Multimodal Indexable Encryption for Mobile Cloud-based Applications (Extended Version)

Bernardo Ferreira, João Leitão, Henrique Domingos
DI, FCT, Universidade NOVA de Lisboa & NOVA LINCS
{bf, jc.leitao, hj}@fct.unl.pt

Abstract

In this paper we propose MIE, a Multimodal Indexable Encryption framework that for the first time allows mobile applications to securely outsource the storage and search of their multimodal data (i.e. data containing multiple media formats) to public clouds with privacy guarantees. MIE is designed as a distributed framework architecture, leveraging on shared cloud repositories that can be accessed simultaneously by multiple users. At its core MIE relies on Distance Preserving Encodings (DPE), a novel family of encoding algorithms with cryptographic properties that we also propose. By applying DPE to multimodal data features, MIE enables high-cost clustering and indexing operations to be handled by cloud servers in a privacy-preserving way. Experiments show that MIE achieves better performance and scalability when compared with the state of art, with measurable impact on mobile resources and battery life.

I. Introduction

Mobile devices currently permeate everyday life, surpassing the sales of PCs and Laptops by six times [44] and being responsible for more than 70% of multimedia consumption on the Internet [15]. The advent of mobile devices and tablets has changed the way users produce and manipulate data. On the one hand, users now produce larger quantities of multimodal data (i.e. data containing various media formats such as photos, audio, and text) through their mobile devices [22]. On the other hand, data access and sharing is expected to be ubiquitous [14].

Due to resource limitations (computational power, battery life, and storage capacity) and increasingly larger collections of data produced and accessed by users[1], mobile devices have been a key driving factor for cloud computing solutions and the outsourcing of both data storage and processing [44]. This trend is also known as Mobile Cloud Computing [19]. In such solutions, the cloud effectively operates as a natural extension to the limited storage and computational resources of mobile devices. Furthermore, given such large datasets, being able to efficiently search and retrieve relevant subsets of their data becomes of increased importance for users.

However outsourcing to the cloud inherently leads to dependability and privacy challenges, especially when data and computations are sensitive or of critical nature. This is a natural observation as outsourcing data and computations also entails outsourcing control over them [13]. Recent news have proven that users' privacy is not protected when using cloud services [54]. Governments impose increasing pressure on technological companies to disclose users' data and build insecure backdoors [16], [26]. Malicious or simply careless cloud system administrators have been responsible for critical data disclosures [12], [21]. Finally, one also has to consider internet hackers exploiting software and hardware vulnerabilities in the cloud providers' infrastructure [40].

A common approach for dealing with these dependability issues is to use end-to-end encryption schemes, where users' devices are responsible for encrypting all data before sending it to the cloud [4], [42]. However these schemes restrict functionalities available to users, including efficient data sharing and searching operations through the cloud infrastructure. While data sharing can easily be achieved through key distribution [36], searching encrypted data is a non trivial challenge.

The research community has tried to address this challenge by proposing Searchable Symmetric Encryption (SSE) schemes [2], [11], [17], [27], [34], [35], [49], [55]. Originally designed for exact-match searching in text documents, SSE schemes allow querying encrypted data in sub-linear time, by having users index their data (i.e.

This technical report was partially published in: B. Ferreira, J. Leitão, H. Domingos. Multimodal Indexable Encryption for Mobile Cloud-based Applications. In 47th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'17). June 2017.

[1]In cloud-backed multimedia storage apps like iCloud Photos: http://www.apple.com/icloud/photos; and Google Photos: https://www.google.com/photos.

build a compact dictionary of the data; e.g. with the unique keywords of each text document) and upload both encrypted index and data to the cloud. However extending SSE to richer queries [2], [9], [57] and other media domains [20], [41] has proven challenging. On one hand, indexing computations of multimodal data and rich media types (including images, audio, and video) are too expensive, especially for mobile client devices and considering that training tasks (i.e. clustering and machine learning algorithms) also have to be performed before data can be efficiently indexed [18]. On the other hand, the few existing approaches [2], [9], [41], [57] are still limited to static collections (i.e. data can't be added, updated, or removed dynamically after deployment and initial load of a repository).

On a side note, searching encrypted data in sub-linear time is only possible by revealing some information patterns to adversaries with each query, including if the query has been performed before and which data objects, although encrypted, were returned by it (search and access patterns [17], respectively). This note is important, as it will be leveraged in the core design of our solution as explained next.

In this paper we propose a novel framework to tackle the practical challenges of supporting mobile applications dynamically storing, sharing, and searching multimodal data in public cloud infrastructures while preserving privacy. We call our proposal MIE - Multimodal Indexable Encryption. MIE leverages from two insights: on the one hand, the leakage of search and access patterns has been proven unavoidable in order to search encrypted data in sub-linear time [48]; on the other hand, in practical deployments where many queries are submitted concurrently by multiple users, these patterns are eventually revealed for the entire index space (i.e. for all possible queries). Leveraging these insights, we contrive MIE to reveal information patterns with each add/update operation, instead of each query. This will allow users to dynamically update and search multimodal repositories while securely outsourcing indexing and training computations to the cloud, which we later show to be the heaviest and more unsuitable computations for mobile applications.

To support MIE's operations we propose a novel family of encoding algorithms with cryptographic properties, called DPE - Distance Preserving Encodings. DPE schemes securely encode data while preserving a controllable distance function between plaintexts. By extracting feature-vectors from multimodal data and encoding them with DPE, users are able to outsource training and indexing computations to the cloud in a privacy-preserving way. We formally define DPE and present two efficient implementations: one for dense media types (e.g. images, audio, and video) and another for sparse media (e.g. text). DPE is of particular interest on itself and can be easily integrated in other secure protocols.

We implemented both an Android and Desktop applications on top of our MIE framework for those platforms. These applications, designed to support the storage and search of multimodal data containing text and image formats, are used to experimentally validate MIE's performance, scalability, and battery consumption in mobile devices. Since (as far as we know) MIE is the first endeavor in multimodal encrypted search, we also implemented a recent SSE scheme from the literature [11], extended it to support multimodal searching, and experimentally compared its performance with MIE. Our implementations are open source and publicly available at: https://github.com/bernymac/MIE.

In summary, this paper makes the following contributions:

- We propose an alternative design to dynamically updating and searching encrypted multimodal data that allows the secure outsourcing of training and indexing computations. We call our proposal MIE - Multimodal Indexable Encryption (§V).
- To support MIE's operations we propose a new family of cryptographic primitives that preserve a controllable distance function between plaintexts. We call our proposal DPE - Distance Preserving Encodings (§IV);
- We formally prove the security properties of our proposals under the standard security model, i.e. without resorting to heuristic models like Random Oracles, which may not have secure implementations in practice [24];
- We implement MIE, both for Desktop and Mobile (Android) devices (§VI), and a multimodal SSE scheme based on the recent literature [11], evaluating and comparing both in terms of performance and scalability across different operations (§VII). Real-world datasets and public commercial clouds (Amazon EC2) are used in these experiments.

Scheme	**Search Time**	**Update Time**	**Client Storage**	**Revocation Size**	**Query Type**	**Search Leakage**	**Update Leakage**
Kamara'12 [35]	$O(m/n)$	$O(m/n)$	$O(1)$	–	Text Match	$ID(w), ID(d)$	$ID(w)$
Kamara'13 [34]	$O(log\|F\|.m/n)$	$O(log\|F\|.n)$	$O(1)$	–	Text Match	$ID(w), ID(d)$	–
Naveed'14 [49]	$O(m/n)$	$O(m/n)$	$O(1)$	–	Text Match	$ID(w), ID(d)$	$ID(w)$
Hahn'14 [27]	$O(m/n)$ (amort.)	$O(m/n)$	$O(n)$	–	Text Match	$ID(w), ID(d)$	–
Cash'14 [11]	$O(m/n)$	$O(m/n)$	$O(n)$	$O(m)$	Text Match	$ID(w), ID(d)$	–
Stefanov'14 [55]	$O(m/n + \log m)$	$O(\log^2 N)$	$O(m)$	–	Text Match	$ID(w), ID(d)$	– (forward private)
Cao'14 [9]	$O(n^2)$	$O(n^2)$	$O(1)$	–	Text Ranked	$ID(w), ID(d)$	$ID(w), freq(w)$
Ferreira'15 [20]	$O(m/n)$	$O(m/n)$	$O(1)$	–	Image Ranked	$ID(w), ID(d)$	$ID(w), freq(w)$
MSSE	$O(m/n)$	$O(m/n)$	$O(n)$	$O(m)$	Multimodal	$ID(w), ID(d), freq(w)$	–
Hom-MSSE	$O(m/n)$	$O(m/n)$	$O(n)$	$O(m)$	Multimodal	$ID(w), ID(d)$	–
MIE	$O(m/n)$	$O(m/n)$	$O(1)$	–	Multimodal	$ID(w), ID(d)$	$ID(w), freq(w)$

TABLE I: Overview of average complexities for the literature on SSE, our work (MIE), and two multimodal SSE schemes (MSSE and Hom-MSSE) designed for baseline experimental comparison by extending the recent literature on SSE [11] (more details in the Evaluation Section §VII). The reader should note that although MIE displays the same search and update time complexities as its two multimodal alternatives, it resorts to more efficient cryptographic primitives, resulting in faster operation time in practice (as will be revealed in §VII). Table Legend: n is the number of unique keywords (or similar concept in other medias, e.g. a keypoint in an image), m is the total number of index entries (keywords or other), $|F|$ is the number of data-objects, $ID(w)$ is the deterministic id of a keyword being queried or added to a data-object, $ID(d)$ represents the ids of the data-objects returned by a query (i.e. that contain the queried keyword), and $freq(w)$ is the frequency of a keyword in data-objects being updated or returned by a query.

