InstantCryptoGram: Secure Image Retrieval Service

Minghui Li[†‡], Mingxue Zhang[†], Qian Wang[†‡],
Sherman S. M. Chow[§¶], Minxin Du[†], Yanjiao Chen[†] and Chenliang Li[†]
[†]School of Cyber Science and Engineering, and School of Computer Science, Wuhan University, P. R. China
[‡]Collaborative Innovation Center of Geospatial Technology, P. R. China
[§]Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong
[¶]Institute of Theoretical Computer Science and Communications, The Chinese University of Hong Kong, Hong Kong

***Abstract*—Image retrieval is crucial for social media sites such as Instagram to identify similar images and make recommendations for users who share similar interests. To get rid of the storage burden and computation for image retrieval, outsourcing to a remote cloud is now a trend. Yet, privacy concerns mandate the use of encryption before outsourcing the images. We need a secure way for retrieving images from a not-fully-trusted server.**

This paper proposes InstantCryptoGram, a secure image retrieval service. We first design a new data structure called sub-simhash, which fits for the inverted index used by many searchable symmetric encryption schemes. It leads to our modular solution that supports efficient similarity queries and updates over encrypted images. Our experiments on Amazon AWS EC2 over representative datasets show that our scheme is efficient and accurate in finding similar images while preserving privacy.

I. Introduction

With the vast usage of mobile devices, an avalanche of images is continuously generated every day. This phenomenon gives birth to numerous photo-sharing sites such as Instagram and Flickr. Various social networks (Facebook, Tumblr, *etc.*) are also becoming photo-centric which allow users to upload and share photos with friends/followers. Users are generally interested in photos which share similar contents with what they posted, like photos taken at the same place/event, or with similar kinds of food/pets, as also shown by the popularity of hashtags for annotating images. Two users who share similar interests might like to reach out to each other. Image retrieval is thus widely used to recommend similar images [1].

Albeit useful, image retrieval can be very time-consuming for popular service providers who bear millions of images uploaded every day. A popular trend now is to outsource the image retrieval task to a commercial cloud server (*e.g.*, Amazon EC2) to get rid of tremendous local storage and computation burdens, which ultimately benefits the users.

While users might have accepted that their often-sensitive images are revealed to the service providers for utilizing their services, users are not willing to extend their trusts to the storage server at the back-end, which is not much different from a public cloud from the user perspective. The service provider can encrypt the images before outsourcing. Yet, how to enable efficient image retrieval over a large-scale encrypted image collection remains a big challenge.

A. Related Work

The information retrieval community has devoted great efforts to enable efficient image search (*e.g.*, [2]), but privacy is rarely taken into consideration. Huang *et al.* [3] developed a biometric identification system over encrypted fingerprint, yet its search complexity is linear in the database size. At another extreme, the image retrieval scheme of Weng *et al.* [4] simply uses deterministic hash functions which is efficient. The price is that the privacy of the images solely relies on the *one-wayness* of the hash functions and the omissions of parts of the hashes. The authors argue empirically that recovery is difficult. Yet, it is not a provable security guarantee. Li *et al.* [5] made use of linear transformation methods (*e.g.*, matrix multiplications) to secure image features, which suffer from high computation complexity (at least $O(n^{2+})$ for multiplying two n-dimensional matrices). Ferreira *et al.* [6] proposed to deterministically encrypt the pixels and randomly permute their positions [7]. One can thus compare the HSV histograms of the encrypted images without a search trapdoor.

Cryptographers have proposed searchable symmetric encryption (SSE) for supporting sub-linear search over encrypted data. Later schemes are *dynamic* which support addition and removal of files [8]. Recent research [9] shows that additional features such as forward security can be generically obtained. Nevertheless, the focus has been on *exact* searches over text files. Structured encryption [7], [9], [10] extended SSE to cover data structure in general, yet the concrete instantiations just work with some basic structures such as matrix and graph.

Similarity encrypted search is generally done by using less-efficient asymmetric primitive (*e.g.*, pairing, which can be outsourced in batch though [11]) to operate on each possibility, thus does not scale for large image databases. Zhang *et al.* [12] presented an image search system requiring a trusted party which generates a lot of user-specific secret keys for two servers. It works by first creating a cluster representative for the images uploaded by each user, then relying on homomorphic encryption to support the computation of Euclidean distances between an encrypted query and the ciphertexts of every representatives from each user. Notably, its correct functionality crucially relies on a special kind of "key-transformation" of a homomorphic encryption scheme[1]. Yet, the scheme was broken in 2014 [13], namely, a decryption key can be recovered by $O(1)$ arithmetic operations with $O(1)$ of plaintext-ciphertext pairs. This attack works

[1]It is suggested for "applications where semantic security is not required and one-wayness security is sufficient." See http://ia.cr/2012/193. Citation is omitted for minimizing references to papers with/relying on insecure result(s).

978-1-5386-4128-6/18/$31.00 ©2018 IEEE

even without accessing a transformation oracle (but never formally analyzed, *cf.*, related-key attacks), which nullifies the performance demonstrated by their experiment [12]. That said, one can resort to additive homomorphic encryption such as Paillier [14] to compute the square of the Euclidean distance. To minimize the leakage of the distances, Elmehdwi *et al.* [15] proposed using additional protocols for securely computing minimum and bit-decomposition to derive a list of nearest neighbors. Wang *et al.* [16] designed an SSE for feature-rich data (such as images) via the use of locality-sensitive hashes (LSH). One of their schemes does not rely on (additive) homomorphic encryption. However, their schemes suffer from linear (in the dataset size) search complexity. Cui *et al.* [17] also designed an LSH-based scheme. Yet, building index is expensive, namely, the number of atomic operations, such as encryption of the image features is linear in the product of the number of files, the dimension of feature vector, and the number of times LSH is applied (which is correlated with the accuracy). Even the search time is sublinear in the number of files, it is still inefficient due to the way the index is built. Specifically, for each entry of the feature vector, it needs to probe and retrieve a number of indices, depending on how many LSH'es are used. Practically, it requires thousands of atomic operations for a single retrieval query. (See Section V.)

B. Our Contributions

We design an image retrieval service that supports similarity search and updates over encrypted data based on "*sub-simhash*", which "divide-and-conquer" [18] over simhash [19]. The novelty of our sub-simhash-based design is that it is a *general* data structure fitting perfectly with any encrypted (inverted) index structure [8], [9]. We can then leverage SSE for pre-indexing before performing similarity search in a much smaller scale. Note that we are not proposing a new SSE scheme, nor a "fully-secure" solution which the users encrypt their images under their own (public) keys before uploading.[2]

Our scheme also fits well with the prevalent model (*e.g.*, [15], [20]) of two-servers. Expensive tasks including building the searchable encrypted index, image retrieval, and image update can be delegated to the cloud. Using SIFT [21] and simhash, the service provider just needs to perform the basic jobs of processing individual image, namely, receiving (and subsequently encrypting) a user-uploaded image and returning (and subsequently decrypting) relevant images for the users with matching interests. Both are needed for any service provider who needs to ensure the images confidentiality.

Finally, we performed experiments over representative real-world datasets on Amazon AWS EC2. Our results demonstrate the great efficiency improvements over an existing scheme for mobile image sharing [17], without sacrificing accuracy.

II. Preliminaries and Problem Formulation

We present technical preliminaries for the rest of our papers and formulates our secure image retrieval problem.

[2]The latter probably requires some trust assumptions and managements of the private keys, and heavyweight tools such as fully homomorphic encryption.

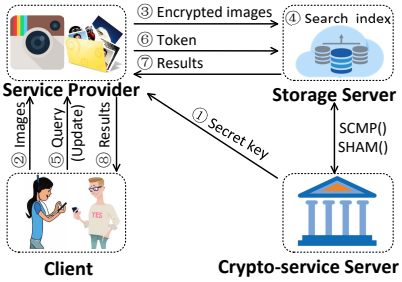


Fig. 1: System architecture of InstantCryptoGram

A. SimHash

Simhash is a hashing technique which not only aims to produce a unique signature of the input (like cryptographic hash functions), but also ensures that "similar inputs" lead to similar outputs (as binary strings) after applying simhash. (We also call its output as simhash when there is no ambiguity.) In other words, when applied on images, its main characteristic is that similar images differ in a small number of bit positions [19]. Specifically, for an f-dimensional feature vector extracted from the image, it generates its corresponding simhash by comparing each element value of the feature vector with the averaged element value of the vector. The i-th entry of the simhash is set as 1 if the i-th element value is larger than the averaged element value; 0 otherwise.

