Practical Privacy-Preserving Content-Based Retrieval in Cloud Image Repositories

Bernardo Ferreira, *Student Member, IEEE*, João Rodrigues, João Leitão, *Member, IEEE*, and Henrique Domingos, *Member, IEEE*

Abstract—Storage requirements for visual data have been increasing in recent years, following the emergence of many highly interactive multimedia services and applications for mobile devices in both personal and corporate scenarios. 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 security 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 shared image repositories. Our proposal is based on IES-CBIR, a novel Image Encryption Scheme that exhibits Content-Based Image Retrieval properties. The framework enables both encrypted storage and searching using Content-Based Image Retrieval queries while preserving privacy against honest-but-curious cloud administrators. We have built a prototype of the proposed framework, formally analyzed and proven its security properties, and experimentally evaluated its performance and retrieval 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 paves the way for new practical application scenarios.

Index Terms—Data and computation outsourcing, encrypted data processing, searchable encryption, content-based image retrieval

1 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, especially through mobile devices, is growing at an ever increasing rate. The storage needs for such large amounts of data in resource-constrained mobile devices 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 also leads to the need for content-based search and retrieval solutions (CBIR) [3].

Despite the fact that data outsourcing, especially to cloud computing infrastructures, seems a natural solution to support large scale image storage and retrieval systems, it also raises new challenges in terms of data privacy control. This is a consequence of outsourcing data, which usually implies releasing control (and some times even effective ownership) over it [4]. Recent incidents have provided clear evidence that privacy should not be expected to be preserved by cloud providers [5], [6]. Furthermore, malicious or simply careless 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 the danger these threats pose 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 unacceptable 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 largely unpractical, namely those requiring fully homomorphic encryption, which is still computationally too expensive [12]. Nonetheless, partially homomorphic encryption schemes [13], [14], [15], [16] and symmetric-key solutions (or property-preserving schemes) supporting specific search patterns [17], [18], [19] are interesting alternatives, yielding more practical results while providing a good trade-off 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. This prohibitive complexity is even

• *The authors are with the Faculdade de Ciências e Tecnologia, Universidade NOVA de Lisboa and NOVA LINCS, Caparica P-2829-516, Portugal. E-mail: bernardo.ferreira.pt@ieee.org, jm.rodrigues@campus.fct.unl.pt, {jc.leitao, hj}@fct.unl.pt.*

Manuscript received 3 Nov. 2016; revised 10 Feb. 2017; accepted 12 Feb. 2017. Date of publication 16 Feb. 2017; date of current version 4 Sept. 2019. Recommended for acceptance by B. Bhargava.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2017.2669999

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

further exacerbated if we consider mobile (resource constrained) clients, which are already responsible for more than 30 percent of internet traffic [1].

To address these challenges we propose a new secure framework for privacy preserving outsourced storage, search, and retrieval of large-scale, dynamically updated image repositories. We base our proposal on IES-CBIR, a novel Image Encryption Scheme (IES) with Content-Based Image Retrieval properties. Key to the design of IES-CBIR is the observation that in image processing, distinct feature types can be separated and encrypted with different cryptographic algorithms. As an example, image color and texture data can be separated in such a way that CBIR in the encrypted domain can be performed on one feature type while the other remains fully randomized and protected with semantically-secure cryptography. Following this observation, and considering that texture is usually more relevant than color in object recognition [20], in IES-CBIR we make the following security-oriented tradeoff: we choose to privilege the protection of image contents, by encrypting texture information with probabilistic (semantically-secure) encryption [21]; then we controllably 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 guarantees.[2] Notably, our solution allows outsourcing servers to generate and update an index used to efficiently process and reply to queries, a task that in many state of art solutions must be managed by client devices. As we show further ahead in the paper, our new methodology leads to optimized computation and communication overheads with non-negligible impact on system performance and mobile battery consumption.

In summary, this paper makes the following contributions: (i) We formally define IES-CBIR, a novel Image Encryption Scheme with Content-Based Image Retrieval properties, 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 to be performed by the client (i.e., indexing of dynamically added/updated images), hence circumventing performance pitfalls that exist in current state of art proposals [15], [16], [17], [18], [19]; (iii) We formally prove the security of our framework and 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 mobile; (v) And finally we show that the retrieval precision and recall of the proposed solution is on par with the current state of art [15], [17].

The work presented in this paper was first introduced in [22]. Here we extend our exposition significantly by discussing two use cases where IES-CBIR and the proposed framework can be applied with immediate benefits. We further provide a complete formal security evaluation of our proposals and a performance analysis of the search operation of our framework in comparison with relevant previous works. Additionally we provide a statistical security analysis of IES-CBIR and its entropy levels at each step of encryption and the complete description of all framework operations.