II. Related Work

Searching encrypted data is currently a hot research topic, with the increasing popularity of storage and computation cloud services and the security issues they bring. In the last decades, relevant advances have been achieved in powerful cryptographic mechanisms that allow generic computations on encrypted data, including Fully Homomorphic Encryption [23] and Oblivious RAM [56]. However such techniques still remain too expensive to be practical: e.g. computing an AES decryption circuit through fully homomorphic encryption is at least 10^9 times slower [23]; while developing an SSE scheme on top of Oblivious-RAM, protecting access patterns, increases query data-transfer overheads by at least 128 times compared to *conventional* SSE, and by at least 1.75 times compared to downloading the entire database with each query [48].

Searchable Symmetric Encryption (SSE) [17] strives for a practical balance between efficiency and security. Originally designed for exact-match search over static collections of text documents of a single user, SSE schemes are able to achieve sub-linear search performance by initially revealing no information regarding the encrypted data and then gradually revealing some information patterns with each search operation [17]. These leaked information patterns include: search patterns, i.e. has this query been issued before, which is leaked by a deterministic hash of the query; and access patterns, i.e. which data objects are returned by each query, which is leaked by the deterministic identifiers of the objects. Extending SSE for dynamic collections, where documents can be added, updated, and deleted at runtime, initially lead to the further disclosure of update patterns [35], [49] (i.e. if new documents share contents with previously stored documents, leaked by deterministic hashes of the new document's keywords).

Recent dynamic SSE schemes were able to overcome the update leakage issue by increasing operational overhead [34], [55] and/or requiring client storage that grows linearly with the number of unique keywords [6], [11], [27]. Recent works also introduced the concept of forward privacy [6], [55], which states that updates should leak no information even when combined with old query tokens. However in practical scenarios with many queries being submitted by multiple users simultaneously, such guarantees can not hold for long periods.

With the exception of [49], dynamic SSE schemes described so far depend on heuristic models, like the Random Oracle model, which may not have secure implementations in practice and that have been highly criticized in recent years [24]. Making them secure under standard security assumptions requires further client processing and largely increased communication overhead [11], [55], turning these solutions unpractical.

Supporting richer query expressiveness in SSE has not been easy to achieve. The first SSE-based schemes for ranked retrieval were either based on insecure cryptographic primitives [57], or required heavy client processing and search time linear with the index size [9]. These SSE schemes also further revealed frequency patterns, i.e. how many times each queried keyword appears in retrieved documents. Hiding this information has only been possible by assuming the existence of a user-controlled cryptographic module in the cloud server, which would perform multi-party computation with the server, besides encrypting the index with an additively-homomorphic encryption scheme [2]. Furthermore these ranked SSE schemes have so far been restricted to static document collections, as they depend on pre-computed and immutable ranking scores that would need to be refreshed and re-encrypted with each document addition, update, or removal.

SSE schemes are usually designed for single writer and single reader/searcher scenarios [2], [11], [27], [55]. Some SSE schemes extend this model to support multiple searchers, however it must be a single writer to generate searching tokens for all other users [9], [57]. Searchable encryption in the public key setting (also know as PEKS) [5] allows the opposite: multiple writers can use a public key to write data, but only a single reader can use the respective private key to search that data. In [53], a multi-key searchable encryption scheme supporting multiple writers and searchers was proposed. However this approach is based on bilinear maps on elliptic curves (which are an order of magnitude slower than conventional symmetric cryptography), has linear-time search performance, and although it supports multiple users it does not address user access control and revocation issues.

Besides text documents, SSE-based schemes have also been designed for other media domains such as images [20], [41]. However, the overhead imposed on client devices in text ranked searching is even more noticeable in the context of images, as machine learning tasks (also known as training) are usually required before dense media types (i.e. images, audio, and video) can be indexed [18]. Furthermore, both training and indexing of dense media data are computationally intensive operations. Some of these performance issues were addressed in [20], however this approach was limited to color features in the image domain. Hence, and to the best of our knowledge, this paper presents the first endeavor in supporting encrypted storage and search of multiple media formats simultaneously (i.e. multimodal data) in a practical way, while supporting resource-restricted mobile devices. Table I provides a summary review of the recent literature on SSE and comparison with our approach across multiple distinguishing factors.

III. Technical Overview

In this section we present an overview of MIE and the system and adversary models that we consider. We start with some notations and fundamental concepts: we call **multimodal data-object**, or simply **object**, to an aggregation of data with multiple media formats or modalities (i.e. an object containing text, image, audio, and/or video; examples are annotated images, wikipedia pages, and personal health records [47]); a **repository** is a collection of multimodal data-objects; **features** are characterizations of objects in some particular media type (e.g. the text modality of an object can be characterized by its most relevant textual keywords [43], while the image modality by a set of visual points of interest [3]); **feature-vectors** are vectorial representations of features, describing an object across its multiple modalities. Feature-vectors are essential components to enable efficient search in repositories containing large collections of multimodal objects.

Multimodal searching uses a multimodal object as query for searching in a multimodal repository. Search results are usually obtained for each media format in separate and aggregated through a multimodal merging function [47].

Indexing takes a collection of data-objects and constructs a dictionary describing them under some features (e.g. which keywords appear in each text document) [43]. This dictionary, called index, forms a compressed representation of the data and allows searching in sub-linear time (e.g. searching for a keyword becomes equivalent to one dictionary access, instead of linearly scanning all text documents).

Training tasks are machine learning operations, such as the k-means clustering algorithm [28], used to find homogeneous groups of objects in dense, high-dimensional data [1]. These groups are used to build more compact representations of high-dimensional data-objects. Example: an object-recognition algorithm [3] finds multiple points of interest in an image. Training a collection of such keypoints from different images yields a group of Distinctive Keypoints [50]. Representing all keypoints of an image in a compact way can then be achieved by finding the most similar Distinctive Keypoint of each and building an histogram with their frequencies.

A. System Model and Architecture

In this paper we focus on the challenges inherent to building practical, secure, and searchable cloud-backed multimodal data repositories especially tailored for mobile devices. We consider a system with multiple readers and

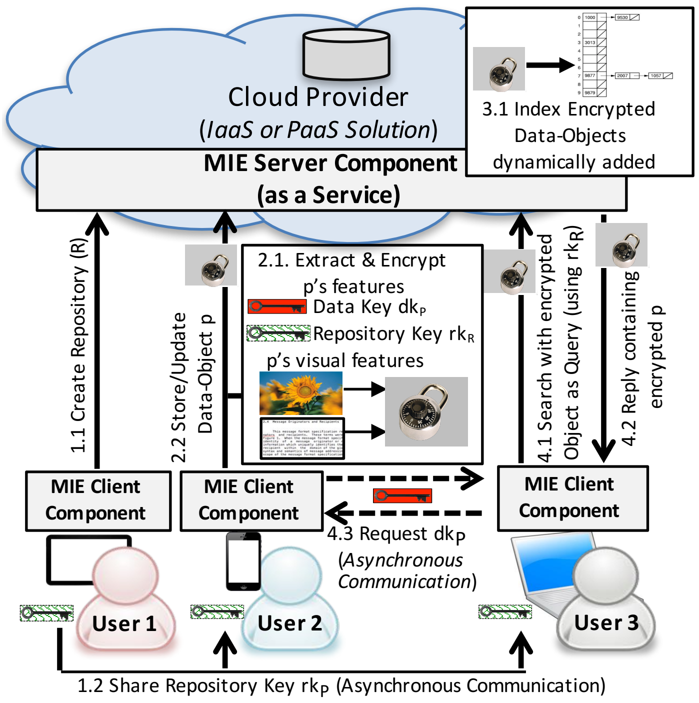
Cloud Provider
(IaaS or PaaS Solution)
MIE Server Component
(as a Service)
3.1 Index Encrypted Data-Objects dynamically added
1.1 Create Repository (R)
2.2 Store/Update Data-Object p
2.1. Extract & Encrypt p's features
Data Key dkP
Repository Key rkR
p's visual features
4.1 Search with encrypted Object as Query (using rkR)
4.2 Reply containing encrypted p
MIE Client Component
MIE Client Component
MIE Client Component
4.3 Request dkP
(Asynchronous Communication)
User 1
User 2
User 3
1.2 Share Repository Key rkP (Asynchronous Communication)

Fig. 1: System model with example interactions between users and the cloud infrastructure, considering image and text media domains.

writers (the **Users**) who store, share, and search data through multiple independent repositories hosted by a **Cloud Server** (or simply **Server**). We assume all data is outsourced to these repositories in the form of data-objects that may contain multiple media formats. A repository is created by one user, and can be used by multiple (authorized) users besides herself. Authorized users can upload their own multimodal data-objects, search through the use of multimodal queries, and retrieve/read objects stored in a repository. Figure 1 provides a high level overview of the described system model.

Upon the creation of a repository, we delegate on the user that created it the task of generating and sharing a **Repository Key** with his trusted users. This cryptographic key allows users to search and add/update objects in that particular repository. More concretely, it is used in the indexing of new/updated objects, as well as in the generation of searching trapdoors. In addition to repository keys we also employ **Data Keys**, used to encrypt the data-objects themselves (using a semantically secure block-cipher, such as AES in CTR mode [36]). Data keys offer users a fine-grained access control over who accesses the full contents of their data-objects; nonetheless they should be seen as an optional functionality, and they can be discarded from the system design in use cases where fine-grained access-control is not required (for instance, by encrypting all data-objects with a shared master key).

When adding (or updating) data-objects in a repository, a user will first process them and extract their feature-vectors in their different modalities. These feature vectors are then encrypted with a Distance Preserving Encoding (DPE, detailed in §IV) and uploaded to the cloud server for training and indexing, alongside the encrypted data-object.

Authorized users with a repository key can also issue multimodal queries, using data-objects with any number of (supported) modalities as queries. To this end they process their query objects the same way as for new data-objects, extracting and encrypting their feature-vectors with DPE and sending them to the cloud server. After receiving an encrypted multimodal query, the cloud server returns the ranked top k matches for it, where k is a configurable parameter. Each of these k matches contains a pair of encrypted data-object and metadata, the later containing deterministic identifiers for the object and its owner (unless data keys have been removed from the system's design, to fully access its contents the querying user will still need to ask the object's owner for its data key).

