ and tag, $\widetilde{\mathcal{I}}$ and $\mathcal{I}$, $\widetilde{t}$ and t, $\widetilde{\mathbf{B}}$ and $\mathbf{B}$ are indistinguishable. Therefore, the simulated view $\widetilde{\mathcal{V}}_{SS}$ is indistinguishable from $\mathcal{V}_{SS}$. □

Then, we quantify the information leakage in the view of the CS. The CS stores the encrypted images and the encrypted pieces, and responds to the request of the SS.

Definition 11 [View of CS ($\mathcal{V}_{CS}$)]: $\mathcal{V}_{CS}$ is the information leakage obtained by the CS and is defined as $\mathcal{V}_{CS} = EDB$, where EDB includes the encrypted images $(I_1^*, I_2^*, \ldots, I_N^*)$ and the encrypted pieces $(Piece_1^*, Piece_2^*, \ldots, Piece_N^*)$.

Theorem 2 (Indistinguishability for CS): Let $\mathcal{A}$ be a P.P.T. adversary, Sim_{CS} be a P.P.T. simulator, and $\widetilde{\mathcal{V}}_{CS}$ is the simulated view. For a negligible ϵ, we have $|\Pr[\mathcal{A}(\mathcal{V}_{CS}) = 1] - \Pr[\mathcal{A}(\widetilde{\mathcal{V}}_{CS}) = 1]| < \epsilon$, i.e., $\mathcal{V}_{CS}$ and $\widetilde{\mathcal{V}}_{CS}$ are computationally indistinguishable.

Proof: Sim_{CS} can simulate $\widetilde{EDB}$ according to $\mathcal{V}_{CS}$. For the storage at each address, Sim_{CS} first generates a random string $\widetilde{I}^*$ that has the same length as I^*. Then Sim_{CS} generates random strings $\widetilde{Piece}^*$ that has the same length as the encrypted pieces $Piece^*$. Due to semantic security of the symmetric encryption, $\widetilde{I}^*$ and I^*, $\widetilde{Piece}^*$ and $Piece^*$ are indistinguishable. Therefore, the simulated view $\widetilde{\mathcal{V}}_{CS}$ is indistinguishable from $\mathcal{V}_{CS}$. □

TABLE 2. Performance comparison.

	Our scheme	Scheme of [5]
Index size	$O(NnL)$	$O(N) + O(N)$
Retrieval time	$O(1)$	$O(N) + O(\delta)$
Imageprocess comp.	$(2nL)PRF + (nLk)LSH$ $+1SS + (nL)ENC$	$(n+3nk)PRF$
QueryGen comp.	$(nL)PRF + (nLk)LSH$	$(n+3nk)PRF$
Retrieval comp.	$nLl \cdot (d+1)(PRF+CP)$	$(N+\delta)CP + \delta MUL$ $+(2nk)PRF$
ImageRecover comp.	$(m+1)DEC + mPRF + 1SR$	$1DEC$
User&owner Comm.	$--$	1

B. EFFICIENCY ANALYSIS

In this section, we discuss the performance of our scheme and make comparisons to the scheme proposed by Zou *et al.* [5] that is most similar to ours.

The factors we focus on to evaluate performance include the index size, retrieval time, image processing computation, query generation computation, retrieval computation, image recovery computation, and communication times between the image owner and the user. Table 2 summarizes the complexity or the operations required for these phases, where *PRF* denotes evaluation of a pseudo-random function, *LSH* denotes evaluation of a locality-sensitive hashing, *CP* denotes a numerical comparison, *MUL* denotes a multiplication operation, *ENC* denotes a symmetric encryption operation, *DEC* denotes a symmetric decryption operation, *SS* denotes a sharing operation in secret sharing, *SR* denotes a reconstruction operation in secret sharing, N denotes the total number of images, and δ denotes the number of matched images in the first round of comparison in [5].

Owing to the properties of cuckoo hashing, our retrieval time $O(1)$ is constant and our index size $O(NnL)$ is linearly increased with the number of tags in N images. Suppose the index load factor τ is defined as $\tau = p/lw$, where p is the number of occupied buckets, and lw is the size of the cuckoo hashing. So the size of our index design is $(NnL) \cdot u/\tau$, where u is the bit length of each bucket. In the process of image processing, the user performs $n \cdot L \cdot k$ LSH to generate profile vectors for the image. Then the user invokes Share algorithm to share the secret key and performs $n \cdot L$ PRFs to generate *Tag*, Finally, the user performs $n \cdot L$ PRFs and $n \cdot L$ symmetric encryption to generate $Piece^*$. To generate the query, the user performs $n \cdot L \cdot k$ LSH to generate profile vectors for the query image and then performs $n \cdot L$ PRFs to generate *Tag*. In the process of retrieval, the user performs $nLl \cdot (d+1)$ PRFs to locate buckets and performs $nLl \cdot (d+1)$ comparisons to compare contents in the buckets with tags. In the process of image recovery, the user performs m PRFs to generate keys for decrypting encrypted pieces, performs m symmetric decryption to recover pieces of the secret key, invokes Reconst to reconstruct the secret key, and finally performs symmetric decryption algorithm to decrypt the encrypted image.

