ImageProof: Enabling Authentication for Large-Scale Image Retrieval

Shangwei Guo†, Jianliang Xu†, Ce Zhang†, Cheng Xu†, and Tao Xiang‡

†*Department of Computer Science, Hong Kong Baptist University, Hong Kong, China*
{csswguo,xujl,cezhang,chengxu}@comp.hkbu.edu.hk
‡*College of Computer Science, Chongqing University, Chongqing, China*
txiang@cqu.edu.cn

***Abstract*—With the explosive growth of online images and the popularity of search engines, a great demand has arisen for small and medium-sized enterprises to build and outsource large-scale image retrieval systems to cloud platforms. While reducing storage and retrieval burdens, enterprises are at risk of facing untrusted cloud service providers. In this paper, we take the first step in studying the problem of query authentication for large-scale image retrieval. Due to the large size of image files, the main challenges are to (i) design efficient authenticated data structures (ADSs) and (ii) balance search, communication, and verification complexities. To address these challenges, we propose two novel ADSs, the Merkle randomized k-d tree and the Merkle inverted index with cuckoo filters, to ensure the integrity of query results in each step of image retrieval. For each ADS, we develop corresponding search and verification algorithms on the basis of a series of systemic design strategies. Furthermore, we put together the ADSs and algorithms to design the final authentication scheme for image retrieval, which we name ImageProof. We also propose several optimization techniques to improve the performance of the proposed ImageProof scheme. Security analysis and extensive experiments are performed to show the robustness and efficiency of ImageProof.**

I. Introduction

Content-based image retrieval (CBIR) is a system for searching for images with similar content from a large image database [1]. Among the various existing CBIR schemes, scale invariant feature transform (SIFT) is a widely used method that detects and extracts an image's local features, based on which a nearest-neighbor indexing algorithm is enabled to identify similar images [2]. SIFT and its variants have been implemented in many CBIR systems because of their scale and rotation invariant characteristics [3]–[5].

With the advances in data-as-a-service (DaaS) technology, a small or medium-sized enterprise can build a cost-efficient, large-scale SIFT-based image retrieval system on cloud platforms such as Google Similar Images, Amazon Flow, and Alibaba Image Search [6]. While this allows enterprises to extricate themselves from heavy storage, computation, and communication burdens, untrustworthy service providers bring new security issues.

Existing works on secure image retrieval mainly focus on the honest-but-curious model [7], [8]. In these systems, it is assumed that the service provider (SP) is trusted to return genuine search results. However, this assumption may not always be held due to various reasons, including software/hardware malfunctions, hack attacks, or even corporate dishonesty [9], [10]. For example, as reported, in the Search for ExtraTerrestrial Intelligence at Home (SETI@home) project, 1% of the users have attempted to forge their contributions in order to move up on the Web listings of top contributors [11], [12]. The search results cannot be trusted once the listings are maliciously modified. As such, how to ensure the integrity of search results for large-scale image retrieval remains a pressing problem that needs to be addressed.

Query authentication techniques, which guarantee the integrity of search results for outsourced databases, are promising solutions to defend against an untrustworthy SP [13]–[19]. The basic idea is that the data owner builds a well-designed authenticated data structure (ADS) and outsources it to the SP, together with the original database. Upon receiving a query request, the SP computes the results, as well as a cryptographic proof, known as verification object (VO), on the basis of the ADS. Both the search results and the VO are returned to the client. Using the VO, the client can verify whether the received search results are indeed genuine.

Although query authentication has been studied extensively, no authentication techniques exist for large-scale image retrieval. The challenges are twofold. On the one hand, CBIR is a large, complex system and designing a query authentication scheme for such a system is a big challenge in itself. Particularly, a SIFT-based image retrieval method generally requires two steps when searching for similar images: bag-of-visual-words (BoVW) encoding and inverted index search. Specifically, BoVW encoding transforms a query image into a sparse BoVW vector, while an inverted index is used to facilitate image search with such vectors. However, existing query authentication schemes can only be applied to one step of SIFT-based image retrieval. For example, the scheme proposed in [15] can only support authenticated searches over an inverted index. To the best of our knowledge, none of the existing query authentication schemes can achieve both authenticated BoVW encoding and inverted index search. On the other hand, the client usually has only limited storage, communication, and computation resources. Because of the large size of image files, the client's computation and communication costs would be prohibitively high if we only outsource the inverted index search step and apply an existing technique to this step. Outsourcing both steps of image retrieval and designing a systematic authentication scheme for them would

be more desirable.

To address the above issues, in this paper, we propose a robust and efficient query authentication scheme, named *ImageProof*, for large-scale SIFT-based image retrieval with large or medium-sized codebooks. Specifically, we develop two novel ADSs for each step of image retrieval. The first ADS, the Merkle randomized k-d tree, is designed for verifying the integrity of BoVW encoding by integrating the randomized k-d tree and the Merkle hash tree. The other one is the Merkle inverted index with cuckoo filters that is designed to ensure the integrity of the top-k image search step. To make the proposed techniques more efficient, we compress the partial VO when traversing the randomized k-d tree, employ a set membership test during inverted index search, and carefully design the image search and verification algorithms. Moreover, several optimization techniques are proposed to further improve the performance of our system.

To summarize, our contributions made in this paper are as follows:

- To the best of our knowledge, this is the first attempt to study query authentication for large-scale image retrieval. As an initial exploration, we propose an efficient authentication scheme, ImageProof, for SIFT-based image retrieval with large or medium-sized codebooks.
- We propose two novel ADSs, the Merkle randomized k-d tree and the Merkle inverted index with cuckoo filters, for the proposed ImageProof scheme. We also systematically design image search and verification algorithms to make the scheme robust.
- We develop several optimization techniques to further reduce the costs of both the SP and the client.
- We perform a security analysis and conduct experiments to evaluate the query authentication performance under various system settings.

The rest of this paper is organized as follows. Section II introduces the SIFT-based image retrieval scheme and several preliminaries. Section III gives a formal definition of the CBIR authentication problem. Section IV presents the two proposed ADSs, followed by a description of the overall ImageProof scheme in Section V and the optimization techniques in Section VI. Section VII shows the experimental results. The related works are reviewed in Section VIII. Finally, Section IX concludes the paper.

II. Background and Preliminaries

In this section, we present some background knowledge and preliminaries. Table I summarizes the symbols frequently used in this paper.

A. SIFT-based Image Retrieval

We focus on searching for top-k similar images from an image database. Generally, the search process of a SIFT-based image retrieval method consists of two steps: BoVW encoding and inverted index search [2]. In this paper, we employ a commonly used technique, approximate k-means (AKM) [20], for BoVW encoding. In the following, we formally define the top-k image search problem and provide an overview of the data structures and index search algorithms.

TABLE I
Summary of symbols

Symbol	Description
Q	Query image
I	An image in the database
w_{c_i}	Weight of cluster c_i
p_{c_i}	Impact value of the first posting of Merkle inverted list
B_Q (B_I)	BoVW vector of Q (I)
f_{I,c_i}	Frequency of cluster c_i in B_I
$S(Q,I)$	Similarity between Q and I
p_Q (p_I)	Impact value vector of Q (I)
$\{I_{top_i}\}_{i=1}^k$	Top-k similar images
s_k	Similarity between Q and the k-th similar image

In SIFT-based image retrieval, an image is represented by a set of feature vectors extracted from the image. A large number of clusters, also known as *codebook* or *vocabulary*, are pre-trained on the basis of the feature vectors of the images in the database. Given an image, AKM is used to find the approximate nearest cluster for each feature vector of the image. By counting the frequency of these approximate nearest clusters, a sparse BoVW vector is obtained for the image, in which the i-th value represents the number of feature vectors that are approximately nearest to the i-th cluster. Let Q be the query image, I be an image in the database, and B_Q, B_I be the BoVW vectors of Q, I respectively. The similarity between Q and I is defined by the similarity between their BoVW vectors B_Q, B_I.

Similarity Measure. Before formally defining the similarity measure, we first present some notations:

- w_{c_i}, the weight of a pre-trained cluster c_i;
- n_C, the number of clusters in the database;
- n_D, the number of images in the database;
- n_{D,c_i}, the number of images that have at least one feature vector approximating to cluster c_i;
- f_{I,c_i}, the frequency of cluster c_i in B_I;
- p_{I,c_i}, the impact of c_i on I;

and we define

$$w_{c_i} = \ln \frac{n_D}{n_{D,c_i}}, \qquad p_{I,c_i} = \frac{w_{c_i} \cdot f_{I,c_i}}{||B_I||}, \tag{1}$$

where $||B_I||$ is the L_2-norm of B_I [21].