All remote communications between users and the server should be encrypted and authenticated through secure communication protocols (TLS/SSL [36]). Key sharing interactions can be done asynchronously and out-of-band by resorting to broadcast encryption [17] or a key-sharing protocol based on public-key authentication [36]. User authentication and access control can be achieved through different mechanisms found in the literature, such as sharing authorization tokens between trusted users [17]. This discussion, however, is orthogonal to the main focus of the paper as these mechanisms can easily be integrated into our solution.

B. Adversary Model

In this work we aim at protecting the privacy of users' data and queries. Similar to previous approaches from the literature [2], [11], [17], [27], [55], we consider as main adversary the cloud administrator. This adversary acts in a *honest-but-curious* way, operating the cloud's infrastructure and possibly eavesdropping on users' data, but nonetheless is expected to fulfill its contract agreements and correctly perform operations when asked. We assume that the cloud administrator has access to all data stored on disk or in RAM on any device physically connected to the server, and passing through the network from or to the cloud. Throughout this paper we prove the security of our proposals against such an adversary. We also assume the cloud provider to protect its infrastructure from Internet hackers, as it is in its best interest to protect its infrastructure, its clients, and its reputation.

A stronger adversary that should also be considered is a *malicious user*, i.e. a user of the system who deviates from his expected behavior. Malicious users are an open problem for any multi-user application, as they may be given access to multiple repository and data keys before being discovered, and can more easily eavesdrop on other users' data. In this work we can mitigate the effect of this adversary by providing user access control enforcement and revocation mechanisms, complemented with public-key authentication and periodic key refreshment. Furthermore, we do not consider integrity or availability threats, as they can be handled by different mechanisms orthogonal to the contributions of this paper [38]. Finally, we assume that the higher-level applications using our work can control the amount of background information they reveal, as this may be sensitive and can be leveraged by adversaries for breaking security [10]. In §V-A we discuss possible attack vectors on our work and how to mitigate them.

C. Application Use Case

To provide examples of applications that could benefit from our work, we now briefly discuss a use case and explain the mapping of concrete entities between it and the previously introduced system model.

Personal Health Records. The number of mobile applications leveraging sensorial data for personal health tracking is growing by a large faction [37]. Moreover, major cloud operators are now offering centralized storage and computation services for such critical health data, under the form of Personal Health Records (PHR) [45]. PHR may contain information regarding users' health conditions under multiple media formats, extracted from their mobile devices' sensors, as well as from medical consultations and healthcare exams performed by healthcare professionals at different medical centers. The availability of this information not only ensures a better healthcare service for patients, but also offers a high potential for the exchange of medical information among different healthcare practitioners and institutes, for medical research purposes and to assist in the treatment of patients with similar conditions.

In this scenario, patients or medical doctors on their behalf (i.e. the Users), outsource their PHR directly from their mobile devices or IoT healthcare devices to a cloud-based backend (i.e. the Cloud Server). Because PHRs belong to the patients, these records can be protected by Data Keys only known to them (and possibly shared with trusted doctors with the patients explicit permission). On the other hand, Repository Keys can be shared between medical doctors and centers, organized in alliance based or medical-specialty based repositories between cooperating professionals. Doctors can then search on these repositories, requesting data keys to PHRs that might be of their interest directly to the respective patients.

IV. Distance-Preserving Encoding

In this section we propose a new family of encoding algorithms, called Distance Preserving Encodings (DPE). Our proposal of DPE comes from the generalization and formal analysis of the main principles behind different existing mechanisms for privacy-preserving nearest-neighbor and similarity computations [7], [20], [33], [41]. DPE is the basis of this work and our new approach to searching multimodal encrypted data. Nonetheless its abstract

Algorithm 1 The "ideal" $\mathcal{F}_{\text{DPE}}$ functionality: all information leaked to the server is specified here.

$\mathcal{F}_{\text{DPE}}$ is specified as a trusted third-party, which mediates inputs and outputs between the client and the server, modeling the information leaked to the later. $\mathcal{F}_{\text{DPE}}$ accepts one command, $\mathcal{F}_{\text{DPE}}$.Distance which is identical to DPE.Distance, the only algorithm in DPE where interaction between the server and the client occurs.

- On receiving command $\mathcal{F}_{\text{DPE}}$.Distance(e_1, e_2) from the client:
  - $\mathcal{F}_{\text{DPE}}$ returns $D_{e_1}^{e_2} = d_e(e_1, e_2) = d_p(p_1, p_2)$, i f $d_p(p_1, p_2) < t$. Otherwise, it returns $D_{e_1}^{e_2} = d_e(e_1, e_2) = t$.
  - **Distance Leakage:** $\mathcal{F}_{\text{DPE}}$ also leaks to the server: ID_{e_1}, ID_{e_2} (deterministic identifiers of e_1 and e_2), and $D_{e_1}^{e_2}$.

concept may have interesting applications in other contexts, and as such we present it as an independent building block that doesn't explicitly depend on external aspects of the system using it. We start this section by formally defining DPE. Then we present two efficient implementations of DPE, one applied to dense media types (e.g. images), and another for sparse media (e.g. text). Both implementations are used in §VI to implement an efficient Multimodal Indexable Encryption prototype.

A. DPE Definition

Informally, Distance Preserving Encodings (DPE) are a family of encoding schemes that preserve a controllable distance function between plaintexts, by means of their respective encodings. We say the distance function is controllable, meaning that on instantiation of a DPE scheme a security threshold parameter should be defined, which will allow controlling the amount of information leaked by encodings. More specifically, DPE encodings should only preserve distances between plaintexts up to the value of the threshold. For greater distances, nothing should be leaked by DPE encodings. This threshold allows defining an upper bound on information leakage and security, as it will limit the adversarial ability to perform statistical attacks and establish a distance relation between different plaintexts in the application domain. More formally:

Definition 1 (Distance Preserving Encoding)**.** *A Distance Preserving Encoding (DPE) scheme is a collection of three polynomial-time algorithms (*KEYGEN, ENCODE, DISTANCE*) run by a client and a server, such that:*

• $K, t \leftarrow$ KEYGEN(1^k)*: is a probabilistic key generation algorithm run by the client to setup the scheme. It takes the security parameter k and returns a secret key K and a distance threshold t, both function of and polynomially bounded by k.*

• $e \leftarrow$ ENCODE(K, p)*: is a deterministic algorithm run by the client to encode plaintext p with key K, with p polynomially bounded by k. It outputs an encoding e.*

• $D \leftarrow$ DISTANCE(e_1, e_2)*: is a deterministic algorithm run by the server that takes as input two encodings e_1 and e_2. For plaintext distance function $[0, 1] \leftarrow d_p(\cdot, \cdot)$ and encoded distance function $[0, 1] \leftarrow d_e(\cdot, \cdot)$ (possibly $d_p = d_e$) with inputs polynomially bounded by k, it outputs $D = d_e(e_1, e_2) = d_p(p_1, p_2)$, if $d_p(p_1, p_2) < t$. Otherwise it outputs $D = t$.*

Given the definition of DPE, we formalize in Algorithm 1 an ideal functionality $\mathcal{F}_{\text{DPE}}$, which represents the protocol interactions between the client and the server and that captures all information leaked by these. In $\mathcal{F}_{\text{DPE}}$ we consider as adversary the honest-but-curious cloud provider (as defined in §III-B), which can only attack the server passively. We remark that the information leaked is limited (due to threshold t) and easy to specify. Nonetheless, an adversary can still leverage this leakage to learn some statistics about the data being encoded, and it's up to the applications using DPE to ensure those statistics are not sensitive. In the following we present two implementations of DPE and formally prove that they are secure realizations of $\mathcal{F}_{\text{DPE}}$.

B. A DPE Implementation for Dense Data

Rich media types, including images, audio, and video are characterized by their high-dimensionality and high-density [1]. High dimensionality means that multiple coordinates (the dimensions) are required to describe a point (i.e. a feature-vector) in these media types. As an example, consider the SURF [3] feature extraction algorithm for images, which computes feature-vectors of 64 dimensions. High density means that in all dimensions necessary

Algorithm 2 Dense-DPE Implementation

```
1: function KEYGEN($N, M, \Delta$)
2:     $A \leftarrow \mathcal{G}(M \times N)$                      ▷ Generate A
3:     $w \leftarrow \mathcal{G}^{[0,\Delta]}(M)$                 ▷ Generate w, limited by 0 and $\Delta$
4:     $t \leftarrow Func(\Delta)$                                ▷ $t$ is controlled by $\Delta$
5:     return $K = \{A, w\},\ t$
6: function ENCODE($p, K = \{A, w\}$)
7:     $e \leftarrow Q(\Delta^{-1}.(A.p + w))$                    ▷ $Q(.)$ is fixed
8:     return $e$
9: function DISTANCE($e_1, e_2$)
10:    $D \leftarrow NormHamm(e_1, e_2)$                          ▷ Equal to $Eucl(p_1, p_2)$ if $D < t$
11:    return $D$
```

to describe a feature-vector, most will have a rational value different from zero (even if close, e.g. 0.01). This is defined in clear contrast to sparse media types such as text, where a document only has a finite subset of keywords from the whole english vocabulary [43] (or any other language) and non-existing keywords can simply be omitted from a feature-vector characterization of the document (e.g. a keyword-frequency histogram).

A DPE implementation for dense data should be able to efficiently encode high-dimensional feature-vectors, while preserving some parametrizable distance function between them. To achieve this goal we extend the encoding proposed by Boufounos et al. [7] for privacy-preserving nearest neighbors. This encoding cryptographically protects feature vectors by transforming them through universal scalar quantization [7]. Moreover, it preserves Euclidean [43] distances between plaintext feature-vectors, through the normalized Hamming [43] distances between encodings, but only up to a tunable threshold t. For plaintext distances greater than t, the distance between encodings conveys no information and will tend to a constant value. More concretely, feature vectors are transformed through the following function:

$$e(x) = Q(\Delta^{-1}(Ax + w)) \tag{1}$$

where $x \in \mathbb{R}^N$ is a N-dimensional feature vector given as input, $A \in \mathbb{R}^{M \times N}$ is a random matrix with independent and identically distributed elements (M is a tunable parameter representing the output size and basically controls the noise introduced by the encoding), Δ is a tunable scaling factor operating element-wise which controls the distance threshold t, $w \in \mathbb{R}^M$ is an additive dither uniformly distributed in $[0, \Delta]$, and $Q(.)$ is a scalar quantizer with non-contiguous intervals such that scalar values in $[2v, 2v + 1)$ quantize to 1 and values in $[2v + 1, 2v + 2)$ quantize to 0, for any v. Finally, $\{A, w\}$ compose the secret key of this scheme.

