Privacy-Preserving Content-Based Image Retrieval in the Cloud (Extended Version)

Bernardo Ferreira, João Rodrigues, João Leitão, Henrique Domingos
Nova University of Lisbon & NOVA LINCS
bernardo.ferreira.pt@ieee.org,jm.rodrigues@campus.fct.unl.pt,{jc.leitao,hj}@fct.unl.pt

Abstract

Storage requirements for visual data have been increasing in recent years, following the emergence of many new highly interactive multimedia services and applications for both personal and corporate use. This has been a key driving factor for the adoption of cloud-based data outsourcing solutions. However, outsourcing data storage to the Cloud also leads to new challenges that must be carefully addressed, especially regarding privacy. In this paper we propose a secure framework for outsourced privacy-preserving storage and retrieval in large image repositories. Our proposal is based on IES-CBIR, a novel Image Encryption Scheme that displays Content-Based Image Retrieval properties. Our solution enables both encrypted storage and searching using CBIR queries while preserving privacy. We have built a prototype of the proposed framework, formally analyzed and proven its security properties, and experimentally evaluated its performance and precision. Our results show that IES-CBIR is provably secure, allows more efficient operations than existing proposals, both in terms of time and space complexity, and enables more reliable practical application scenarios.

I. Introduction

Nowadays visual data is responsible for one of the largest shares of global Internet traffic in both corporate and personal use scenarios [1]. The amount of images, graphics, and photos being generated and shared everyday is growing at an ever increasing rate. The storage needs for such large amounts of data has been a driving factor for data outsourcing services such as the ones leveraging Cloud Storage and Computing solutions. Such services (*e.g.* Instagram and Flickr) have been reported to be among the largest growing internet services [2]. Additionally, the availability of large amounts of images in public and private repositories naturally leads to the need for content-based search and retrieval solutions (CBIR) [3].

Despite the fact that data outsourcing (*i.e.* resorting to cloud computing infrastructures) seems a natural solution to support large scale image storage and retrieval systems, it actually raises new challenges in terms of data control and privacy. This is a consequence of outsourcing data, which usually implies releasing control (and some times even full ownership) over it [4]. Recent news have provided clear proofs that privacy should not be expected to be preserved from Cloud providers [5], [6]. Furthermore, malicious system administrators working for the providers have full access to data on the hosting cloud machines [7], [8]. Finally, external hackers can exploit software vulnerabilities to gain unauthorized access to servers [9]. The recent incident with the iCloud image storage service and celebrity photo leakage [10] illustrates, in some part, the importance of these threats for cloud-based visual data stores.

The conventional approach to address privacy in this context is to encrypt sensitive data before outsourcing it and run all computations on the client side [11]. However, this imposes too much client-overhead, as data must continuously be downloaded, decrypted, processed and securely re-uploaded. Many applications cannot cope with this overhead, particularly online and mobile applications operating over very large datasets such as image repositories with CBIR services. A more viable approach would be to outsource computations and perform operations over the encrypted data on the server side. Existing proposals in this domain remain on the theoretical realm, namely those requiring fully homomorphic encryption, which are still computationally too expensive [12]. Nonetheless, partially homomorphic encryption schemes [13]–[16] and symmetric-key solutions designed for addressing specific searching problems [17]–[19] are an interesting alternative, yielding more practical results while providing a good tradeoff

Scheme	Information Leakage	Search Time	Update Time	Local Index Size	CBIR Alg.
SSE [17]	$ID_I + ID_{vw_I}$	$O(FE_I$+$Cluster_{fv_I}$+put(vw_I))	$O(E_{AES}(I)$+put(C_I) + get(Idx) + $D_{OPE}(Idx)$+FE_I+$Cluster_{fv_I}$+ $Update_{vw_I}(Idx)$+$E_{OPE}(Idx)$+put(Idx))	$O(\|CB\|$+$\|vw\| \times \|Rep\|)$	Local Color
PKHE [15]	ID_I+$size_I$+ID_{vw_I}	$O(E_{Paillier}(I)$+put(C_I))	($O(E_{Paillier}$(I) + put(C_I))	–	SIFT
This Work	ID_I+$size_I$+ID_{vw_I}	$O(E_{IES-CBIR}(I)$+put(C_I))	$O(E_{IES-CBIR}(I)$ + put(C_I))	–	Global Color

TABLE I: Overview of information leakage and average complexities (on the client-side) for the most relevant privacy-preserving CBIR approaches and our work, where: ID_I is a deterministic identifier of an image I being stored/updated or being searched for as query image; put(x) and get(x) represent the complexity of respectively, sending and retrieving data item x to/from the server; FE_I is the *Feature Extraction* of I and *fv* is the extracted *Feature-Vector*; vw_I are the visual words of I, resulting from its clustering (more details on this operation in Sec. IV-B); E_S represents encryption with scheme S and C_P is the resulting ciphertext when applied to plaintext P; *D* is the decryption operation; *Idx* is the index; $|vw|$ is the total number of visual words in the repository; $|Rep|$ is the number of images in the repository; and $|CB|$ is the size of the codebook used for clustering.

between security (privacy) and usability. Unfortunately, even these solutions are too computationally complex for wide adoption, particularly regarding the support of privacy-preserving CBIR over large-scale, dynamically updated[1] image repositories, and even more if we consider mobile (resource constrained) clients, which already represent the source for more than 30% of internet traffic.

To address these challenges we propose a new secure framework for the privacy-preserving outsourced storage, search, and retrieval of large-scale, dynamically updated image repositories. We base our proposal on another contribution of this work: IES-CBIR, a novel Image Encryption Scheme with Content-Based Image Retrieval properties. Key to the design of IES-CBIR is the observation that in images, color information can be separated from texture information, enabling the use of different encryption techniques with different properties for protecting each of these features. Following this observation, and considering that texture is usually more relevant than color in object recognition [20], in IES-CBIR we make the following tradeoff: we choose to privilege the protection of image contents, by encrypting texture information with probabilistic (semantically secure) encryption [21]; then we somewhat relax the security on color features, by using deterministic encryption on image color information. This methodology allows privacy-preserving CBIR based on color information to be performed directly on the outsourced servers with high security[2]. Notably, our solution allows the outsourced servers to generate and update the resource index used to support queries, a task that in many state of the art solutions must be managed by the user devices which, as we later show in the paper, leads to excessive computational and/or communication overheads with impact on performance.

This paper makes the following contributions: (i) we formally define IES-CBIR and propose an efficient construction that achieves its functionality; (ii) we show how to design an outsourced image storage, search, and retrieval framework by leveraging IES-CBIR to avoid most heavy computations (i.e. indexing of dynamically added/updated images) to be performed by the client, hence avoiding performance pitfalls that exist in previous approaches [15]–[19]; (iii) we formally prove the security of our proposed framework based on IES-CBIR; (iv) we experimentally show that when compared with competing alternatives [15], [17], our framework provides increased scalability, performance (from user's perspective), and lower bandwidth consumption, allowing client applications to be increasingly lightweight and potentially mobile; and finally (v) we show that the retrieval precision and recall [22] of the proposed solution is on par with the current state of art.

II. Related Work

Previous proposals for supporting outsourced storage, search and retrieval of images in the encrypted domain can be broadly divided in two classes: Searchable Symmetric Encryption (SSE) based approaches and Public-Key partially-Homomorphic approaches (PKHE). SSE has been widely used in the past by the research community, both for text [23]–[25] and image [17]–[19] search/retrieval. In SSE-based solutions, clients process and encrypt their data

[1]By dynamically updated we mean a repository where clients continually add, update, and remove images.

[2]And shows potential for extension to many other applicational domains.

before outsourcing it to the Cloud. From this processing, an index is created, encrypted, and stored in the outsourced infrastructure, which allows clients to search their data efficiently and in a secure way. Data is typically encrypted with a probabilistic symmetric-key encryption scheme, while the index is protected through a combination of probabilistic and deterministic (or even order-preserving [26]) encryption. Unfortunately, SSE-based approaches in general share the following limitations:

(**i**) clients either require a trusted proxy [18] or have to index their images (and encrypt that index) locally [17], which entails the use of additional computational power on their side and limits the practicality of the solution for lightweight and mobile devices. This is even more limiting if we consider dynamic application scenarios, where images are constantly being added, updated, and removed. In such dynamic cases, SSE works require multiple rounds of communication for updating image repositories and their indexes. For instance, [17] uses repository-wide statistics (inverse-document frequencies), which change as the repositories are updated and thus force re-construction and re-encryption of the index which might require clients performing such task to download and decrypt the full contents of the repository. Additionally, in [17] index values are encrypted with an order-preserving encryption scheme whose security depends on the plaintext domain distribution, and with multiple updates this distribution will change, also forcing index re-construction and re-encryption;

(**ii**) clients have to transfer additional data to the cloud (instead of just uploading images, they also have to retrieve and re-upload their encrypted index with each repository update). This leads to additional bandwidth usage, negatively impacting the storage operations latency as perceived by users;

(**iii**) as SSE works use deterministic identifiers and search trapdoors to provide their functionality with practical performance, they leak what is known as *search, access, similarity*, and *update* patterns [18], [23]–[25], which intuitively means that they reveal, respectively: if a new query has been submitted before; which images are returned by each query; which images are similar to a given query image (in case of similarity/ranked search); and which images (previously searched) are similar to a new image being inserted. As demonstrated in [27] (and even if only the *search* and *access* patterns are revealed [23]), in any long-lived system with many queries being executed and all index entries being accessed at some point in time, or when facing an adversary capable of adaptively querying the whole system [23], these leakages result in exposing as much information as a fully deterministic encryption scheme, albeit with much higher computational overhead. Nonetheless, the reader should note that deterministic schemes (and SSE-based schemes with the referred leakages) can still be provably-secure, as long as the higher-level applications leveraging them control the amount of background information leaked to adversaries [27].

The alternatives to SSE that can be found in the literature [15], [16] are based on public-key partially-homomorphic encryption (PKHE) schemes such as Paillier [13] or ElGammal [14] (which allow additions and multiplications on the encrypted domain, respectively). In these approaches, clients encrypt images pixel by pixel with a PKHE scheme, allowing the cloud to process and index their encrypted images on their behalf and thus avoiding many of the practical issues of SSE-based solutions. Unfortunately, PKHE works present much higher time and space complexities. For instance, Hsu et al. [15] proposed a high-precision CBIR algorithm in the encrypted domain, by resorting to the Paillier cryptosystem [13]. However, their approach results in significative ciphertext expansion (for a secure key size of at least 1024 bits, each pixel is transformed from its traditional 24 bits representation into 2048 ciphertext bits), slow encryption and decryption times (as we will experimentally demonstrate in our evaluation Section V-1), and in scalability issues (the “ciphertext blowup” problem [28]). Furthermore, their work was later shown to be either insecure or computationally intractable for the cloud server [29]. Zheng et al. [16] proposed a variant of that work to overcome some of its limitations, by replacing Paillier ciphertexts with pointers to a ciphertext table (built by mapping those pointers to all possible ciphertext pixel values). While this approach reduces the number of encryption operations and minimizes ciphertext expansion, it still presents a significant computational overhead which limits its practicality.