2. The proposed solution also shows potential for extension to other applicational domains.

2 Related Work

Previous proposals for supporting outsourced storage, search, and retrieval of images in the encrypted domain can be broadly divided in two classes: those based on Searchable Symmetric Encryption (SSE) techniques and those based on Public-Key partially-Homomorphic Encryption (PKHE). SSE has been widely used in the past by the research community, especially for text data [23], [24], [25]. In the image domain, even though not identified as SSE schemes, multiple systems use the same (or similar) techniques for image search/retrieval [17], [18], [19], [26]. For simplicity, we refer to these as S-based solutions. In SSE-based solutions, clients process their data before encrypting and 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 probabilistic symmetric-key encryption schemes, while the index is protected through a combination of probabilistic and deterministic (or even order-preserving [27]) 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], [26], which entails the use of additional computational power on their side and limits the practicality of such solutions for resource-constrained mobile devices. This effect is further exacerbated when considering dynamic scenarios, where images are constantly being added, updated, and removed. In such dynamic scenarios, SSE solutions usually require multiple rounds of communication for updating image repositories and their indexes. For instance, a previous approach by Lu et al. [17] uses repository-wide statistics (e.g., inverse-document frequencies), which change as the repositories are updated and thus force the re-construction and re-encryption of the index, requiring clients to download and decrypt the full contents of the repository. Additionally index values are encrypted with an order-preserving encryption scheme that depends on plaintext domain distribution. With multiple updates this distribution changes, again requiring the re-construction and re-encryption of the index. This is an important issue from a security viewpoint. Other approaches from the literature require multiple rounds of communication for performing such operations [18], [24], [26];
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 latency of storage operations as perceived by users and being a particular issue for cloud-backed deployments;

TABLE 1
Overview of Information Leakage and Average Complexities (on the Client-Side) for the Most Relevant Privacy-Preserving CBIR Approaches and IES-CBIR (This Work)

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

iii) As SSE works use deterministic tokens to provide their functionality with practical performance. Deterministic tokens include unique document identifiers and deterministic encryptions of keywords. Consequently they leak what are known as *search*, *access*, *similarity*, and *update* patterns [18], [23], [24], [25], i.e., they reveal respectively: if a query has been submitted before; which images are returned for 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. These leakage patterns result in exposing as much information as a fully deterministic encryption scheme, albeit with much higher computational overhead. This is demonstrated in [28] and is particularly evident in long-lived system with many queries being executed concurrently and all index entries being accessed. 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 (including plaintext distribution knowledge) [28].

The alternatives to SSE that can be found in the literature [15], [16] are based on public-key partially-homomorphic encryption 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 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 significant ciphertext expansion (for a secure key size of 1,024 bits, each pixel is transformed from its traditional 24 bits representation into 2,048 ciphertext bits), slow encryption and decryption times (as we experimentally demonstrate in Evaluation Section 5.1), and in limited scalability (the "ciphertext blowup" problem [29], i.e. when ciphertext values reach their arithmetic group limits through multiple multiplications). Furthermore their work was later shown to be either insecure or computationally intractable for a typical cloud server [30]. Zheng et al. [16] proposed a variant of that work, overcoming some of its drawbacks by replacing Paillier ciphertexts with pointers to a ciphertext table with all possible ciphertext pixel values. This approach can potentially reduce the number of encryption operations and minimize ciphertext expansion in some use cases. However Paillier encryptions still present a significant computational overhead, limiting the practicality of the approach [16].

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. [31] which aims at providing privacy-preserving single image template matching performed by third-party clouds. However, this work doesn't support large-scale repositories, 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 requires 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. [32], which proposes 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. 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). In our proposal we focus on allowing the generation of indexes over large collections of encrypted images by an untrusted third party and the efficient and precise resolution of user queries over these large image collections.

The reader should note that most research works on privacy-preserving image retrieval are not actually proven secure. In some works the underlaying cryptographic primitives used are well known 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 approaches rely on small modifications that can actually compromise security. Such is the case of [15] and [31], as discussed in the previous paragraphs, and [19] which bases its security solely on hashing functions which can be compromised through dictionary attacks [21].

Table 1 summarizes key aspects of the state of art, 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 clients. We also implemented these works and experimentally compare them with a prototype of our framework in Section 5. In the Table, the *Information Leakage* column represents the leakage of all system operations (particularly the update, search, and remove operations) as a whole; *Local Index Size* represents a maximum bound on the possible index size on the client device; and the CBIR Algorithm column represents

the CBIR algorithms used in each work: local color histograms [17], SIFT [33], and global color histograms [34]. Also in the table, ID_I is a deterministic identifier of an image I being stored/updated or used as a query image in a search; 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 Section 4.2); 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; $|Rep|$ is the size of the repository; and $|CB|$ is the size of the clustering codebook.

3 Technical Overview

To overcome the limitations of the state of art, we propose a framework for 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, which allows us to design outsourced image repository systems that support content-based image retrieval based on color features, while protecting the privacy of both image owners and other users issuing queries. Regarding the state-of-art, IES-CBIR shows comparable retrieval precision and higher computational performance than previous approaches as perceived by 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 work presented in Section 5, where the performance of a IES-CBIR system is compared against the state-of-art SSE [17] and PKHE [15] based approaches.

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, where each user accesses one or more repositories to 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 images in the repositories, in conjunction with the respective repository keys.

In the following sections we present the system model for our proposed framework (Section 3.1), followed by its adversary model and security assumptions (Section 3.2) and by some relevant use cases we envision for the application of our proposal (Section 3.3).

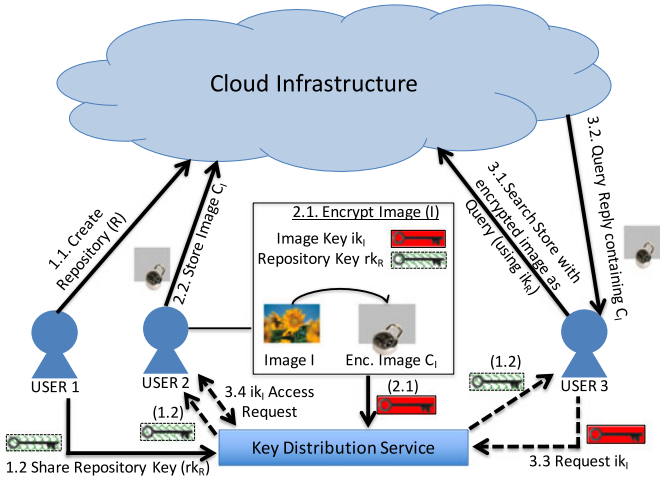

Fig. 1. System model overview of the proposed framework.