The previous scheme suffers from a main applicability limitation: secret key $\{A, w\}$ has size proportional to the input and output sizes (N and M respectively). This approach leads to large key sizes and limits flexibility of deployment, as a change on input/output length (e.g. user changes the type of features used for indexing and searching) forces the generation and sharing of a new secret key with the appropriate size. To solve this issue, we introduce a Pseudo-Random Generator (PRG) $\mathcal{G}$ [36] in the key generation algorithm of the previous scheme, instantiated with some random bits of entropy as cryptographic seed. The random values in A and w will be generated through $\mathcal{G}$, and for a Probabilistic Polynomial-Time (PPT) bounded adversary these values are indistinguishable from true random values [36].

Algorithm 2 describes our implementation in detail, which we call Dense-DPE. Consistent with our definition for DPE, Dense-DPE only reveals a distance function between the feature-vectors of data-objects, and this function is limited by threshold t. Furthermore we can prove that:

Theorem 1. *Dense-DPE securely realizes functionality $\mathcal{F}_{\text{DPE}}$ against honest-but-curious PPT adversaries.*

Proof. The proof involves showing that a simulator $\mathcal{S}$, interacting with the client only through $\mathcal{F}_{\text{DPE}}$ (the ideal experiment), can simulate the view of the server in a real interaction with the client through an instance of Dense-

Algorithm 3 Sparse-DPE Implementation

```
 1: function KEYGEN($k$)
 2:     $K \leftarrow \mathcal{G}(k)$
 3:     $t \leftarrow 0$
 4:     return $K,\ t$
 5: function ENCODE($p, K$)
 6:     $e \leftarrow \mathcal{P}_K(p)$
 7:     return $e$
 8: function DISTANCE($e_1, e_2$)
 9:     if $e_1 == e_2$ then
10:         $D \leftarrow 0$
11:     else
12:         $D \leftarrow 1$
13:     return $D$
```

DPE (the real experiment), and that the two experiments are indistinguishable even when combined with the adaptively influenced inputs to the client (apart from a negligible probability [36]).

$\mathcal{S}$ starts by initializing a simulated data-objects collection $L' = \{ID_{e_i}, p'_i\}_{i=0}^{*}$ and a simulated distance map $M' = \{ID_{e_i},\ \{ID_{e_j},\ D'^{p_j}_{p_i}\}_{j=0}^{*}\}_{i=0}^{*}$, whose entries represent distinct data-objects and a list containing their closest objects and a simulated distance between them, respectively. Then, when $\mathcal{S}$ receives the Distance command from $\mathcal{F}_{\text{DPE}}$ with its *Distance Leakage*=$\{ID_{e_1}, ID_{e_2}, D_{e_1}^{e_2}\}$, it creates simulated data-objects p'_1 and p'_2 with simulated length N', fills them with uniformly random bits and stores them in $L'[ID_{e_1}]$ and $L'[ID_{e_2}]$. Then $\mathcal{S}$ checks if $D_{e_1}^{e_2} < t$. If that is the case, $\mathcal{S}$ knows that the distance between the plaintexts has been preserved and adds $\{ID_{e_2}, D_{e_1}^{e_2}\}$ to $M'[ID_{e_1}]$ and $\{ID_{e_1}, D_{e_1}^{e_2}\}$ to $M'[ID_{e_2}]$. Otherwise, $\mathcal{S}$ randomly chooses a simulated distance value D' such that $1 > D' \geq t$, and adds $\{ID_{e_2}, D'\}$ to $M'[ID_{e_1}]$ and $\{ID_{e_1}, D'\}$ to $M'[ID_{e_2}]$ instead.

Due to the properties of the encoding function used [7] and of the Pseudo-Random Generator $\mathcal{G}$ [36], p_1 and p_2 will be indistinguishable from their simulated counterparts p'_1 and p'_2 for PPT adversaries. The correctness of the implementation, in particular that only Euclidean distances between plaintexts up to threshold t will be preserved, is inherited from the the correctness of the encoding function, which is proven in [7]. Moreover, if $D_{e_1}^{e_2} \geq t$, it will also be indistinguishable from the simulated distance D', hence concluding the proof. $\square$

C. A DPE Implementation for Sparse Data

Since in sparse media types, such as text data, feature-vectors are much smaller compared with dense media types, more efficient algorithms can be used to index and search sparse media. More concretely, to index and search in sparse data, we only need to compare the different non-null values in its feature-vectors for equality[2] (e.g. the keywords of each text document). Translating this to the DPE definition, our DPE implementation for Sparse Data will have a similarity distance threshold of $t = 0$, meaning that it will only reveal if two keywords are equal, and nothing will be revealed even if they are only one character apart.

To achieve the above goals, we base our DPE implementation for Sparse Data on a Pseudo-Random Function (PRF) [36]. More concretely, given a feature-vector from a sparse data-object (i.e. a text document), we apply:

$$f(x) = \mathcal{P}_K(x) \tag{2}$$

where x is a single keyword and $\mathcal{P}$ is a PRF, instantiated with secret key K. In practice, $\mathcal{P}$ can be implemented as a keyed hash function. Algorithm 3 provides the full details of our implementation, which we call Sparse-DPE. Furthermore, we can prove that:

[2]Edit distance and cryptographic schemes such as [33] could be used to construct an alternative Sparse-DPE implementation with threshold distances greater than zero. However, exact string matching complemented with light client-side techniques such as stemming and spell-checking wields similar search precision in ranked text retrieval [43].

Scheme	P-FV	E-FV1 $d_p = 0$	E-FV2 $d_p = 0.3$	E-FV3 $d_p = 0.7$	E-FV4 $d_p = 1$
Dense-DPE ($t = 0.5$)	0.5557	0.0	0.3085	0.59375	0.5585
Sparse-DPE ($t = 0$)	1.0	0.0	1.0	1.0	1.0

TABLE II: Encoded (i.e. normalized Hamming) distances between DPE encodings and: their original plaintext feature-vector P-FV; and encoded feature-vectors E-FV1 through E-FV4, with varied plaintext (i.e. Euclidean) distances d_p between their plaintexts and the original P-FV. In Sparse-DPE, since $t = 0$, distances above t have a different yet constant value, in this case 1.

Theorem 2. *Sparse-DPE securely realizes functionality $\mathcal{F}_{\text{DPE}}$ against honest-but-curious PPT adversaries.*

Proof. The proof is similar to the one of Theorem 1. Simulator $\mathcal{S}$ starts by initializing a simulated data-objects collection $L' = \{ID_{e_i}, p'_i\}_{i=0}^{*}$ and a simulated distance map $M' = \{ID_{e_i}, \{ID_{e_j}, D'^{p_j}_{p_i}\}_{j=0}^{*}\}_{i=0}^{*}$. When it receives the Distance command with leakage $\{ID_{e_1}, ID_{e_2}, D^{e_2}_{e_1}\}$, it checks if $D^{e_2}_{e_1} = 0$. If that is the case, it creates a simulated data-object p' with uniformly random bit strings of simulated length N', and sets $L'[ID_{e_1}]$ and $L'[ID_{e_2}]$ to p'. Otherwise, it creates two distinct simulated data-objects p'_1 and p'_2 and a simulated distance D' such that $1 \geq D' > 0$, sets $L'[ID_{e_1}] = p'_1$, $L'[ID_{e_2}] = p'_2$, and adds $\{ID_{e_2}, D'\}$ to $M'[ID_{e_1}]$ (and vice-versa). From the properties of Pseudo-Random Functions (PRFs) [36], p_1 will be indistinguishable from p' and p'_1, and p_2 will be indistinguishable from p' and p'_2. Moreover, PRFs also guarantee that $D^{e_2}_{e_1}$ will only be zero if and only if $p_1 == p_2$, thus proving the security and correctness of the implementation. $\square$

Table II presents a summary evaluation of the entropy generated by DPE encodings, by analyzing distance functions between both encoded and plaintext feature-vectors.

V. Multimodal Indexable Encryption

In this section we describe in detail our Multimodal Indexable Encryption (MIE) proposal. The main insight behind MIE is that in practical scenarios where many queries are submitted by multiple users concurrently, the semantic security guarantees initially offered by SSE schemes will not hold for long, as the information patterns leaked with each query will eventually be revealed for the entire index space. However those initial guarantees are only possible by having users train and index their data before uploading it to the cloud, which are heavy operations especially for mobile devices. Leveraging this insight, in MIE we outsource training and indexing computations from user's devices to cloud servers. This is done in a privacy-preserving way by having users extract feature-vectors from the different media formats, encode them with DPE, and upload the encodings to the cloud for computation. The practical result of our approach, on one hand, is that instead of revealing information patterns when queries are performed, as in previous SSE schemes, we reveal them at data creation/update time (namely search, access, and frequency patterns). On the other hand, this approach allows us to effectively support mobile devices dynamically updating and searching multimodal repositories, with increased performance and scalability (see §VII for experimental results).

From a systems perspective, MIE is defined as a distributed framework with two main components: one running in the client device(s), which processes data-objects, extracts feature-vectors in their different modalities, and encrypts them; and another (untrusted) running in the cloud servers, which performs training tasks and indexes data-objects through their encoded features. More formally:

Definition 2 (Multimodal Indexable Encryption)**.** *A Multimodal Indexable Encryption framework is a collection of five polynomial time algorithms (*CreateRepository, Train, Update, Remove, Search*) executed collaboratively between a user and a server, such that:*

• $\mathbf{rk_R} \leftarrow \mathbf{CreateRepository}(\mathbf{ID_R}, \mathbf{1^{sp_R}}, \{\mathbf{ID_{m_i}}\}_{\mathbf{i=0}}^{\mathbf{n}})$*: is an operation started by the user to initialize a new repository identified by ID_R. It also takes as input a security parameter sp_R and the n modalities to be supported by R ($\{ID_{m_i}\}_{i=0}^{n}$). It creates a repository representation on the server side and outputs a repository key rk_R.*

Algorithm 4 The ideal functionality $\mathcal{F}_{\text{MIE}}$: all information leaked to the server is specified here.