From Table 2, we could see that our scheme is more efficient in retrieval, but more complex in image processing

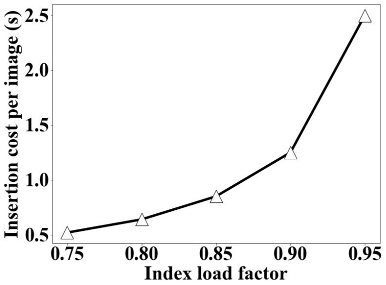


FIGURE 3. Insertion cost.

and image recovery compared with [5]. That's because in our scheme the key used for decrypting image is reconstructed by the user himself and in order to help the user recover the key, the image owner preprocess the key using the secret sharing techniques. And this also reduces the communication overhead between the users and the image owners. In [5], the server sends encrypted retrieval results to the image owner, and then image owner returns to the user the decrypted image. However, in our scheme, the users have no need to communicate with the image owner.

VII. PERFORMANCE EVALUATION

We use Python to implement the prototype of our scheme. We deploy the prototype on a PC with Intel Core i7 3.4 GHz CPU and 8 GB RAM. We use the computer vision library Opencv3 to process images and the OpenSSL library to achieve symmetric encryption and PRFs. The feature extraction algorithm is implemented using Oriented FAST and Rotated BRIEF (ORB) [11]. The extended locality-sensitive hashing is implemented using random bits sampling based *eLSH*. We conduct our experiments on the UK Bench database [27] to test retrieval accuracy and system performance. Each group in the UK Bench database contains 4 images. The images in the same group capture the same object under different angles and illuminations. Therefore we consider images in a group as similar images. From the database, we randomly choose 3,000 images to build the secure index. We evaluate our system from the following aspects: image processing cost, query overhead, and retrieval accuracy. And all of the experimental results are the average of 100 operations.

A. IMAGE PROCESSING COST

The parameters of the experiments are $n = 120$, $t = 10$, *eLSH* arguements $(k, L) = (30, 80)$, and we test the computation cost of uploading one image. The experimental results show that in the process of image processing, it takes $0.003s$ to share the secret key, $0.056s$ to generate profile vectors, $0.081s$ to generate tags and encrypted pieces.

B. QUERY COST

Whether a query is for retrieval, insertion, or deletion, the bandwidth is constant with respect to $n \times L$ tags in *Tag*. The parameter of the experiments are $l \times w = 30$ million, probe value $d = 3$, $n = 120$, $(k, L) = (30, 80)$. We test tag insertion cost for uploading 1 image, tag deletion cost for removing 1 image, and retrieval cost for querying 1 image respectively.

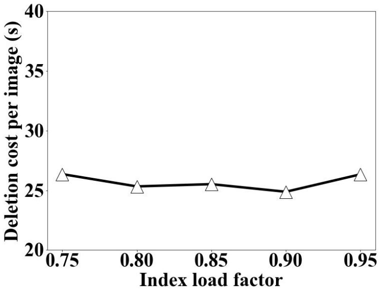


FIGURE 4. Deletion cost.

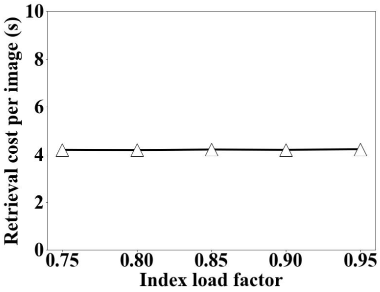


FIGURE 5. Retrieval cost.

- Insertion cost. Tags insertion may cause collisions in cuckoo hashing and further incur kick-away operations. As the index load factor τ grows, cuckoo kick-away operations occur more frequently which results in the growth of time cost, as shown in Fig.3.
- Deletion cost. After receiving the deletion query from the user, the SS generates $n \times L$ trapdoors, accesses to $n \times L \times (d+1)$ buckets, and finally empties $n \times L$ buckets related to the query. As shown in Fig.4, the deletion cost is nearly constant under different τ.
- Retrieval cost. Same as the tags deletion, after receiving the retrieval query, the SS generates $n \times L$ trapdoors. Then SS accesses $n \times L \times (d + 1)$ buckets in the cuckoo hashing, and filters contents in these buckets. As shown in Fig.5, the time cost under different τ is constant.