B. Notations

Given a vector $\boldsymbol{v}$, we refer to the i-th element as $\boldsymbol{v}_i$ and to its total number of elements as $\#\boldsymbol{v}$. Let the original image set be $\mathbf{g} = \{g_1, g_2, \ldots, g_{\#\mathbf{g}}\}$ and the corresponding extracted feature vector set be $\mathbf{b} = \{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_{\#\mathbf{g}}\}$, where $\boldsymbol{b}_i = (b_{i,1}, b_{i,2}, \ldots, b_{i,f})$ is an f-dimensional feature vector of image g_i by the widely used SIFT descriptors [21]. The simhashes derived from the feature vectors are denoted by $\mathbf{B} = \{\mathsf{B}_1, \mathsf{B}_2, \ldots, \mathsf{B}_{\#\mathbf{g}}\}$. Each B_i is a bit-vector (while each component of the feature $b_{i,j}$ is not a bit). An encrypted form of message x is denoted by $[\![x]\!]_{pk}$, where pk is the encryption key. Encrypted features are denoted by $\mathbf{c} = \{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_{\#\mathbf{g}}\}$ where $\boldsymbol{c}_i$ is a ciphertext of $\boldsymbol{b}_i$, *i.e.*, $c_{i,j} = [\![\boldsymbol{b}_{i,j}]\!]_{pk}$. Finally, we use $\|$ to denote string concatenation and $\oplus$ to denote XOR.

C. System Model

Our system model consists of four main kinds of entities. A mobile client $\mathcal{U}$ is a registered user who uploads images and searches for relevant images from other clients. Here we consider relevancy based on the features of the images.[3] The image gallery service provider $\mathcal{IG}$ (*e.g.*, Instagram) provides photo related services to $\mathcal{U}$. The storage server $\mathcal{SS}$ is a cloud server employed by $\mathcal{IG}$ which stores the encrypted images and encrypted index for $\mathcal{IG}$. It thus entertains requests for image retrieval and updates from $\mathcal{U}$. The crypto-service server $\mathcal{CS}$ is

[3]Suggestion based on contexts not related to the image features, say, people visited St. Paul's Cathedral, Mdina may also be interested in the love statue in St. Julians, can be supported by techniques such as collaborative filtering.

another server employed by $\mathcal{IG}$, which is responsible for key management and providing cryptographic services.

Figure 1 illustrates a typical workflow. Without privacy, the system only consists of $\mathcal{U}$ and $\mathcal{IG}$. $\mathcal{IG}$ builds the search index itself based on the images data uploaded from $\mathcal{U}$, and probes the index to find the similar images when $\mathcal{U}$ is making a query. When considering privacy, the workflow involves all four entities as follows. In the initialization phase, $\mathcal{CS}$ generates and distributes the public/private keys to $\mathcal{IG}$. Either $\mathcal{U}$ or $\mathcal{IG}$ can encrypt the user-images which will be uploaded to $\mathcal{SS}$.

Our system supports secure image retrieval via the following workflow. $\mathcal{SS}$ cooperates with $\mathcal{CS}$ to build the encrypted search index ψ. When $\mathcal{U}$ issues a query for similar images, it first uploads the query image based on which $\mathcal{IG}$ generates the search token τ_q. $\mathcal{SS}$ then uses τ_q to probe the search index ψ. Finally, $\mathcal{SS}$ returns to $\mathcal{IG}$ (and $\mathcal{U}$) the identifier (ID) set corresponding to the similar images. For adding or deleting an image, $\mathcal{CS}$ sends an update (*i.e.*, add or delete) token τ_u to $\mathcal{SS}$, who will generate the updated index ψ'.

D. Discussion on the Two-Server Model / Trust Assumption

Our goal is to protect the confidentiality of the original images and the corresponding features vectors from the storage server $\mathcal{SS}$[4]. For practical efficiency, we leverage the non-colluding assumption (*e.g.* [17], [20]). It requires that the two servers ($\mathcal{SS}$ and $\mathcal{CS}$ in our case) are not colluding with each other. Yet, they are curious to infer as much private information as possible. We assume they dutifully execute the pre-defined protocols. We also assume that $\mathcal{IG}$ can see the images in clear. Note that most photo services post-processes images uploaded by users. Without these assumptions, we probably need heavy-weight tools such as public-key fully homomorphic encryption plus insider-secure multi-key homomorphic signatures [22] or delegation using indistinguishability obfuscation (*e.g.*, [23]).

From the functionality perspective, our $\mathcal{CS}$ helps in computing simhashes over encrypted feature vectors and subsequently adding them to the encrypted index. These computations appear to be inherent for security. $\mathcal{CS}$ can be a fortified internal host without high computational power and huge storage. In particular, the storage and the searching are delegated to $\mathcal{SS}$.

E. Cryptographic Details

1) Encrypted Image Retrieval: An encrypted image retrieval system consists of the following algorithms/protocols:

- $K \leftarrow \mathsf{Gen}(1^\lambda)$: is a key generation algorithm run by $\mathcal{CS}$, which takes as input a security parameter λ and outputs the key set $K = \{k_\mathsf{d}, k_\mathsf{sse}\}$, where k_d and k_sse are for data encryption and SSE respectively.
- $\mathbf{c} \leftarrow \mathsf{EncFeat}(k_\mathsf{d}, \mathbf{b})$: is an algorithm run by $\mathcal{IG}$. It uses the key k_d to encrypt the feature vectors $\mathbf{b}$ extracted from the images, and outputs the ciphertexts $\mathbf{c}$.
- $\{\mathbf{B}; \perp\} \leftarrow \mathsf{SimhashComp}(\mathbf{c}; k_\mathsf{d})$: $\mathcal{SS}$ and $\mathcal{CS}$ runs a protocol[5] with the ciphertexts $\mathbf{c}$ and the key k_d as their respective inputs. $\mathcal{SS}$ obtains simhashes $\mathbf{B}$.
- $\mathbf{S} \leftarrow \mathsf{SimhashDictBuild}(\mathbf{B})$: Given the simhashes, $\mathcal{SS}$ outputs the sub-simhash dictionary $\mathbf{S} = \{S^v\}_{v=1}^{div}$.
- $\psi \leftarrow \mathsf{IndexBuild}(\mathbf{S}; k_\mathsf{sse})$: Given the sub-simhash dictionary $\mathbf{S}$ from $\mathcal{SS}$ and the key k_sse stored by $\mathcal{CS}$. $\mathcal{CS}$ outputs the encrypted search index ψ to be stored by $\mathcal{SS}$.
- $(\perp; \mathsf{ID}_q) \leftarrow \mathsf{Search}(K, g_q; \psi)$: is a protocol between $\mathcal{IG}$ and $\mathcal{SS}$. $\mathcal{IG}$ takes as input the key set K and the query image g_q, while $\mathcal{SS}$ takes as input the index ψ. It outputs ID_q, the ID set of similar images.
- $(\perp; \psi') \leftarrow \mathsf{Update}(K, g_u; \psi)$: is an algorithm run between $\mathcal{IG}$ and $\mathcal{SS}$. $\mathcal{IG}$ takes as input the key set K and the update image g_u, while $\mathcal{SS}$ takes as input the index ψ. As a result, $\mathcal{SS}$ obtains an updated index ψ'.

2) Paillier Encryption: The Paillier encryption [14] is a well-known public-key cryptosystem due to its additively homomorphic property, namely, for any two messages x_1 and x_2, $[\![x_1]\!]_{pk} \cdot [\![x_2]\!]_{pk} = [\![x_1 + x_2]\!]_{pk}$ and $([\![x_1]\!]_{pk})^{x_2} = [\![x_1 \cdot x_2]\!]_{pk}$.

3) Secure Computation Protocols: Secure two-party computation protocols are well studied by cryptographers [24], [3], [25], [26]. Below describes the interfaces of two such protocols used by our scheme, with details deferred to the appendix.

Secure Comparison Protocol (SCMP): $(x; \perp) \leftarrow \mathsf{SCMP}(([\![a_1]\!]_{pk}, [\![a_2]\!]_{pk}); sk)$ is run between $\mathcal{SS}$ and $\mathcal{CS}$, where $\mathcal{SS}$ holds two encrypted values $[\![a_1]\!]_{pk}$ and $[\![a_2]\!]_{pk}$ under Paillier encryption, and $\mathcal{CS}$ has the corresponding secret key sk. It outputs a bit indicating if $a_1 > a_2$. Looking ahead, we use SCMP to compute the simhash of each image.