- On receiving command $\mathcal{F}_{\text{MIE}}$.CreateRepository(ID_R) from the client:
    - $\mathcal{F}_{\text{MIE}}$ creates a new repository R and initializes the required data-structures.
    - **Setup Leakage:** $\mathcal{F}_{\text{MIE}}$ sends to the server the deterministic identifier ID_R.
- On receiving command $\mathcal{F}_{\text{MIE}}$.train($ID_R, \{ID_{m_i}, ip_{m_i}\}_{i=0}^{n}$):
    - $\mathcal{F}_{\text{MIE}}$ internally initializes R's indexing structures in its n modalities, and trains them (i.e. performs machine learning tasks) with the objects stored in R, if needed (as defined by the indexing parameters $\{ID_{m_i}, ip_{m_i}\}_{i=0}^{n}$). Then $\mathcal{F}_{\text{MIE}}$ indexes R's data-objects, storing the results in its indexing structures.
    - **Train Leakage:** $\mathcal{F}_{\text{MIE}}$ sends ID_R and $\{ID_{m_i}, ip_{m_i}\}_{i=0}^{n}$ to the cloud server.
- On receiving command $\mathcal{F}_{\text{MIE}}$.Update($ID_R, ID_p, p, \{ID_{m_i}, fvs_{m_i}^{p}\}_{i=0}^{n}$):
    - If p already exists in repository R, it is first removed through the $\mathcal{F}_{\text{MIE}}$.remove operation.
    - $\mathcal{F}_{\text{MIE}}$ internally stores p and $\{fvs_{m_i}^{p}\}_{i=0}^{n}$ in R.
    - If the TRAIN command has already been invoked, $\mathcal{F}_{\text{MIE}}$ indexes p through its feature vectors ($\{ID_{m_i}, fv_{m_i}^{p}\}_{i=0}^{n}$).
    - **Update Leakage:** $\mathcal{F}_{\text{MIE}}$ sends to the server ID_R, ID_p, $\{ID_{fv_j^{m_i}}, freq_p^{fv_j^{m_i}}\}_{j=0}^{|p|}\}_{i=0}^{n}$ (the ids of p's feature-vectors and their frequencies), and $\{\{\{fv_j^{m_i}, fv_k^{m_i}, d(fv_j^{m_i}, fv_k^{m_i})\}_{j=0}^{|p|}\}_{k=0}^{|fv^{m_i}|}\}_{i=0}^{n}$ (distances between the feature-vectors in p and all other feature-vectors already stored in the repository).
- On receiving command $\mathcal{F}_{\text{MIE}}$.remove(ID_R,ID_p):
    - $\mathcal{F}_{\text{MIE}}$ internally removes p from R, as well as its feature-vectors $\{fvs_{m_i}^{p}\}_{i=0}^{n}$ and any references to p in R's indexing structures.
    - **Removal Leakage:** $\mathcal{F}_{\text{MIE}}$ sends ID_R and ID_p to the server.
- On receiving command $\mathcal{F}_{\text{MIE}}$.search(ID_R, $\{ID_{m_i}$, $fvs_{m_i}^{q}\}_{i=0}^{n}$, k):
    - If the $\mathcal{F}_{\text{MIE}}$.TRAIN command hasn't been invoked yet for repository R, $\mathcal{F}_{\text{MIE}}$ performs a linear search through R's data-objects, comparing their feature-vectors with q's feature-vectors and returning the k most similar results according to all modalities.
    - Otherwise, $\mathcal{F}_{\text{MIE}}$ accesses R's indexing structures in the n modalities present in q, and returns to the user the k closest data-objects in the repository in sub-linear time.
    - **Search Leakage:** $\mathcal{F}_{\text{MIE}}$ sends to the server ID_R, k, ID_Q (a deterministic id of q generated by $\mathcal{F}_{\text{MIE}}$), $\{ID_{fv_j^{m_i}}, freq_q^{fv_j^{m_i}}\}_{j=0}^{|q_{m_i}|}\}_{i=0}^{n}$ (the ids of the feature-vectors in q and their frequencies), and $\{\{\{fv_j^{m_i}, fv_k^{m_i}, d(fv_j^{m_i}, fv_k^{m_i})\}_{j=0}^{|q|}\}_{k=0}^{|fv^{m_i}|}\}_{i=0}^{n}$ (distances between the feature-vectors in q and all other feature-vectors stored in R).

- $\mathbf{Train(ID_R, rk_R, \{ID_{m_i}, ip_{m_i}\}_{i=0}^{n})}$*: operation invoked by the user to initialize repository R's indexing structures, by performing machine learning tasks (i.e. automatic training procedures), and index its data-objects, if any. The user also inputs the repository key and the indexing algorithms to be used as indexing parameters (*$\{ID_{m_i}, ip_{m_i}\}_{i=0}^{n}$*, one for each modality; examples of indexing parameters are Inverted List Index and Single Pass In Memory Indexing [43], more details in §VI). This algorithm can be invoked multiple times with different indexing parameters. Note however, that training procedures are only required in dense media types (e.g. images, audio, and video). In a repository containing only sparse media types (e.g. text), this operation will only index existing objects, if any.*
- $\mathbf{Update(ID_R, ID_p, p, dk_p, rk_R, \{ID_{m_i}\}_{i=0}^{n})}$*: is the operation used to dynamically add or update a data-object p in repository R. In addition to p, it also takes as input *ID_R* and *ID_p* (deterministic identifiers of R and p, respectively), *dk_p* (data key to be used in the encryption of p), *rk_R* (repository key of R) and *$\{ID_{m_i}\}_{i=0}^{n}$* (the modalities represented in p). If the* TRAIN *algorithm has already been invoked in R, p is indexed in its modalities. Otherwise p's indexing is performed when the* TRAIN *algorithm is invoked for the first time.*
- $\mathbf{Remove(ID_R, ID_p)}$*: is an operation that allows a user to fully remove a data-object p from repository R*

and its indexing structures.

- $\{\mathbf{ID_{p_i}}, \mathbf{p_i}, \mathbf{score^q_{p_i}}\}^{\mathbf{k}}_{\mathbf{i=0}} \leftarrow$ **Search**($\mathbf{ID_R}$, $\mathbf{q}$, $\mathbf{rk_R}$, $\{\mathbf{ID_{m_i}}\}^{\mathbf{n}}_{\mathbf{i=0}}$, $\mathbf{k}$)*: is issued by a user to search in repository R with object q as query, returning the k most relevant data-objects in the repository. Also takes as input the repository key rk_R and the modalities represented in q ($\{ID_{m_i}\}^n_{i=0}$). If the* TRAIN *algorithm has been invoked previously for R, the server replies to the query in sub-linear time by accessing R's indexing structures. Otherwise it performs a linear search through R's objects.*

Given MIE's definition, Algorithm 4 presents an idealized functionality for MIE ($\mathcal{F}_{\mathrm{MIE}}$) and Algorithms 5 through 9 detail our MIE's implementation based on DPE (respectively, operations CreateRepository, Train, Update, Remove, and Search). The main difference between our MIE implementation and functionality $\mathcal{F}_{\mathrm{MIE}}$ is the use of DPE. Hence, the main argument in proving security lies in showing that by using DPE's algorithms, our MIE implementation doesn't leak anything further to the server beyond what is specified in $\mathcal{F}_{\mathrm{MIE}}$. Furthermore, we can prove that:

Theorem 3. *The DPE-based MIE implementation presented in Algorithms 5-9 securely realizes functionality $\mathcal{F}_{\mathrm{MIE}}$ against honest-but-curious PPT adversaries.*

Proof. This security proof is straightforward, since DPE is used as a blackbox component and our MIE implementation involves no other cryptographic protocol. All information leaked to the cloud server by DPE (i.e. distance leakage) is easily derived from the information leaked by $\mathcal{F}_{\mathrm{MIE}}$. As such, simulator $\mathcal{S}$ can simulate all the interactions in the protocol using the information it obtains from $\mathcal{F}_{\mathrm{MIE}}$. The details are straightforward and hence omitted. $\square$

A. Additional Security Considerations

Applications using MIE have provable security guarantees, equivalent to the ones of previous SSE schemes in practical deployments frequently queried [11], of the information leaked by each operation. However, the impact of this information leakage and to what extent it can be leveraged by adversaries in inference attacks is not yet fully understood. Recent advances have been achieved in this field, with passive [10], [30] and active [58] attacks being proposed, in the text domain, for both query and plaintext recovery. However the efficiency of these attacks depends on very strong assumptions. Passive attacks require almost complete document set knowledge, i.e. adversaries must know the contents of a large subset of all encrypted data. For instance, the best known attack [10] requires 95% document knowledge to achieve 58% query recovery rate. with 75% document knowledge, query recovery drops to values close to 0%. Active attacks can have very strong consequences, but require the adversarial ability of injecting maliciously crafted documents, which must still be encrypted by the client. This means that when deploying a SSE scheme (including MIE), users should control the source of their documents and protect their devices from external hacking.

Regarding other media domains and multimodal data, while keywords in the text domain have a straight semantical meaning, the same may not hold for similar concepts in richer media (including audio, images, and video). Attacks over these domains and their impact are still an open area of research and an interesting future research direction, nonetheless we argue that further background information (controllable by users) may be required for adversaries to achieve acceptable recovery rates in these medias.