Aside from the SSE and PKHE research directions, there have been other works following similar approaches to what we propose in this paper, although for different purposes. An example is the work by Nourian et al. [30] which aims at providing privacy-preserving single image template matching performed by third-party clouds. The work doesn’t support large-scale repositories however as it only allows linear searching, requires the template being matched to be re-encrypted for comparison with each different image in a repository, and depends on the availability of public images as noise for encryption which can be easily found by an attacker using popular high-availability repositories for dictionary attacks or by tracking users' traffic. Another example is the more theoretical work by Chase et al. [31], which propose a set of algorithms for the encryption of several data structures including matrix-based datatypes such as images, while enabling queries to be performed over the ciphertext. However their main motivation is to extract partial information about a single encrypted data object (such as the color of a given pixel in an image) whereas we focus on allowing the generation of indexes over large collections of encrypted images by an untrusted third party and the resolution of user queries over these large collections.

The reader should also note that most works on privacy-preserving image retrieval are not actually proven secure. In some works the underlaying cryptographic primitives used have already been proven secure and are used as intended [17], meaning that their security correctness may be easy to infer despite the lack of a formal security analysis. However, other works rely on small modifications that can actually compromise security. Such is the case, for instance, of [15] and [30] as we discussed in the previous paragraphs, and [19] which bases its security solely on hashing functions which can be trivially compromised through dictionary attacks [21].

Table I summaries some aspects of the previous works discussed here, by comparing our work with the most relevant approaches from SSE [17] and the PKHE [15] research contexts, in terms of information leakage and computational complexity for the clients. These works were also implemented and will be experimentally compared with a prototype of our framework in Section V. The *Information Leakage* column represents the leakage of all system operations (particularly the update, search, and remove operations) as a whole. The column *Local Index Size* represents a maximum bound on the possible index size on the clients' side. In PKHE [15] and our work, the cloud extracts and indexes the images' feature-vectors, however in SSE works (e.g. [17]) it's the client who extracts image features and builds/updates the index with each repository update, so its size becomes especially relevant for the overhead imposed on clients. The final column represents the CBIR algorithms used in each work: local color histograms [17], SIFT [32], and global color histograms [33].

III. Technical Overview

In this work we propose a framework for the privacy-preserving outsourced storage, search, and retrieval of images in large-scale, dynamically updated repositories. Our framework is composed of two main components: an image encryption component, executed on client devices; and a storage, indexing, and searching component (in the encrypted domain), executed in the outsourcing server (e.g. a cloud provider). We base this framework on a new encryption scheme specifically designed for images, called IES-CBIR. IES-CBIR allows us to design outsourced image repository systems that support content-based image retrieval (CBIR) based on color features, while protecting the privacy of both image owners and other users issuing queries. Comparing with state-of-art, IES-CBIR shows comparable retrieval precision and higher computational performance than previous approaches as perceived by the clients, since it securely moves indexing computations to the cloud provider's infrastructure and avoids public-key and homomorphic cryptography. IES-CBIR also minimizes ciphertext expansion and consequently bandwidth and outsourced space requirements, reinforcing the positive impact on user-perceived latency. These benefits are further illustrated in our experimental analysis in Sec. V, where the performance of a IES-CBIR system is compared against the state of the art SSE [17] and PKHE [15] based approaches.

In the next subsections we present the system model for our proposed framework (Sec. III-A), followed by its adversary model and security assumptions (Sec. III-B) and by some relevant use cases we envision for the application of our proposal (sec. III-C). For the remainder of the paper we use the following terminology: a *repository* is a collection of images which is stored in the infrastructure of a cloud provider; the *cloud server*, or just *cloud*, is the outsourcing infrastructure that acts as a server both for storage and computation over images; *users* are the clients of our system, possibly using lightweight mobile devices. Each user accesses one or more repositories where they can search, add, and update images at any time; *Repository keys* are secret cryptographic keys that are used to search, add and update images in the repositories (each repository has its own repository key); *Image keys* are secret keys used for encrypting and decrypting the images in the repositories, in conjunction with the respective repository keys (each image has its own image key).

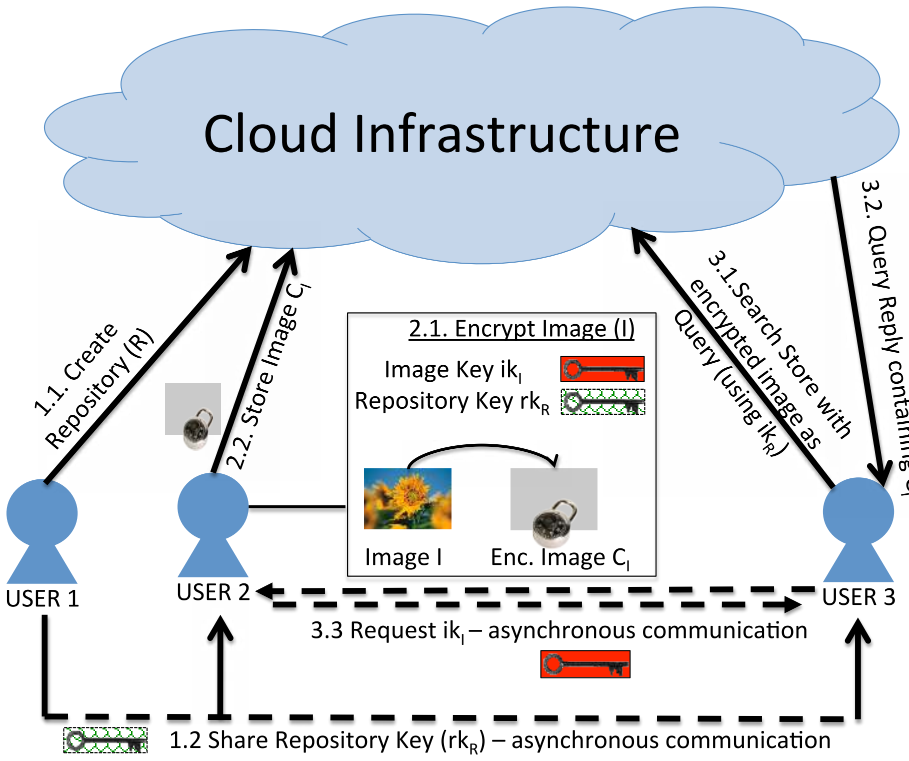
Cloud Infrastructure
1.1. Create Repository (R)
2.2. Store Image C_I
2.1. Encrypt Image (I)
Image Key ik_I
Repository Key rk_R
Image I
Enc. Image C_I
3.1.Search Store with encrypted image as Query (using ik_R)
3.2. Query Reply containing C_I
USER 1
USER 2
USER 3
3.3 Request ik_I – asynchronous communication
1.2 Share Repository Key (rk_R) – asynchronous communication

Fig. 1: System Model Overview

A. *System Model and Architecture*

We now describe an example system model and architecture envisioned for using our framework and IES-CBIR. In this model, we consider two main entities: the *cloud* and (multiple) *users* (Fig. 1). Images are outsourced to *repositories* managed by the cloud. Each repository is used by multiples *Users*, where they can both add their own images and/or search using a query image. Users can also request access to stored images from their creators/owners. Our objective is to ensure the privacy of users, hence all data sent to the cloud is encrypted.

Repositories are created by a single user. Upon the creation of a repository, a new *repository key* is generated by that user and then shared with other trusted users, allowing them to search on the repository and store new images. To add/update images, a user further needs an *image key* (locally) generated for that image. Image keys are kept secret by their users, meaning that even users capable of searching in a repository (i.e. with access to the repository key) will need to ask the owners of specific images for access to them. Note that using specific keys per-image should be seen as optional in our framework, i.e. if the users of a repository prefer to avoid further key management work and are willing to sacrifice fine-grained access control, they can use the same image key for all images or in groups of images.

When the cloud receives an encrypted image for storage it extracts its relevant features (in our framework, we use global color features [33]) and indexes the image based on these features. The same action is performed for a query image, which after being encrypted by a user with a repository key, is then processed by the cloud and have its features extracted and matched with the repository's index. The reply to a query will contain a (configurable) *k* number of encrypted images and respective metadata, which include each image's id and the id of the user that owns each of the images. To fully decrypt and access the contents of an image, besides the repository key, the querying user will further require the image key for that specific image.

It should be noted that all key sharing interactions can be done asynchronously and out-of-band, by resorting to a trusted key-sharing protocol with public-key authentication such as Needham-Schroeder-Lowe [21]. User authorization and revocation can also be easily achieved through the sharing (and refreshment when user revocations are issued) of repository-specific tokens between trusted users, and its request in the framework operations. Nonetheless, we find these discussions to be orthogonal to the main focus of this paper, as the mechanisms involved can be easily integrated into our framework.

B. Adversary Model and Assumptions

In this work we aim at protecting the privacy of users' images and queries. The main adversary we consider is the *malicious cloud administrator*, who operates the cloud's infrastructure and servers. Similar to many previous works found in the literature [17], [18], [23]–[26], we assume an honest-but-curious cloud model [4], that is, the cloud is seen as a passive adversary that is expected to correctly perform operations when asked (i.e. fulfill its contract agreements), but may eavesdrop on users' data. We assume that a malicious cloud administrator has access to all data stored on disk or in RAM on any device physically located at the cloud, and passing through the network from or to the cloud. In Section IV-C, we formally prove the security of our framework against such an adversary.

A stronger adversary that should also be considered in this work is a *malicious user*, i.e. a user of the system who deviates from his expected behavior. Malicious users are an open problem for any multi-user application, as they may be given access to multiple repository and image keys before being discovered, and can more easily eavesdrop on other users' images. In this work we focus on protecting our proposals and proving their security against the malicious cloud administrator, and leave the malicious users issue as an open and interesting future research direction. Nonetheless, we acknowledge that different orthogonal mechanisms can be deployed to mitigate the challenges introduced by malicious users, including access control techniques, repository access revocation and periodic key refreshment. We also assume that distinct adversaries (i.e. a malicious user and the cloud administrator) can collude with each other, as they will not gain any further advantage as a whole against the system. Furthermore, we don't consider integrity or availability threats, as they can be handled by different mechanisms orthogonal to the contributions of this paper [34]–[36]. Finally, we assume that the higher-level applications using our framework control the amount of background information leaked, as this information may be sensitive and can be leveraged by adversaries for breaking security.

C. Relevant Use Cases

As a way to illustrate the broad applicability of our system model, we now briefly discuss two relevant use cases and explain the mapping of concrete entities between models.

Personal Health Records Personal health records (PHR) storage is being offered today as an outsourced service by major cloud operators[3]. PHR may contain both textual and/or image information (e.g. colored MRAs, skin cancer photos, among others) from previous medical consults or exams of several patients followed by different medical doctors at different healthcare centers. The availability of this information, not only ensures a better service towards patients, but also offers a high potential for the exchange of healthcare information among different medical professionals and institutes to assist them in treating patients with similar conditions, as well as for research proposes. In this scenario, medical doctors are *users* of the system, and outsource PHR of their patients to a *cloud-based* backend. In the cloud, PHR are organized in alliance-based repositories between cooperating professionals and/or medical specialty-based repositories. Because PHRs contain sensitive information and belong to the patients, these records can be protected by an image key only known to the patient. Repository keys are shared among all cooperating medical doctors of all medical centers involved in this effort. Doctors can then perform search operations on these repositories, and indirectly request the image keys to PHRs that might be of their interest, through the physician following the patient to whom those records belong to.