C. RETRIEVAL ACCURACY

Retrieval accuracy is affected by threshold value m and *eLSH* arguments (k, L). From Fig. 6, 7, 8, and 9, we could observe the influence of threshold value on the retrieval accuracy and the retrieval recall in our scheme. The growth of the threshold value causes the standard for measuring image similarity to become stricter, and leads to better retrieval accuracy and lower retrieval recall. Fig. 6 and 7 demonstrate the impact of L on retrieval accuracy and retrieval recall of our scheme, while Fig. 8 and 9 demonstrate the impact of k. The growth of L increases the retrieval recall and reduce the retrieval accuracy, while the increase of k leads to the increase of retrieval accuracy and the decrease of retrieval recall. The impact

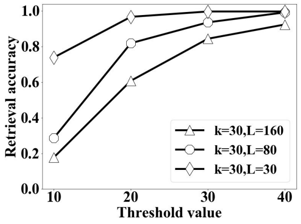


FIGURE 6. Accuracy for different *L*.

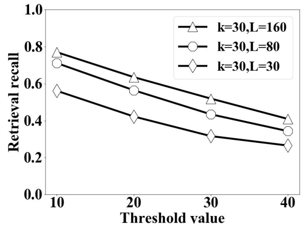


FIGURE 7. Recall for different *L*.

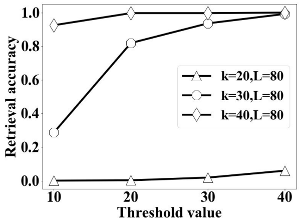


FIGURE 8. Accuracy for different *k*.

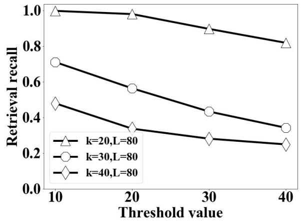


FIGURE 9. Recall for different *k*.

results of L and k are consistent with the characteristics of *eLSH* described in section III-A.

VIII. CONCLUSION

In this paper, we proposed and implemented a privacy-preserving content-based image retrieval and sharing scheme, which can be used for friend recommendation in social multimedia applications. We measured image similarity through image visual features. We used *eLSH* to reduce the dimensionality of visual features and realize similarity search on visual features. We designed the index based on cuckoo hashing to speed up the similarity search. Based on secret sharing, we allowed the user to query and recover images on his own, which eliminates key management overhead and access control overhead compared with other schemes. Finally, we implemented a prototype to evaluate the efficiency of our proposed scheme. The results showed that our scheme achieves practical performance under the UK Bench database.

REFERENCES

[1] C. West. (2019). *17 Instagram Stats Marketers Need to Know For 2019*. [Online]. Available: https://sproutsocial.com/insights/instagram-stats/

[2] Z. Xia, Y. Zhu, X. Sun, Z. Qin, and K. Ren, "Towards privacy-preserving content-based image retrieval in cloud computing," *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 276–286, Jan. 2018.

[3] Y. Xu, J. Gong, L. Xiong, Z. Xu, J. Wang, and Y.-Q. Shi, "A privacy-preserving content-based image retrieval method in cloud environment," *J. Vis. Commun. Image Represent.*, vol. 43, pp. 164–172, 2017.

[4] J. Yuan, S. Yu, and L. Guo, "SEISA: Secure and efficient encrypted image search with access control," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 2083–2091.

[5] Q. Zou, J. Wang, J. Ye, J. Shen, and X. Chen, "Efficient and secure encrypted image search in mobile cloud computing," *Soft Comput.*, vol. 21, no. 11, pp. 2959–2969, Jun. 2017.

[6] Y. Wang, M. Miao, J. Shen, and J. Wang, "Towards efficient privacy-preserving encrypted image search in cloud computing," *Soft Comput.*, vol. 23, no. 6, pp. 2101–2112, Mar. 2019.

[7] M. Li, M. Zhang, Q. Wang, S. S. M. Chow, M. Du, Y. Chen, and C. Lit, "InstantCryptoGram: Secure image retrieval service," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 2222–2230.

[8] U. Sharif, Z. Mehmood, T. Mahmood, M. A. Javid, A. Rehman, and T. Saba, "Scene analysis and search using local features and support vector machine for effective content-based image retrieval," *Artif. Intell. Rev.*, vol. 52, no. 2, pp. 901–925, Aug. 2019.