3.1 System Model and Architecture

We now describe the 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* that reside in 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.

Each repository is 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 add/update images. To add/update images (but not search), a user further needs an *image key* 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 an option in our framework, i.e., if the users of a repository prefer to avoid further key management overhead and are willing to sacrifice fine-grained access control, they can use the same image key for all images in a repository.

When the cloud receives an encrypted image for storage it extracts its relevant features (in our framework, we use global color features [34]) 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 has its features extracted and matched with the repository's index. The reply to a query will contain k (a tunable system parameter) 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 by resorting to a key distribution service, implemented either in a centralized way (using protocols such as Kerberos [35] or in a distributed fashion (through

asynchronous communications or protocols such as Diffie-Hellman [21]). User authorization and revocation can also be easily achieved, for instance, 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 contribution, as the mechanisms involved can be easily integrated into our framework.

3.2 Adversary Model

In this work we focus at protecting the privacy of users' images and queries. The main adversary we consider is the *cloud administrator*, who operates the cloud's infrastructure and servers. Similar to many previous works found in the literature [17], [18], [23], [24], [25], [27], we assume an *honest-but-curious* adversary [4], meaning that 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 and disclosure users' data. We assume that a malicious cloud administrator has access to all data stored on disk or in RAM on any device located at the cloud infrastructure, and passing through the network from or to the cloud. In Section 4.4, we formally prove the security of our framework against such an adversary.

A stronger adversary that should also be considered is the *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 *honest-but-curious* cloud administrator, and leave the malicious users challenge as a 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 [23]. Furthermore, we don't consider integrity or availability threats, as these can be handled by different mechanisms that are orthogonal to our contributions [36], [37], [38].

3.3 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 (PHR). Personal health records 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. A repository key is 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 and with her authorization.

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 image repositories storage to a *cloud provider*, while coping 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 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.

4 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 (Section 4.1). Then we explain how the framework's component in the cloud side leverages IES-CBIR properties to store, index, and search images while preserving their privacy (Section 4.2). Finally we detail the protocols employed by our framework (Section 4.3) and formally prove its security (Section 4.4).

4.1 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 that 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 [39]. 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, safeguarding 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. Indeed, color information is given from pixel color values in the different channels of a particular color model; while texture information is given by the (relative) position of pixels and strong

3. E.g., https://www.healthvault.com, http://www.cleardata.com/

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 observations are further supported by the most recent works in image reconstruction [40], which not only depend on local features extracted from sub-segments of 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 remarks 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 Section 4.4). We define IES-CBIR as:

Definition 1 (IES-CBIR). *An Image Encryption Scheme with CBIR properties 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*;*
- 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*;*
- Enc(I, rk, ik)*: takes as input an image* I *and the cryptographic keys* $\{rk, ik\}$*, returning an encrypted image* C_I*;*
- Dec(C_I, rk, ik)*: takes as input an encrypted image* C_I *and keys* $\{rk, ik\}$*, returning the decrypted image* I*;*
- TrpGen(Q, rk)*: takes as input a query image* Q *and a repository key* rk*, returning a searching trapdoor* C_Q*;*

4.1.1 Key Generation

As already discussed (in Section 3.1) 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. Repository keys deterministically map a pixel's color value in a color channel to some new random value.[4] To prevent images from increasing in size after encryption (i.e., prevent ciphertext expansion), encrypted pixels should be in the same range of values as their original plaintexts (usually 8 bits per color channel). As such, we build repository keys in IES-CBIR by performing random permutations of all possible pixel color values in each color channel. Leveraging the HSV color space ((H) hue, (S) saturation, (V) value/brightness), we perform three independent random permutations of the values in range $[0..100]$. This range represents all possible color values in the HSV color space, and each permutation is used for a different color channel, resulting in three repository sub-keys: rk_H, rk_S, rk_V. Permutations are performed by a Pseudo-Random Generator (PRG) [21] $\mathcal{G}$ parameterized with the security parameter sp_{rk} as random seed (in our implementation we use an AES-based PRG [21] for $\mathcal{G}$ and 128 bits for sp_{rk}). Besides limiting ciphertext expansion, this approach allows image processing operations to be executed in the encrypted domain without alterations, including image indexing, searching, and compressing operations

$$rk_z \leftarrow RandPerm(\mathcal{G}_{sp_{rk}}, [0..100]) : \ \forall z \in (H, S, V), \\ rk = \{rk_H, rk_S, rk_V\}. \tag{1}$$

Equation (1) formalizes the algorithm for generating repository keys. In contrast, image keys are generated by requesting a number of pseudorandom bits to $\mathcal{G}$ (initialized with some random seed) equal to sp_{ik} (we use 128 bits in our implementation). Image keys will be used as a cryptographic seeds for the probabilistic encryption step of IES-CBIR.

4.1.2 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 $rk = \{rk_H, rk_S, rk_V\}$. Equation (2) represents this operation, where $\mathcal{P}_{rk}(p_z)$ is the encryption of pixel p in color component z through $\mathcal{P}$ and key rk_z, and c_{p_z} is the resulting ciphertext

$$c_{p_z} \leftarrow \mathcal{P}_{rk_z}(p_z) : \forall z \in (H, S, V), \ \forall p \in I, \ \forall c_p \in C_I. \tag{2}$$

This step of encryption securely hides color values of 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. In this step 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, shifting that column r positions downward, 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).