The similarity between Q and I, $S(Q,I)$, is defined on the basis of the cosine distance, i.e.,

$$S(Q,I) = \sum_{i=1}^{n_C} p_{Q,c_i} \cdot p_{I,c_i}. \tag{2}$$

Because the BoVW vector is sparse, most impact values of p_{Q,c_i} would be zero. As such, the similarity between Q and I can be simplified as

$$S(Q,I) = \sum_{p_{Q,c_i} \neq 0} p_{Q,c_i} \cdot p_{I,c_i}. \tag{3}$$

Definition 1. (**Top-k Image Search**) Given a query image Q, find k most similar images $\{I_{top_i}\}_{i=1}^k$ such that, for $\forall i \in [1,k]$ and each image $I \notin \{I_{top_i}\}_{i=1}^k$ in the database,

$$S(Q, I_{top_i}) \geq S(Q,I). \tag{4}$$

Approximate k-Means. Approximate k-means (AKM), a

computationally efficient variation of the original k-means algorithm, is used for efficient BoVW encoding in SIFT-based image retrieval [20]. Given a feature vector, instead of finding the *exact* nearest cluster among a huge number of pre-trained clusters, AKM uses the randomized k-d trees to obtain an *approximate* nearest cluster.

More specifically, a set of randomized k-d trees are first constructed over the pre-trained clusters. In contrast to a regular k-d tree, at each level of the index tree, a randomized k-d tree chooses the split dimension *randomly* from the dimensions with the largest variances. When processing a given feature vector, all the trees are traversed until leaf nodes are reached, on the basis of a global priority queue that maintains the distances of the feature vector to each indexed subspace. The search is terminated after a pre-defined number of tree traversals is reached, and the nearest cluster obtained so far is returned as the result.

Inverted Index. The inverted index in SIFT-based image retrieval is used for efficient similarity evaluation over BoVW vectors [21]. It consists of two major components: clusters and their corresponding inverted lists. An inverted list contains all the images that have at least one feature vector with this cluster c_i as the nearest cluster. It is represented by a sequence of $\langle I, value \rangle$ postings, where I is an image identifier and $value$ is the corresponding weight. In this paper, we focus on the widely used impact-ordered inverted index [15], [22], in which the weight is the impact value p_{I,c_i} of the corresponding image and the postings are sorted in descending order.

B. Cryptographic Primitives and Preliminaries

Digital Signature. A digital signature scheme is designed for verifying the authenticity and integrity of messages. It typically consists of three algorithms: (*i*) a key generation algorithm that outputs the private key and a corresponding public key, (*ii*) a signing algorithm that, given a message and a private key, computes a signature, and (*iii*) a verifying algorithm that takes the message, signature, and public key as input and outputs either acceptance or rejection.

Merkle Hash Tree. A cryptographic hash function, denoted by $h(\cdot)$, takes an arbitrary-length string as input and outputs a fixed-length bit string. It is a one-way function that is collision resistant (i.e., finding two messages which have the same hash value is computationally hard). A Merkle hash tree (MH-tree) is an authenticated binary tree, enabling users to verify individual data objects without retrieving the entire database [23]. A simple example of an MH-tree is shown in Fig. 1. Each leaf node in the MHtree stores the digest of an object, computed using a cryptographic hash function $h(\cdot)$. Each internal node stores a digest computed on the concatenation of the digests of its children, e.g., $h_{N_2} = h(h_{N_4}|h_{N_5})$ and $h_{N_7} = h(h(o_7)|h(o_8))$. The root hash is signed for verification. To authenticate the integrity of object o_6, one only needs to verify the root hash reconstructed from o_6, its sibling's hash $h(o_5)$, and the adjacent hashes h_{N_7}, h_{N_2} on the leaf-to-root path.

Cuckoo Filter. A cuckoo filter is a data structure supporting approximate set membership tests [24]. Specifically, a cuckoo filter is a compact variant of a cuckoo hash table that uses a small f-bit fingerprint to represent data. The filter

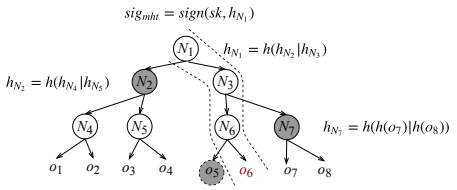

Fig. 1. An example of a Merkle hash tree.

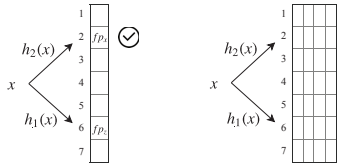

Fig. 2. A cuckoo filter, two hash values per item and four slots per bucket.

is implemented as an array of buckets associated with two hash functions, $h_1(\cdot)$ and $h_2(\cdot)$, which determine two alternate buckets of an input item. Fig. 2 illustrates an example of inserting x into a hash table of 7 buckets, where the hashes of x are 2 and 6. If either of these two buckets is free, x's fingerprint fp_x is inserted into that bucket. Otherwise, the filter would recursively kick existing data to their alternate buckets until space is found or attempts are exhausted. A bucket can have multiple slots. Fig. 2 also shows a cuckoo hash table of 7 buckets, where each bucket has 4 slots. To look up an item, we check both buckets to see whether either contains the fingerprint of the item. If no matching fingerprint is found, the item is definitely not in the filter. If a matching fingerprint is found, the item *might* be in the filter. False positives occur when a matching fingerprint of another item is stored in either of the two buckets.

Compared with traditional Bloom filters, cuckoo filters have three major advantages [24]: (1) supporting dynamic deletions; (2) achieving a better lookup performance; and (3) consuming less space when the target false positive rate (FPR) is less than 3%.

III. Problem Definition

System Overview. As illustrated in Fig. 3, our system consists of three parties: (*i*) image owner, (*ii*) service provider (SP), and (*iii*) client. During system setup, the image owner first extracts the feature vectors from the images using an existing feature extraction method and builds an authenticated data structure (ADS). Then, the image owner outsources the image dataset and its ADS to the SP. Later, when receiving a query request containing a set of feature vectors extracted from the query image, the SP executes BoVW encoding and searches the top-k similar images according to Definition 1. Furthermore, the SP constructs a verification object (VO) for the search results on the basis of the ADS and returns both the results and the VO to the client.

Threat Model. We now define the potential threats of the outsourced image database. In this paper, we assume a malicious threat model in which the SP could return incorrect search results (e.g., faked or low-ranked images). To protect the client from receiving wrong image results, the client can use the VO returned by the SP to verify the integrity of

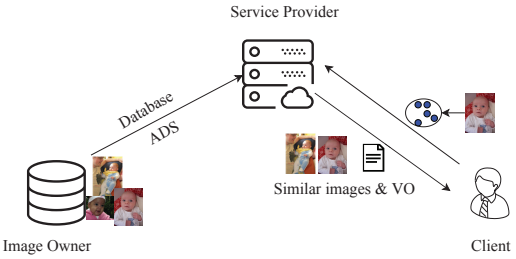

Fig. 3. Architecture of the proposed authenticated image retrieval system.

the results. More specifically, the results should satisfy the following security properties:

- **Soundness**: The results must be the images which are outsourced from the image owner and have not been tampered with.
- **Completeness**: The results include the k most similar images, i.e., the similarity values of the other images are smaller than those of the returned images.

Thus, the problem here is how to design the ADS so that efficient query processing and result verification can be supported. In what follows, we will propose two novel ADSs, together with the corresponding index search and verification algorithms. Note that the accuracy of our authenticated SIFT-based image search algorithms is the same as that of the original algorithms. In addition, our proposed ADSs and algorithms can be extended to work with other SIFT-based methods that improve the accuracy of image search, such as [25].

IV. Authenticated Randomized k-d Tree and Inverted Index with Cuckoo Filters

In this section, we propose two ADSs, namely the Merkle randomized k-d tree and the Merkle inverted index with cuckoo filters. They are used to achieve authenticated BoVW encoding and inverted index search, respectively.

A. Merkle Randomized k-d Tree

1) Data Structure: The first step of SIFT-based image retrieval is BoVW encoding, which is done by the AKM algorithm, as discussed in Section II. In order to authenticate this step, we propose the Merkle randomized k-d tree (MRKD-tree), which is an integration of the MH-tree [23] and the randomized k-d tree built on top of the pre-trained feature clusters.

The MRKD-tree consists of two types of nodes, internal nodes and leaf nodes. An internal node has three components, i.e., the splitting hyperplane, the pointers pointing to its child nodes, and a digest. The first two components are the same as in the randomized k-d tree. The digest is defined as follows.

Definition 2. (**Digest of Internal Node**) Let l_{N_i} be the splitting hyperplane of an internal node N_i of the MRKD-tree, and '|' be the string concatenation operator. The digest of N_i is defined as

$$h_{N_i} = h(l_{N_i}|h_{N_i^l}|h_{N_i^r}), \tag{5}$$

where $h_{N_i^l}$ ($h_{N_i^r}$) is the digest of the left (right) child of N_i.

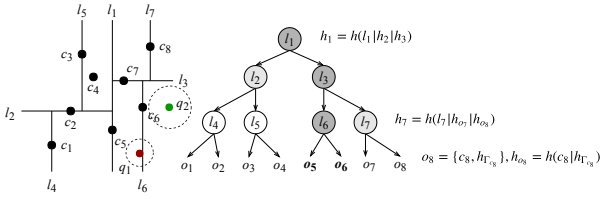

Fig. 4. An example of the MRKD-tree and VO generation for query q_1, q_2.

In contrast, a leaf node records a certain number of feature clusters, the digests of the clusters' inverted lists (detailed in the next section), and a digest about the leaf node itself.