Secure Hamming Distance Protocol (SHAM): $(x; \perp) \leftarrow \mathsf{SHAM}(([\![a_1]\!], [\![a_2]\!]); sk)$ is run between $\mathcal{SS}$ and $\mathcal{CS}$. x is the Hamming distance between two strings a_1 and a_2, *i.e.*, the number of positions at which the corresponding bits differ.

III. Our Secure Modular Construction

Locating relevant parts of the database without missing too many results is the key idea of our system.

A. Overview

We transform the feature vector of an image into a simhash. Similarity is defined by having Hamming distance from the query smaller than a given threshold. To allow seamless integration with any inverted-index-based SSE scheme, we design a sub-simhash data structure. It divides each simhash into several parts, where each is populated to different indices.

In more details, for each part, the images that share the same sub-simhash are linked together as a linked list (which can be replaced by other data structure [9]). Similarity searches are done by probing different sub-simhashes of the query image. In other words, image retrieval probes the inverted index for possible matches, starting with the first sub-simhash of the query image, and repeats the probing for its subsequent sub-simhashes. This will cover a large portion of similar images.

B. Basic Design without Privacy Guarantee

The basic construction without privacy guarantees is run between $\mathcal{U}$ and $\mathcal{IG}$, which consists of four main phases.

[4]As most SSE literatures, we do not consider private information retrieval and assume $\mathcal{SS}$ can learn the image identifier (ID) in clear.

[5]$(x; y) \leftarrow P(u; v)$ denotes that parties $\mathcal{A}$ and $\mathcal{B}$ run the protocol P with u (or x) and v (or y) being $\mathcal{A}$'s and $\mathcal{B}$'s inputs (or outputs) correspondingly.

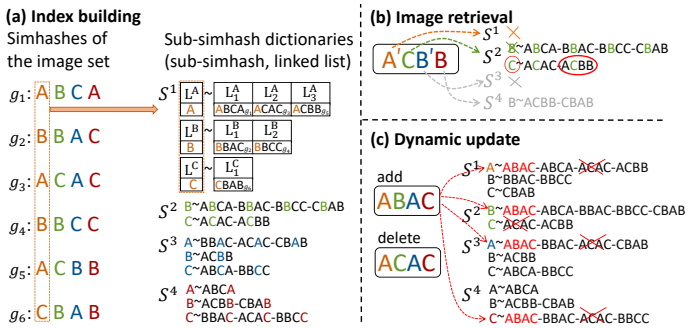


Fig. 2: Example of the basic construction

1) Computing Simhash: $\mathcal{IG}$ first extracts the feature vector $\boldsymbol{b}_i$ of each image g_i exploiting the SIFT method, and then computes the simhash B_i based on the feature vector. Finally, it can obtain all the simhashes $\mathbf{B} = \langle \mathsf{B}_i \rangle_{i=1}^{\#\mathbf{g}}$ of images $\langle g_i \rangle_{i=1}^{\#\mathbf{g}}$.

2) Building Sub-Simhash Dictionary: We exploit "divide-and-conquer" [18] over simhash [19] to build sub-simhash. $\mathcal{IG}$ divides each simhash into sub-simhash and builds the dictionaries accordingly. Specifically, the f-bit simhash B_i is divided into div parts, where each part in $\langle \mathsf{B}_i^v \rangle_{v=1}^{div}$ is a $\lceil \frac{f}{div} \rceil$-bit simhash named as sub-simhash. (If the last part is shorter than $\lceil \frac{f}{div} \rceil$ bits, $\mathcal{IG}$ can append a number of dummy bits.) div is an important parameter which manifests itself in various occasions. For the v-th part, $\mathcal{IG}$ builds the dictionary S^v that stores $(tag, value)$ pairs, where the tag is the v-th sub-simhash selected from $\langle \mathsf{B}_i^v \rangle_{i=1}^{\#\mathbf{g}}$, and $value$ is a linked list L^{tag} which stores the images sharing same sub-simhash in their v-th part.

3) Image Retrieval: $\mathcal{IG}$ probes div dictionaries one after another for the linked list(s) whose tag is equal to the corresponding sub-simhash $\langle \mathsf{B}_q^v \rangle_{v=1}^{div}$ of the query image g_q. For each node of the matched linked list, $\mathcal{IG}$ figures out whether it is similar to the query by comparing the Hamming distance with the threshold z. It is crucial to set z smaller than div to ensure that the query has at least one identical part with the similar images. In other words, if the sub-simhash of the query cannot be found, the Hamming distance between the query and all the images must be larger than the threshold.

4) Dynamic Update: When $\mathcal{U}$ wants to add an image g_u with simhash B_u to the dataset stored on $\mathcal{IG}$, $\mathcal{IG}$ first locates among all the div sub-simhash dictionaries the linked list whose tag is in $\langle \mathsf{B}_u^v \rangle_{v=1}^{div}$, and then inserts B_u to the matched linked list as a new head node. For deleting an existing image g_u, $\mathcal{IG}$ first locates the matched linked lists based $\langle \mathsf{B}_u^v \rangle_{v=1}^{div}$, and deletes the node corresponding to g_u from them (with appropriate housekeeping for linked lists).

Illustrative Example. Figure 2 illustrates how our basic design works by a toy example for image set $\langle g_i \rangle_{i=1}^{6}$. Here, we assume that the simhash is divided into 4 parts (*i.e.*, $div = 4$) and the threshold is set to $z = 3$ (which is less than div). For brevity, the f-bit simhash is denoted by the concatenation of 4 capital letters, where different letters refer to different $\lceil \frac{f}{4} \rceil$-bit sub-simhashes. Take the first part which marked in orange for example, g_1, g_3, and g_5 share the same sub-simhash A but they may have a different second part. Therefore, A is set as the tag and the corresponding $value$ is the linked list that links ABCA, ACAC, and ACBB together since they share the same first part, as shown in Fig. 2 (a). Analogously, based on these sub-simhashes, $\mathcal{IG}$ builds other $(tag, value)$ pairs, which are the components of the $div = 4$ simhash dictionaries $\langle S^v \rangle_{k=1}^{4}$.

Fig. 2 (b) gives an illustration of querying an image with simhash A′CB′B where A′ (B′) has 1 bit of difference from A (B). $\mathcal{IG}$ first probes the dictionary S^1 for the first part of the query with $tag = \mathsf{A}'$. As S^1 contains no such tag, $\mathcal{IG}$ probes S^2 for the second part of the query. The matched linked list tagged with C is L^C. (It is the second row in S^2. For brevity, the notation L^C is omitted from S^2 in the figure.) Then $\mathcal{IG}$ traverses each node L_j^C $(j \in (1, 2))$ of L^C to determine whether the Hamming distance between the node and the submitted query A′CB′B is smaller than z. It is clear that the Hamming distance between A′CB′B and ACAC is larger than the threshold 3, but that between A′CB′B and ACBB, which is 2, is smaller than the threshold. Therefore, the corresponding image whose simhash is ACBB is deemed as a similar image.

After the first dictionary with an identical sub-simhash is identified, we can choose to traverse other dictionaries. This possibly leads to either duplicate or additional search results. For example, the fourth sub-simhash of the query, B, also presents in dictionary S^4. If we traverse S^4, we will get ACBB which already presents in the result from traversing S^2. Traversing only one dictionary helps us to achieve high efficiency, while traversing more can improve the accuracy.

Setting threshold $z < div$ implies that an identical sub-simhash must exist. Basing on identical sub-simhashes may miss some similar results with differing bits scattered around different parts of the simhash. Since simhash is based on average value of features, such cases are rare. Section V will show by experiments that we achieve a favorable accuracy. This echoes with the accuracy of privacy-preserving collaborative filtering by applying "divide-and-conquer" over LSH [27].

Dynamic Updates. Adding and deleting an image are illustrated in Fig. 2 (c). Here we consider adding an image with ABAC as its simhash and deleting an image corresponding to ACAC. When adding ABAC, for the first part A, $\mathcal{IG}$ locates the linked list L^A in the dictionary S^1 and then inserts it as the head of the linked list. Analogously, this simhash is also inserted into the head of the matched linked lists in other dictionaries (*i.e.*, L^B in S^2, L^A in S^3, and L^C in S^4).

When deleting ACAC, based on its sub-simhash, $\mathcal{IG}$ locates the matched linked list in each dictionary. Then the node ACAC is deleted, and its adjacency nodes are connected.

C. A Generic Construction for Image Retrieval with Privacy

In the following we show how to integrate the above basic construction with an arbitrary inverted-index-based SSE.