Storage for Mobile Users Existing studies have shown that Internet users are increasingly mobile [1]. Since mobile clients usually have limited computational and storage resources, they tend to rely on cloud services for storing and processing bulky data such as images. In this scenario, mobile clients (*users*) want to delegate their private

[3]e.g, https://www.healthvault.com, http://www.cleardata.com/

image repositories storage to a *cloud provider*, while copping with the limitations of their device's storage capability, computational power, and battery life. Additionally, clients might be interested in allowing their images to be searched (and eventually accessed) by other users (either friends, family, or co-workers). Privacy can be relevant for instance when a user is a public figure or has access to sensitive material. Additionally, one might imagine that some companies could have interest in accessing the images owned by a given user, for instance when performing background checks on prospective new employees, among other cases.

IV. A Privacy-Preserving CBIR Framework

In this section we present the design and details of our proposed framework. We start by formally defining IES-CBIR and its algorithms (Sec. IV-A). Then we explain how the framework's component in the cloud server side leverages IES-CBIR properties to store, index and search images while preserving their privacy (Sec. IV-B). Finally we detail the system protocols of our framework and formally prove its security (Sec. IV-C).

A. IES-CBIR Design and Implementation

The main component on the users' side leverages a novel cryptographic scheme specifically designed for images and privacy preserving CBIR, dubbed *IES-CBIR*. Before describing IES-CBIR in detail, we give a definition of image privacy underlines our work.

Informally, we define image privacy as the ability to keep the contents of an image secret to public (or simply unauthorized) disclosure [37]. Generally speaking, image contents are characterized by the combination of its color and texture information. These two components form what one can readily identify in an image: objects, people, etc. As such, to safeguard image privacy entails preventing unauthorized entities from recognizing objects in those images. We further remark that image color and texture informations can be separated from each other. In fact, color information is given from pixel color values in the different channels of some color models; while texture information is given by the (relative) position of pixels and strong color changes across neighboring pixels. We also remark that texture information is usually more relevant in images for object recognition [20]. Finally, we conclude that no sub-component alone (i.e. color or texture information) can be used to infer the precise contents of an image, as color information on itself is usually ambiguous (e.g. strong blue can translate into sky, ocean, etc.) and texture information depends not only on pixel positions but also on their color values. These conclusions are further supported by the most recent works in image reconstruction [38], which not only depend on local features extracted from sub-parts of the images (in this work we focus on global features extracted from each image as a whole), but also on those local features not being encrypted.

Leveraging the previous definition and observations, we design IES-CBIR, an image encryption scheme that separates color from texture information, applying different encryption techniques for protecting each: emphasizing that texture is usually more relevant than color for object recognition [20], we design IES-CBIR to protect image texture with probabilistic encryption and color information with deterministic encryption. This way, content-based image indexing and retrieval, based on color information, can be performed on the cloud servers in a privacy-preserving way and without intervention of users, while texture information remains protected with the highest level of security (we provide a detailed and formal security evaluation in Sec. IV-C). We define IES-CBIR as:

Definition 1 (IES-CBIR)**.** *An Image Encryption Scheme with CBIR properties (IES-CBIR) is a tuple (*GenRK, GenIK, Enc, Dec, TrpGen*) of five polynomial-time algorithms run by a user, where:*

- GenRK*(sp_{rk}): is a probabilistic algorithm that takes as input the security parameter $sp_{rk} \in \mathbb{N}$ and generates a repository key rk with length polynomially bounded by sp_{rk};*
- GenIK*(sp_{ik}): is a probabilistic algorithm that takes as input the security parameter $sp_{ik} \in \mathbb{N}$ and generates an image key ik with length polynomially bounded by sp_{ik};*
- Enc*(I, rk, ik): is an algorithm that takes as input an image I and the cryptographic keys $\{rk, ik\}$ and returns an encrypted image C_I;*
- Dec*(C_I, rk, ik): is an algorithm that takes as input an encrypted image C_I and keys $\{rk, ik\}$ and returns the*

decrypted image I;

• TRPGEN*(Q, rk): is an algorithm that takes as input a query image Q and a repository key rk and returns a searching trapdoor C_Q;*

1) Key Generation: As already discussed in our system model (Sec. III-A), IES-CBIR works with two different types of cryptographic keys, repository keys (rk) and image keys (ik), which are generated by the GENRK and GENIK algorithms respectively. In our design and implementation of IES-CBIR, we specify that repository keys are generated by making three independent random permutations of all values in the range $[0..100]$. These permutations can be generated through a Pseudo-Random Generator (PRG) [21] $\mathcal{G}$, parameterized with some random seed (in our implementation we instantiate $\mathcal{G}$ with an AES-based PRG [21]). This results in 3 sub-keys: rk_H, rk_S, rk_V (Eq. 1). The range $[0..100]$ represents all possible color values in the HSV color space ((H) hue, (S) saturation, (V) value/brightness), and each different sub-key is attributed to a color channel. We choose to maintain the same domain of possible values with encryption in order to reduce ciphertext expansion and to allow image processing operations to be done "as is", including CBIR, indexing and image compression/decompression. Therefore, key $rk = \{rk_H, rk_S, rk_V\}$ enables deterministic encryption of color information by (pseudo)randomly permuting all pixel color values in the three HSV color channels[4]. In this case, sp_{rk} takes the value $303 \times 8 = 2424$ bits.

$$rk_H, rk_S, rk_V \leftarrow RandPerm(\mathcal{G}, [0..100]),\ rk = \{rk_H, rk_S, rk_V\} \tag{1}$$

In contrast, key ik is generated by requesting 128 (the sp_{ik}) pseudorandom bits to $\mathcal{G}$. ik will be used as a cryptographic seed for the probabilistic encryption counterpart of IES-CBIR.

2) Encryption: Image encryption in IES-CBIR is achieved through two main steps and a final (optional) step: i) pixel color values encryption, ii) pixel positions permutation, and iii) image compression. The goal of the first step is to protect image color features, through the application of a Pseudo-Random Permutation (PRP) [21] $\mathcal{P}$ on all pixel color values. Although we could use a standard PRP construction to instantiate $\mathcal{P}$ (such as an AES-based PRP [21]), we chose to conceive a specific color-domain PRP, allowing us to preserve the format of encrypted images. Our construction encrypts pixel color values by deterministically replacing them, in each color channel, using repository key $rk = \{rk_H, rk_S, rk_V\}$. Eq. 2 represents this operation, where $\mathcal{P}_{rk}(p_x)$ is the encryption of pixel p's value in color component x through $\mathcal{P}$ and key rk_x.

$$C_I \leftarrow \mathcal{P}_{rk_x}(p_x) : \forall x \in (H, S, V),\ \forall p \in I \tag{2}$$

This step of encryption securely hides the color values of the encrypted pixels. However, due to the deterministic properties of $\mathcal{P}$ (a requirement to enable CBIR in the encrypted domain), patterns present in the original image (which denote its texture) will remain visible. To fully protect image contents, we rely on a second probabilistic step in our encryption algorithm: (pseudo)random pixel position permutation, through pixel rows and columns shifting. This step consists in the following: a PRG $\mathcal{G}$ is instantiated with a previously generated image key ik (operation GENIK above) as cryptographic seed. Then, for each pixel column, we request from $\mathcal{G}$ a new pseudorandom value r between 1 and the image height, and do a downward shift on that column of r positions, overflowing to its beginning. After all columns have been randomly shifted, we repeat the procedure for the rows (with random values ranging between 1 and the image width). Eqs. 3 and 4 formally describe this step, where w and h are, respectively, the width and height of image I. Note that this encryption algorithm has no ciphertext expansion (i.e, after encryption the image has the same width and height as before).

$$C_I(x, y) \leftarrow C_I(x, (y + r)\ mod\ h) : \forall x \in \{1, .., w\}, \forall y \in \{1, .., h\} \tag{3}$$

$$C_I(x, y) \leftarrow C_I((x + r)\ mod\ w, y) : \forall x \in \{1, .., w\}, \forall y \in \{1, .., h\} \tag{4}$$

This second step is probabilistic, as each new image will have a new pseudorandomly generated ik, even if the same image is stored multiple times with different names (if the same image key is used for all images, then a random *iv* must also be used as input to $\mathcal{G}$). Moreover, this step effectively hides existing texture patterns in the image, making

[4]Instead of encrypting pixel color values with deterministic encryption, we could also use Order-Preserving Encryption [26], slightly increasing image retrieval precision at the expense of greater information leakage.

it computationally unfeasible to extrapolate correlations between the plaintext and ciphertext. We choose to shift rows and columns, instead of pseudorandomly permuting all single pixel positions, because its more efficient (only $w + h$ pseudorandom values are required instead of $w \times h$) and we obtain similar robustness against cryptanalysis even for images as small as 16×16 pixels (see security analysis in Sec. IV-C).

The final, optional step in our encryption algorithm is to perform image compression. This is possible due to the format-preserving properties of IES-CBIR, and can be achieved through the use of a non-lossy image compression scheme such as PNG, directly over the encrypted image (additionally one can use more generic file compression algorithms such as ZIP or RAR). This step allows to control a tradeoff between computational requirements and encryption time with that of network traffic and cloud storage requirements.

3) Decryption: The decryption algorithm applies the different steps of encryption in the opposing order, or more formally, through the ordered application of the transformations denoted by Eqs. 5, 6, and 7 (after decompressing the ciphertext if required). Note that the r random values must be generated in the same order as in the encryption.

$$C_I((x + r) \ mod \ w, y) \leftarrow C_I(x, y) : \forall x \in \{1, .., w\}, \forall y \in \{1, .., h\} \tag{5}$$

$$C_I(x, (y + r) \ mod \ h) \leftarrow C_I(x, y) : \forall x \in \{1, .., w\}, \forall y \in \{1, .., h\} \tag{6}$$

$$I \leftarrow \mathcal{P}_{rk_x}(c_{p_x}) : \forall x \in (H, S, V), \ \forall c_p \in C_I \tag{7}$$

4) Searching-Trapdoor Generation: The TRPGEN algorithm generates searching trapdoors that users can leverage to search over image repositories. Trapdoor generation requires a query image Q as input, as well as the repository key rk. This means that users with access to rk will be able to access color values of all images stored in that repository. However, users can't access texture information (and hence full image contents) without the corresponding image keys, and can't use rk to search other repositories. Given rk, the TRPGEN algorithm operates in a similar fashion to the ENC algorithm (Eq. 8, where the image key is substituted by a new randomly generated ik). This means that searching trapdoors are also decryptable, and can be stored in the repositories as new images as long as the querying users locally save the generated image keys.

$$Trp(Q, rk) \leftarrow Enc(Q, rk, ik) \tag{8}$$

B. CBIR in the Encrypted Domain

On the cloud's side, the received encrypted images are processed and indexed for CBIR before being persistently stored. IES-CBIR enables these operations (for color features) to be performed over their ciphertexts, using algorithms that operate on non-encrypted images and without requiring any modifications. Encrypted image processing has two main steps: feature extraction and feature indexing. Feature extraction consists in processing an image and extracting a reduced set of feature vectors that describe it. In this work we focus on color features in the HSV color model and their representation as color histograms. For each encrypted image and each HSV color channel, the cloud server builds a color histogram by counting the number of pixels in each intensity level. This yields 3 color histograms with 101 entries each.