Definition 3. (**Digest of Leaf Node**) Let τ be the number of feature clusters in a leaf node N_f. Each cluster c_i is associated with a digest of the corresponding Merkle inverted list $h_{\Gamma_{c_i}}$ (to be detailed in Section IV-B). The digest of N_f is defined as

$$h_{N_f} = h(c_1|h_{\Gamma_{c_1}}|\cdots|c_\tau|h_{\Gamma_{c_\tau}}). \tag{6}$$

Note that the digests of Merkle inverted lists are included in leaf nodes in order to connect the MRKD-tree with the ADS proposed for authenticated inverted index search.

MRKD-tree Example. Fig. 4 shows an example of an MRKD-tree. The tree first chooses l_1 to divide the plane into two subspaces. Then, the tree repeats the splitting operation until each leaf node contains only one cluster. After that, we can build the MRKD-tree through the following steps: (i) generate the digests of leaf nodes, e.g., $h_{o_8} = h(c_8|h_{\Gamma_{c_8}})$; (ii) compute the digests of internal nodes in a bottom-up fashion, e.g., h_7 is computed as $h(l_7|h_{o_7}|h_{o_8})$.

2) Authenticated Query Processing: The query of the MRKD-tree takes a set of feature vectors $Q = \{q_1, q_2, \cdots, q_{n_Q}\}$ and their corresponding thresholds $T = \{t_1, t_2, \cdots, t_{n_Q}\}$. For each feature vector q_i, we want to find the leaf nodes whose (minimum) distances to q_i are shorter than t_i and output the feature clusters in those leaf nodes.

Query Processing and VO Generation. We propose a new algorithm, called *MRKDSearch*, for authenticated query processing over the MRKD-tree. The main principle is that we generate a single VO for all feature vectors by maximizing the use of shared tree nodes.

Algorithm 1 describes the query processing and VO generation process. It takes the feature vectors Q, the thresholds T, and the root node as input. C and VO_C are initialized as $\emptyset$. The algorithm recursively traverses the MRKD-tree for all feature vectors in one go. If an internal node has a distance longer than t_i for some q_i, the subtree is opened for further exploration Otherwise, if there exists no q_i whose distance is longer than t_i, the subtree can be pruned and the node's digest is inserted into VO_C. Fig. 4 shows an example of *MRKDSearch* with two feature vectors q_1 and q_2. Suppose that the query results are c_5, c_6 for q_1 and c_6 for q_2. The VO contains the digests of all traversed nodes (marked in grey). Note that the digests of h_1 and h_7 are shared for the queries of q_1 and q_2.

Verification. After receiving C and VO_C, the client needs to run the following two subtasks to verify the integrity of the search results: (i) reconstruct and verify the correctness of the digest of root node, (ii) check whether all valid clusters

Algorithm 1: *MRKDSearch*: Searching clusters and generating verification object.

Input : Feature vectors Q, thresholds T, node N
Output: C, VO_C

if $q = \emptyset$ **then**
  Append h_N to VO_C;
  return
if N *is a leaf node* **then**
  Add the clusters in N to C;
  Append the clusters' digests $\{h_{\Gamma_{c_i}}\}$ to VO_C;
  return
Append l_N to VO_C;
Initialize q^l, q^r, t^l, t^r as $\emptyset$;
foreach q_i *in* q **do**
  if $dist(q_i, N) \leq t_i$ **then**
    Add q_i, t_i to q^l, t^l if $dist(q_i, N^l) \leq t_i$;
    Add q_i, t_i to q^r, t^r if $dist(q_i, N^r) \leq t_i$;
    // N^l, N^r, left and right children of N
MRKDSearch(q^l, t^l, N^l);
MRKDSearch(q^r, t^r, N^r);
return

are contained in C. Again, take Fig. 4 as an example. The client can reconstruct the digest of the root node h_1 from the VO for q_1, q_2. After verifying the correctness of the digest of the MRKD-tree, the client can check whether the distances between q_1 (resp. q_2) and leaf nodes o_5, o_6 (resp. o_6) are shorter than the given thresholds while the distances between q_1, q_2 and the internal nodes denoted by l_7 and l_2 are longer.

3) Complexity Analysis: For a query image with n_Q feature vectors, the VO size of the proposed *MRKDSearch* is $O\left(n_Q(\log n_C - \log n_Q)\right)$ in the worst case and $\log n_C$ in the best case (all feature vectors have the same result). Without sharing nodes, the total VO size would be $O(n_Q \log n_C)$ in both the worst and best cases. Obviously, the VO size of the proposed *MRKDSearch* is smaller than the scheme without sharing nodes and the benefit will increase with n_Q.

The verification process consists of verifying the soundness and completeness of the randomized k-d tree search. The time complexity of the verification is $O(n_Q \log n_C)$. Compared with the scheme without sharing nodes, the space complexity of the proposed scheme is reduced from $O(n_Q \log n_c)$ to $O\left(n_Q(\log n_c - \log n_Q)\right)$.

B. Merkle Inverted Index with Cuckoo Filters

1) Data Structure: The second step of SIFT-based image retrieval is inverted index search, and we use the impact-ordered inverted index as the basic data structure. To guarantee the query integrity of this step, we propose the Merkle inverted index with cuckoo filters.

Each Merkle inverted list Γ_{c_i} in the proposed Merkle inverted index consists of five components, i.e., associated cluster c_i, cluster weight w_{c_i}, posting list P_{c_i}, cuckoo filter Θ_{c_i}, and digest $h_{\Gamma_{c_i}}$. Each posting in P_{c_i} is a tuple of image id, impact value, and its digest. The first two elements are the same as in the postings of the original impact-ordered inverted index. The digest of a posting is defined as follows.

Definition 4. (**Digest of Posting**) Let $pos_{c_i,j}$ be a posting in P_{c_i}. The first two elements of $pos_{c_i,j}$ are I and p_{I,c_i}. The digest of $pos_{c_i,j}$ is defined as

$$h_{pos_{c_i,j}} = h(I|p_{I,c_i}|h_{pos_{c_i,j+1}}), \tag{7}$$

where $h_{pos_{c_i,j+1}}$ is the digest of the next posting.

The cuckoo filter of Γ_{c_i} is initialized by inserting all the image ids contained in P_{c_i}. The usage of the cuckoo filter will be introduced later in this section. After the cuckoo filer is built, we define the digest of Γ_{c_i} as follows.

Definition 5. (**Digest of Inverted List**) Let $h_{pos_{c_i,1}}$ be the digest of the first posting, w_{c_i} be the weight of c_i, and Θ_{c_i} be the initialized cuckoo filter. The digest of Γ_{c_i} is defined as:

$$h_{\Gamma_{c_i}} = h(w_{c_i}|h(\Theta_{c_i})|h_{pos_{c_1,1}}). \tag{8}$$

Note that, as discussed earlier in Section IV-A, this digest will be embedded in the MRKD-tree.

Example. Table II shows an example of the Merkle posting lists Γ_{c_5} and Γ_{c_6}. The weights of c_5 and c_6 are $2\sqrt{2}$ and $\sqrt{2}$, respectively. The digest of each posting is computed according to Definition 4, e.g., $h_{pos_{5,1}} = h(0.1|0.35|h_{pos_{5,2}})$.

2) Authenticated Query Processing: The inverted index search takes BoVW vector B_Q and parameter k as input, and finds the top-k most similar images. We are interested in the posting lists where $p_{Q,c_i} \neq 0$, as similar images can only be found from those lists.

Let $\Gamma_Q = \{\Gamma_{c_i}\}_{p_{Q,c_i}\neq 0}$, $P_Q = \{P_{c_i}\}_{p_{Q,c_i}\neq 0}$, and s_k be the similarity score between Q and the k-th most similar image I_{top_k}. To achieve authenticated inverted index search, the SP should prove to the client that except for $\{I_{top_i}\}_{i=1}^k$, all other images in P_Q have similarity scores smaller than s_k. Instead of returning all postings in P_Q, Pang et al. [15] developed a search algorithm that pops and returns part of the postings to prove the integrity of query results. With the help of cuckoo filters, we here propose an improved algorithm, called *InvSearch*, which aims to ensure the integrity of top-k search with fewer postings.

We start by defining some notations:

- $\overline{P}_{c_i}$: the list of postings that have been popped;
- $\widetilde{P}_{c_i}$: the list of remaining postings, $P_{c_i} = \overline{P}_{c_i} + \widetilde{P}_{c_i}$;
- $\overline{P}_I$: the set of popped posting lists where I appears;
- $S^L(Q, I)$ (resp. $S^U(Q, I)$): lower (resp. upper) bound of similarity between Q and I;
- π: maximal similarity score of the images in $\{\widetilde{P}_{c_i}\}$;
- π^U: upper bound of π.

To ensure the correctness of inverted index search, the similarity scores should satisfy the following termination conditions:

1. $s_k^L \geq \pi^U$, where s_k^L is the lower bound of s_k;
2. $s_k^L \geq S^U(Q, I)$, for each $I \in \{\overline{P}_{c_i}\}$ but $\notin \{I_{top_i}\}_{i=1}^k$.

The termination conditions mean that the similarities between Q and $\{I_{top_i}\}_{i=1}^k$ should be higher than those of the images not yet popped (Condition 1) or popped (Condition 2).