VI. IMPLEMENTATION

One of the advantages of our approach lies in its flexibility of deployment and its capacity to integrate different algorithms for feature extraction (client side) and both training and indexing computations (server side). MIE is agnostic to the information retrieval techniques used on either side, and they can be used in the encrypted domain without any major modifications from their original plaintext algorithms. With this in mind, we implemented a prototype version of MIE to experimentally validate its design and compare it with the most relevant approaches from the literature. These experimental results are detailed in §VII, while for now we focus on our prototype description. The user-side component of MIE was developed as an Android Service, using a mixture of Java with Android's SDK and C++ with Android's Native Development Kit. The cloud server component was fully developed in C++.

Algorithm 5 Create New Repository

```
1: function USER($U$).CREATEREPOSITORY($ID_R$, $sp_R$)
2:     $rk1_R \leftarrow$ DENSE-DPE.Keygen($sp_{m_i}$)
3:     $rk2_R \leftarrow$ SPARSE-DPE.Keygen($sp_{m_i}$)
4:     CLOUD.CreateRepository($ID_R$)
5:     RepUsers.ShareKey($\{rk1_R, rk2_R\}$)
6:     return $\{rk1_R, rk2_R\}$
7: procedure CLOUD.CREATEREPOSITORY($ID_R$)
8:     $Rep[ID_R] \leftarrow$ InitializeRepository()
9:     $Fvs[ID_R] \leftarrow$ InitializeFeatureVectorsList()
```

Algorithm 6 Train Repository

```
1: procedure USER($U$).TRAIN($ID_R, \{rk1_R, rk2_R\}, ID_{m_i}, ip_{m_i}\}_{i=0}^{n}$)
2:     CLOUD.Train($ID_R, \{ID_{m_i}, ip_{m_i}\}_{i=0}^{n}$)
3: procedure CLOUD.TRAIN($ID_R$, $\{ID_{m_i}, ip_{m_i}\}_{i=0}^{n}$)
4:     for all $\{ID_{m_i}, ip_{m_i}\}_{i=0}^{n}$ do
5:         $Idx[ID_R][ID_{m_i}] \leftarrow$ InitializeIndex($ID_{m_i}, ip_{m_i}$)
6:         if DenseMediaType($ID_{m_i}$) then
7:             $CB_R^{m_i} \leftarrow$ TrainIndex($Idx[ID_R][ID_{m_i}], ip_{m_i}, Fvs[ID_R]$)
8:         IndexData($Idx[ID_R][ID_{m_i}], Fvs[ID_R]$)
```

In order to showcase its multimodality, we implemented our prototype supporting text and image data. Text feature extraction on the user's side is performed through standard keyword stemming, stop-words removal, and histogram extraction [43], followed by Sparse-DPE encoding. Regarding image feature extraction, since our Dense-DPE implementation currently preserves Euclidean distances between plaintext feature-vectors, it is more suitable for floating-point image descriptors. As such, we use the *SURF* descriptor extraction algorithm [3] and *Dense Pyramid* feature detection [39] for our prototype implementation. Dense-DPE was instantiated with threshold $t = 0.5$ and output size equal to the input size (64 dimensions for *SURF* feature-vectors). As cryptographic algorithms' implementations, we use HMAC-SHA1 as implementation of Pseudo-Random Functions (PRFs), AES in CTR mode for data-objects encryption, and an AES-based Pseudo-Random Number Generator (PRNG) for random number generation. OpenSSL 1.0.2 [51] and OpenCV 2.4.10 [31] were compiled for Android integration and support MIE's user-side cryptographic and image retrieval computations, respectively (all remaining computations, including text feature-extraction, were implemented by us).

On the server side we use an index per modality, for each repository (as previously discussed in MIE's design). Both for text and image data, the inverted index [43] approach is used, where each index key represents a distinct keyword and index values compose a list of all object identifiers containing the keyword. Since this type of index was originally designed for text data, we use the Bag-Of-Visual-Words (BOVW) model as an intermediary step to represent image features as visual words [50]. In this model, feature-vectors extracted from a repository's images are clustered in a machine-learning step (MIE's training operation), through a clustering algorithm such as k-means [50]. This training step selects a number of representative feature-vectors (1.000 in our experiments) which are called visual words. After this step, when adding/updating or searching, the different feature-vectors of the input image can be matched with the selected visual words, and the most similar ones are used henceforth to represent each feature-vector. This way, the frequency of visual words in an image become similar to the frequency of keywords in text documents. Each visual word is given an index key, and a tree-like structure is built over all visual words, through hierarchical k-means [50], in order to improve visual word comparison performance (we use a visual-words tree of height 3 and width 10).

To further improve scalability, if an index (of any modality) grows too large to fit in the cloud server's main

Algorithm 7 Add/Update Object in Repository

```
1: procedure USER($U$).UPDATE($ID_R$, $ID_p$, $p$, $dk_p$, $\{rk1_R, rk2_R\}$, $\{ID_{m_i}\}_{i=0}^n$)
2:     for all $\{ID_{m_i}\}_{i=0}^n$ do
3:         $fvs_{m_i}^p \leftarrow$ ExtractFeatureVectors($p, ID_{m_i}$)
4:         if Dense-Media($ID_{m_i}$) then
5:             $efvs_{m_i}^p \leftarrow$ DENSE-DPE.Encode($fvs_{m_i}^p, rk1_R$)
6:         else
7:             $efvs_{m_i}^p \leftarrow$ SPARSE-DPE.Encode($fvs_{m_i}^p, rk2_R$)
8:     $e \leftarrow Enc(dk_p, p)$
9:     CLOUD.Update($ID_R, ID_p, e, \{ID_{m_i}, efvs_{m_i}^p\}_{i=0}^n$)
10: procedure CLOUD.UPDATE($ID_R, ID_p, e, \{ID_{m_i}, efvs_{m_i}^p\}_{i=0}^n$)
11:     CLOUD.Remove($ID_R, ID_p$)
12:     $Rep[ID_R][ID_p] \leftarrow e$
13:     $Fvs[ID_R][ID_p] \leftarrow \{efvs_{m_i}^p\}_{i=0}^n$
14:     if IsTrained($ID_R$) then
15:         for all $\{ID_{m_i}\}_{i=0}^n$ do
16:             for all $fv \in efvs_{m_i}^p$ do
17:                 if $Idx[ID_R][ID_{m_i}][fv][ID_p] == \{\}$ then $Idx[ID_R][ID_{m_i}][fv][ID_p] \leftarrow 0$
18:                 $Idx[ID_R][ID_{m_i}][fv][ID_p] + +$
```

Algorithm 8 Remove Object from Repository

```
1: procedure USER($U$).REMOVE($ID_R, ID_p$)
2:     CLOUD.Remove($ID_R, ID_p$)
3: procedure CLOUD.REMOVE($ID_R, ID_p$)
4:     if $Rep[ID_R][ID_p]! = \{\}$ then
5:         $Rep[ID_R][ID_p] \leftarrow \{\}$; $Fvs[ID_R][ID_p] \leftarrow \{\}$
6:         if IsTrained($ID_R$) then
7:             for all $\{ID_{m_i}\}_{i=0}^n$ do
8:                 for all $fv \in Idx[ID_R][ID_{m_i}]$ do
9:                     $Idx[ID_R][ID_{m_i}][fv]$.Remove($ID_p$)
```

memory, champion posting lists [43] are used to ensure that only the top ranked data-objects for each index entry are kept in memory, while the full index is stored in disk and periodically merged with updated/newly added index entries. This technique improves scalability without impacting retrieval precision. Again we remark that due to the properties of MIE and DPE, only small modifications are required for these techniques to work in the encrypted domain (such as applying k-means over normalized Hamming distances due to Dense-DPE properties, instead of Euclidean distances as in its original design).

To rank search results, the TF-IDF [43] weighting function is used both for images and text. Nonetheless more complex functions could be used without loss of generality (e.g. BM25 [43]). Finally, to enable multimodal querying (simultaneous search with multiple media query formats) we use the logarithmic inverse square rank fusion approach [46]. This approach allows us to separately search in the different modalities and then merge all obtained results into the final set of multimodal results, according to the rankings in each modality.

Training and k-means computations in the cloud side are done using OpenCV 2.4.10, and all other computations (including indexing and searching) were implemented by us. Once again we remark that the prototype described is one of many information retrieval combinations made possible by MIE's design, and should be seen as a reference implementation. To showcase the potential of our framework, we also implemented a simple Android and desktop applications which exercise all operations provided by MIE.

Algorithm 9 Search Repository with Object as Query

```
1: function User($U$).Search($ID_R, q, \{rk1_R, rk2_R\}, \{ID_{m_i}\}_{i=0}^{n}, k$)
2:     for all $\{ID_{m_i}\}_{i=0}^{n}$ do
3:         $fvs_{m_i}^{q} \leftarrow$ ExtractFeatureVectors($q, ID_{m_i}$)
4:         if Dense-Media($ID_{m_i}$) then
5:             $efvs_{m_i}^{q} \leftarrow$ Dense-DPE.Encode($fvs_{m_i}^{q}, rk1_R$)
6:         else
7:             $efvs_{m_i}^{q} \leftarrow$ Sparse-DPE.Encode($fvs_{m_i}^{q}, rk2_R$)
8:     $\{ID_{p_i}, p_i, score_{p_i}^{q}\}_{i=0}^{k} \leftarrow$ Cloud.Search($ID_R$, $\{ID_{m_i}, efvs_{m_i}^{q}\}_{i=0}^{n}$, $k$)
9:     return $\{ID_{p_i}, p_i, score_{p_i}^{q}\}_{i=0}^{k}$
10: function Cloud.Search($ID_R$, $\{ID_{m_i}, efvs_{m_i}^{q}\}_{i=0}^{n}$, $k$)
11:     for all $\{ID_{m_i}, efvs_{m_i}^{q}\}_{i=0}^{n}$ do
12:         if IsTrained($ID_R$) then
13:             $hist_{m_i}^{q} \leftarrow$ ClusterizeAndSort($CB_R^{m_i}, fvs_{m_i}^{q}$)
14:             $Res_{m_i} \leftarrow Idx[ID_R][ID_{m_i}]$.IndexSearch($hist_{m_i}^{q}$)
15:         else
16:             $Res_{m_i} \leftarrow$ LinearRankedSearch($efvs_{m_i}^{q}, Fvs[ID_R]$)
17:         $Res_{m_i} \leftarrow$ Sort($Res_{m_i}$)
18:     $Res \leftarrow$ FusionRank($\{ID_{m_i}, Res_{m_i}\}_{i=0}^{n}, k$)
19:     return $\{ID_{p_i}, Rep[ID_{p_i}], Res[ID_{p_i}]\}_{i=0}^{k}$
```

VII. Experimental Evaluation

In this section we experimentally evaluate MIE, through the prototype implementation described in the previous section. For experimental baseline comparison, we also extended a recent SSE scheme from the literature [11] to support ranked multimodal querying and implemented two variants: one that is a simple extension of its mechanisms and hence leaks search, access, and frequency patterns; and another where the user encrypts the index with an additively-homomorphic encryption scheme [52], protecting frequency patterns when performing queries. We refer to these schemes as MSSE and Hom-MSSE, respectively, and their full implementation details can be found in Appendix sections A and A, respectively.