1) Initialization: $\mathsf{Gen}(1^\lambda)$: Given a security parameter λ, $\mathcal{CS}$ generates the following keys uniformly at random: $k_\mathsf{d} = (pk, sk)$ for Paillier encryption and k_sse for the SSE scheme. The output is $K = \{k_\mathsf{d}, k_\mathsf{sse}\}$ which is sent to $\mathcal{IG}$ securely.

$\mathsf{EncFeat}(k_{\mathsf{d}}, \mathbf{b})$: For each feature vector $\boldsymbol{b}_i$ in $\mathbf{b}$, $\mathcal{IG}$ conducts Paillier encryption element-wise to obtain $\boldsymbol{c}_i$, where $c_{i,j} = [\![b_{i,j}]\!]_{pk}$. The encrypted features set $\mathbf{c}$ is sent to $\mathcal{SS}$.

2) Building Index: $\mathsf{SimhashComp}(\mathbf{c}; k_{\mathsf{d}})$: $\mathcal{SS}$, holding $\mathbf{c} = \langle \boldsymbol{c}_i \rangle_{i=1}^{\#\mathbf{g}}$, cooperatively computes the simhash of each image with $\mathcal{CS}$ who holds the decryption key k_{d}. To avoid computation of the average which involves division by f, $\mathcal{SS}$ instead conducts element-wise multiplications to have $\langle [\![f \cdot b_{i,j}]\!]_{pk} \rangle_{j=1}^{f}$, and homomorphically sums up the f elements to obtain $[\![bs_i]\!]_{pk} = [\![\Sigma_{j=1}^{f} b_{i,j}]\!]_{pk}$. Then, $\mathcal{SS}$ and $\mathcal{CS}$ jointly execute the SCMP protocol (in Section II-E) to compare between $[\![f \cdot b_{i,j}]\!]_{pk}$ and $[\![bs_i]\!]_{pk}$. The output of SCMP determines the j-th bit of simhash B_i. By repeating the above operations for all i, $\mathcal{SS}$ obtains simhashes $\mathbf{B} = \langle \mathsf{B}_i \rangle_{i=1}^{\#\mathbf{g}}$ of all the images[6].

$\mathsf{SimhashDictBuild}(\mathbf{B})$: $\mathcal{SS}$ builds the sub-simhash dictionaries $\mathbf{S}$ from $\mathbf{B}$ as in Section III-B2 and then sends $\mathbf{S}$ to $\mathcal{CS}$.

$\mathsf{IndexBuild}(\mathbf{S}; k_{\mathsf{sse}})$: $\mathcal{CS}$ uses the underlying SSE to build an encrypted search index ψ of $\mathbf{S} = \langle S^v \rangle_{v=1}^{div}$ with key k_{sse}. The inverted index of "keyword-files" in the classical setting of SSE exactly corresponds to our sub-simhash structure. To be specific, each tag, which is a sub-simhash for different parts of the image simhash, will be used as the "keyword". The list of files associated with a "keyword" will store all simhashes sharing the same part of sub-simhash. Simply put, any SSE can directly build the encrypted index ψ based on $\mathbf{S}$.

3) Image Retrieval: For an image query g_q from $\mathcal{U}$, $\mathcal{U}$ first uploads g_q to $\mathcal{IG}$. $\mathcal{IG}$ then cooperates with $\mathcal{SS}$ to run $\mathsf{Search}(K, g_q; \psi)$ to find the similar images:

For each sub-simhash of g_q, $\mathcal{IG}$ generates the corresponding search tokens from the underlying SSE. Note that it is easy to differentiate between different parts by appending the part-number (*e.g.*, sub-simhash-"Part"-1). $\mathcal{IG}$ then submits the list of tokens from the underlying SSE as a single token τ_q to $\mathcal{SS}$.

Using τ_q, $\mathcal{SS}$ probes ψ to obtain candidate images with a sub-simhash part identical with the corresponding part of g_q. $\mathcal{SS}$ can then determine if an image is similar by evaluating its Hamming distance from g_q and checking if it is smaller than z, the given threshold[7]. Lastly, $\mathsf{ID}_q = \langle \mathsf{id}_i^q \rangle_{i=1}^{\#\mathsf{ID}_q}$ is returned to $\mathcal{U}$, where id_i^q is the identifier of the i-th similar image.

4) Dynamic Updates: When adding a new image or deleting an existing image g_u, $\mathcal{U}$ submits g_u to $\mathcal{IG}$. $\mathcal{IG}$ then cooperates with $\mathcal{SS}$ to run $\mathsf{Update}(K, g_u; \psi)$: $\mathcal{IG}$ uses K to generate the update (*i.e.*, add or delete) token τ_u for image g_u, which will be used to update the index ψ to ψ' on $\mathcal{SS}$ side.

D. Secure Image Retrieval Service: An Instantiation

Now we present an instantiation to illustrate how to seamlessly integrate our proposed structure with a linked-list-based, provably secure dynamic SSE scheme [8]. It is easy to see that the linked lists can be replaced by parallelism-friendly dynamic encrypted data structure, *e.g.*, cascaded triangles [9].

$\mathsf{IndexBuild}(\mathbf{S}; k_{\mathsf{sse}})$: We first randomly shuffle all the nodes of each linked-list in the sub-simhash dictionaries and place them randomly in the search array $\mathsf{A_s}$. We also store the head address of each linked list in the search dictionary $\mathsf{T_s}$ for locating all the nodes in the linked list. Besides, the neighbor information of each node is stored in the update dictionary $\mathsf{T_d}$.

Algorithm 1 $\psi \leftarrow \mathsf{IndexBuild}(\mathbf{S}; k_{\mathsf{sse}})$

Initialize $\langle \mathsf{A_s}^v \rangle_{v=1}^{div}$, $\langle \mathsf{T_s}^v \rangle_{v=1}^{div}$, $\langle \mathsf{T_d}^v \rangle_{v=1}^{div}$, π, and $ctr = 1$;
for $v = 1$ to div **do**
  for each $tag\ \omega \in S^v$ **do**
    for $j = 1$ to $\#\mathsf{L}^\omega$ **do**
      Set $\mathsf{N}_j^\omega := \mathsf{id}_j^\omega \| [\![\mathsf{L}_j^\omega]\!]_{pk} \| \pi(ctr+1)$;
      Set $\mathsf{D}_j^\omega := F_{K_1}(\mathsf{id}_{j-1}^\omega) \| \pi(ctr-1) \| \pi(ctr) \| \pi(ctr+1) \| F_{K_1}(\mathsf{id}_{j+1}^\omega)$;
      Sample $\mathsf{R}_j^\omega \xleftarrow{\$} \{0,1\}^\lambda$;
      Set $\mathsf{A_s}^v[\pi(ctr)] := (\mathsf{N}_j^\omega \oplus \mathcal{H}(P_{K_3}(\omega) \| \mathsf{R}_j^\omega)) \| \mathsf{R}_j^\omega$;
      Set $\mathsf{T_d}^v[F_{K_1}(\mathsf{id}_j^\omega)] := \mathsf{D}_j^\omega \oplus G_{K_2}(\mathsf{id}_j^\omega)$;
      Set $ctr := ctr + 1$;
    end for
    Set $\mathsf{T_s}^v[F_{K_1}(\omega)] := (addr_s(\mathsf{N}_1^\omega) \| F_{K_1}(\mathsf{id}_1^\omega) \| P_{K_3}(\omega)) \oplus G_{K_2}(\omega)$;
  end for
end for
Return the encrypted index $\psi := \{\mathsf{A_s}, \mathsf{T_s}, \mathsf{T_d}\}$;

As specified in Alg. 1, $\mathcal{CS}$ first initializes the arrays and dictionaries $\mathsf{A_s} = \langle \mathsf{A_s}^v \rangle_{v=1}^{div}$, $\mathsf{T_s} = \langle \mathsf{T_s}^v \rangle_{v=1}^{div}$, $\mathsf{T_d} = \langle \mathsf{T_d}^v \rangle_{v=1}^{div}$ of size $\#\mathbf{g}$, and a counter ctr. We use a random function π which maps an integer to a random address of the array maintained by $\mathcal{SS}$. We set $k_{\mathsf{sse}} = \{K_1, K_2, K_3\}$, where K_1, K_2, K_3 are keys for pseudorandom functions F, G, P, respectively. Details of parameter settings for these functions can be found in [8].