It is easy to compute a lower bound of $S(Q, I)$ by aggregating all similarity scores from $\overline{P}_I$, i.e.,

$$S^L(Q, I) = \sum_{c_i \in \overline{P}_I} p_{Q,c_i} \cdot p_{I,c_i}. \tag{9}$$

TABLE II
AN EXAMPLE OF THE MERKLE INVERTED INDEX AND THE PROCEDURE OF *PostingSearch* FOR A TOP-2 SEARCH.

c_i	$h_{\Gamma_{c_i}}$	w_{c_i}	Θ_i	**Posting Lists**						
c_5	$h(2\sqrt{2}\|h(\Theta_{c_5})\|h_{pos_{5,1}})$	$2\sqrt{2}$	Θ_{c_5}	$\mapsto$ $\langle 1, 0.34, h_{pos_{5,1}}\rangle$	$\langle 3, 0.26, h_{pos_{5,2}}\rangle$	$\langle 4, 0.25, h_{pos_{5,3}}\rangle$	$\langle 10, 0.17, h_{pos_{5,4}}\rangle$	$\langle 7, 0.11, h_{pos5,5}\rangle$	$\langle 2, 0.09, h_{pos_{5,6}}\rangle$	...
c_6	$h(\sqrt{2}\|h(\Theta_{c_6})\|h_{pos_{6,1}})$	$\sqrt{2}$	Θ_{c_6}	$\mapsto$ $\langle 5, 0.41, h_{pos_{6,1}}\rangle$	$\langle 8, 0.32, h_{pos_{6,2}}\rangle$	$\langle 3, 0.28, h_{pos_{6,3}}\rangle$	$\langle 6, 0.25, h_{pos_{6,4}}\rangle$	$\langle 4, 0.10, h_{pos_{6,5}}\rangle$	$\langle 9, 0.07, h_{pos_{6,6}}\rangle$	...

Step	**Popped Postings**			**Description**
1	$\langle 1, 0.34, h_{pos_{5,1}}\rangle$	$\langle 3, 0.26, h_{pos_{5,2}}\rangle$		top-2 images
	$\langle 5, 0.41, h_{pos_{6,1}}\rangle$	$\langle 8, 0.32, h_{pos_{6,2}}\rangle$	$\langle 3, 0.28, h_{pos_{6,3}}\rangle$	
2	$\langle 4, 0.25, h_{pos_{5,3}}\rangle$			Condition 1
3	$\langle 6, 0.25, h_{pos_{6,4}}\rangle$	$\langle 4, 0.1, h_{pos_{6,5}}\rangle$		Condition 2

A simple way to obtain S^U and π^U is to use their maximal upper bounds, i.e.,

$$S_{max}(Q,I) = \sum_{c_i \in P_Q} max(p_{Q,c_i} \cdot p_{c_i},\ p_{Q,c_i} \cdot p_{I,c_i}),$$
$$\pi_{max} = \sum_{c_i \in P_Q} p_{Q,c_i} \cdot p_{c_i}, \tag{10}$$

where p_{c_i} is the highest impact value of the postings in $\widetilde{P}_{c_i}$ and $p_{I,c_i} = 0$ if I does not appear in P_{c_i}.

The algorithm developed in [15] used the above maximal bounds to terminate a search. However, because S_{max} and π_{max} are maximal, the SP has to examine a huge number of postings to meet the termination conditions. It remains a challenge to tighten the upper bounds of $S(Q,I)$ and π.

The implicit assumption of Equation (10) is that every posting list in P_Q contains I. However, the probability that a posting list contains I is small because of the sparseness of BoVW vectors. Actually, determining whether I is in a posting list is a set membership problem. On the basis of this observation, we propose to use cuckoo filters to obtain tighter upper bounds of $S(Q,I)$ and π .

Taking advantage of the cuckoo filter, we can estimate whether an image I is in a posting list with a high probability. Thus, it is easy to obtain the set of posting lists $\{\widetilde{P}_{c_i}\}$ that contains I with a high probability. Denote this set as $\widetilde{P}_I$. Then, the upper bound of $S(Q,I)$ can be estimated as follows:

$$S^U(Q,I) = \sum_{c_i \in \widetilde{P}_I} max(p_{Q,c_i} \cdot p_{c_i}, p_{Q,c_i} \cdot p_{I,c_i}). \tag{11}$$

Let γ be an upper bound of the frequency of the most frequent image in $\{\widetilde{P}_{c_i}\}$. Further denote by $\widetilde{P}_{max_\gamma}$ the set of top-γ posting lists that have the largest values of $p_{Q,c_i} \cdot p_{c_i}$. The upper bound of π can be estimated as follows:

$$\pi^U = \sum_{c_i \in \widetilde{P}_{max_\gamma}} p_{Q,c_i} \cdot p_{c_i}. \tag{12}$$

Given a set of cuckoo filters $\{\Theta\}$, γ can be computed by scanning the buckets of all filters. As detailed in Algorithm 2, the algorithm goes through every bucket of all the filters and keeps track of the highest frequency of the fingerprints, max_{fp}. Finally, the algorithm returns the double of max_{fp}, since each image has two alternate buckets.

Lemma 1. Let π be the maximum similarity score of the images not yet popped and π^U be the value computed using Equation (12). We must have $\pi \leq \pi^U$.

Proof Sketch. We first prove that γ is an upper bound of the frequency of the most frequent image in $\{\widetilde{P}_{c_i}\}$. Let I_f be the most frequent image with fingerprint fp. The fingerprint and hash buckets of I are the same for all the cuckoo filters. As such, the frequency of I_f is equal to or less than the frequency of fp in the two alternate buckets of all cuckoo filters and further bounded by γ.

Let I' be the image with the similarity score π. Then

$$\pi = \sum_{c_i \in P_Q} p_{Q,c_i} \cdot p_{I',c_i} \leq \sum_{c_i \in \widetilde{P}_{max_{|I'|}}} p_{Q,c_i} \cdot p_{c_i} \tag{13}$$

where $|I'|$ is the frequency of I' in $\{\widetilde{P}_{c_i}\}$. Since $max_{|I'|} \leq max_{|I_f|} \leq \gamma$, we have

$$\pi \leq \sum_{c_i \in \widetilde{P}_{max_{|I'|}}} p_{Q,c_i} \cdot p_{c_i} \leq \sum_{c_i \in \widetilde{P}_{max_\gamma}} p_{Q,c_i} \cdot p_{c_i} = \pi^U. \tag{14}$$

□

Algorithm 2: *MaxCount*: Computing an upper bound of the frequency of the most frequent image in $\widetilde{P}$.

Input : Filter set $\{\Theta_{c_i}\}$
1 $max_{fp} \leftarrow 0$ **for** *each bucket* i **do**
2 $max'_{fp} \leftarrow$the count of most frequent fingerprint in the i-th buckets of all cuckoo filters in $\{\Theta_{c_i}\}$;
3 **if** $max_{fp} < max'_{fp}$ **then**
4 $max_{fp} \leftarrow max'_{fp}$
5 $\gamma \leftarrow 2 \cdot max_{fp}$;
6 **return** γ

Query Processing and VO Generation. Algorithm 3 outlines the algorithm to search the postings to meet the termination conditions. Theoretically, to prove the integrity of query results, the SP has to examine enough postings containing the top-k images $\{I_{top_i}\}_{i=1}^k$. On the basis of this observation, we first pop up all postings containing $\{I_{top_i}\}_{i=1}^k$ and their preceding postings (Line 1). The search algorithm then proceeds to pop up the relevant postings until the termination conditions are met.

Example. Table II illustrates the top-2 search procedure for the query $Q = \{q_1, q_2\}$ in Fig. 4. The BoVW vector of Q is (0,0,0,0,1,1,0,0) and $p_{Q,c_5} = 2, p_{Q,c_6} = 1$. The top-2 most similar images are 1 and 3. The *PostingSearch* algorithm executes as follows: (1) pop up the postings containing 1 and 3 and their preceding postings; (2) check Condition 1, compute γ (assuming $\gamma = 2$), and pop up $pos_{5,3}$ so that Condition 1 is met; (3) check Condition 2 and pop up $pos_{6,4}$ and $pos_{6,5}$ to make sure that image 4 has a similarity score smaller than 1 and 3. Assume that we know from the cuckoo filter Θ_{c_5} that images 5, 8, and 6 do not appear in Γ_{c_5}. Thus, it is guaranteed that they have a lower similarity, even without popping up additional postings.

Algorithm 3: *PostingSearch*: Searching through impact-ordered inverted index.

Input : BoVW vector B_Q, posting lists P

1 Examine the posting lists and pop up the postings containing the top-k images;
2 *UpdateBounds*;
3 **while** *Condition 1 fails* **do**
4 Pop up the postings until Condition 1 is met;
5 *UpdateBounds*;
6 **while** *Condition 2 fails* **do**
7 **for** *each image* I^* *whose* $S^U(Q, I^*) > s_k$ **do**
8 Pop up the postings until the ones containing I^*;
9 *UpdateBounds*;
10 **return** $\{I_{top_i}\}_{i=1}^k$, $\{\overline{P}_{c_i}\}$;
11 **Function** *UpdateBounds* **is**
12 Delete the popped images from their cuckoo filters;
13 **foreach** *popped image* I **do**
14 calculate or update $S^L(Q, I)$ and $S^U(Q, I)$;
15 $\gamma \leftarrow$ *MaxCount*$(\{\Theta_{c_i}\}_{p_{c_i} \neq 0})$;
16 $\pi \leftarrow \sum_{c_i \in P_{max_\gamma}} p_{Q,c_i} \cdot p_{c_i}$;

Algorithm 4: *InvSearch*: Searching and generating VO through impact-ordered inverted index.