4. Instead of encrypting pixel color values with deterministic encryption, we could also use Order-Preserving Encryption [27], slightly increasing image retrieval precision at the expense of greater information leakage.

Equations (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)

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

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

The above 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 it computationally unfeasible to extrapolate correlations between 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 Section 4.4).

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 any 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.

4.1.3 Decryption

The decryption algorithm applies the different steps of encryption in the inverse order, or more formally, through the ordered application of the transformations denoted by Equations (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

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

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

$$p_z \leftarrow \mathcal{P}_{rk_z}(c_{p_z}) : \forall z \in (H,S,V),\ \forall p \in I,\ \forall c_p \in C_I. \tag{7}$$

4.1.4 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 (Equation (8), where the image key is substituted by a new ik randomly generated for the query). This means that searching trapdoors are also decryptable, and can be stored in the repositories as new images as long as users locally save the image keys generated for the queries

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

4.2 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. In our model, this yields three color histograms with entries in range [0, 100], which are the admissible values for each HSV channel (i.e., each histogram has 101 entries).

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 [41] 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 [41]) 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 Distance). Finally, the cloud server builds an inverted list index, with all visual words as keys and the list of images most close to them (plus a frequency score) as values. 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 search trapdoors for some query images of their choice. When a new search 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 (key to ensuring scalability). After receiving search results, users can explicitly request full access to images by requesting the corresponding image keys from their owners.

4.3 Framework Protocols

In the next paragraphs we detail the protocols of our framework based on IES-CBIR. In these protocols we omit operations related with the request and sharing of keys, as these are orthogonal to the scope of our work.

Instantiate a New Repository. We start by describing the operation used by a user U to create a new repository R (Algorithm 1). 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 (lines 3-6). The user then sends the initialization parameters, pseudorandom ids (including his own id) and encrypted images to the cloud server (line 7). The cloud starts by initializing the storage space Rep_R and index Idx_R for R (lines 11-12), and then extracts the color feature-vectors (histograms) of all the d initial images (lines 13-14). Then it hierarchically clusters these d feature-vectors, building codebook CB_R (line 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 (lines 17-23).

Store/Update Image. Algorithm 2 illustrates the procedure followed by a user U to store a new image I in repository R, or update it if it already exists. 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 Algorithm 1's instructions (since Algorithm 1 also stores a group of initial images), where the only difference is the creation of codebook CB_{ID_R} in Algorithm 1. We point to the presentation of Algorithm 1 for a detailed explanation of each instruction.

Search with an Image as Query. Algorithm 3 sketches the procedure to search in a repository R with query image Q. 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 (line 2). Then she sends it to the cloud server, along with k and ID_R, as parameters for the Search remote invocation (line 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} (lines 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 (lines 12-15). In this set, 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 (line 18).

Algorithm 1. Operation 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**

Algorithm 2. Operation 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**

Access an Image. Algorithm 4 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 3. Operation Search with 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: $vw_{C_Q} = \{ID_{vw_i}, freq^{C_Q}_{vw_i}\}_{i=0}^{|vw_{C_Q}|} \leftarrow CB_R$.Stem($fv_{C_Q}$)
10: **for all** $\{ID_{vw_i}, freq^{C_Q}_{vw_i}\}_{i=0}^{|vw_{C_Q}|}$ **do**
11: $PL_{vw_i} = \{ID_{I_j}, freq^{C_{I_j}}_{vw_i}\}_{j=0}^{|PL_{vw_i}|} \leftarrow Idx_R[ID_{vw_i}]$
12: **for all** $\{ID_{I_j}, freq^{C_{I_j}}_{vw_i}\}_{j=0}^{|PL_{vw_i}|}$ **do**
13: $score^Q_{I_j} \leftarrow$ScaledTfIdf $(freq^{C_Q}_{vw_i}, freq^{C_{I_j}}_{vw_i}, |Rep_{ID_R}|, |PL_{vw_i}|)$
14: $\{C_{I_j}, ID_{U_j}\} \leftarrow Rep_R[ID_{I_j}]$
15: $qr[ID_{I_j}] \leftarrow \{C_{I_j}, qr[ID_{I_j}].score + score^Q_{I_j}, ID_{U_j}\})$
16: **end for**
17: **end for**
18: **return** resize(k, Sort(qr))
19: **end procedure**

Algorithm 4. Operation Access 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**

Remove an Image. Algorithm 5 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 (lines 3-5). In this protocol we assume the presence of an authorization mechanism, which enforces that users can only remove their own images.

Algorithm 5. Operation Remove 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**

4.4 Security Analysis and Proofs

In this section we prove the security properties of our work. Our security proofs follow the *real/ideal* (or simulation-based) paradigm that is conventional in secure multi-party computations [42], including the SSE literature [23], [24], [25]. This paradigm has the advantage of clearly specifying what information is leaked by each operation (through an *idealized functionality*) and proving that a concrete implementation (in this case, our framework based on IES-CBIR) securely fulfills it and leaks nothing else to adversaries.

Algorithm 6 formalizes the *ideal functionality* $\mathcal{F}$ of our framework, which specifies all information leaked. In $\mathcal{F}$ we consider as adversary the honest-but-curious cloud administrator (Section 3.2), which corrupts the cloud server passively. As stated in the Related Work Section 2, the leakage functions specified in Algorithm 6 are equivalent to the *search, access, similarity*, and *update* leakages of SSE-based works [18], [23], [24], [25], particularly for any long-lived system with many queries being executed as expected in real-world application scenarios. Furthermore applications using our framework can ensure that the information leaked will not compromise security, by limiting the amount of background information made available to adversaries [28].