Experimental Test-Bench In the following we will present perfomance results for the MIE, MSSE, and Hom-MSSE alternatives, comparing results both from Desktop and Mobile clients and analyzing them to the grain of each sub-operation. As Mobile client device we used a 2013 Nexus 7 Android Tablet, equipped with a Qualcomm Snapdragon S4 Pro quad-core 1.5Ghz CPU, 2 GB RAM running Android Lolipop 5.1.0. As Desktop client we used a Macbook Pro with Mac OS X 10.11, 4GB of RAM, and 2.3Ghz quad-core Core i7 CPU. For the Cloud server, we used an Amazon EC2 m3.large instance, where the average round-trip time for client-server communications is 52.160 ms. In these experiments, the mobile client is connected to the Internet through WIFI 802.11g and the Desktop Client through an ethernet cable (100 mb/s). As dataset we used the MIR-Flickr dataset [29], which contains one million images and their user defined textual tags extracted from the Flickr social network.

Experimental Evaluation Roadmap The goal of our experimental work is to answer the following questions: **i**) what are the implications on user perceived performance (i.e, time consumed by user devices) to process and upload multimodal data to a cloud infrastructure, considering different devices (mobile and desktop) and how performance evolves as we scale the size of the data set in a scenario where a single user is accessing the repository (§VII-A)? **ii**) As MIE was designed to support multiple users and facilitate concurrent accesses to repositories, what are the implication on user perceived latency when two clients concurrently add objects to the same repository (§VII-B)? **iii**) What is the user perceived performance associated with searching a repository using MIE and the concurrent schemes (§VII-C)? **iv**) What is the retrieval precision obtained by MIE, in comparison with the concurrent schemes and with plaintext retrieval (§VII-D)? And finally, **v**) what are the implications of the different schemes on the

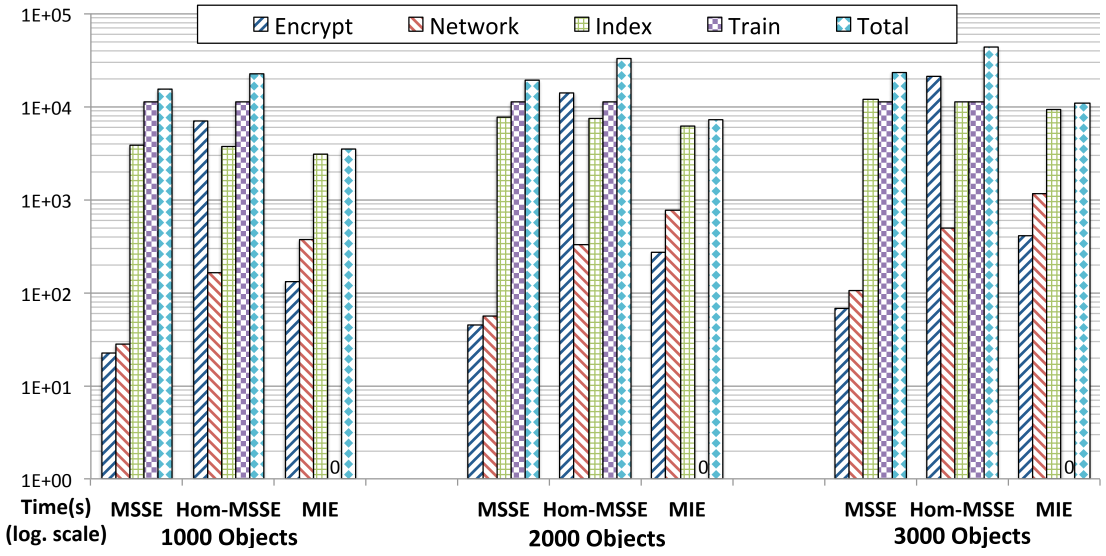

Fig. 2: Performance of the update operation in a mobile device

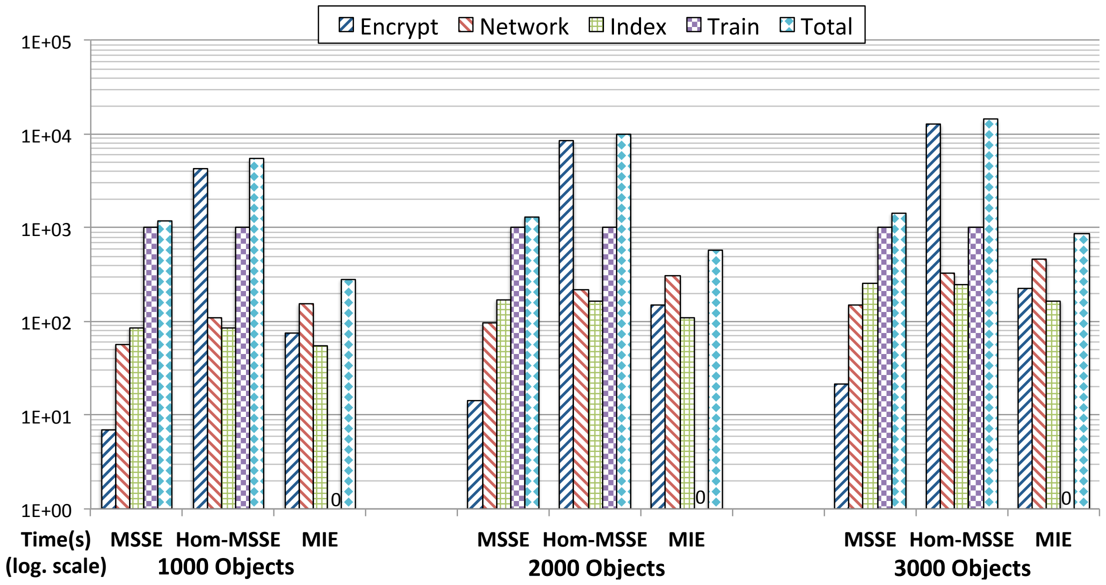

Fig. 3: Performance of the update operation in a desktop device

battery life of mobile devices when users upload new multimodal content to a repository, and how this varies as dataset sizes grow (§VII-E)?

A. *Single User Scenario*

Figures 2 and 3 report the results for the time consumed by respectively, a client executing in a mobile device and in a desktop computer, when initializing a repository and uploading a variable number of multimodal data objects (varying from $1,000$ to $3,000$). Notice that the y-axis in these figures is presented in a logarithmic scale for improved readability. Results are divided between sub-operations: *Encrypt* represents the performance of encryption operations in the three schemes; *Network* represents the time spent with communications and uploading data to the cloud server; *Index* is the time spent by the client extracting multimodal feature-vectors and indexing them; *Train* is the performance of the training operation, where machine learning tasks are performed; *Total* represents the sum of all sub-operations.

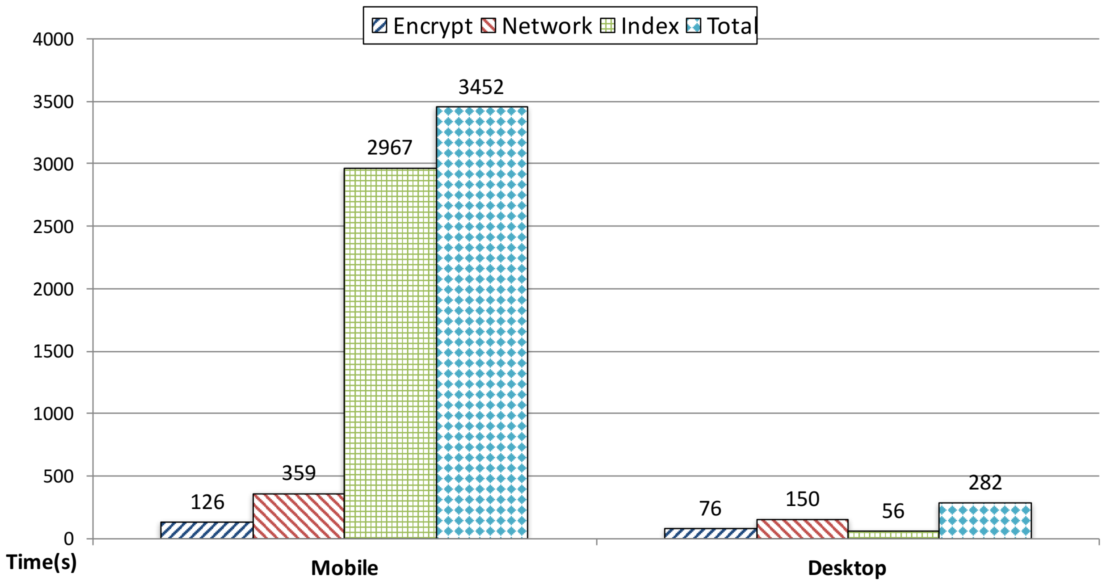

Fig. 4: Multiple client simultaneous update performance, with 1 mobile and 1 desktop client where each upload $1,000$ data-objects

We start by noting that when compared with MSSE and Hom-MSSE, the client that leverages MIE (both in desktop and mobile devices) does not consume any time on the training operation. This is due to MIE's ability to offload this heavy computational step to the cloud in a secure way.