Input : BoVW vector B_Q, Merkle posting lists Γ_Q

1 $\{I_{top_i}\}_{i=1}^k, \{\overline{P}_{c_i}\} \leftarrow$ *PostingSearch*(B_Q, P);
2 **foreach** $\Gamma_{c_i} \in \Gamma_Q$ **do**
3 Add w_{c_i} to VO_{inv};
4 **foreach** $\overline{P}_{c_i} \in \{\overline{P}_{c_i}\}$ **do**
5 **foreach** $pos \in \overline{P}_{c_i}$ **do**
6 Add $pos.I$, $pos.p_{i,c_i}$ to VO_{inv}
7 **if** $\widetilde{P_{c_i}}$ *is empty* **then**
8 Add $h_{\Theta_{c_i}}$ to VO_{inv};
9 **else**
10 Add the digest of the first posting in $\widetilde{P}_{c_i}$ to VO_{inv};
11 Add Θ_{c_i} to VO_{inv};
12 **return** $\{I_{top_i}\}_{i=1}^k$ and VO_{inv}

With the *PostingSearch* algorithm and the Merkle inverted index, we now present the *InvSearch* algorithm to perform a top-k search and generate the VO. Algorithm 4 describes the proposed algorithm. The SP first runs Algorithm 3 and obtains $\{I_{top_i}\}_{i=1}^k$ and $\overline{P}_{c_i}$ (Line 1). Then, the information needed to reconstruct the digest of each posting list is added to the VO (Lines 3-6). If all postings in a posting list are popped, the SP would return the digest of the cuckoo filter; otherwise the SP returns the cuckoo filter as part of the VO (Line 7-11). Note that, to prove the integrity of posting search, the SP also fetches the digest of the next posting of each posting list and adds it to the VO (Line 10). Finally, the SP returns $\{I_{top_i}\}_{i=1}^k$ and VO_{inv} to the client.

In the above example, the VO_{inv} of Q includes w_{c_5}, w_{c_6}, filters $\Theta_{c_5}, \Theta_{c_6}$, image ids 4, 5, 8, and 6 and their impact values, and the digests of the postings $pos_{5,4}$, $pos_{6,6}$.

Verification. After receiving $\{I_{top_i}\}_{i=1}^k$ and VO_{inv}, the client first reconstructs and verifies the digests of each posting list. If verified, the client proceeds to check the popped images. The upper bound π^U and the upper and lower bounds of the similarity score for each image will be computed. After that, the client checks whether the termination conditions are satisfied. The top-k images are verified as valid if all the above procedures are passed.

In our example, the client can reconstruct $h_{\Gamma_{c_5}}$ from w_{c_5}, Θ_{c_5}, and the corresponding posting information. After deleting the received images from the corresponding filters, the client can compute γ and π^U. Through computing the upper bounds of the received images, the client knows that 1 and 3 are indeed the top-2 similar images.

3) Complexity Analysis: There is no theoretical upper bound of the communication and computation costs of the proposed *InvSearch* algorithm. The proportion of the popped postings depends on the distribution of the inverted lists and parameter k. Ideally, the termination conditions are satisfied once the postings containing the top-k images are popped. In this case, the VO size is minimal. In the worst case, all postings in the relevant inverted lists have to be returned to prove the integrity of search results.

For the verification process, the client only needs to verify the correctness of the termination conditions via performing a series of hash operations, including reconstructing the digests of the relevant posting lists and deleting and looking up images from the cuckoo filters. Two hash operations are required for each deletion and lookup operation with the cuckoo filters, and the total number of these operations depends on the number of popped images.

V. ImageProof: Authenticated Image Retrieval

In this section, we present the overall ImageProof scheme. It uses the ADSs developed in the last section to support authenticated large-scale SIFT-based image retrieval.

A. ADS Generation

Given an image database, the image owner first signs each image with a signature of the image id I and its raw data img_I. Formally, the signature of an image I is

$$sig_I = sign(sk, h(I|h(img_I))), \tag{15}$$

where sk is the image owner's private key. Next, the image owner invokes the same index building procedures as those in a normal SIFT-based image retrieval system. During this step, feature vectors are extracted from raw image data, which are then used to train the feature clusters. After that, a forest of randomized k-d trees is built upon the trained clusters to facilitate BoVW encoding, and the impact-ordered inverted index is built on the basis of the encoded BoVW vectors, as discussed in Section II. Finally, the image owner applies the algorithms proposed in Section IV to generate the ADSs for the randomized k-d trees and inverted index: (i) for each posting list in the inverted index, the digests with cuckoo filters are generated; (ii) the MRKD-trees are constructed on the basis of the randomized k-d trees; (iii) the hash of the digests of the root nodes, as the digest of ImageProof, is generated and signed using the image owner's private key. With the generated ADSs, the image owner outsources the image database along with the indexes and ADSs to the SP. The image owner's public key and the database's signature are published to the clients. Fig. 5 gives an overview of the ADSs employed in our ImageProof scheme.

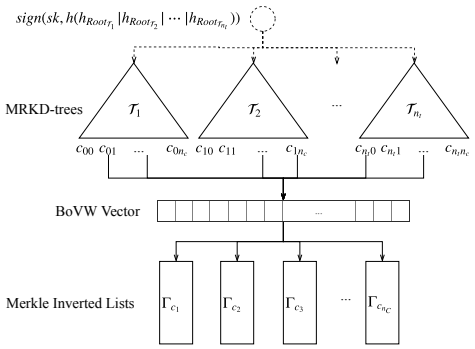

Fig. 5. An overview of ADSs for ImageProof.

Algorithm 5: Top-k Image Search: Searching similar images and generating VO.

Input : Feature vectors Q, search parameter k

Search across the randomized k-d trees and compute threshold t_i for each q_i;
for *each MRDK-tree* $\mathcal{T}_i$ **do**
  $VO_{C,i}, C_i \leftarrow$ *MRKDSearch*$(Q, t_i, Root_{\mathcal{T}_i})$;
$B_Q \leftarrow$ Count the frequency of the approximate nearest clusters;
$\{I_{top_i}\}_{i=1}^{k}, VO_{inv} \leftarrow$*InvSearch*$(B_Q, \Gamma_Q)$;
Fetch signatures $\{sign_i\}_{i=1}^{k}$ of $\{I_{top_i}\}_{i=1}^{k}$;
$VO \leftarrow \big\{\{VO_{C,i}\}_{i=1}^{n_t}, \bigcup_{i=1}^{n_t} C_i, \{sig_{top_i}\}_{i=1}^{k}\big\}$;
return $\{I_{top_i}\}_{i=1}^{k}$ and VO

B. Authenticated Query Processing

Upon receiving the search parameter k and a set of feature vectors $Q = \{q_1, q_2, \cdots, q_{n_Q}\}$ from the client, the SP finds the top-k most similar images and computes the VO using the indexes and ADSs outsourced by the image owner. More specifically, the SP first searches the approximate nearest neighbors and finds the auxiliary threshold for each feature vector. Then, the SP uses Algorithm 1 to compute $\{VO_{C,i}\}_{i=1}^{n_t}$ for the BoVW encoding, where n_t is the number of MRKD-trees. After that, the SP encodes the BoVW B_Q, and next runs Algorithm 4 to search the top-k images and generate VO_{inv} for the inverted index search. Finally, the SP combines $\{VO_{C,i}\}_{i=1}^{n_t}$, $\bigcup_{i=1}^{n_t} C_i$, VO_{inv}, and the corresponding image signatures as the final VO, and sends it, together with the top-k results, to the client. The overall query processing algorithm is summarized in Algorithm 5.

Example. We now present the complete authenticated query processing for the query $Q = q_1, q_2$ in Fig. 4. The SP first obtains the VO_C and cluster candidates c_5, c_6 by executing Algorithm 1. The BoVW vector of Q is encoded as (0,0,0,0,1,1,0,0). Then, the SP runs Algorithm 4 to search the top-2 similar images 1 and 3 and generate VO_{inv}. The final VO of the whole query is $VO = \{VO_C, \{c_5, c_6\}, VO_{inv}, \{sig_1, sig_3\}\}$, where sig_1 and sig_3 are the signatures of images 1 and 3, respectively.

C. Result Verification

After receiving the top-k image results and the VO, the client verifies the integrity of the image retrieval as follows: (i) check the correctness of the termination conditions and compute the digests of the posting lists; (ii) verify the integrity of BoVW encoding; (iii) verify the integrity of the MRKD-trees; (iv) verify the integrity of raw image data. The verification procedure of Q in the above example is an integration of the verification procedures in sections IV-A and IV-B. The client first reconstructs the digests of $\Gamma_{c_5}, \Gamma_{c_6}$ and the digest of ImageProof. Then, the client checks whether the digest of MRKD-trees is valid against the published signature. Finally, the client also needs to verify the signatures of sig_1, sig_3.