The proof that our framework 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 (i.e., the adversary) in a real interaction with the user through an instance of our framework (the real experiment), and that the two experiments would be indistinguishable (apart from a negligible probability [21]), 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 a query to a repository, the server basically learns the frequency of its visual words (and consequently its similarity to other images in the repository), but nothing more. In the real experiment, the user will invoke (through our framework 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 cloud server beyond what is specified in $\mathcal{F}$.

Formally $\mathcal{S}$ can simulate the view of the cloud server 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, from a brute-forcing point of view, there is a limitation on the size of the images being stored and searched. If we consider: (i) 128 bits as a minimum bound for security (as recommended, for instance, for block-cipher algorithms such as AES [21]), and (ii) that the brute-forcing effort on an image encryption is $w^h \times h^w$ (w and h are the image width and height, respectively), since w and h random values need to be resolved, with possible values ranging in [0..h] and [0..w] respectively; than images should be at least 16×16 pixels to achieve 128 bits security, since $w^h \times h^w = 16^{16} \times 16^{16} = 2^{128}$. For images smaller than that, an adversary can possibly compromise the probabilistic counterpart of IES-CBIR 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. *The 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 16×16 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. □

Algorithm 6. 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.
  - **Search Leakage** $\mathcal{F}$ sends to the server ID_R, ID_Q (a deterministic id for Q generated by $\mathcal{F}$), k, w_Q (width of Q in pixels), h_Q (height of Q), and $\{ID_{vw_j}, freq_{ID_{vw_j}}^{ID_Q}\}_{j=0}^{|vw_Q|}$ (the visual words in Q and their frequencies).
- $\mathcal{F}$.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.

When it receives the CreateRepository message from $\mathcal{F}$ with its *Setup Leakage* = $\{ID_R, ID_U, d, n, m, \{ID_{I_i}, w_{I_i}, h_{I_i}, \{ID_{vw_j}, freq_{ID_{vw_j}}^{ID_{I_i}}\}_{j=0}^{|vw_{I_i}|}\}_{i=0}^{d}\}$, $\mathcal{S}$ initializes some data structures: **(i)** A simulated repository $Rep'_R = \{ID_{I_i}\{I'_i, ID_{U_i}\}\}_{i=0}^{*}$, 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 $\{ID_{vw_i}, vw'_i\}_{i=0}^{n}$, where vw'_i is a simulated visual word; **(iii)** a simulated index $Idx'_R = \{ID_{vw_i}, PL'_{vw_i}\}_{i=0}^{n}$, where $PL'_{vw} = \{ID_{I_i}, freq_{ID_{vw}}^{ID_{I_i}}\}_{i=0}^{*}$ is a simulated posting list of the images that contain vw' and respective frequencies; **(iv)** a simulated search list $Search'_R = \{ID_{Q_i}, Q'_i, \{ID_{vw_j}, freq_{ID_{vw_j}}^{ID_{Q_i}}\}_{j=0}^{|vw_{Q_i}|}\}_{i=0}^{*}$ that stores all performed queries; **(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 performs hierarchical clustering, resulting in 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 through two steps, first by a pseudorandom permutation [21] of their color values and then by a random swapping of their pixel positions through a pseudorandom generator [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 I to be 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 *Search Leakage* = $\{ID_R, ID_Q, k, w_Q, h_Q, \{ID_{vw_j}, freq_{ID_{vw_j}}^{ID_Q}\}_{j=0}^{|vw_Q|}\}$, $\mathcal{S}$ 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 $vw'_Q = \{ID_{vw_i}, vw'_i, \ freq^{ID_Q}_{ID_{vw_i}}\}^{|vw'_Q|}_{i=0}$. Then $\mathcal{S}$ accesses $Search'_R$ and stores $\{Q', \{ID_{vw_j}, \ freq^{ID_Q}_{ID_{vw_j}}\}^{|vw_Q|}_{j=0}\}$ in position $[ID_Q]$. 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.

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.

5 Experimental Evaluation

In this section we experimentally evaluate our proposal. To this end we have implemented, in the Java language, a prototype of our IES-CBIR based framework (as described in Section 4) and prototypes of competing relevant alternatives for baseline comparison: i) the SSE 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*). We compare our approach with these two solutions, because they represent the state of art in the two ends of the spectrum of encrypted CBIR solutions: [17] represents SSE based solutions, where the index is built before data is outsourced to the cloud infrastructure; while [15] represents solutions based on partially-homomorphic encryption, where complex image processing algorithms can be performed over outsourced encrypted data. The code of these prototypes is open source and available at: https://github.com/bernymac/IES-CBIR.

Using 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 and cloud computations. 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 [43]. 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 1,000 low-resolution images with a JPEG compressed size of 29.8 MB; and the Inria Holidays Dataset [44], containing 1,491 high-resolution images with total JPEG compressed size of 2.85 GB.

Fig. 2 shows an example of an image from the Holidays Dataset and the result of its IES-CBIR encryption. We present our results in the following order: first we discuss the performance and scalability of our solution when storing and searching images, comparing it with the alternative approaches; then we study the achieved retrieval precision; finally we analyse the statistical entropy generated by IES-CBIR encryption algorithm.

Fig. 2. Example image from the Holidays dataset and its encryption with IES-CBIR.