Based on each dictionary in $\langle S^v \rangle_{v=1}^{div}$, $\mathcal{SS}$ computes N_j^ω for the j-th node of the linked list L^ω that shares the $tag\ \omega$, where id_j^ω is the ID of the j-th image in L^ω, L_j^ω is the f-bit simhash, and $\pi(ctr+1)$ is the address of the next node in array $\mathsf{A_s}^v$. Meanwhile, D_j^ω consists of the neighbor information of each node (*i.e.*, the prior and following nodes of L_j^ω and the corresponding positions in search array $\mathsf{A_s}^v$). Then $\mathcal{SS}$ encrypts N_j^ω by $\mathcal{H}$-based encryption [8] and stores it at address $\pi(ctr)$ of $\mathsf{A_s}^v$, where $\mathcal{H}$ is a hash function. Besides, D_j^ω will be stored in the update dictionary $\mathsf{T_d}^v$ in a ciphertext form. By increasing ctr, $\mathcal{SS}$ can get the $(j+1)$-th node of the linked list L^ω and encrypt the information of this node by the above operations. For each linked list L^ω, the address of the head node N_1^ω in $\mathsf{A_s}^v$ will be further encrypted and stored in the search dictionary $\mathsf{T_s}^v$. Finally, we obtain the encrypted index $\psi = (\mathsf{A_s}, \mathsf{T_s}, \mathsf{T_d})$, which will be sent to $\mathcal{SS}$

$\mathsf{Search}(K, g_q; \psi)$: As shown in Alg. 2, $\mathcal{IG}$ first computes the simhash B_q for the query image g_q and divides B_q into div parts. For the sub-simhashes $\langle \mathsf{B}_q^v \rangle_{v=1}^{div}$, $\mathcal{IG}$ generates the corresponding search tokens $\langle \tau_q^v \rangle_{v=1}^{div}$ and submits them to $\mathcal{SS}$.

$\mathcal{SS}$ parses the search token τ_q^v as $(\tau_1^v, \tau_2^v, \tau_3^v)$ and returns $\perp$ if τ_1^v is not present in $\mathsf{T_s}^v$. Otherwise, $\mathcal{SS}$ recovers the address $addr_1$ of the head of the matched linked list and looks up $\mathsf{A_s}^v[addr_1]$ to obtain N_1^q, which corresponds to the first node of

[6]One can make the output as a ciphertext for higher security against $\mathcal{SS}$. (See IV-A and [28].) But $\mathcal{IG}$ or $\mathcal{CS}$ then needs to build the encrypted index.

[7]Secure comparison protocol can be used to process an encrypted threshold.

Algorithm 2 $(\perp; \mathsf{ID}_q) \leftarrow \mathsf{Search}((K, g_q); \psi)$

$\mathcal{IG}$: Compute the simhash B_q of image g_q;
$\mathcal{IG}$: Compute the search token:
$\tau_q^v := \langle (F_{K_1}(\mathsf{B}_q^v), G_{K_2}(\mathsf{B}_q^v), [\![\mathsf{B}_q]\!]_{pk}) \rangle_{v=1}^{div}$;
At $\mathcal{SS}$ side:
Initialize set $\mathsf{ID}_q := \emptyset$;
for $v = 1$ to div **do**
  Parse τ_q^v as $(\tau_1^v, \tau_2^v, \tau_3^v)$;
  if $\mathsf{T}_\mathsf{s}^v[\tau_1^v] = \perp$ **then** Continue;
  else
    Parse $\mathsf{T}_\mathsf{s}^v[\tau_1^v] \oplus \tau_2^v$ as $addr_1 \| F_{K_1}(\mathsf{id}_1^q) \| P_{K_3}(\mathsf{B}_q^v)$;
    Look up $\mathsf{A}_\mathsf{s}^v[addr_1] = (\mathsf{NH}_1^q \| \mathsf{R}_1^q)$;
    Compute $\mathsf{N}_1^q := \mathsf{NH}_1^q \oplus \mathcal{H}(P_{K_3}(\mathsf{B}_q^v) \| \mathsf{R}_1^q)$;
    Set counter $j := 1$;
    while $addr_j \neq \mathsf{null}$ **do**
      Parse N_j^q as $\mathsf{id}_j^q \| [\![\mathsf{L}_j^q]\!]_{pk} \| addr_{j+1}$;
      $\mathcal{SS}\&\mathcal{CS}$: $(x, \perp) \leftarrow \mathsf{SHAM}([\![\mathsf{L}_j^q]\!]_{pk}, [\![\mathsf{B}_q]\!]_{pk}; sk)$;
      if $x \leq z$ **then** Add id_j^q to set ID_q; **end if**
      Look up $\mathsf{A}_\mathsf{s}^i[addr_{j+1}] = (\mathsf{NH}_{j+1}^q \| \mathsf{R}_{j+1}^q)$;
      Compute $\mathsf{N}_{j+1}^q := \mathsf{NH}_{j+1}^q \oplus \mathcal{H}(P_{K_3}(\mathsf{B}_q^v) \| \mathsf{R}_{j+1}^q)$;
      Set $j := j + 1$;
    end while
  end if
  Return set ID_q of similar images to $\mathcal{IG}$;
end for

Algorithm 3 $(\perp; \psi') \leftarrow \mathsf{Update}((K, g_u); \psi)$

$\mathcal{IG}$: Compute the simhash B_u of image g_u;
1) Adding a new image:
$\mathcal{IG}$: Compute the add token $\tau_u := \langle \tau_u^v \rangle_{v=1}^{div}$ where $\tau_u^v := (F_{K_1}(\mathsf{B}_u^v), G_{K_2}(\mathsf{B}_u^v), P_{K_3}(\mathsf{B}_q^v), F_{\mathsf{K}_1}(\mathsf{id}_u), G_{K_2}(\mathsf{id}_u), \mathsf{N}_u^v)$, and $\mathsf{N}_u^v := (\mathsf{id}_u \| [\![\mathsf{B}_u]\!]_{pk} \| \mathsf{null}) \oplus \mathcal{H}(P_{K_3}(\mathsf{B}_q^v) \| \mathsf{R}_u^v) \| \mathsf{R}_u^v$;
At $\mathcal{SS}$ side:
for $v = 1$ to div **do**
  Parse τ_u^v as $(\tau_1^v, \tau_2^v, \tau_3^v, \tau_4^v, \tau_5^v, \tau_6^v)$;
  Choose a free node with address $addr_1$;
  if $\mathsf{T}_\mathsf{s}^v[\tau_1^v] = \perp$ **then**
    Set $\mathsf{A}_\mathsf{s}^v[addr_1] := \tau_6^v$;
    Set $\mathsf{T}_\mathsf{s}^v[\tau_1^v] := (addr_1 \| \tau_4 \| \tau_3) \oplus \tau_2$;
    Set $\mathsf{T}_\mathsf{d}^v[\tau_3] := (\mathbf{0} \| \mathbf{0} \| addr_1 \| \mathbf{0} \| \mathbf{0}) \oplus \tau_5$;
  else
    Parse $\mathsf{T}_\mathsf{s}^v[\tau_1^v] \oplus \tau_2^v$ as $(addr_2 \| F_{K_1}(\mathsf{id}_1^u) \| \tau_3)$;
    Parse τ_6 as $(\mathsf{NH}_u^v \| \mathsf{R}_u^v)$;
    Set $\mathsf{A}_\mathsf{s}^v[addr_1] := (\mathsf{NH}_u^v \oplus (\mathbf{0} \| addr_2)) \| \mathsf{R}_u^v$;
    Update head node $\mathsf{T}_\mathsf{s}^v[\tau_1^v] := (addr_1 \| \tau_4 \| \tau_3) \oplus \tau_2$;
    Set $\mathsf{T}_\mathsf{d}^v[\tau_4] := (\mathbf{0} \| \mathbf{0} \| addr_1 \| addr_2 \| F_{K_1}(\mathsf{id}_1^u)) \oplus \tau_5$;
    Set $\mathsf{T}_\mathsf{d}^v[F_{K_1}(\mathsf{id}_1^u)] := \mathsf{T}_\mathsf{d}^v[F_{K_1}(\mathsf{id}_1^u)] \oplus (\tau_4 \| addr_1 \| \mathbf{0})$;
  end if
end for
2) Deleting an existing image:
$\mathcal{IG}$: Compute the delete token $\tau_u := (F_{K_1}(\mathsf{id}_u), G_{K_2}(\mathsf{id}_u))$;
At $\mathcal{SS}$ side:
Parse τ_u as (τ_1, τ_2) and return $\perp$ if τ_1 is not in T_d;
for $v = 1$ to div **do**
  Parse $\mathsf{T}_\mathsf{d}^v[\tau_1] \oplus \tau_2$ as $(s_1 \| a_1 \| a_2 \| a_3 \| s_2)$;
  Parse $\mathsf{A}_\mathsf{s}^v[a_1]$ as $\mathsf{NH}_u^v \| \mathsf{R}_u^v$;
  Update $\mathsf{A}_\mathsf{s}^v[a_1] := \mathsf{NH}_u^v \oplus (\mathbf{0} \| a_2) \oplus (\mathbf{0} \| a_3) \| \mathsf{R}_u^v$;
    $\mathsf{T}_\mathsf{d}^v[s_1] := \mathsf{T}_\mathsf{d}^v[s_1] \oplus (\mathbf{0} \| a_2 \| \tau_1) \oplus (\mathbf{0} \| a_3 \| s_2)$;
    $\mathsf{T}_\mathsf{d}^v[s_2] := \mathsf{T}_\mathsf{d}^v[s_2] \oplus (\tau_1 \| a_2 \| \mathbf{0}) \oplus (s_1 \| a_1 \| \mathbf{0})$;
end for
$\mathcal{SS}$: Obtain an updated index $\psi' := \{\mathsf{A}_\mathsf{s}, \mathsf{T}_\mathsf{s}, \mathsf{T}_\mathsf{d}\}$;