D. Security Analysis

Theorem 1. Our proposed authenticated image retrieval scheme, ImageProof, satisfies the security properties defined in Section III.

Proof Sketch. We classify the possible attacks into three cases:

1) the SP forges a BoVW vector B'_Q, which is different from the genuine BoVW vector B_Q;
2) the SP computes B_Q honestly but forges a top-k image set $\{I'_{top_i}\}_{i=1}^{k}$, which is different from the genuine image set $\{I_{top_i}\}_{i=1}^{k}$;
3) the SP computes B_Q, $\{I_{top_i}\}_{i=1}^{k}$ honestly, but returns fake image data.

For case 3), due to the cryptographic digital signature scheme, the SP (any probabilistic polynomial-time adversary) cannot forge a legal signature with a non-negligible probability which satisfies $sign(sk, h(I, img_I)) = sign(sk, h(I', img_{I'}))$, where $I \neq I'$. Therefore, the case 3) attack cannot succeed. For case 1), in order to forge a legal B'_Q, the SP has to tamper with some nodes of the MRKD-trees. In this case, due to the collision resistance property of the hash function, the digest of the MRKD-trees would be different from the original one. Thus, this attack would fail when the client verifies the digest against the signature of the MRKD-trees. For case 2), similar to case 1), the SP has to tamper with some cuckoo filters or postings of the Merkle inverted index in order to generate a legal top-k image set. Thus, the digests of the corresponding posting lists would deviate from the original ones and, hence, could not pass the verification. □

VI. Optimizations

This section proposes two optimization techniques that further improve the performance of ImageProof.

A. Compressing Nearest Neighbor Candidates

Recall that in Algorithm 5, a set of clusters are generated for each feature vector as part of the VO by searching the MRKD-trees. To verify the integrity of BoVW encoding, the client needs to check the correctness of the nearest neighbor among all the candidates.

Inspired by [17], instead of returning each cluster in full dimensions, we return some partial dimensions of a cluster which are enough to prove whether the cluster is the nearest neighbor among all candidates. Let c_{q_i} be the nearest cluster of q_i among all candidates. To prove that $dist(q_i, c_i) \geq dist(q_i, c_{q_i})$ for another candidate c_i, the SP only needs to return part of c_i, c'_i, such that $dist(q_i, c'_i) \geq dist(q_i, c_{q_i})$. We use an MH-tree to guarantee the integrity of c'_i and the digest of the MH-tree is embedded into the MRKD-trees.

In practice, the clusters generated for different feature vectors may overlap. To further reduce the communication complexity, we leverage a sharing strategy to generate cluster candidates. Assume that c_i is a nearest cluster candidate for both q_i and q_j. The SP returns part of c_i, c_i', such that $dist(q_i, c_i') > dist(q_i, c_{q_i})$, $dist(q_j, c_i') > dist(q_j, c_{q_j})$.

With this optimization, because the SP has to find c' and the client needs to reconstruct the digest of the MH-tree, the computational cost would increase. There is a trade-off between the computation and communication costs.

B. Frequency-Grouped Inverted Index

Instead of using an ordinary impact-ordered inverted index, the second optimization proposes to use a frequency-grouped inverted index as the underlying structure to improve the performance of ImageProof. Here, the frequency-grouped inverted index is a special kind of impacted-ordered inverted index, which groups the images with the same frequency count into one posting. The new posting consists of two components, i.e., the frequency count and a list of image ids with the L_2-norm values of their corresponding BoVW vectors. The list is ordered in ascending order of the L_2-norm values. The impact value of the posting is set to that of the first image, which is also the largest impact value of all images in the list. The frequency-grouped inverted list is sorted in descending order of the impact values of the postings. The usage of frequency-grouped inverted index is motivated by the following observations. First, in a typical inverted list, most frequency counts are small [26] and images with the same frequency count can be combined into a prefix component. Second, the L_2-norm value of an image's BoVW vector are constant, which means they can be shared in multiple posting lists.

Data Structure. Taking advantage of the frequency-grouped inverted index introduced above, we propose a new ADS, the frequency-grouped Merkle inverted index with cuckoo filters.

Similar to the impact-ordered Merkle inverted index, each frequency-grouped Merkle inverted list $\Gamma^f_{c_i}$ in the proposed data structure consists of five components, i.e., associated cluster c_i, cluster weight w_{c_i}, cuckoo filter Θ_{c_i}, posting list $P^f_{c_i}$, and digest $h_{\Gamma^f_{c_i}}$. Each posting in $P^f_{c_i}$ is a tuple of frequency, a list of image ids with the L_2-norm values of their corresponding BoVW vectors, and its digest. The digest of a posting is defined as follows.

Definition 6. (**Digest of Posting**) Let $l_{B_{I_i}}$ be the L_2-norm value of the BoVW vector of I_i, $pos^f_{c_i,j}$ be a posting in $P^f_{c_i}$, and $(I_1, l_{B_{I_1}}; \cdots; I_{n_g}, l_{B_{I_{n_g}}})$ be the list of images and their corresponding L_2-norm values. The digest of $pos^f_{c_i,j}$ is defined as:

$$h_{pos^f_{c_i,j}} = h(I_1|l_{B_{I_1}}|\cdots|I_{n_g}|l_{B_{I_{n_g}}}|h_{pos^f_{c_i,j+1}}). \quad (16)$$

Each Merkle inverted list $\Gamma^f_{c_i}$ is also associated with a cuckoo filter, Θ_{c_i}, which is initialized with all image ids in $P^f_{c_i}$. With the cuckoo filer, we define the digest of $\Gamma^f_{c_i}$ as follows.

Definition 7. (**Digest of Inverted List**) Let $h_{pos^f_{c_i,1}}$ be the digest of the first posting in $P^f_{c_i}$, w_{c_i} be the weight of c_i, and Θ_{c_i} be the initialized cuckoo filter. The digest of $\Gamma^f_{c_i}$ is defined as

$$h_{\Gamma^f_{c_i}} = h(w_{c_i}|h(\Theta_{c_i})|h_{pos^f_{c_i,1}}). \quad (17)$$

TABLE III
AN EXAMPLE OF THE POSTING LIST OF A FREQUENCY-GROUPED MERKLE INVERTED INDEX.

Component	Value
c_i	c_5
$h_{\Gamma^f_{c_i}}$	$h(2\sqrt{2}\|h(\Theta_{c_5})\|h^f_{pos_{5,1}})$
w_{c_i}	$2\sqrt{2}$
Θ_{c_i}	Θ_{c_5}
Posting List	$\langle 4, (1, 33.3; 10, 66.6), h_{pos^f_{5,1}}\rangle$
	$\langle 5, (3, 54.4), h_{pos^f_{5,2}}\rangle$
	$\langle 3, (4, 33.9; 7, 77.1; 2, 94.3), h_{pos^f_{5,3}}\rangle$
	...

Example. Table III shows an example of the frequency-grouped version $\Gamma^f_{c_5}$ of Γ_{c_5} in Table II. $\Gamma^f_{c_5}$ consists of c_5, w_{c_5}, Θ_{c_5}, $h_{\Gamma^f_{c_5}}$, and the posting list, among which c_5, w_{c_5}, and Θ_{c_5} are the same as in Γ_{c_5}. We show the first three postings, and they are sorted in descending order of their impact values. For example, $pos^f_{5,1}$ stores (i) frequency 4 count, (ii) image ids 1 and 10 and their respective L_2-norm values 33.3 and 66.6, and (iii) the digest of $pos^f_{5,1}$, $h^f_{pos_{5,1}} = h(4|1|33.3|10|66.6|h_{pos^f_{5,2}})$.

The communication cost can be further reduced by using the compaction and compression techniques, such as compact representation of frequency counts and d-gaps [26], [27]. The compact representation of frequency counts is mainly based on the observation that the counts are usually small and can be more efficiently encoded [26]. In $\Gamma^f_{c_i}$, d-gaps can sort the images in each posting (expect the first image) in document-order and take the differences between consecutive values for compression.

VII. EXPERIMENTS

In this section, we experimentally evaluate the performance of the proposed ADSs and query processing techniques.

A. Experimental Setup

For performance evaluation, we use the MirFlickr1M dataset [28], which consists of one million images collected from Flickr. Two feature descriptors are involved, i.e., SIFT and speeded up robust features (SURF) [3]. A SIFT (resp. SURF) feature vector is a 128 (resp. 64) -dimensional vector, and about 800 (resp. 700) million feature vectors are extracted from the dataset, from which feature clusters are trained.

In the experiments, we use the recommended parameters for AKM [29]. The number of randomized k-d trees, n_t, is 8, and each leaf node has two clusters at most. The AKM search stops after exploring 32 leaf nodes. Following [24], the capacity of the cuckoo filters is set to 60% of the maximal length of the posting lists and the fingerprint size is 8 bits. The experimental results are obtained on the basis of an average of 10 randomly selected query images. Because the number of feature vectors varies with different images, we generate query feature vectors by choosing the first certain number of feature vectors extracted from the query image. The default settings

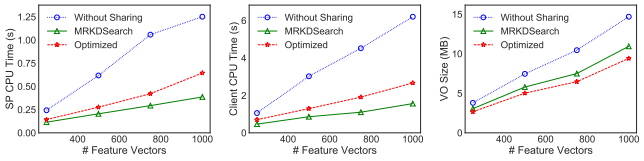

Fig. 6. BoVW performance as the number of SIFT feature vectors in a query increases.