Upon extracting these features, the cloud can perform feature indexing to speedup query execution. In this work, we use the Bag-Of-Visual-Words (BOVW) representation [39] to build a *vocabulary tree* and an *inverted list index* for each repository. We choose this approach for indexing as it shows good search performance and scalability properties. In the BOVW model, feature-vectors are hierarchically clustered (for instance, using the *k-means* algorithm [39]) into a vocabulary tree (also known as *codebook*), where each node denotes a representative feature-vector in the collection and leaf nodes are selected as the most representative nodes (called *visual words*). This clustering step requires a training dataset, so in the prototype implementation of our framework based on IES-CBIR, we request an initial image collection from users when creating a new repository. After the creation of the codebook, additional images can be stored dynamically by hierarchically stemming them against it. This stemming returns the closest visual words to the image, according to some distance function (in our prototype we use the Hamming/L1 Distance). Finally, the cloud server builds an inverted list index, with all visual words as keys and, as values, the list of images

Algorithm 1 The ideal functionality of our framework, $\mathcal{F}$; all information leaked is specified here.

$\mathcal{F}$ is specified as a trusted third-party, which mediates inputs and outputs between a user and the cloud server, modeling all information leaked to the later. $\mathcal{F}$ accepts four commands, with inputs identical to the commands of the cloud server:

- $\mathcal{F}$.CreateRepository($ID_R, n, m, ID_U, \{ID_{I_i}, I_i\}_{i=0}^{d}$) - Upon receiving this command from the user identified by ID_U:
    - ○ $\mathcal{F}$ initializes a new repository Rep_R and creates a new index Idx_R with size n. Then $\mathcal{F}$ stores and indexes the initial set of images $\{ID_{I_i}, I_i\}_{i=0}^{d}$, creating in the process the clustering codebook CB_R with height m and leaf width n.
    - ○ **Setup Leakage** $\mathcal{F}$ sends to the cloud server the deterministic identifiers of the repository (ID_R), of the user creating it (ID_U) and of each initial image (ID_{I_i}). Additionally, $\mathcal{F}$ sends initialization parameters n and m and, for each initial image I, $\mathcal{F}$ sends its width in pixels (w_I), its height (h_I), and deterministic identifiers of the visual words contained in I and their frequency ($\{ID_{vw_j}, freq_{ID_{vw_j}}^{ID_I}\}_{j=0}^{|vw_I|}$).
- $\mathcal{F}$.StoreImage(ID_R, ID_I, I, ID_U) - If R doesn't exist, $\mathcal{F}$ returns an error. Otherwise:
    - ○ $\mathcal{F}$ persistently stores image I in Rep_R and indexes it, updating Idx_R in the process.
    - ○ **Storage Leakage** In addition, $\mathcal{F}$ sends to the server $ID_R, ID_I, ID_U, w_I, h_I$ and $\{ID_{vw_j}, freq_{ID_{vw_j}}^{ID_I}\}_{j=0}^{|vw_I|}$.
- $\mathcal{F}$.Search(ID_R, Q, k) - If R doesn't exist, $\mathcal{F}$ returns an error. Otherwise:
    - ○ $\mathcal{F}$ processes query image Q and returns the most relevant image results in descending order, according to Idx_R.
    - ○ **Similarity Search Leakage** $\mathcal{F}$ sends to the server ID_R, ID_Q (a deterministic id for Q generated by $\mathcal{F}$), k, w_Q, and h_Q. It also sends the query results and their relevance scores ($\{ID_{I_i}, score_{ID_{I_i}}^{ID_Q}, ID_{U_{I_i}}\}_{i=0}^{k}$).
- $\mathcal{F}$.Remove(ID_R, ID_I) - If R doesn't exist or I isn't an image of R, $\mathcal{F}$ returns an error. Otherwise:
    - ○ $\mathcal{F}$ removes the image identified by ID_I from Rep_R and from Idx_R.
    - ○ **Remove Leakage** $\mathcal{F}$ sends to the server ID_R and ID_I.

(and a frequency score for the visual word in the image) most close to them. This type of list is known as a *Posting List* [3].

After processing and indexing encrypted images, the cloud server can receive search requests from users, through the submission of searching trapdoors for some query images of their choice. When a new searching trapdoor is received, the cloud server extracts its color feature-vectors and finds their closest visual words by stemming them against the codebook. The query's visual words are used to access the repository's index, obtaining the corresponding posting lists in the process. Then, for each image referenced in at least one posting list, a search score is calculated for that image (in our implementation we use a "scaled tf-idf" scoring function [17]). Finally, the cloud returns the top k images to the user, according to their scores (k is a configurable parameter). The BOVW approach guarantees that only the most relevant images (a fraction of the repository) have to be compared in the scoring step (ensuring scalability). After receiving these ranked results, users can explicitly request full access to images by requesting the corresponding image keys from their owners.

C. Framework Protocols and Security Analysis

We start this sub-section by providing an *idealized functionality* for our framework. Then we present the details of our IES-CBIR based framework construction and formally prove it securely materializes the idealized functionality. Our security proofs follow the *real/ideal* paradigm that is standard in secure multi-party computations [40].

Algorithm 1 formalizes the *ideal functionality* $\mathcal{F}$ of our framework. In $\mathcal{F}$ we consider as adversary the honest-but-curious cloud provider (Section III-B), which corrupts the cloud server passively. As stated in Section II, the leakage functions specified in Alg. 1 are equivalent to the *search, access, similarity* and *update* leakages of SSE-based works [18], [23]–[25], particularly for any long-lived system with many queries being executed as expected from real-world application scenarios. Furthermore, applications using our framework can ensure that the information leaked will not compromise their security, by limiting the amount of background information made available to an adversary [27].

In the next paragraphs we detail the protocols of our IES-CBIR based framework's construction, which securely fulfills the idealized functionality $\mathcal{F}$: respectively the *Create Repository*, *Store Image*, *Search with Query Image*, *Remove Image* and *Access Image*. In these protocols we omit operations related with the request and sharing of keys, as these are orthogonal to the scope of the paper.

Instantiate a new Repository We start by describing the operation used by a user U to create a new repository R (Alg. 2). On the user's side, the protocol takes as input the repository id (ID_R), the security parameters for the required keys (sp_{rk}, sp_{ik}), some initialization parameters (height m and leaf width n of the clustering codebook), and an initial collection of d images for the repository along with their user-defined ids ($\{id_{I_i}, I_i\}_{i=0}^{d}$). In the protocol, the user starts by locally generating a repository key rk_R for the repository, through the IES-CBIR.GenRK algorithm (line 2). Then, for each image I in the initial group of images, the user generates a new image key ik_I and encrypts the image with ik_I and rk_R (3-6). The user then sends the initialization parameters, pseudorandom ids (including his own id) and encrypted images to the cloud server (7). The cloud starts by initializing the storage space Rep_R and index Idx_R for R (11-12), and then extracts the color feature-vectors (histograms) of all the d initial images (13-14). Then it hierarchically clusters these d feature-vectors, building codebook CB_R (16). Finally, it stems the feature-vectors against CB_R to determine their visual words representations, stores these and their frequencies in Idx_R, and stores each image (with its user id) in Rep_R (17-23).

Algorithm 2 Create New Repository

```
 1: procedure USER($ID_U$).CREATEREPOSITORY($ID_R$, $sp_{rk}$, $sp_{ik}$, $n$, $m$, $\{ID_{I_i}, I_i\}_{i=0}^{d}$)
 2:     $rk_R \leftarrow$ IES-CBIR.GenRK($sp_{rk}$)
 3:     for all $\{ID_{I_i}, I_i\}_{i=0}^{d}$ do
 4:         $ik_{I_i} \leftarrow$ IES-CBIR.GenIK($sp_{ik}$)
 5:         $C_{I_i} \leftarrow$ IES-CBIR.Enc($I_i, rk_R, ik_{I_i}$)
 6:     end for
 7:     CLOUD.CreateRepository($ID_R, n, m, ID_U, \{ID_{I_i}, C_{I_i}\}_{i=0}^{d}$)
 8:     return $\{rk_R, \{ik_{I_i}\}_{i=0}^{d}\}$
 9: end procedure
10: procedure CLOUD.CREATEREPOSITORY($ID_R$, $n$, $m$, $ID_U$, $\{ID_{I_i}, C_{I_i}\}_{i=0}^{d}$)
11:     $Rep_R = \{ID_{I_i}, \{C_{I_i}, ID_{U_i}\}\}_{i=0}^{*} \leftarrow$ InitiateRepository()
12:     $Idx_R = \{ID_{vw_i}, \{ID_{I_j}, freq_{vw_i}^{I_j}\}_{j=0}^{*}\}_{i=0}^{n} \leftarrow$ InitiateIndex($n$)
13:     for all $\{C_{I_i}\}_{i=0}^{d}$ do
14:         $fv_{C_{I_i}} = \{hist_H, hist_S, hist_V\} \leftarrow$ ExtractFeatures($C_{I_i}$)
15:     end for
16:     $CB_R \leftarrow$ ClusterFeaturesIntoCodebook($n, m, \{fv_{C_{I_i}}\}_{i=0}^{d}$)
17:     for all $\{ID_{I_i}, C_{I_i}, fv_{C_{I_i}}\}_{i=0}^{d}$ do
18:         $vw_{C_{I_i}} = \{ID_{vw_j}, freq_{vw_j}^{C_{I_i}}\}_{j=0}^{|vw_{C_{I_i}}|} \leftarrow CB_R$.Stem($fv_{C_{I_i}}$)
19:         for all $\{ID_{vw_j}, freq_{vw_j}^{C_{I_i}}\}_{j=0}^{|vw_{C_{I_i}}|}$ do
20:             $Idx_R[ID_{vw_j}]$.add($\{ID_{I_i}, freq_{vw_j}^{C_{I_i}}\}$)
21:         end for
22:         $Rep_R[ID_{I_i}] \leftarrow \{C_{I_i}, ID_U\}$
23:     end for
24: end procedure
```

Store/Update Image Algorithm 3 illustrates the procedure followed by a user U to store a new image I, or update it if it already exists, in repository R. U is assumed to have access to R and rk_R. U starts with inputs ID_R, rk_R, image I and security parameter sp_{ik}. The algorithm is straightforward and basically consists in a sub-group of alg. 2's instructions (since alg. 2 also stores a group of initial images), where the only difference is the creation of codebook CB_{ID_R} in alg. 2. We point to alg. 2 for a detailed explanation of each instruction.

Search with an Image as Query Alg. 4 sketches the procedure to search in a repository R with query image Q.