5.1 Store/Update Performance

In these experiments we used the larger-sized Holidays dataset to analyse the performance of our system, with and without image compression (labeled *IES-CBIR w/ compression* and *IES-CBIR no compression* respectively) with the SSE and PKHE alternatives. To this end we have measured the time taken by the Update Image operation considering two distinct workloads: one where the 1,491 (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 1,491 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 that emerge when 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 for each repository, as described in Section 4.2) and *Cloud Storage*,[5] 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 elapsed time for each operation from the users' perspective (without considering cloud computations, which are performed asynchronously). Each experiment shows the average of 10 independent runs.

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 processing throughput presented by 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

5. The cloud storage metric captures the amount of time spent in transferring data between the client and the cloud infrastructure. This is fundamentally affected by the amount of data that needs to be transferred, and hence this metric is closely related with ciphertext expansion and bandwidth consumption.

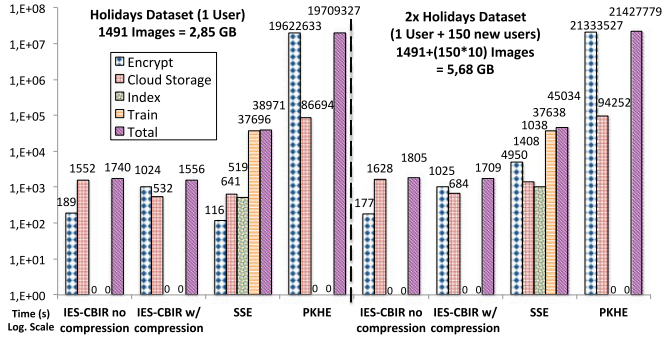

Fig. 3. Performance for the *store/update image* operation (log scale).

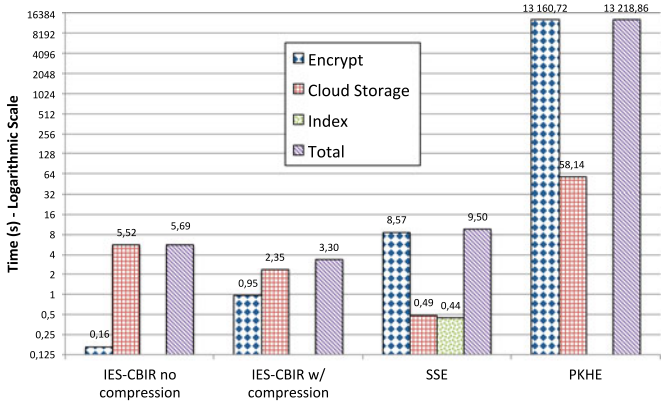

Fig. 4. Performance of the *search with query image* operation, for all analyzed alternatives ($\log_2$ scale).

the cloud server); and *SSE* requires expensive initial training performed by the user that creates the repository, in order to build the clustering codebook required for indexing, and also requires the user to locally index his images before uploading them. Results for the *PKHE* system had to be simulated from the results with a smaller subset (of 10 images, which took approximately 36 hours to encrypt and upload), as experiments would take approximately 228 days in the first experiment and 248 days in the second.

An argument that could be made in favor of the *SSE* approach is that the initial training may only have to be performed once, amortizing its cost in the long run. However our second workload, where multiple users try to add additional images to the repository, shows that this 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 percent compared to 83 percent for *SSE* (ignoring its training/clustering time) and 8 percent 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 in a coordinated fashion 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 slower than its compressed counterpart.

5.2 Search Performance

Fig. 4 shows the experimental results for the SEARCH OPERATION, comparing 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 remaining approaches). The results showed here represent the performance for searching in the Inria Holidays dataset [44] 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 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 percent 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 variants) has higher Cloud computation time, as most of the work involved in this operation is done by the cloud server instead of the user device (as in the *SSE* approach), that overhead is still smaller than the difference between the encryption overheads in the two systems. Moreover, overhead 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].

5.3 Retrieval Precision and Recall

We start by defining the metrics used herein [45]: 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 [20] 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 [44]. 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 ($[1 \ldots 1{,}000]$). Similarly 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.

Fig. 5 summarizes the results. Our framework shows similar precision and recall as the compared alternatives, with a small variation of about 6 percent. This small difference is the advantage gained by these alternatives through the sacrifice of performance and scalability. Nonetheless, the reader should note that our approach can be extended to also consider texture information in its CBIR algorithm,

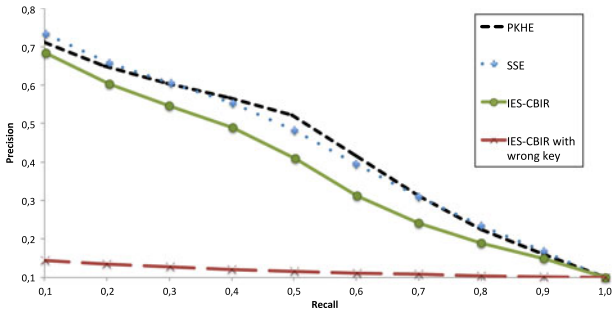

Fig. 5. Precision versus recall graph for the Wang dataset.

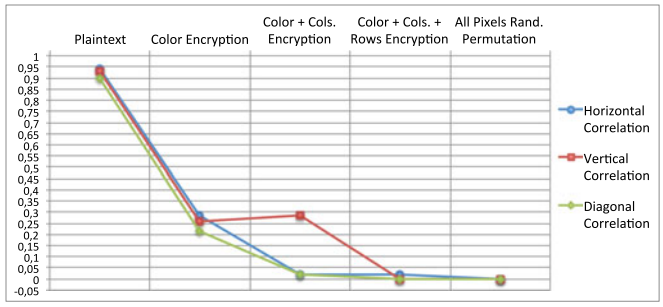

Fig. 6. Average vertical, horizontal and diagonal correlation between all pixels of all images in the Wang dataset.