the matched linked list. Then $\mathcal{SS}$ retrieves the matched linked list to find similar images until $addr_j = \mathsf{null}$. Specifically, for N_j^q that contains the j-th node's encrypted simhash $[\![\mathsf{L}_j^q]\!]_{pk}$ of the matched linked list, $\mathcal{SS}$ and $\mathcal{CS}$ jointly compute the Hamming distance between L_j^q and B_q using SHAM protocol (in Section II-E). When the Hamming distance is smaller than the threshold z, the ID id_j^q of the matched image is added to the set ID_q. Then the $(j+1)$-th node's information can be extracted via $addr_{j+1}$. After detecting all the nodes of the matched linked lists, $\mathcal{SS}$ can create the set ID_q with the ID of all the similar images. Finally, ID_q will be returned to $\mathcal{IG}$.

$\mathsf{Update}(K, g_u; \psi)$: As shown in Alg. 3, $\mathcal{IG}$ first computes the simhashes $\langle \mathsf{B}_u^v \rangle_{v=1}^{div}$ and the update tokens $\langle \tau_u^v \rangle_{v=1}^{div}$.

Adding an image: The update token τ_u^v can be parsed into six parts. $\mathcal{SS}$ first chooses a free position from the array A_s^v that will be used to store the update image's information. If τ_1^v is not present in T_s^v, this update node will be set as the only node of a new linked list. Otherwise, $\mathcal{SS}$ adds the new node to the head of the matched linked list and updates the neighbor information. To be specific, $\mathcal{SS}$ stores the information of the update node in A_s^v, updates T_s^v to save the address of the new head node, and updates the neighbor information of T_d^v.

Deleting an image: For simplicity, we assume $\mathcal{SS}$ first searches for the identifier of the image g_u to be deleted. $\mathcal{SS}$ parses the search tokens τ_u as (τ_1, τ_2) and returns $\perp$ if τ_1 is not present in T_d^v. Otherwise, $\mathcal{SS}$ looks up T_d^v and recovers the adjacency information, where s_1 and s_2 contain the "encrypted" ID of the neighbor nodes, and a_1 and a_3 are the addresses of the neighbor nodes. $\mathcal{SS}$ then links the pointer of the previous node to the next node in A_s^v and updates the related entries of the neighbor nodes in T_d^v.

IV. Theoretical Analysis

A. Security

Security of our system is easy to be argued thanks to its modular design from any SSE scheme and any secure protocols (for SCMP and SHAM functionalities) which in turn rely on any additive homomorphic encryption scheme (Paillier in our case). Our system inherits any leakage incurred by the underlying SSE scheme, such as the neighbor nodes during update in our instantiation [8]. When instantiated with another SSE scheme, *e.g.*, [9], the leakage would be different.

On the $\mathcal{SS}$ side, it only receives ciphertexts at the beginning. In the image retrieval (resp. dynamic update) phase, $\mathcal{SS}$ obtains the search (resp. update) tokens which encrypts the query image in the form of Paillier ciphertext as well. In

	SCMP (ms/KB)		SHAM (ms/KB)	
Phase	Time	Bandwidth	Time	Bandwidth
Offline	131.23	22.34	227.50	23.07
Online	4.48	9.64	27.00	20.58

TABLE I: Reference costs of SCHP & SHAM

addition, while the Hamming distance (the output of the SHAM subroutine) and the threshold is revealed to $\mathcal{SS}$, the dataset and the query image are encrypted, so $\mathcal{SS}$ still cannot build any meaningful relation of the dataset to recover the original images and the corresponding features.

On the $\mathcal{CS}$ side, the secure two-party computation protocols (for SCMP and SHAM) guarantee that $\mathcal{CS}$ can learn nothing about both the input and the output of $\mathcal{SS}$. With the non-colluding assumption, $\mathcal{SS}$ and $\mathcal{IG}$ (obviously) will not actively reveal the ciphertexts to the $\mathcal{CS}$. Only the simhash output by SimhashComp is revealed to $\mathcal{SS}$, which is far from the feature vector [29]. For higher security, existing transformation can turn the output of SCMP into a ciphertext [28].

B. Efficiency

For the computation overhead, encrypting data and constructing SSE-based index are one-time processes, we thus focuses on image search and update phases. For searching, the traversal done by $\mathcal{SS}$ depends on the length of the linked lists. For databases with a lot of supposedly different images, the images will be distributed across $2^{\lceil \frac{f}{div} \rceil}$ different linked lists, which is the number of possibilities for a sub-simhash. Even if we probe multiple dictionaries for an even higher accuracy, the average search complexity is $O((\alpha/2^{\alpha}) \cdot \#\mathbf{g})$ where $\alpha = \lceil \frac{f}{div} \rceil$. To add an image, $\mathcal{SS}$ retrieves the pointer to the head of the corresponding linked list, which takes $O(1)$ time. To delete an image, after the node is searched, $\mathcal{SS}$ can easily find the address of the node corresponding to update image and update the encrypted index. The latter step is $O(1)$.

For the storage overhead, the index in the basic scheme mainly consists of the sub-simhash dictionaries, which need $O(div \cdot \#\mathbf{g})$ space. In the secure scheme, the index contains $\mathsf{A_s}$, $\mathsf{T_s}$, and $\mathsf{T_d}$, where the storage complexity is $O(div \cdot (\#\mathbf{g})^2)$ for the array $\mathsf{A_s}$, and $O(div \cdot \#\mathbf{g})$ for $\mathsf{T_s}$ and $\mathsf{T_d}$.

V. Experiment Evaluation

Motivated by outsourcing, we deployed $\mathcal{SS}$ and $\mathcal{CS}$ on Amazon AWS EC2 instances "c4.8xlarge" (same as [17]) with a 36-core Intel Xeon CPU at 2.90GHz and 60GB RAM for our experiments. $\mathcal{SS}$ and $\mathcal{CS}$ run Java runtime environment (JRE) 1.8 on Windows Server 2012R2. $\mathcal{IG}$ and $\mathcal{U}$ run on Windows 10 desktop, with a 4-core Intel CPU at 2.90GHz and 12GB RAM.

Our experiments used five well-known real datasets[8] which are also used in the artificial intelligence community (*e.g.*, for large-scale image retrieval [30]), namely, `Holidays`: 1491 personal holiday photos underwent various transformations,

[8]Last retrieved in 2018 from http://lear.inrialpes.fr/people/jegou/data.php, http://www.robots.ox.ac.uk/~vgg/data/oxbuildings, https://archive.org/details/ukbench, https://authors.library.caltech.edu/7694, http://press.liacs.nl/mirflickr

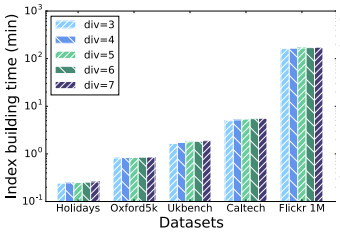


Fig. 3: Index-building time

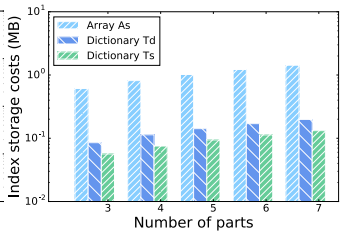


Fig. 4: Index storage costs

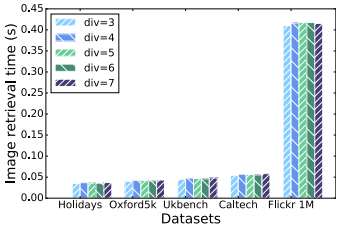


Fig. 5: Image retrieval time

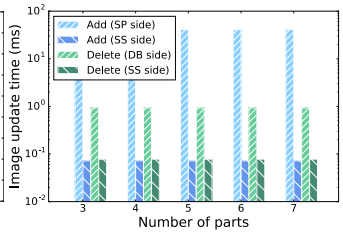


Fig. 6: Image update time

`Oxford5K`: 5062 images of 11 landmarks, each is represented by 5 queries, `Ukbench`: 10200 images of 2550 different scenes/objects, `Caltech-256`: 30608 images from 256 categories, and `MIR Flickr 1M`: 1 million images from Flickr.