Algorithm 3 Store/Update Image

```
 1: procedure USER($ID_U$).UPDATEIMAGE($ID_R, rk_R, ID_I, I, sp_{ik}$)
 2:     $ik_I \leftarrow$ IES-CBIR.GenIK($sp_{ik}$)
 3:     $C_I \leftarrow$ IES-CBIR.Enc($I, rk_R, ik_I$)
 4:     cloud.StoreImage($ID_R, ID_I, C_I, ID_U$)
 5:     return $\{ik_I\}$
 6: end procedure
 7: procedure CLOUD.UPDATEIMAGE($ID_R, ID_I, C_I, ID_U$)
 8:     if $Rep_R$.contains($ID_I$) then
 9:         cloud.Remove($ID_R, ID_I$)
10:     end if
11:     $fv_{C_I} = \{hist_H, hist_S, hist_V\} \leftarrow$ ExtractFeatures($C_I$)
12:     $vw_{C_I} = \{ID_{vw_i}, freq_{vw_i}^{C_I}\}_{i=0}^{|vw_{C_I}|} \leftarrow CB_{ID_R}$.Stem($fv_{C_I}$)
13:     for all $\{ID_{vw_i}, freq_{vw_i}^{C_I}\}_{i=0}^{|vw_{C_I}|}$ do
14:         $Idx_R[ID_{vw_i}]$.add($\{ID_I, freq_{vw_i}^{C_I}\}$)
15:     end for
16:     $Rep_R[ID_I] \leftarrow \{C_I, ID_U\}$
17: end procedure
```

The input for this operation on the user side is ID_R, Q, repository key rk_R and parameter k (the number of most similar results to be returned). User U starts by generating Q's searching trapdoor C_Q, through IES-CBIR.GenTrp algorithm (2). Then he sends it to the cloud server, along with k and ID_R, as parameters for the Search remote invocation (3). The cloud starts by extracting C_Q's feature-vector, stems it against CB_R to determine its visual words vw_{C_Q}, and accesses Idx_R with them to retrieve the respective posting lists PL_{vw} (8-11). Then, for each image referenced in each of the posting lists retrieved, the cloud calculates its *scaled tf-idf score* [17] and adds it to the set of results for the query (12-15). In the set of results, scores for the same image but different visual word are summed. Finally, the cloud sorts this set by descending score and returns the top k to the user (18).

Algorithm 4 Search with an Image as Query

```
 1: procedure USER($ID_U$).SEARCH($ID_R, Q, rk_R, k$)
 2:     $C_Q \leftarrow$ IES-CBIR.GenTrp($Q, rk_R$)
 3:     $rankedImgDistances \leftarrow$ cloud.Search($ID_R, C_Q, k$)
 4:     return $rankedImgDistances$
 5: end procedure
 6: procedure CLOUD.SEARCH($ID_R, C_Q, k$)
 7:     $qr \leftarrow$ InitiateQueryResults()
 8:     $fv_{C_Q} = \{hist_H, hist_S, hist_V\} \leftarrow$ ExtractFeatures($C_Q$)
 9:     $vw_{C_Q} = \{ID_{vw_i}, freq_{vw_i}^{C_Q}\}_{i=0}^{|vw_{C_Q}|} \leftarrow CB_R$.Stem($fv_{C_Q}$)
10:     for all $\{ID_{vw_i}, freq_{vw_i}^{C_Q}\}_{i=0}^{|vw_{C_Q}|}$ do
11:         $PL_{vw_i} = \{ID_{I_j}, freq_{vw_i}^{C_{I_j}}\}_{j=0}^{|PL_{vw_i}|} \leftarrow Idx_R[ID_{vw_i}]$
12:         for all $\{ID_{I_j}, freq_{vw_i}^{C_{I_j}}\}_{j=0}^{|PL_{vw_i}|}$ do
13:             $score_{I_j}^Q \leftarrow$ScaledTfIdf ($freq_{vw_i}^{C_Q}$, $freq_{vw_i}^{C_{I_j}}$, $|Rep_{ID_R}|$, $|PL_{vw_i}|$)
14:             $\{C_{I_j}, ID_{U_j}\} \leftarrow Rep_R[ID_{I_j}]$
15:             $qr[ID_{I_j}] \leftarrow \{C_{I_j}, qr[ID_{I_j}].score + score_{I_j}^Q, ID_{U_j}\}$)
16:         end for
17:     end for
18:     return resize($k$, Sort($qr$))
19: end procedure
```

Access an Image Alg. 5 illustrates the protocol to access an encrypted image C_I previously returned by a search. This algorithm can be executed by a user after he has been given access to the image key ik_I by its owner, user ID_{U_I}. The protocol is a straightforward application of IES-CBIR.Dec algorithm, with inputs C_I, rk_R and ik_I.

Algorithm 5 Access an Image

```
1: procedure User($ID_U$).Access($C_I, rk_R, ik_I$)
2:     $I \leftarrow$ IES-CBIR.Dec($C_I, rk_R, ik_I$)
3:     return $I$
4: end procedure
```

Algorithm 6 Remove an Image

```
1: procedure Cloud.Remove($ID_R, ID_I$)
2:     $Rep_R[ID_I] = \{\}$
3:     for all $PL_{vw} \in Idx_R$ do
4:         $PL_{vw}$.Remove($ID_I$)
5:     end for
6: end procedure
```

Remove an Image Alg. 6 shows the protocol for removing an image I from repository R. Since the algorithm is very simple, we only show the cloud computation part. The cloud server takes as input pseudorandom ids ID_R and ID_I, and starts by removing C_I from Rep_R (line 2). Then, for each posting list in Idx_R, it removes the reference and frequency for image I, if they exist (3-5). In this protocol we assume again the presence of an authorization mechanism, which enforces that users can only remove their own images.

Security Analysis and Proofs The proof that our framework's construction securely realizes $\mathcal{F}$ involves showing that a simulator $\mathcal{S}$, interacting with a user only through $\mathcal{F}$ (the ideal experiment), can simulate the view of the cloud server in a real interaction with the user through an instance of our construction (the real experiment), and that the two experiments would be indistinguishable (apart from a *negligible* probability), even when combined with the *adaptively* influenced inputs of the client. The essential rational that justifies our security properties is as follows: In the ideal functionality $\mathcal{F}$, when the user stores an image or sends it as query to a repository, the server basically learns its similarity to the images stored there, based on a distance function over their color histograms, and nothing more. In the real experiment, the user will invoke (through our construction's protocols) the algorithms of IES-CBIR to achieve the same functionality. Thus, the crucial point in proving security is to show that IES-CBIR leaks no additional information to the server.

Formally, a simulator can simulate the view of the adversary randomly, based only on the size (number of images) of the repository. The only difference between this simulation and the real execution is the following: in the real execution there is a limitation on the size (in terms of pixel width and height) of the images being stored and searched. In fact, IES-CBIR algorithms can only be proven computationally secure for images with at least 16×16 pixels of width and height, respectively. For images smaller than that, a Probabilistic Polynomial-Time (PPT) bounded adversary can compromise the probabilistic counterpart of IES-CBIR encryption in useful time. In the simulation, such limitation does not exist. As such, the proof must show that if this requirement on the size of images is respected by the user, security properties will hold and the real and ideal experiments will be indistinguishable.

Theorem 1. *Our framework's construction based on IES-CBIR securely realizes $\mathcal{F}$ against honest-but-curious PPT adversaries, provided that all images used as input have at least 16x16 pixels of width and height, respectively.*

Proof: Simulator $\mathcal{S}$ interacts with functionality $\mathcal{F}$ and the cloud server, translating each message it receives from $\mathcal{F}$ into a set of simulated messages in the interaction between the server and the user in our framework.

• When it receives the CreateRepository message from $\mathcal{F}$ with its *Setup Leakage* = $\{ID_R, ID_U, d, n, m, \{ID_{I_i}, w_{I_i}, h_{I_i}, \{ID_{vw_j}, freq^{ID_{I_i}}_{ID_{vw_j}}\}^{|vw_{I_i}|}_{j=0}\}^d_{i=0}\}$, $\mathcal{S}$ initializes some data structures: **(i)** A simulated repository $Rep'_R = \{ID_{I_i}\{I'_i, ID_{U_{I_i}}\}\}^*_{i=0}$, which will simulate the contents of images as they are stored in the server; **(ii)** A simulated codebook CB'_R, with height m and leafs $\{ID_{vw_i}, vw'_i\}^n_{i=0}$, where vw'_i is a simulated visual word; **(iii)** a simulated index $Idx'_R = \{ID_{vw_i}, PL'_{vw_i}\}^n_{i=0}$, where $PL'_{vw} = \{ID_{I_i}, freq^{ID_{I_i}}_{ID_{vw}}\}^*_{i=0}$ is a simulated posting list of the images that contain vw' and respective frequencies; **(iv)** a simulated search map $M'_R = \{ID_Q, Q', searchHistory'_Q\}$, where

$searchHistory'_Q = \{ID_{I_i}, score_{I_i}\}_{i=0}^{v}$ is the search history of query Q' and v is the count of distinct results ever returned by query Q'; **(v)** a simulated removal list $Rem'_R = \{ID_I\}$ that stores the ids of removed images. Then, $\mathcal{S}$ creates an initial group of d simulated images. For each image I' in this group, $\mathcal{S}$ creates $w_I \times h_I$ uniformly randomly sampled pixels from the HSV color range ($H, S, V \in [0..100]$), fills I' with these, and sets $R'[ID_I] = \{I', ID_U\}$. $\mathcal{S}$ also creates a simulated color feature-vector fv'_I, by extracting the color features of I'. Then $\mathcal{S}$ takes all simulated feature-vectors and hierarchically clusters them into the simulated codebook CB'_R with height m and leaf width n. CB'_R leaf nodes $\{ID_{vw_i}, vw'_i\}_{i=0}^{n}$ are used to fill Idx'_R keys. Finally, $\mathcal{S}$ stems each fv'_I against CB'_R, yielding $vw'_I = \{vw'_i, ID_{vw_i}, freq_{ID_{vw_i}}^{ID_I}\}_{i=0}^{|vw'_I|}$, and inserts $\{ID_I, freq_{ID_{vw}}^{ID_I}\}$ in $PL'_{vw} = Idx'_R[ID_{vw}],\ \forall vw' \in vw'_I$.

Since the encryption algorithm in IES-CBIR has the pixels encrypted twice, first by a pseudorandom permutation (PRP) [21] of their color values, and then by a random swapping of their pixel positions through a pseudorandom generator (PRG) [21], and due to the properties of these cryptographic primitives (PRPs and PRGs), I and I' will be computationally indistinguishable. Consequently, fv_I and fv'_I will also be indistinguishable, as well as vw_I and vw'_I, CB_R and CB'_R, and Idx_R and Idx'_R. However, due to the use of PRG in randomly shifting pixel rows and columns, the computational indistinguishability of I and I' will not only depend on sp_{ik} (security parameter of the *image key*), but also on the size of I in pixels width and height. More specifically, the computational complexity of a distinguisher $\mathcal{D}$, executed by $\mathcal{S}$, in distinguishing I from I' will be $w^h \times h^w$, as $\mathcal{D}$ has to resolve w random values, each with a possible value range of $[0..h]$, and h additional random values with a possible value range of $[0..w]$. If we consider 128 as minimum security bound for sp_{ik} (as recommended for AES encryption [21]), then $w^h \times h^w$ should be at least 2^{128}. Since an image of 16×16 pixels of width and height leads to $16^{16} \times 16^{16} = 2^{128}$, this represents the minimum security bound for ensuring that I is indistinguishable from I'.