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 picking random images from those repositories.

The second precision test consisted in using the evaluation package of the Holidays dataset (available online [44] for calculating the mAP of a group of 500 pre-defined queries. Table 2 shows the results. In this experiment, *PKHE* achieved the best result, as expected due to the use of the SIFT retrieval algorithm [33]. *IES-CBIR* achieved the second highest results, followed by *SSE*. Retrieval precision results for the *PKHE* system (in both experiments) were not substantially different from the other systems, even though it uses strong texture-based image features (in particular, SIFT). SIFT features were originally designed for object-recognition, and we believe that their use to search by example in image repositories (such as the ones used in our experiments and in the literature) does not leverage its full potential.

Comparing the results of the two experiments, 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 in the Inria dataset [44]. Based on all results presented so far, we conclude that a framework leveraging IES-CBIR can achieve a good trade-off between precision/recall and performance/scalability.

5.4 Experimental Security Evaluation

To finalize the experimental section of our work, we made a statistical analysis to experimentally assess the entropy level in IES-CBIR encrypted images. In this analysis we leveraged the co-variance correlation function of [31], measuring correlation levels between all horizontally, vertically, and diagonally adjacent pixels. Measurements were taken for pixels of original plaintext images, at different steps of IES-CBIR encryption process, and for a complete random permutation of all pixel positions. In this experiment correlation results range between $[-1 \ldots 1]$, with images of higher entropy closer to 0. As dataset we used the low-resolution Wang Dataset [20], 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. 6. The first set of points in the figure represents the plaintext images; the second represents IES-CBIR 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.

TABLE 2
Mean Average Precision (mAP) for the Holidays Dataset

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

The results show that color encryption alone lowers 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 respectively), but at a much higher performance cost ($w \times l$ random numbers and permutations required instead of $w + l$) and with small improvement in terms of correlation.

These results also show that there is a direct function between decreasing image correlation and performance. With only color encryption, one obtains very high performance with the highest correlation. The correlation obtained by column and row shifting (with color encryption) and by permutation of all pixels (also with color encryption) is very similar. However, we note that performance wise there is a significant performance difference. Considering the effort for generation of pseudorandom numbers, the first alternative, for an image with 16×16 pixels, needs $16 + 16 = 32$ values while the second alternative requires $16 \times 16 = 256$. Considering larger images with 3.264×2.448 pixels, as in the Inria Holidays Dataset, the difference between shifting rows and columns compared to permuting all pixels goes from $3.264 + 2.448 = 5.712$ to $3.264 \times 2.448 = 7.990.272$. We therefore conclude that IES-CBIR, with color encryption and column and row permutation, offers the most adequate balance between performance and resulting pixel correlation.

6 Conclusion

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 is 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 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 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 revealed 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. An interesting future work direction is to investigate the applicability of our methodology—i.e., the separation of information contexts when processing data (color and texture in this contribution)—in other domains beyond image data.

Acknowledgments

This work was supported by FCT/MCTES through project NOVA LINCS (UID/CEC/04516/2013) and the European Union, through project LightKone (grant agreement no. 732505).