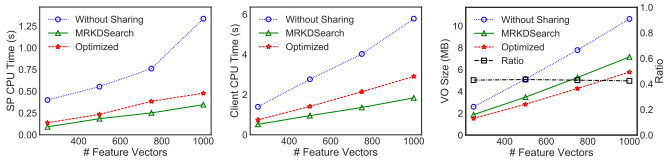

Fig. 7. BoVW performance as the number of SURF feature vectors in a query increases.

in the experiments are 10 (k), 1 million (codebook size), 0.5 million (dataset size), and 500 (the number of feature vectors in a query). We run our experiments on CentOS with Xeon 2.2 GHz E5-2630 v4 CPU and 256 GB RAM. Further, we choose SHA3-256 as the cryptographic hash function.

Three query authentication schemes are tested in our experiments.

- **Baseline**: The scheme that just combines the proposed *MRKDSearch* without sharing nodes and the scheme in [15]. Instead of checking the termination conditions for each popped posting, we execute the checking operations after popping up a certain number of postings to improve the efficiency of [15].
- **ImageProof**: The scheme proposed in Section V.
- **Optimized**: The ImageProof scheme optimized with the techniques presented in Section VI.

B. BoVW Encoding Evaluation

In this set of experiments, we focus on the BoVW encoding step and evaluate the performance of the three schemes by varying the number of feature vectors in a query and the size of the codebook (i.e., trained clusters).

1) Varying # Feature Vectors: Figs. 6 and 7 show the performance of the three schemes as the number of feature vectors in a query increases. In general, the two proposed schemes, *MRKDSearch* and its optimization, perform better than the baseline scheme. The performance gap increases as the number of feature vectors increases.

While *MRKDSearch* achieves the best performance in terms of computation cost for both the SP and the client, its optimization has the best performance in terms of communication cost. The reason is that in the optimization, both the SP and the client need to do more computation in order to obtain a smaller sized feature vector. It is a trade-off between the communication and computation cost. Because SIFT and SURF exhibit similar performance trends, we use the SURF descriptor in the following experiments.

2) Varying Codebook Size: Fig. 8 shows the results of BoVW encoding as the codebook size increases. With an increase in the codebook size from 0.25 to 1 million, the height of the MRKD-trees only increases by one or two. Thus, the query and verification costs are almost the same, while the

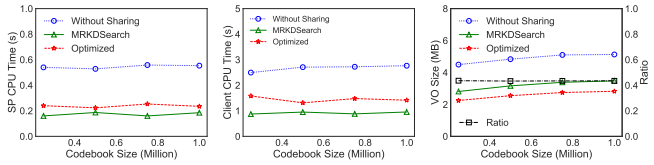

Fig. 8. BoVW performance as the codebook size increases.

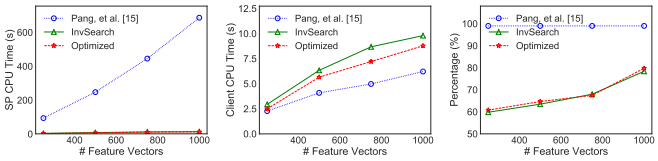

Fig. 9. Inverted index performance as the number of feature vectors increases.

VO size increases slightly. The performance of the proposed schemes is insensitive to the codebook size.

3) Ratio of Shared Nodes: In Figs. 7 and 8, we also plot the average ratio of the shared nodes against all traversed nodes. From Fig. 7, we can observe that the average ratio decreases slightly as the number of feature vectors increases. Fig. 8 shows that the average ratio is stable for different codebook sizes. The results indicate that the average ratio varies between 0.4 to 0.5, which justifies the motivation for sharing MRKD-tree nodes during BoVW encoding.

C. Inverted Index Evaluation

We now study the performance of the authenticated inverted index schemes under different settings, including the number of feature vectors in a query, the codebook size, and the parameter k. We report the SP CPU time, the client CPU time, and the percentage of popped postings.

1) Varying # Feature Vectors: Fig. 9 plots the results of the three authenticated inverted index schemes as the number of feature vectors in a query increases. The scheme proposed in [15] takes hundreds of seconds to process a query. Because of the loose upper bounds, its search cannot stop until almost all postings are popped up. The proposed *InvSearch* and its optimization can stop earlier and take less time in searching for similar images.

2) Varying Codebook Size: Fig. 10 shows the impact of codebook size. As the codebook size increases, the number of posting lists increases and the average length of posting lists decreases. Thus, the SP CPU time and the client CPU time of all schemes decrease. While nearly all postings are popped up in the baseline scheme, the popped postings in *InvSearch* and its optimization generally decrease as the codebook size increases.

3) Varying Parameter k: In Fig. 11, we show the impact of k on the performance of the three schemes. Two observations are made. First, as k increases, more postings will be returned to prove the integrity of the search results. Thus, the percentage of popped postings increases for both *InvSearch* and the optimized scheme. Second, while the optimized scheme reduces the VO size and hence the client CPU time, it has a similar SP CPU time to *InvSearch*. This is because the optimization groups the images with the same frequency count in each posting list of the inverted index, which does not affect the termination conditions of the image search process.

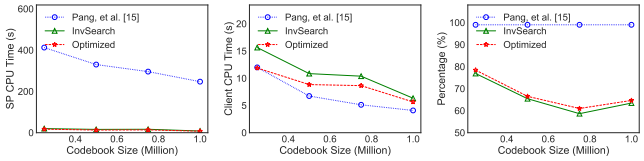

Fig. 10. Inverted index performance as codebook size increases.

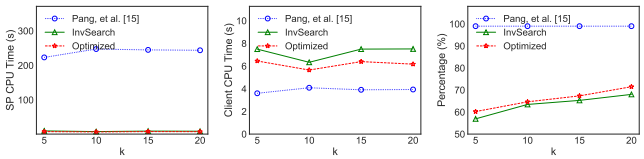

Fig. 11. Inverted index performance as k increases.

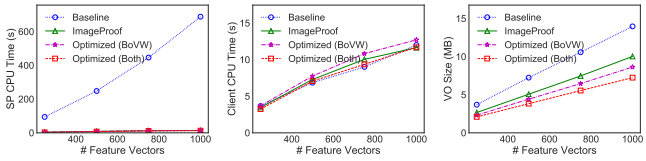

Fig. 12. Overall performance as the number of feature vectors increases.

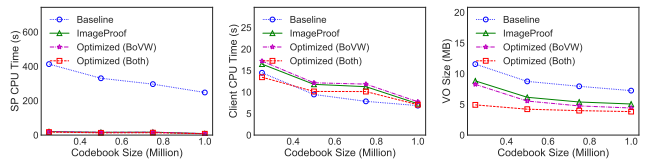

Fig. 13. Overall performance as codebook size increases.

D. Overall Evaluation

Finally, we present the overall performance of the complete authenticated image retrieval schemes. We show the performance advantages of each proposed optimization separately. We denote the ImageProof scheme with the first optimization of BoVW encoding as Optimized (BoVW), and that with the second optimization of Merkle inverted index as well as Optimized (Both).

1) Varying # Feature Vectors: Fig. 12 plots the results with different numbers of feature vectors in a query. We can observe that the communication and computation costs of all schemes increase with the number of feature vectors. As expected, Optimized (BoVW) reduces the VO size at the expense of the client CPU time. The client CPU time of Optimized (Both) is smaller than ImageProof. This is because in Optimized (Both), the images with the same frequency count are grouped and, hence, the time needed to reconstructing the digest of each frequency-grouped posting list is reduced.

2) Varying Codebook Size: The effect of different codebook sizes is shown in Fig. 13. The communication and computation costs of all schemes decrease as the codebook size increases. This is mainly due to the reduced number of images indexed in each inverted list.

3) Varying Dataset Size: Fig. 14 shows the results as the dataset size increases from 0.25 to 1 million. It is observed that, consistent with the previous experiments, while ImageProof has an inferior client CPU time, its SP CPU time and VO size are much lower than those of Baseline. Comparing the two optimizations of ImageProof, Optimized (Both) achieves a better performance in terms of both the client CPU time and the VO size. Because of the compact representation of frequency-grouped inverted index in Optimized (Both), this performance advantage increases with the dataset size.

VIII. Related Work

The key to achieving query authentication is to design ADSs through integrating existing indexes with cryptographic tools. According to different types of databases, we can classify existing query authentication schemes into two categories, i.e., schemes for low-dimensional and schemes for high-dimensional databases. In the following, we review the most relevant research in each category.