• When an image I is stored with *Storage Leakage* = $\{ID_R, ID_I, ID_{U_I}, w, h, \{ID_{vw_j}, freq_{ID_{vw_j}}^{ID_I}\}_{j=0}^{|vw_I|}\}$, $\mathcal{S}$ creates a simulated image I' with size $w \times h$ in pixels, and stores it in $R'[ID_I]$, along with ID_{U_I}. I' pixels are uniformly randomly sampled from the HSV color range ($H, S, V \in [0..100]$). Then $\mathcal{S}$ extracts from I' the simulated color feature-vector fv'_I and stems it against CB'_R. This yields visual words $vw'_I = \{vw'_i, ID_{vw_i}, freq_{ID_{vw_i}}^{ID_I}\}_{i=0}^{|vw'_I|}$. Finally, $\mathcal{S}$ adds $\{ID_I, freq_{ID_{vw}}^{ID_I}\}$ to $PL'_{vw} = Idx'_R[ID_{vw}], \forall vw' \in vw'_I$. As with the initial images stored when creating a repository, I and I' will be indistinguishable due to IES-CBIR Encryption, as long as $w_I, h_I \geq 16$. Consequently, fv_I and fv'_I, vw_I and vw'_I, and Idx_R and Idx'_R will also be computationally indistinguishable, respectively.

• When a query image Q is searched for with *Similarity Search Leakage* = $\{ID_R, ID_Q, k, w_Q, h_Q, \{ID_{I_i}, score_{ID_{I_i}}^{ID_Q}, ID_{U_{I_i}}\}_{i=0}^{*}\}$, $\mathcal{S}$ creates a simulated query image Q' with size $w_Q \times h_Q$. Then Q' is filled with pixels uniformly randomly sampled from the HSV color range ($H, S, V \in [0..100]$), and its simulated color feature-vector fv'_Q is extracted. $\mathcal{S}$ stems fv'_Q with CB'_R, getting visual words $vw'_Q = \{vw'_i, ID_{vw_i}, freq_{ID_{vw_i}}^{ID_Q}\}_{i=0}^{|vw'_Q|}$. For each $vw' \in vw'_Q$, $\mathcal{S}$ accesses $Idx'_R[ID_{vw}]$, retrieves $PL'_{vw} = \{ID_{I_i},\ freq_{ID_{vw}}^{ID_{I_i}}\}_{i=0}^{|PL'_{vw}|}$, and calculates $\{score_{ID_{I_i}}^{ID_Q}\}_{i=0}^{|PL'_{vw}|}$, where the search scores are given by the *scaled tf-idf* function. Finally, $\mathcal{S}$ gets $searchHistory'_Q$ from $M[ID_Q]$ and updates it with the new search results $\{ID_{I_i}, score_{ID_{I_i}}^{ID_Q}\}_{i=0}^{*}$ (if $M[ID_Q]$ doesn't exist yet, it also sets $M[ID_Q] = \{Q',\ searchHistory'_Q\}$). Since the search algorithm in IES-CBIR is based on IES-CBIR's Encryption algorithm, whose output was already proven indistinguishable from the simulated output, Q and Q' will also be indistinguishable as long as $w_Q, h_Q \geq 16$. Consequently, fv_Q and vw_Q will also be indistinguishable from fv'_Q and vw'_Q, respectively, and the real search history $searchHistory_Q$ will be indistinguishable from the simulated search history $searchHistory'_Q$.

• When an image I is removed with *Remove Leakage* = $\{ID_R, ID_I\}$, $\mathcal{S}$ sets $Rem_R[ID_I] = 1$. The indistinguishability of the remove token comes from the indistinguishability of PRPs used in the generation of deterministic identifiers ID_I and ID_R. ■

V. Experimental Evaluation

In this section we experimentally evaluate our proposals and compare them with some of the most relevant competing alternatives. To this end we have implemented, in the Java language, a prototype of our IES-CBIR based framework (as described in Sections IV-A and IV-C) and prototypes of the competing relevant works: i) the SSE

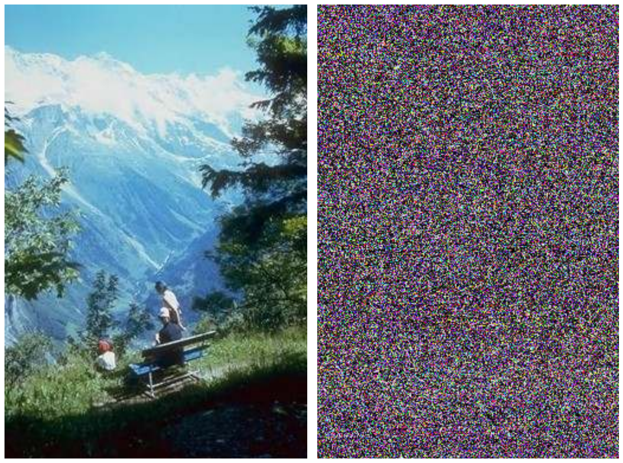

Fig. 2: Example of a IES-CBIR image encryption from the Holidays Dataset

solution proposed in [17], based on Bag of Visual Words indexing and Order-Preserving Encryption (labelled *SSE* in the graphics presented in this section); and (ii) a system leveraging the Paillier cryptosystem described in [15] (labeled *PKHE*). Leveraging these prototypes, we conducted an experimental evaluation of the performance and precision of our solution and the competing works. All experimental assessments were carried out using Amazon EC2 instances, both for user simulation and for running the cloud storage and computation component. To simulate geographic distance, user processes were executed in Oregon's data-center instances, while the cloud component was deployed in a North-Virginia's data-center instance. User instances, in our framework's testing scenario, were of the *general-purpose m3.medium* type and the cloud server was of the *m3.large* type[5]. In the competing works testing scenarios user instances had to be increased to the *m3.large* type, as they have to perform heavier computations. For testing purposes, we used two image datasets: the Wang dataset [20], containing 1000 low-resolution images with a JPEG compressed size of 29.8 MB; and the Inria Holidays Dataset [41], containing 1491 high-resolution images with total JPEG compressed size of 2.85 GB. Figure 2 shows an example of an image from the Holidays Dataset and its IES-CBIR encryption. We present our results in the following order: first we discuss the performance and scalability of our solution, comparing it with the alternative approaches; then we study the achieved retrieval precision.

1) Store/Update Performance: In these experiments we used the larger sized Holidays dataset to compare the performance of our system, with and without image compression (labeled *IES-CBIR w/ compression* and *IES-CBIR no compression* respectively) with that the competing alternatives SSE and PKHE. To this end we have measured the time taken by the UPDATE IMAGE operation with two distinct workloads: one where the 1491 (2.85 GB) images from the Holidays Dataset are used to create and populate a new image repository in the cloud, and another where after the initial upload of 1491 images, another 150 users try to upload 10 new images each concurrently (total 5,68 GB). This last workload allow us to show hidden overheads of updating repositories in some alternatives described in the literature, especially when multiple users try to store new images concurrently. Fig. 3 summarizes the results for each system, in terms of time required for each sub-operation (*Encryption*, *Indexing*, *Training* (i.e. the hierarchical k-means clustering necessary once for each repository, as described in Sec. IV-B) and *Cloud Storage*), as well as a whole (*Total*). The left part of the figure represents the first (static) workload, and the right part represents the second (dynamic) workload. Results are presented in a logarithmic scale and capture the performance from the perspective of users (without considering cloud computations that are performed asynchronously). The experiments show an average of 10 independent runs each.

The results show that *IES-CBIR* with image compression offers overall better performance when compared with the remaining competing alternatives. This is a consequence of the very high cryptographic throughput presented by

[5]http://aws.amazon.com/ec2/instance-types/

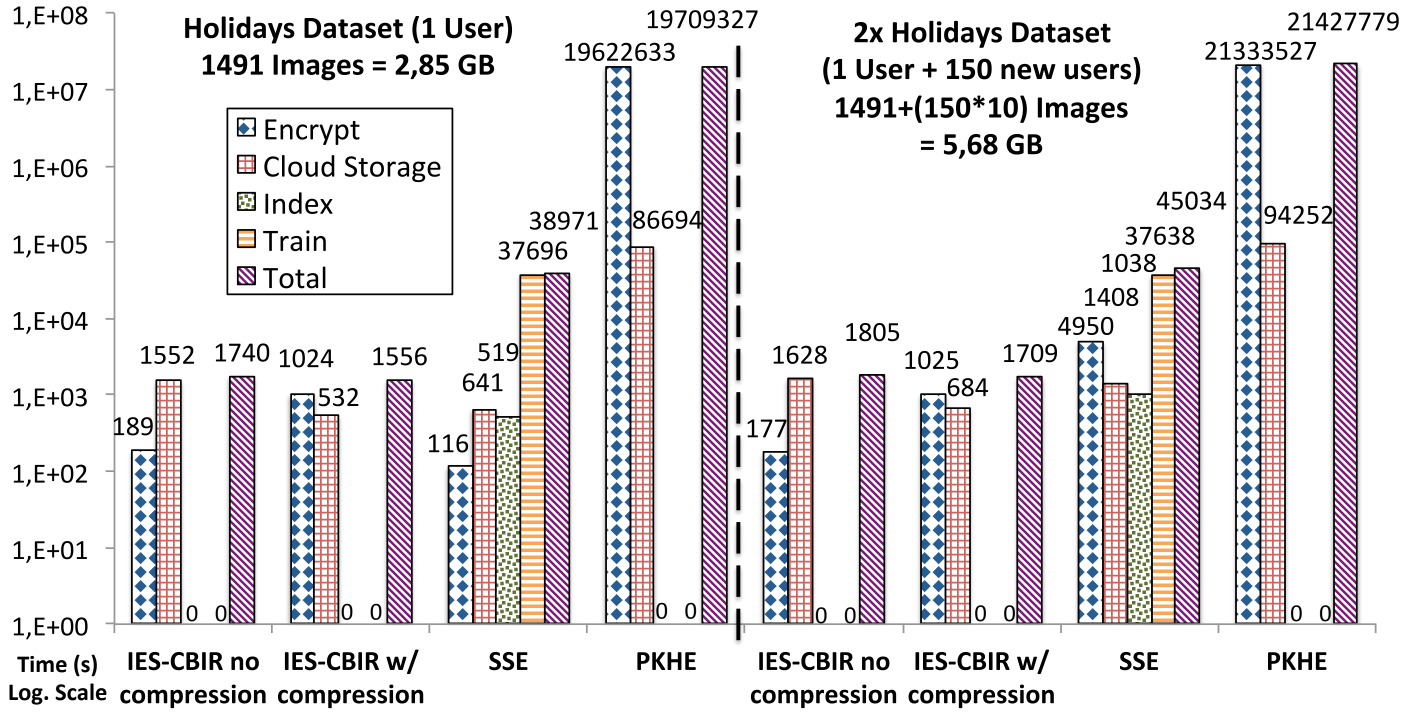

Fig. 3: Performance for the *Store Image* operation (log. scale)

IES-CBIR, combined with the fact that users only have to encrypt images. In contrast, other alternatives either present very slow cryptographic throughput (*PKHE*), or additionally require indexing operations from the users (*SSE*). In more detail: IES-CBIR without compression presents slightly higher cryptographic throughput, but the performance gained there is somehow lost when the user has to upload larger decompressed images; *PKHE* shows prohibitive overheads both due to the low cryptographic throughput of the public-key Paillier cryptosystem and to its high ciphertext expansion (which then has to be uploaded to the cloud server); and *SSE* requires expensive initial training performed by the creating user for each new repository, in order to build the clustering codebook required for indexing, and then also requires the user to index locally his images.