References

[1] M. Meeker, "Internet trends 2015," in *Proc. Code Conf.*, 2015, pp. 1–196.
[2] Global Web Index, "Instagram tops the list of social network growth," 2013. [Online]. Available: http://tinyurl.com/hnwwlzm
[3] C. D. Manning, P. Raghavan, and H. Schütze, *An Introduction to Information Retrieval*, vol. 1. Cambridge, U.K.: Cambridge Univ. Press, 2009.
[4] R. Chow, et al., "Controlling data in the cloud: Outsourcing computation without outsourcing control," in *Proc. ACM Workshop Cloud Comput. Secur.*, 2009, pp. 85–90.
[5] D. Rushe, "Google: Don't expect privacy when sending to Gmail," 2013. [Online]. Available: http://tinyurl.com/kjga34x
[6] G. Greenwald and E. MacAskill, "NSA Prism program taps in to user data of Apple, Google and others," 2013. [Online]. Available: http://tinyurl.com/oea3g8t
[7] A. Chen, "GCreep: Google engineer stalked teens, spied on chats," 2010. [Online]. Available: http://gawker.com/5637234
[8] J. Halderman and S. Schoen, "Lest we remember: Cold-boot attacks on encryption keys," *Commun. ACM*, vol. 52, no. 5, pp. 91–98, 2009.
[9] National vulnerability database, "CVE Statistics," 2014. [Online]. Available: http://web.nvd.nist.gov/view/vuln/statistics
[10] D. Lewis, "iCloud Data Breach: Hacking And Celebrity Photos," 2014. [Online]. Available: https://tinyurl.com/nohznmr
[11] P. Mahajan, et al., "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 *Proc. 32nd Annu. Cryptology Conf. Advances Cryptology*, 2012, pp. 850–867.
[13] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. 17th Int. Conf. Theory Appl. Cryptographic Techn.*, 1999, pp. 223–238.
[14] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Advances in Cryptology*. Berlin, Germany: Springer, 1985.
[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, Nov. 2012.
[16] P. Zheng and J. Huang, "An efficient image homomorphic encryption scheme with small ciphertext expansion," in *Proc. 21st ACM Int. Conf. Multimedia*, 2013, pp. 803–812.
[17] W. Lu, A. Swaminathan, A. L. Varna, and M. Wu, "Enabling search over encrypted multimedia databases," in *Proc. 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 *Proc. IEEE 34th Int. Conf. Distrib. Comput. Syst.*, 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," *IEEE Trans. Inf. Forensics Secur.*, vol. 10, no. 1, pp. 152–167, Jan. 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, Sep. 2001.
[21] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Boca Raton, FL, USA: CRC Press, 2007.
[22] B. Ferreira, J. Rodrigues, J. Leitão, and H. Domingos, "Privacy-preserving content-based image retrieval in the cloud," in *Proc. IEEE 34th Symp. Reliable Distrib. Syst.*, 2015, pp. 11–20.
[23] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, 2006, pp. 79–88.
[24] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *Proc. IEEE 28th Int. Conf. Data Eng.*, 2012, pp. 1156–1167.
[25] F. Hahn and F. Kerschbaum, "Searchable encryption with secure and efficient updates," in *Proc. 13th ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 310–320.
[26] Z. Xia, Y. Zhu, X. Sun, Z. Qin, and K. Ren, "Towards privacy-preserving content-based image retrieval in cloud computing," *IEEE Trans. Cloud Comput.*, vol. PP, no. 99, pp. 1–11, 2015.
[27] R. A. Popa, F. H. Li, and N. Zeldovich, "An ideal-security protocol for order-preserving encoding," in *Proc. IEEE Symp. Secur. Privacy*, May 2013, pp. 463–477.
[28] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2012.
[29] J. R. Troncoso-Pastoriza and F. Perez-Gonzalez, "Secure signal processing in the cloud: Enabling technologies for privacy-preserving multimedia cloud processing," *IEEE Signal Process. Mag.*, vol. 30, no. 2, pp. 29–41, Mar. 2013.
[30] M. Schneider and T. Schneider, "Notes on non-interactive secure comparison in image feature extraction in the encrypted domain with privacy-preserving SIFT," in *Proc. 2nd ACM Workshop Inf. Hiding Multimedia Secur.*, 2014, pp. 135–140.
[31] A. Nourian and M. Maheswaran, "Privacy aware image template matching in clouds using ambient data," *J. Supercomputing*, vol. 66, no. 2, pp. 1049–1070, 2013.
[32] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *Proc. Int. Conf. Theory Appl. Cryptology Inf. Secur.*, 2010, pp. 577–594.
[33] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, Nov. 2004.
[34] M. J. Swain and D. H. Ballard, "Color indexing," *Int. J. Comput. Vis.*, vol. 7, no. 1, pp. 11–32, 1991.
[35] B. C. Neuman and T. Ts'o, "Kerberos: An authentication service for computer networks," *IEEE Commun. Mag.*, vol. 32, no. 9, pp. 33–38, Sep. 1994.
[36] A. Shraer, C. Cachin, A. Cidon, I. Keidar, Y. Michalevsky, and D. Shaket, "Venus: Verification for untrusted cloud storage," in *Proc. ACM Workshop Cloud Comput. Secur. Workshop*, 2010, pp. 19–29.
[37] M. Brandenburger, C. Cachin, and N. Knezevic, "Don't trust the cloud, verify: Integrity and consistency for cloud object stores," in *Proc. 8th ACM Int. Syst. Storage Conf.*, 2015, Art. no. 16.
[38] B. H. Kim and D. Lie, "Caelus: Verifying the consistency of cloud services with battery-powered devices," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 880–896.
[39] G. Danezis and S. Gürses, "A critical review of 10 years of privacy technology," in *Proc. Surveillance Cultures A Global Surveillance Soc.*, 2010, pp. 1–16.
[40] P. Weinzaepfel, H. Jégou, and P. Pérez, "Reconstructing an image from its local descriptors," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2011, pp. 337–344.
[41] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2006, vol. 2, pp. 2161–2168.
[42] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proc. 42nd IEEE Symp. Found. Comput. Sci.*, 2001, pp. 136–145.

[43] Amazon Web Services (AWS), "Amazon EC2 instance types," 2016. [Online]. Available: https://aws.amazon.com/pt/ec2/instance-types/
[44] H. Jegou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *Proc. 10th Eur. Conf. Comput. Vision*, 2008, pp. 304–317.
[45] 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.

Bernardo Ferreira received the BSc, MSc, and PhD degrees from Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa (FCT/UNL), in 2008, 2010, and 2016, respectively. He is currently a researcher with the NOVA LINCS Research Center. His research interests include the topics of cloud privacy, security, and dependability, with special emphasis on searchable encryption. He is a student member of the IEEE.

João Rodrigues received the BSc and MSc degrees from Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa (FCT/UNL), in 2011 and 2013, respectively. He is currently working toward the PhD degree at FCT/UNL, and a student researcher with the NOVA LINCS Research Center. His research interests include cloud data privacy and reliability, with special emphasis on oblivious data transfer protocols.

João Leitão received the master's degree in computer engineering from the Faculdade de Ciências, Universidade de Lisboa (FC/UL), in 2007 and the PhD degree in computer engineering from the Instituto Superior Técnico (IST/UL), in 2012. He is an assistant professor in the Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa (FCT/UNL), and an integrated researcher with the NOVA LINCS Laboratory. His research interests focus on multiple aspects of distributed systems, including scalability, availability, fault-tolerance, and security. He is a member of the IEEE.

Henrique Domingos received the PhD degree in computer science from NOVA Lisbon University (UNL), in 2000. He was an assistant professor with FCT/UNL, from 1994 to 2004, and has been a tenured assistant professor with FCT/UNL, since 2004. He is also a research member of the NOVA LINCS Research Center. His current research focuses on cloud security and privacy, mobile computing usability, security and privacy, dependable distributed computing, and pervasive distributed systems security. He is a member of the IEEE.