Schemes for Low-dimensional Databases. Low-dimensional databases include databases such as spatial databases, key-value databases, and most relational databases. The indexes for low-dimensional databases are B-tree, R-tree, k-d tree, and so on. Combining these indexes with the MH-tree, several authenticated indexes have been designed for low-dimensional databases, including Merkle B-tree [22], Merkle R-tree [18], and Merkle kd-tree [14], [30]. Taking the advantage of the MH-tree, the SP only needs to return part of the ADS as VO and then the client can verify the integrity of query results with the VO which is much smaller than the original index. More recent studies have extended these ADSs to advanced applications [31], [32]. Because low-dimensional index structures are inefficient for high-dimensional data, the above query authentication schemes cannot be applied to image databases directly.

Schemes for High-dimensional Databases. High-dimensional databases include databases of documents, images, etc. The indexes vary according to different search systems and authentication schemes have been proposed accordingly.

Papadopoulos et al. [17] proposed a general scheme for high-dimensional database authentication on the basis of a multi-step nearest neighbor framework and the MH-tree. In [19], Papadopoulos et al. decomposed a d-dimensional range query into d separate 1-dimensional queries and designed a range query authentication scheme for arbitrary dimensions. However, the above two schemes are not sufficient for CBIR. In particular, for the scheme proposed in [17], the communication cost would increase rapidly as the dataset size increases; for the scheme developed in [19], the VO size grows linearly with the number of dimensions, which make it less impractical for images with hundreds or thousands of dimensions.

Query authentication schemes have also been proposed for document databases [15], [16], [33]. For example, Pang et al. [15] proposed the first authenticated inverted index for document search. They designed an ADS based on the inverted index and MH-tree, and proposed corresponding search algorithms to reduce the communication complexity between the SP and the client. Goodrich et al. [33] proposed an authenticated web crawler for conjunctive keyword search. Two cryptographic tools, i.e., the MH-tree and set accumulator, are used to protect the integrity of inverted index search. More recently, Sun et al. [10] and Wan et al. [16] proposed two authentication schemes for conjunctive keyword search and

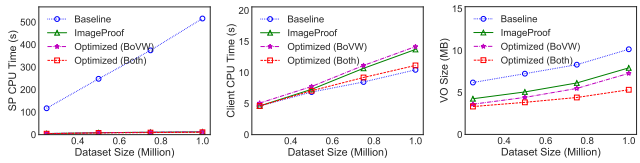

Fig. 14. Overall performance as dataset size increases.

top-k search on encrypted document databases, respectively.

However, the above authentication schemes for document retrieval cannot be applied to SIFT-based image retrieval for two reasons. First, these authentication schemes are just for query authentication of inverted index search and cannot handle authenticated BoVW encoding. Second, these schemes still suffer from some drawbacks. For example, in [15], the SP needs to update and check multiple termination conditions during each step of a posting search. Such operations are time-consuming and the cost would increase rapidly with large posting lists. How to achieve query authentication while maintaining the efficiency for large-scale image retrieval has not been investigated in the literature.

IX. Conclusion

In this paper, we have studied the query authentication problem for outsourced image databases. We have proposed a new authenticated scheme, ImageProof, for large-scale SIFT-based image retrieval with large or medium-sized codebooks. The proposed scheme consists of two novel ADSs, the Merkle randomized k-d tree and the Merkle inverted index with cuckoo filters, to ensure the integrity of search results. We have also proposed the corresponding search and verification algorithms along with several optimization techniques to improve the efficiency of query processing. Experimental results demonstrate that the performance of the proposed ImageProof outperforms the Baseline scheme. The proposed optimization techniques are also capable of improving the efficiency of ImageProof. For future work, we plan to study query authentication problems for other popular image retrieval systems such as convolutional neural network-based image retrieval.

Acknowledgement

The authors are grateful to the anonymous reviewers for their valuable comments and suggestions that improved the quality of this paper. This work was supported by grants from the HK-RGC under Project Nos. 12201018, 12244916, and C1008-16G. Tao Xiang was supported by the National Natural Science Foundation of China under grant No. 61672118.

References

[1] L. Zheng, Y. Yang, and Q. Tian, "SIFT meets CNN: A decade survey of instance retrieval," *TPAMI*, vol. 40, no. 5, pp. 1224–1244, 2018.
[2] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *IJCV*, vol. 60, no. 2, pp. 91–110, 2004.
[3] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," *CVIU*, vol. 110, no. 3, pp. 346–359, 2008.
[4] A. Dutta, R. Arandjelovic, and A. Zissermann, "VGG image search engine (VISE)," http://www.robots.ox.ac.uk/vgg/software/vise/, 2016.
[5] H. Hu, Y. Wang, L. Yang, P. Komlev, L. Huang, J. Huang, Y. Wu, M. Merchant, and A. Sacheti, "Web-scale responsive visual search at bing," in *SIGKDD*, 2018.
[6] Y. Jing, D. Liu, D. Kislyuk, A. Zhai, J. Xu, J. Donahue, and S. Tavel, "Visual search at pinterest," in *SIGKDD*, 2015, pp. 1889–1898.
[7] M. Li, M. Zhang, Q. Wang, S. S. Chow, M. Du, Y. Chen, and C. Li, "InstantCryptoGram: Secure image retrieval service," in *INFOCOM*, 2018.
[8] L. Weng, L. Amsaleg, A. Morton, and S. Marchand-Maillet, "A privacy-preserving framework for large-scale content-based information retrieval," *TIFS*, vol. 10, no. 1, pp. 152–167, 2015.
[9] C. Xu, Q. Chen, H. Hu, J. Xu, and X. Hei, "Authenticating aggregate queries over set-valued data with confidentiality," *TKDE*, vol. 30, no. 4, pp. 630–644, 2018.
[10] W. Sun, X. Liu, W. Lou, Y. T. Hou, and H. Li, "Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data," in *INFOCOM*, 2015, pp. 2110–2118.
[11] L. Kahney, "Cheaters bow to peer pressure," *Wired Magazine*, vol. 2, 2001.
[12] B. Zhang, B. Dong, and H. Wang, "Integrity authentication for sql query evaluation on outsourced databases: A survey," *arXiv preprint*, 2018.
[13] C. Xu, J. Xu, H. Hu, and M. H. Au, "When query authentication meets fine-grained access control: A zero-knowledge approach," in *SIGMOD*, 2018, pp. 147–162.
[14] F. Li, K. Yi, M. Hadjieleftheriou, and G. Kollios, "Proof-infused streams: Enabling authentication of sliding window queries on streams," in *VLDB*, 2007, pp. 147–158.
[15] H. Pang and K. Mouratidis, "Authenticating the query results of text search engines," *PVLDB*, vol. 1, no. 1, pp. 126–137, 2008.
[16] Z. Wan and R. H. Deng, "VPSearch: Achieving verifiability for privacy-preserving multi-keyword search over encrypted cloud data," *TDSC*, 2017.
[17] S. Papadopoulos, L. Wang, Y. Yang, D. Papadias, and P. Karras, "Authenticated multistep nearest neighbor search," *TKDE*, vol. 23, no. 5, pp. 641–654, 2011.
[18] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios, "Authenticated indexing for outsourced spatial databases," *VLDBJ*, vol. 18, no. 3, pp. 631–648, 2009.
[19] D. Papadopoulos, S. Papadopoulos, and N. Triandopoulos, "Taking authenticated range queries to arbitrary dimensions," in *CCS*, 2014, pp. 819–830.
[20] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *CVPR*, 2007, pp. 1–8.
[21] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *CVPR*, vol. 2, 2006, pp. 2161–2168.
[22] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Dynamic authenticated index structures for outsourced databases," in *SIGMOD*, 2006, pp. 121–132.
[23] R. C. Merkle, "A certified digital signature," in *CRYPTO*, 1989, pp. 218–238.
[24] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *CoNEXT*, 2014, pp. 75–88.
[25] L. Zheng, S. Wang, Z. Liu, and Q. Tian, "Fast image retrieval: Query pruning and early termination," *TMM*, vol. 17, no. 5, pp. 648–659, 2015.
[26] J. Zobel and A. Moffat, "Inverted files for text search engines," *CSUR*, vol. 38, no. 2, p. 6, 2006.
[27] H. Yan, S. Ding, and T. Suel, "Inverted index compression and query processing with optimized document ordering," in *WWW*, 2009, pp. 401–410.
[28] M. J. Huiskes, B. Thomee, and M. S. Lew, "New trends and ideas in visual concept detection: the mir flickr retrieval evaluation initiative," in *MIR*, 2010, pp. 527–536.
[29] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *VISSAPP*, 2009, pp. 331–340.
[30] K. Mouratidis, D. Sacharidis, and H. Pang, "Partially materialized digest scheme: An efficient verification method for outsourced databases," *VLDBJ*, vol. 18, no. 1, pp. 363–381, 2009.
[31] D. Wu, B. Choi, J. Xu, and C. S. Jensen, "Authentication of moving top-k spatial keyword queries," *TKDE*, vol. 27, no. 4, pp. 922–935, 2015.
[32] H. Hu, J. Xu, Q. Chen, and Z. Yang, "Authenticating location-based services without compromising location privacy," in *SIGMOD*, 2012, pp. 301–312.
[33] M. T. Goodrich, C. Papamanthou, D. Nguyen, R. Tamassia, C. V. Lopes, O. Ohrimenko, and N. Triandopoulos, "Efficient verification of web-content searching through authenticated web crawlers," *PVLDB*, vol. 5, no. 10, pp. 920–931, 2012.