An argument that could be made in favor of the *SSE* approach is that the initial training only has to be performed once, and its cost is amortized in the long run. However our second workload, where multiple users try to add additional images to the repository, shows that is not the case. In this second workload our solution (with compression) is still the one that offers overall best performance and scalability, showing an overall time increase of 9% compared to 83% for *SSE* (if we discard its training/clustering time) and 8% for *PKHE*. The *SSE* increase is mostly due to users having to retrieve the repository's index, decrypt it, update its entries, encrypt and re-upload it to the cloud server, for each repository update (or bulk of updates). Moreover, this has to be done synchronously between users, to guarantee that the repository index remains in a consistent and correct state. The *PKHE* approach shows the same performance degradation as *IES-CBIR*, which is still prohibitively slow for practical adoption, and IES-CBIR without compression is still slightly slower than its compressed counterpart.

2) Search Performance: Figures 4 and 5 show the experimental results for the SEARCH WITH QUERY IMAGE OPERATION, where the first focuses on the IES-CBIR and *SSE* alternatives, and the second compares all approaches (IES-CBIR with and without compression, *SSE*, and *PKHE*) in a logarithmic scale (necessary because of the high overhead of the *PKHE* solution in comparison with the other approaches). The results showed here represent the performance for searching in the Inria Holidays dataset [41] with a random image chosen from the collection as query (the results represent the average of 100 random runs each). The *Encrypt* and *Index* columns represent local processing done by the querying user, while the *Cloud* column represents not only the network time for transmitting

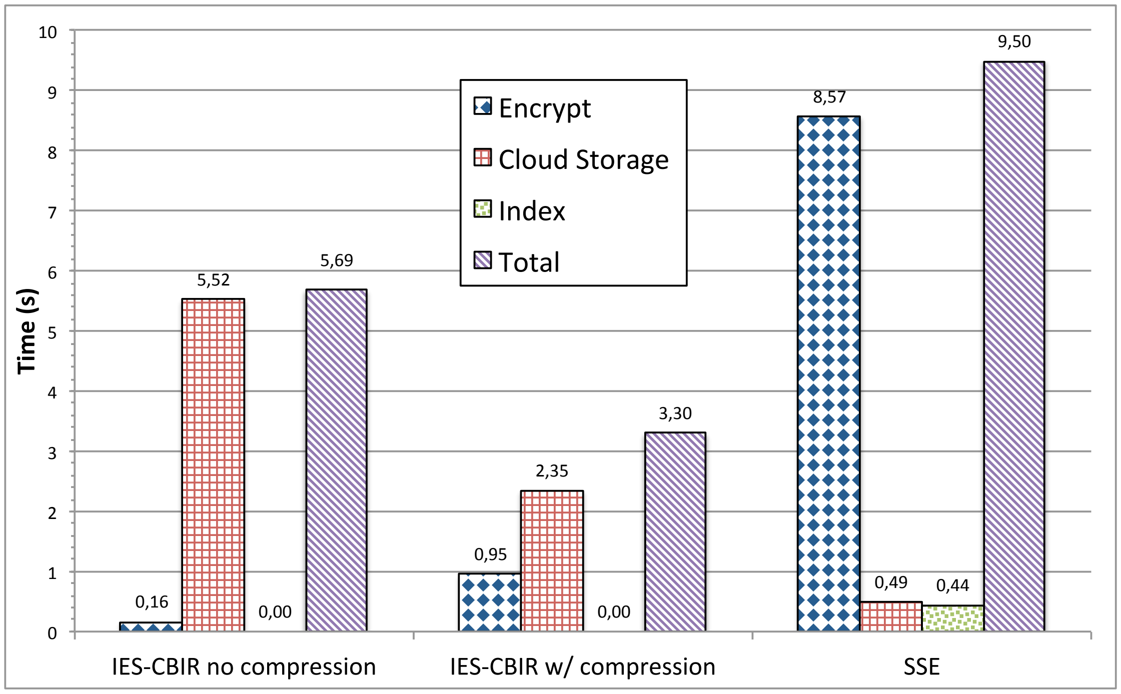

Fig. 4: Performance of the *Search with Query Image* operation, for the IES-CBIR and SSE approaches.

the query and receiving its results, but also the time elapsed by the server in processing and calculating those results.

The results obtained show that IES-CBIR with compression achieves the overall best performance of all alternatives. Compared to the competing alternatives *SSE* and *PKHE*, IES-CBIR obtains an efficiency of 65% and 99,97% respectively. In the *SSE* case, the increased overhead is mainly due to the decryption of the search results, which are encrypted with an Order-Preserving Encryption scheme [17]. Although IES-CBIR (in both of its variants) has higher Cloud computation time, as most of the work involved is done by the cloud server instead of by the user (as in the *SSE* approach), that overhead is still smaller than the difference between the encryption overheads in the two systems and can be further reduced with the less expensive scaling of the cloud server's resources. In the *PKHE* approach, the high overhead is once again consequence of the low throughput and ciphertext expansion of the Paillier cryptosystem [13].

3) Retrieval Precision and Recall: We start by defining the metrics used herein [22]: when a search is done, precision is the number of relevant images retrieved across all returned results; recall is the number of relevant images retrieved from all the relevant results for the query; average precision (AP) is the average of the precision measured each time a new relevant image is retrieved; and mean average precision (mAP) is the mean of APs for a group of queries.

To evaluate the retrieval precision that can be achieved with IES-CBIR, we extracted two metrics: an interpolated recall-precision graph, built with the Wang dataset and all its images as queries; and the mAP of the Holidays dataset, for a group of queries pre-defined by the authors of the dataset [41]. Regarding the first experiment, we used a workload where each image in the dataset is used as query over all others in the repository. We then computed the average precision and recall, for all possible response sizes ($[1, .., 1000]$). Similar to the previous section, we compared the precision of IES-CBIR with it's competing alternatives, *SSE* and *PKHE*. We also assessed the precision that an adversary would achieve if he was to search in the repository with a randomly chosen repository key. Figure 6 summarizes the results. Our framework shows similar precision and recall as the compared alternatives, with a small variation of about 6% in precision compared to *SSE* and *PKHE* systems. This small difference is the advantage gained by these alternatives through the sacrifice of performance and scalability. Nonetheless, the reader should note

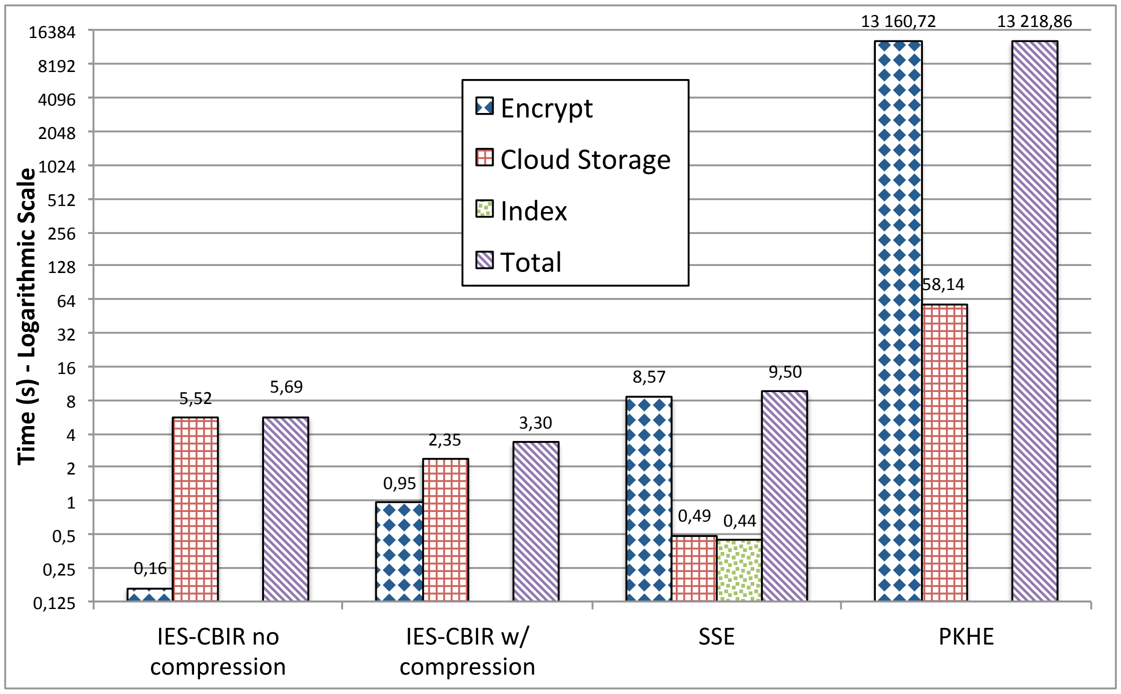

Fig. 5: Performance of the *Search with Query Image* operation, for all analyzed alternatives (log_2 scale).

	IES-CBIR	SSE	PKHE
mAP (%)	54.564	49.075	57.9

TABLE II: Mean Average Precision (mAP) for the holidays dataset

that our approach can be extended to also consider texture information in its CBIR algorithm, increasing retrieval precision at the expense of increased information leakage. Regarding the *IES-CBIR with wrong key* baseline, results show that a malicious user using the framework to search repositories with an incorrect repository key would achieve similar precision as if he was choosing random images from those repositories as answer.

The second precision test we performed consisted in using the evaluation package of the Holidays dataset (available online [41]) for calculating the mAP of a group of 500 pre-defined queries. Table II shows the results. In this experiment, *PKHE* achieved the best result, as expected due to the use of the SIFT retrieval algorithm [32], and *IES-CBIR* achieved the second highest results, followed by *SSE*. Comparing to the results of the previous experiment, we conclude that in some datasets IES-CBIR can actually achieve better precision than some of the competing alternatives, including personal photos and holidays datasets as is the of the Inria dataset [41]. Furthermore, based on all the results presented in this section, we conclude that a framework leveraging IES-CBIR achieves a good trade-off between precision/recall and performance/scalability.