[9] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[10] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Comput. Vis. image Understand.*, vol. 110, no. 3, pp. 346–359, 2008.

[11] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Proc. Int. Conf. Comput. Vis.*, Nov. 2011, pp. 2564–2571.

[12] D. Xiaoding Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy.*, May 2000, pp. 44–55.

[13] X. Yuan, X. Wang, C. Wang, A. C. Squicciarini, and K. Ren, "Towards privacy-preserving and practical image-centric social discovery," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 5, pp. 868–882, Sep. 2018.

[14] K. W. Ahmed, M. Z. Hasan, and N. Mohammed, "Image-centric social discovery using neural network under anonymity constraint," in *Proc. IEEE Int. Conf. Cloud Eng. (ICE)*, Apr. 2017, pp. 238–244.

[15] Facebook. (2019). *Keeping Passwords Secure*. [Online]. Available: https://newsroom.fb.com/news/2019/03/keeping-passwords-secure/

[16] W. Pieters and Q. Tang, "Data is key: Introducing the data-based access control paradigm," in *Proc. IFIP Annu. Conf. Data Appl. Secur. Privacy*. Berlin, Germany: Springer, 2009, pp. 240–251.

[17] A. Boldyreva, T. Tang, and B. Warinschi, "Masking fuzzy-searchable public databases," in *Proc. Int. Conf. Appl. Cryptography Netw. Secur.* Cham, Switzerland: Springer, 2019, pp. 571–591.

[18] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. 13th Annu. ACM Symp. Theory Comput. (STOC)*, 1998, pp. 604–613.

[19] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li, "Modeling LSH for performance tuning," in *Proc. 17th ACM Conf. Inf. Knowl. Mining (CIKM)*, 2008, pp. 669–678.

[20] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe ISN: Efficient indexing for high-dimensional similarity search," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 950–961.

[21] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," *Proc. VLDB*, vol. 99, no. 6, pp. 518–529, 1999.

[22] J. Wang, H. Tao Shen, J. Song, and J. Ji, "Hashing for similarity search: A survey," 2014, *arXiv:1408.2927*. [Online]. Available: http://arxiv.org/abs/1408.2927

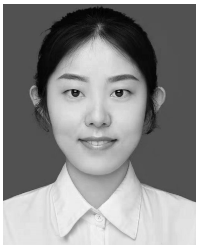
[23] R. Pagh and F. F. Rodler, "Cuckoo hashing," in *Proc. Eur. Symp. Algorithms*. Berlin, Germany: Springer, 2001, pp. 121–133.

[24] Y. Sun, Y. Hua, D. Feng, L. Yang, P. Zuo, S. Cao, and Y. Guo, "A collision-mitigation cuckoo hashing scheme for large-scale storage systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 619–632, Mar. 2017.

[25] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[26] A. Boldyreva and N. Chenette, “Efficient fuzzy search on encrypted data,” in *Proc. Int. Workshop Fast Softw. Encryption*. Berlin, Germany: Springer, 2014, pp. 613–633.

[27] D. Nister and H. Stewenius, “Scalable recognition with a vocabulary tree,” in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, vol. 2, Jun. 2006, pp. 2161–2168.

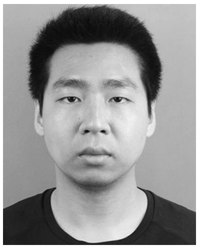


ZONGYE ZHANG received the B.S. degree in information security and the M.S. degree in software engineering from Northeastern University, in 2016 and 2018, respectively, where she is currently pursuing the Ph.D. degree with the Software College. Her research interests include searchable encryption and outsourced database.

FUCAI ZHOU received the Ph.D. degree in computer software and theory from Northeastern University, China, in 2001. He is currently a Professor and the Ph.D. Supervisor with the Software College, Northeastern University. His research interests include cryptography, network security, trusted computing, secure cloud storage, software security, basic theory, and critical technology in electronic commerce.

SHIYUE QIN received the B.S. degree in information security form Northeastern University, China, in 2016, where she is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering. Her research interests include searchable encryption and data privacy.

QIANG JIA received the B.S. and M.S. degrees in software engineering from Liaoning Shihua University, China, in 2014 and 2017, respectively. He is currently pursuing the Ph.D. degree with the Software College, Northeastern University. His research interests include searchable encryption and outsourced database.



ZIFENG XU received the B.S. degree in computer science and the M.E. degree in computer science from Bristol University, U.K., in 2011 and 2012, respectively. He is currently pursuing the Ph.D. degree with the Software College, Northeastern University, China. His research interests include verifiable computation, multiparty computation, and searchable encryption.