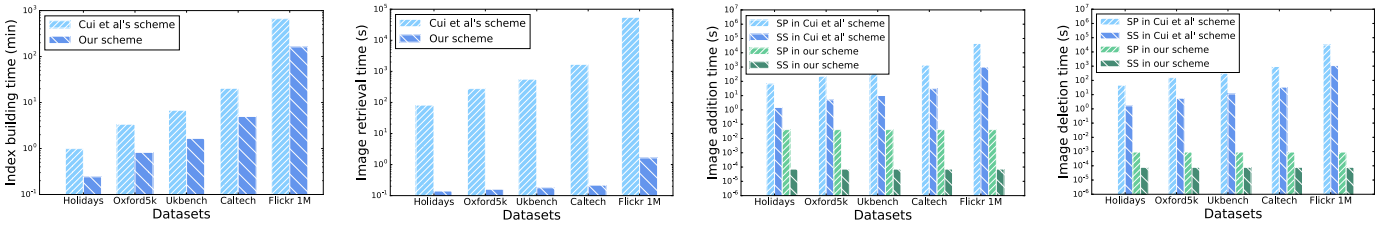
A. Details on Building Blocks

Our experiment used Paillier with a 1024-bit modulus. The bit length of each random mask is set to be 32. Following [3], we used 80-bit wire labels for garbled circuits. Table I presents the time and bandwidth costs for subroutines SCMP and SHAM detailed in Appendix. Several important remarks are in order. First, we utilize conceptually simple protocols which one party ($\mathcal{CS}$) decrypts a randomized ciphertext, while the two parties ($\mathcal{CS}$ and $\mathcal{SS}$) execute a garbled circuit on randomized inputs. The time costs for $\mathcal{CS}$ and $\mathcal{SS}$ are roughly the same and we do not differentiate them. Second, the offline phase, which prepares the garbled circuits and the associated oblivious transfer (OT) protocol, can be executed before seeing any data. Lastly, it is easy to adopt any SCMP [28] and SHAM protocols [3], [25], [26] or use optimized OT protocols.

B. Efficiency

We implement our secure scheme in Section III-D to precisely evaluate its efficiency. Firstly, for sub-simhash structure, we demonstrate how div, the number of the parts in each simhash, affects the performance. Figures 3 and 4 present the costs for computation of building index over different datasets and for storing the encrypted index of the `Holidays` dataset, with div varied from 3 to 7. We can see that the computation overhead grows linearly with the dataset size, but is faintly affected by div. Index storage grows linearly with div.

Figures 5 and 6 demonstrate the impact of div on the time costs of image retrieval and update. From Fig. 5, we can see that the image retrieval on each dataset is nearly unaffected by div, and $\mathcal{SS}$ only needs about 0.41s for searching an image on



(a) Building index phase (b) Image retrieval phase (c) Image adding phase (d) Image deleting phase

Fig. 7: Time costs comparison between InstantCryptoGram and Cui *et al.*'s scheme

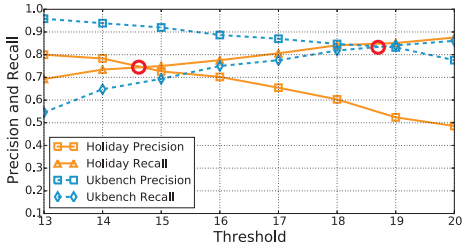


Fig. 8: Precision and recall (vs. threshold) over two datasets

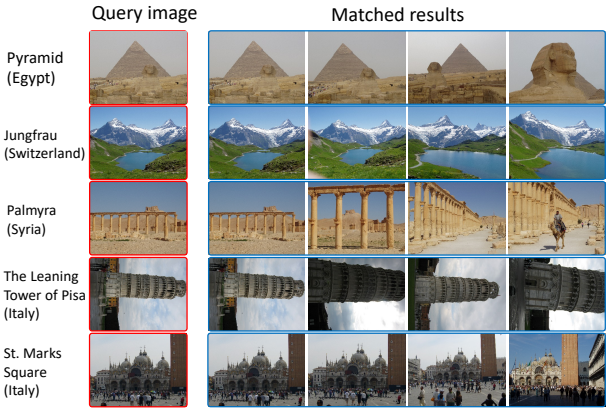


Fig. 9: Sample search results of InstantCryptoGram

the `MIR Flickr 1M` dataset with 1 million records. Fig. 6 shows that the image update time is independent of div. When adding a new image, $\mathcal{IG}$ takes about 0.040s to generate the tokens, while $\mathcal{SS}$ takes 0.072s to update the index. To delete an image, the time costs on both $\mathcal{IG}$ and $\mathcal{SS}$ are less than 1s.

Figure 7 shows in log-scale the overall time costs for each phase compared with Cui *et al.*'s SSE-based image retrieval scheme [17], which is claimed as the state-of-the-art for mobile devices. Here div is set to be 15. (We will show that it leads to a favorable accuracy for the datasets we used). From Fig. 7a, the time cost of Cui *et al.*'s scheme is four times of ours in the index-building phase over all datasets. For image retrieval, addition, and deletion, Fig. 7b, 7c, and 7d show that ours also enjoys several order-of-magnitude improvements.

C. Accuracy

For accuracy, we evaluate *precision*, the fraction of retrieved images that are relevant to the query, and *recall*, the fraction of the relevant images that are successfully retrieved. Figure 8 shows the precision-recall curves with different thresholds z over `Holidays` and `Ukbench` datasets, with each simhash being 128-bit. With the growth of the threshold z, the precision decreases and the recall increases. We thus choose the threshold as the one corresponding to the intersection of the precision and recall curves which give us a precision of 0.75 and a recall of 0.83 respectively for `Holidays` and `Ukbench`.

We also conduct query experiments on the `Holidays` dataset for our motivating OSN application, where each image is resized to 640×480 and transformed into a 128-bit of simhash, with the threshold set to 14. As a highlight illustrated in Fig. 9, when querying a pyramid, the searched results contain the pyramid images from different views and distances, and the sphinx which is co-located with the pyramid is also included. Simply put, while we "aggressively" perform dimension reduction for efficiency, the retrievals are still robust to changes in image scale, noise, illumination, and viewpoint, preserving the nice properties of the SIFT representation.

VI. Conclusion

InstantCryptoGram, our secure image retrieval service, uses an inverted-index for the pre-screening, and secure two-party computation of simple functions including comparison and Hamming distance for similarity test. Our experiments confirmed that its accuracy is high and its overheads are acceptable for large-scale data. With its modular design, the performance can be improved with better and optimized building blocks.

Our work based on a basic IR principle that an efficient search must prune as many irrelevant results as possible in the first stage. It is interesting to consider other image retrieval techniques such as decision trees [31] and neural networks [20] over encrypted data, while achieving acceptable efficiency.

Acknowledgment

The corresponding author Qian, and Yanjiao and Chenliang, are supported in part by National Natural Science Foundation of China (Grant Nos. U1636219, 61373167, 61702380, 61502344), and The Key Program of Natural Science Foundation of Hubei Province (Grant Nos. 2017CFA047, 2017CFB134, 2017AAA125, 2017CFB502). Sherman is supported by General Research Funds (CUHK 14201914) of the Research Grants Council, University Grant Committee of Hong Kong.

REFERENCES

[1] R. A. Negoescu and D. Gatica-Perez, "Analyzing Flickr groups," in *CIVR*. ACM, 2008, pp. 417–426.

[2] H. Bay, A. Ess, T. Tuytelaars, and L. J. V. Gool, "Speeded-up robust features (SURF)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[3] Y. Huang, L. Malka, D. Evans, and J. Katz, "Efficient privacy-preserving biometric identification," in *NDSS*, 2011, pp. 250–267.