4) Experimental Security Evaluation: To finalize the experiments section of our work, we made a statistical analysis to experimentally assess the entropy level in IES-CBIR encrypted images. This consisted in analyzing the correlation level between all horizontally, vertically, and diagonally adjacent pixels, for: original plaintext images, at different steps of IES-CBIR encryption process, and for a complete random permutation of all pixel positions. We used the correlation function of [30], where the obtained values range between $[-1, .., 1]$, and the images with higher entropy get closer to 0. We used the low-resolution Wang Dataset, proving that IES-CBIR can achieve high levels of entropy even for smaller images. All pixels of all images in the dataset were considered, being the average results presented in Fig. 7. The first point in the figure represents the plaintext images; the second represents IES-CBIR

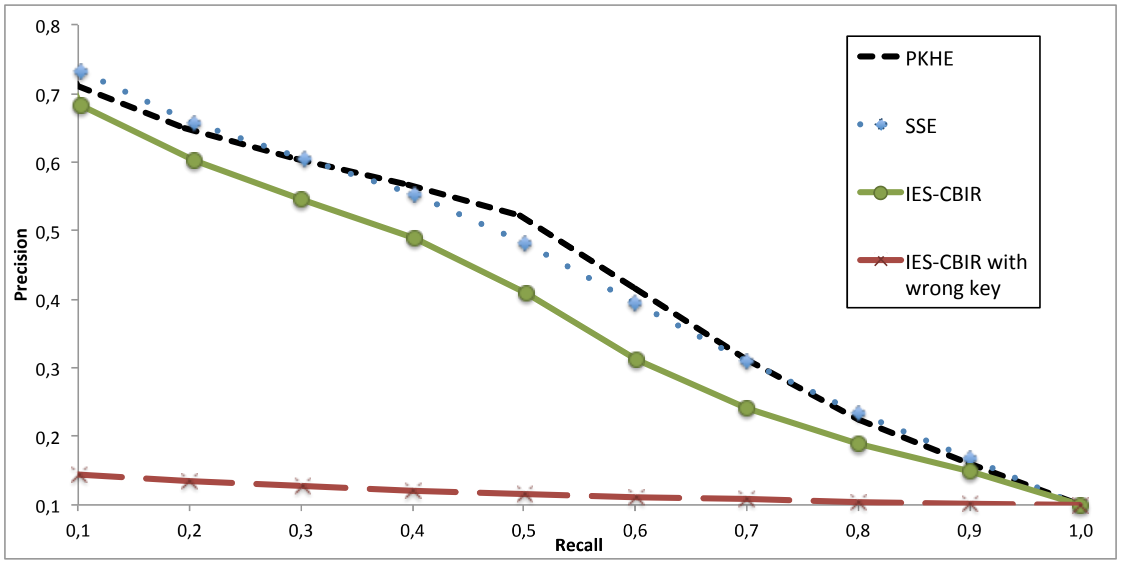

Fig. 6: Precision vs Recall Graph for the Wang Dataset

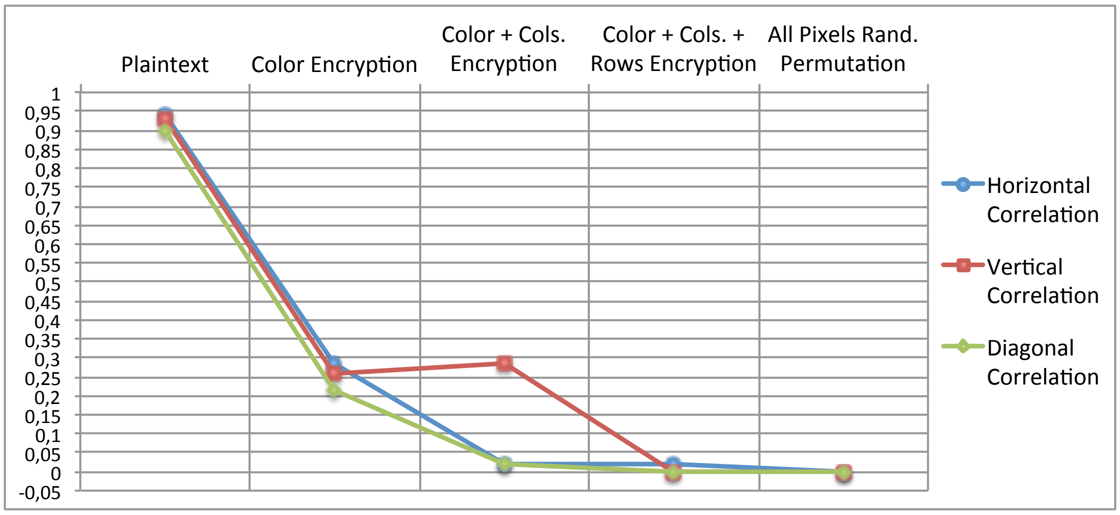

Fig. 7: Average Vertical, Horizontal and Diagonal Correlation between all pixels of all images in the Wang Dataset

color encryption only; the third is color encryption plus columns shifting; the fourth is color encryption plus columns and rows shifting (i.e., full IES-CBIR encryption); and the last point is random permutation of all pixel positions between each others.

The results show that color encryption alone lower pixel correlation levels, albeit not enough (avg. $0,25$ correlation). Adding columns and rows random shifting (texture encryption), correlation level becomes close to 0 ($0,0006$ for vertical and diagonal shifts and $0,02$ for horizontal). With random permutation of all pixels we can further decrease correlation by one order of magnitude ($0,0001$ and $0,00003$), but at a much higher performance cost ($w \times l$ random numbers and permutations required instead of $w + l$) and with little gain in terms of correlation.

VI. Conclusions

In this paper we have proposed a new secure framework for the privacy-preserving outsourced storage, search, and retrieval of large-scale, dynamically updated image repositories, where the reduction of client overheads was a central aspect. In the basis of our framework is a novel cryptographic scheme, specifically designed for images, named IES-CBIR. Key to its design was the observation that in images, color information can be separated from texture information, enabling the use of different encryption techniques with different properties for each one and allowing privacy-preserving Content-Based Image Retrieval to be performed by third-party, untrusted cloud servers. We formally analyzed the security of our proposals, and additional experimental evaluation of implemented prototypes shows that our approach achieves an interesting trade-off between precision and recall in CBIR, while exhibiting high performance and scalability when compared with alternative solutions. As future work we plan to explicitly address orthogonal issues in our design, such as key sharing, user access control, and reliability issues. We also plan to investigate the applicability of our methodology – i.e. the separation of information contexts when processing data (color and texture in this paper) – in other domains beyond image data.

Acknowledgments: This work is partially funded by FCT under projects PEst UID/CEC/04516/2013 and CMUP-ERI/FIA/0048/2013.

References

[1] M. Meeker and L. Wu, "Internet Trends," in *D11 Conf.*, 2013.

[2] Global Web Index, "Instagram tops the list of social network growth," http://blog.globalwebindex.net/instagram-tops-list-of-growth, 2013.

[3] C. D. Manning, P. Raghavan, and H. Schütze, *An Introduction to Information Retrieval*. Cambridge University Press, 2009, vol. 1.

[4] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, "Controlling data in the cloud: outsourcing computation without outsourcing control," in *CCSW'09*, 2009.

[5] D. Rushe, "Google: don't expect privacy when sending to Gmail," http://tinyurl.com/kjga34x, 2013.

[6] G. Greenwald and E. MacAskill, "NSA Prism program taps in to user data of Apple, Google and others," http://tinyurl.com/oea3g8t, 2013.

[7] A. Chen, "GCreep: Google Engineer Stalked Teens, Spied on Chats," http://gawker.com/5637234, 2010.

[8] J. Halderman and S. Schoen, "Lest we remember: cold-boot attacks on encryption keys," in *Commun. ACM*, vol. 52, no. 5, 2009, pp. 91–98.

[9] National Vulnerability Database, "CVE Statistics," http://web.nvd.nist.gov/view/vuln/statistics, 2014.

[10] D. Lewis, "iCloud Data Breach: Hacking And Celebrity Photos," https://tinyurl.com/nohznmr, 2014.

[11] P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish, "Depot: Cloud Storage with Minimal Trust," *ACM Trans. Comput. Syst.*, vol. 29, no. 4, pp. 1–38, Dec. 2011.

[12] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic evaluation of the AES circuit," in *CRYPTO'12*. Springer, 2012, pp. 850–867.

[13] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT'99*, 1999, pp. 223–238.

[14] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Adv. Cryptol.* Springer, 1985, pp. 10–18.

[15] C.-Y. Hsu, C.-S. Lu, and S.-c. Pei, "Image Feature Extraction in Encrypted Domain With Privacy-Preserving SIFT," *IEEE Trans. Image Process.*, vol. 21, no. 11, pp. 4593–4607, 2012.

[16] P. Zheng and J. Huang, "An efficient image homomorphic encryption scheme with small ciphertext expansion," in *MM'13*. ACM, Oct. 2013.

[17] W. Lu, A. Swaminathan, A. L. Varna, and M. Wu, "Enabling Search over Encrypted Multimedia Databases," in *IS&T/SPIE Electron. Imaging*, Feb. 2009, pp. 725 418–725 418–11.

[18] X. Yuan, X. Wang, C. Wang, A. Squicciarini, and K. Ren, "Enabling Privacy-preserving Image-centric Social Discovery," in *ICDCS'14*. IEEE, 2014, pp. 198–207.

[19] 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.

[20] J. Z. Wang, J. Li, and G. Wiederhold, "SIMPLIcity: Semantics-sensitive Integrated Matching for Picture LIbraries," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 9, pp. 947–963, 2001.

[21] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. CRC PRESS, 2007.

[22] H. Müller, W. Müller, D. M. Squire, S. Marchand-Maillet, and T. Pun, "Performance evaluation in content-based image retrieval: overview and proposals," *Pattern Recognit. Lett.*, vol. 22, no. 5, pp. 593–601, 2001.

[23] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable Symmetric Encryption: Improved Denitions and Efcient Constructions," in *CCS'06*, 2006, pp. 79–88.

[24] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient Similarity Search over Encrypted Data," in *ICDE'12*, 2012, pp. 1156–1167.

[25] F. Hahn and F. Kerschbaum, "Searchable Encryption with Secure and Efficient Updates," in *CCS'14*. ACM, 2014, pp. 310–320.

[26] R. A. Popa, F. H. Li, and N. Zeldovich, "An Ideal-Security Protocol for Order-Preserving Encoding," *IEEE S&P*, pp. 463–477, May 2013.

[27] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *NDSS*, 2012.

[28] J. R. Troncoso-Pastoriza and F. Perez-Gonzalez, "Secure signal processing in the cloud: enabling technologies for privacy-preserving multimedia cloud processing," *IEEE SPM*, vol. 30, no. 2, 2013.

[29] M. Schneider and T. Schneider, "Notes on Non-Interactive Secure Comparison in Image Feature Extraction in the Encrypted Domain with Privacy-Preserving SIFT ," in *IH&MMSec14*. ACM, 2014.

[30] A. Nourian and M. Maheswaran, "Privacy aware image template matching in clouds using ambient data," *J. Supercomput.*, vol. 66, no. 2, pp. 1049–1070, 2013.

[31] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," *ASIACRYPT'10*, vol. 6477 LNCS, pp. 577–594, 2010.

[32] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *IJCV*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[33] M. J. Swain and D. H. Ballard, "Color indexing," *Int. J. Comput. Vis.*, vol. 7, no. 1, pp. 11–32, 1991.

[34] A. Shraer, C. Cachin, A. Cidon, I. Keidar, Y. Michalevsky, and D. Shaket, "Venus: Verification for untrusted cloud storage," in *CCSW'10*. ACM, 2010, pp. 19–30.

[35] M. Brandenburger, C. Cachin, and N. Knežević, "Don't trust the cloud, verify: Integrity and consistency for cloud object stores," in *SYSTOR'15*. ACM, 2015.

[36] B. H. Kim and D. Lie, "Caelus: Verifying the consistency of cloud services with battery-powered devices," in *SP'15*. IEEE, 2015.

[37] G. Danezis and S. Gürses, "A critical review of 10 years of Privacy Technology," *Surveill. Cult. A Glob. Surveill. Soc.*, pp. 1–16, 2010.

[38] P. Weinzaepfel, H. Jégou, and P. Pérez, "Reconstructing an image from its local descriptors," in *CVPR'11*. ACM, 2011, pp. 337–344.

[39] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *CVPR*, vol. 2. IEEE, 2006, pp. 2161–2168.

[40] R. Canetti, "Universally Composable Security: A New Paradigm for Cryptographic Protocols," in *FOCS'01*, 2001, pp. 136–145.

[41] H. Jegou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *Comput. Vision-ECCV*. Springer, 2008, pp. 304–317.