[4] L. Weng, L. Amsaleg, A. Morton, and S. Marchand-Maillet, "A privacy-preserving framework for large-scale content-based information retrieval," *IEEE TIFS*, vol. 10, no. 1, pp. 152–167, 2015.

[5] X. Li, Q. Xue, and M. C. Chuah, "CASHEIRS: Cloud assisted scalable hierarchical encrypted based image retrieval system," in *INFOCOM*. IEEE, 2017.

[6] B. Ferreira, J. Rodrigues, J. Leitao, and H. Domingos, "Practical privacy-preserving content-based retrieval in cloud image repositories," *IEEE Transactions on Cloud Computing*, 2017, early access.

[7] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *ASIACRYPT*, 2010, pp. 577–594.

[8] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *CCS*, 2012, pp. 965–976.

[9] R. W. F. Lai and S. S. M. Chow, "Forward-Secure Searchable Encryption on Labeled Bipartite Graphs," in *ACNS*, 2017, pp. 478–497.

[10] Q. Wang, K. Ren, M. Du, Q. Li, and A. Mohaisen, "SecGDB: Graph encryption for exact shortest distance queries with efficient updates," in *Financial Cryptography and Data Security (FC)*, 2017, pp. 79–97.

[11] P. P. Tsang, S. S. M. Chow, and S. W. Smith, "Batch pairing delegation," in *IWSEC*, 2007, pp. 74–90.

[12] L. Zhang, T. Jung, K. Liu, X. Li, X. Ding, J. Gu, and Y. Liu, "PIC: enable large-scale privacy preserving content-based image search on cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3258–3271, 2017, also appeared in Intl' Conf. on Parallel Processing (ICPP) 2015.

[13] D. Vizár and S. Vaudenay, "Cryptanalysis of chosen symmetric homomorphic schemes," in *Central European Crypto Conference*, 2014.

[14] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*, 1999, pp. 223–238.

[15] Y. Elmehdwi, B. K. Samanthula, and W. Jiang, "Secure k-nearest neighbor query over encrypted data in outsourced environments," in *ICDE*. IEEE, 2014, pp. 664–675.

[16] Q. Wang, M. He, M. Du, S. S. M. Chow, R. W. F. Lai, and Q. Zou, "Searchable encryption over feature-rich data," *IEEE Transactions on Dependable and Secure Computing*, 2017, early access.

[17] H. Cui, X. Yuan, and C. Wang, "Harnessing encrypted data in cloud for secure and efficient mobile image sharing," *IEEE Trans on Mobile Comp*, vol. 16, no. 5, pp. 1315–1329, 2017, also in INFOCOM'15.

[18] G. S. Manku, A. Jain, and A. Das Sarma, "Detecting near-duplicates for web crawling," in *WWW*. ACM, 2007, pp. 141–150.

[19] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *STOC*. ACM, 2002, pp. 380–388.

[20] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *S&P*. IEEE, 2017, pp. 19–38.

[21] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Intl. J. Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[22] R. W. F. Lai, R. K. H. Tai, H. W. H. Wong, and S. S. M. Chow, "Multi-key homomorphic signatures unforgeable under insider corruption," Cryptology ePrint Archive, Report 2016/834, 2016.

[23] Y. Chen, S. S. M. Chow, K. Chung, R. W. F. Lai, W. Lin, and H. Zhou, "Cryptography for parallel RAM from indistinguishability obfuscation," in *Innovations in Theoretical Computer Science*, 2016, pp. 179–190.

[24] A. Yao, "How to generate and exchange secrets," in *FOCS*. IEEE, 1986, pp. 162–167.

[25] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits," in *USENIX Security*, 2011.

[26] J. Bringer, H. Chabanne, M. Favre, A. Patey, T. Schneider, and M. Zohner, "GSHADE: faster privacy-preserving distance computation and biometric identification," in *IH&MMSec*, 2014, pp. 187–198.

[27] Y. Zhao and S. S. M. Chow, "Privacy preserving collaborative filtering from asymmetric randomized encoding," in *FC*, 2015, pp. 459–477.

[28] N. Attrapadung, G. Hanaoka, S. Kiyomoto, T. Mimoto, and J. C. N. Schuldt, "A taxonomy of secure two-party comparison protocols and efficient constructions," in *Privacy, Security and Trust*, 2017, to appear.

[29] Z. Li, J. Xie, D. Tu, and Y. Choi, "Sparse signal recovery by stepwise subspace pursuit in compressed sensing," *IJDSN*, vol. 9, 2013.

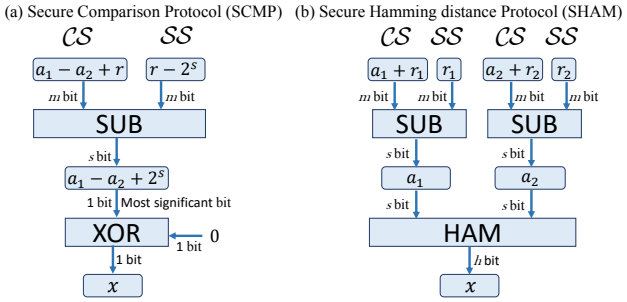


Fig. 10: SCMP & SHAM subroutines

[30] H. Tang and H. Liu, "A novel feature matching strategy for large scale image retrieval," in *IJCAI*, 2016, pp. 2053–2059.

[31] R. K. H. Tai, J. P. K. Ma, Y. Zhao, and S. S. M. Chow, "Privacy-preserving decision trees evaluation via linear functions," in *Euro. Symp. on Research in Comp. Security (ESORICS) Part II*, 2017, pp. 494–512.

APPENDIX

1) Garbled Circuits: Yao [24] proposed garbled circuits for secure two-party computation. It allows two parties holding inputs x and y, respectively, to jointly evaluate $f(x, y)$ for an arbitrary function f represented as a boolean circuit. In details, one party builds the garbled circuit of f by associating each wire with two cryptographic keys (called the wire label), and encrypting the labels of its outgoing wires using an appropriate combination of the two input wire labels, according to the truth table of the gate. The other party obtains via OT the input wire labels without leaking which. It can then decrypt the output value at exactly one outgoing wire and hence evaluate the entire circuit gate-by-gate to obtain $f(x, y)$. No party learns anything about x and y beyond what is implied by $f(x, y)$.

Fig. 10 shows the two (garbled) circuits used in our experiment basing on three atomic circuits [3], [25]. SUB outputs the difference of the two input numbers. XOR returns a bit which is the logical exclusive-OR of two input bits. HAM outputs the Hamming distance between two input binary numbers.

2) Secure Comparison Protocol (SCMP): SCMP works as follows. For the input data $a_1 \in \mathbb{Z}_s$ and $a_2 \in \mathbb{Z}_s$, $\mathcal{SS}$ first selects an random integer $r \in \mathbb{Z}_m$ ($m > s$), then computes $[\![a_1 - a_2 + r]\!]$, and sends the ciphertext to $\mathcal{CS}$. $\mathcal{CS}$ can therefore decrypts the ciphertext to obtain $a_1 - a_2 + r$. Then $\mathcal{SS}$ and $\mathcal{CS}$ collaboratively execute the circuit in Fig. 10 (a), where $\mathcal{SS}$ takes as input $(r - 2^s)$ and $\mathcal{CS}$ takes as input $(a_1 - a_2 + r)$. We use a SUB circuit to obtain s-bit $(a_1 - a_2 + 2^s)$, whose most significant bit indicates the relationship between a_1 and a_2. By taking it and 0 as two inputs for an XOR circuit, we can obtain a bit x, which is equal to 1 when $a_1 > a_2$, 0 otherwise.

3) Secure Hamming Distance Protocol (SHAM): $\mathcal{SS}$ first chooses two random number $r_1, r_2 \in \mathbb{Z}_m$, and adds these random numbers to $[\![a_1]\!]$ and $[\![a_2]\!]$, then $[\![a_1 + r_1]\!]$ and $[\![a_1 + r_2]\!]$ are transferred to $\mathcal{CS}$. We implement the SHAM circuit in Fig. 10 (b) to compute the Hamming distance between a_1 and a_2, where $\mathcal{SS}$ takes as input r_1 and r_2, and $\mathcal{CS}$ takes as input $(a_1 + r_1)$ and $(a_2 + r_2)$. We first use two SUB circuits to obtain a_1 and a_2. Subsequently, we apply HAM circuit resulting an h-bit output x which is the Hamming distance of a_1 and a_2.