Furthermore the time spent on indexing by MIE clients is lower when compared with MSSE and Hom-MSSE. In this step MIE clients only have to extract feature-vectors from the plaintext data-objects in the different modalities. By encrypting those feature-vectors with our Distance-Preserving schemes (DPE), all other indexing computations are securely offload to the cloud server. In contrast, MSSE and Hom-MSSE clients have to perform those operations in their devices, which include: clustering feature-vectors against the training data-structures obtained during the training step (for dense media types); and indexing those feature-vectors (or their clustered versions), storing the results in encrypted indexing structures which are then uploaded to the cloud.

In the Encryption sub-operation, Hom-MSSE clients exhibit the worst performance due to the use of additively-homomorphic encryption. MIE clients waste more time than MSSE in this sub-operation, as DPE is more expensive than the standard cryptographic primitives used in MSSE, and in the Networking sub-operation MIE clients also show worse performance than the competing schemes, as MIE clients have to upload encoded feature-vectors to the cloud while MSSE and Hom-MSSE only have to upload the already processed and encrypted indexing structures. However, and even if we dismiss the cost of the training operation, MIE clients still show lower total execution time than MSEE and Hom-MSSE clients. The average performance cost increase from MIE to MSEE and Hom-MSSE, considering the three datasets and dismissing training costs, is around 9% and 203% respectively.

Concerning the observed performance across different devices (mobile vs desktop), the relative time spent on each operation for each of the evaluated schemes remains mostly unchanged. However, and as expected, CPU intensive operations such as encryption, indexing, and training perform faster on the desktop computer (approximately 1 order of magnitude). This is explained by the difference in CPU power available in each device. Nonetheless, in both devices and across all data set sizes, MIE allows more efficient processing and storage of multimodal data than the competing alternatives. Consequently, MIE also allows users to initialize and load a secure cloud-backed repository with searchable capabilities in much less time than the competing alternatives (by one order of magnitude approximately). This shows the effectiveness of our alternative, which is able to outsource heavy computational steps to the cloud by exposing at create/update time the same information patterns that the remaining alternatives leak when executing search operations.

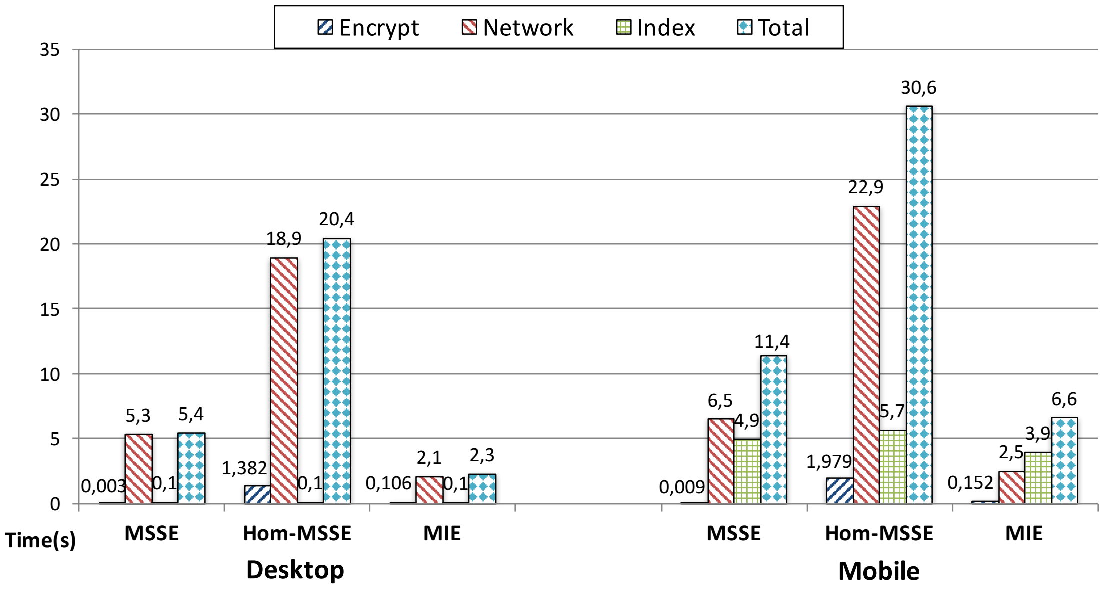

Fig. 5: Performance of the search operation for Mobile and Desktop

B. *Multiple Users Scenario*

We next conducted an experiment where two clients, one executing in a desktop computer and the other on a mobile device, process and upload $1,000$ multimodal data objects each to a single cloud-backed repository. In this experiment we only evaluated the MIE approach, since MSSE and Hom-MSSE (as well as previous SSE schemes [11]) are not easily extendable to multiple users, as they require client-storage that must be consistently synchronized between all users of a repository[3]. Our MIE approach requires no client storage and was designed to enable concurrent write access to data repositories, hence both clients in the experiment can progress at the same time.

Figure 4 summarizes the results for both clients. The figure shows that when compared with the results of the previous sub-sections, both clients are able to make independent progress and both consume the same amount of time when storing a dataset composed of $1,000$ multimodal objects.

C. *Query Performance*

Figure 5 reports the total time required by a client (either on a desktop computer or a mobile device) to perform a query on a repository with $1,000$ multimodal objects and obtain an answer from the cloud infrastructure. In this experiment, since searching is a synchronous operation (contrary to the previous operations that were asynchronous), the *Network* sub-operation contemplates the time spent on communications with the cloud servers and the time the cloud servers take to respond to the query. The results show that in both devices MIE out-performs significantly the competing solutions MSSE and Hom-MSSE. The reasons that explain this are two-fold. First, MIE was designed to only extract feature-vectors from the multimodal object used as query, while the other approaches also have to cluster these feature-vectors with the output of the training task, in order to determine the index positions that should be accessed by the cloud servers. The effect of this is shown in the *Index* sub-operation. Second, MIE requires less computational effort in the cloud servers than the MSSE and Hom-MSSE approaches, which is shown in the *Network* sub-operation. As expected, on mobile devices all solutions take more time than in the desktop computer to process and fetch relevant information for a query, however the increase is proportional across the different schemes.

[3]SSE schemes could use some form of strongly consistent distributed storage in order to keep client state synchronized between users, however such an approach, especially on mobile devices, would increase performance and bandwidth overheads even further.

	Plaintext	MSSE	Hom-MSSE	MIE
mAP (%)	57.938	57.965	57.881	57.562

TABLE III: Mean Average Precision (mAP) for the Holidays dataset [32]

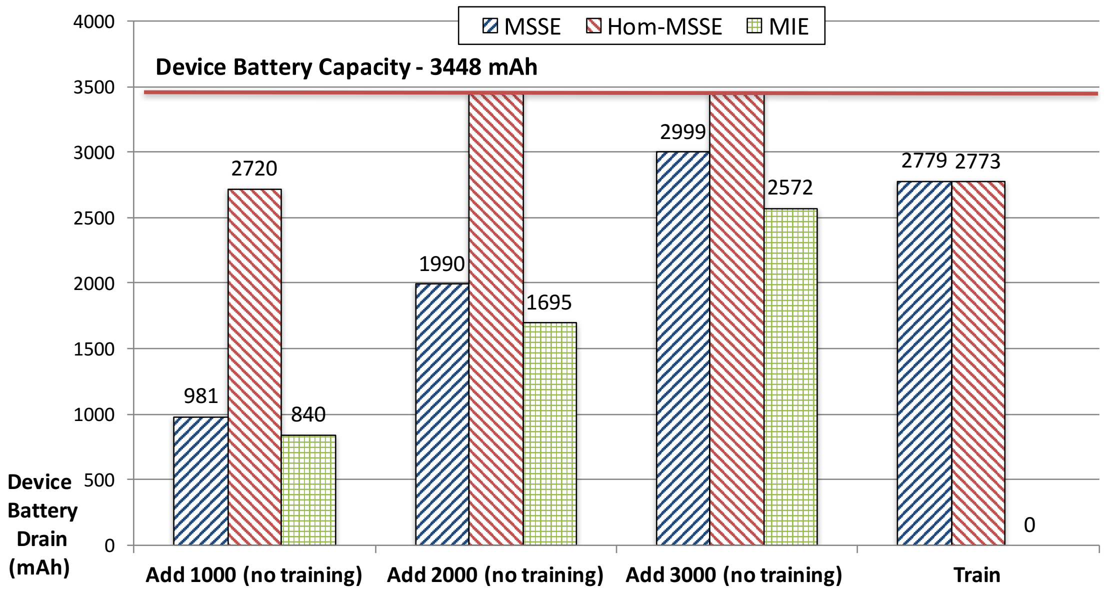

Fig. 6: Mobile energy consumption for the different operations

These results clearly show that not only MIE is more performant than MSSE and Hom-MSSE, but it is also well suited for mobile devices when storing multimodal data on a public cloud infrastructure and when performing queries to retrieve data objects.

D. Query Precision

Dense-DPE, used in the encryption of dense feature-vectors (e.g. those extracted from images), is the only MIE component that may possibly introduce entropy for retrieval operations, affecting query results. As such, we assessed the retrieval precision obtained by MIE and the competing alternatives when querying an image-only repository. This evaluation was performed using the Inria Holidays dataset and its evaluation package [32], measuring the mean average precision (mAP) of 500 queries over a repository of 1491 photos. Table III shows an average of 10 independent executions for MIE, the competing alternatives MSSE and Hom-MSSE, and a plaintext retrieval system based on the same image retrieval techniques.

All assessed systems obtained similar retrieval precision results. Dense-DPE (in MIE) does not meaningfully affect retrieval precision as long as encoded features are at least as large their plaintext versions. Homomorphic encryption (in Hom-MSSE) also seems to preserve the precision of the retrieval algorithms. Finally, we believe that the result of the training operation may have a more meaningful impact on retrieval precision than any other component in the framework system, as clustering is a NP-Hard problem and only an approximated solution can be found [28].

E. Mobile Energy Consumption

As one of our goals is to provide adequate support to mobile devices, it is relevant to measure the draining of energy from a mobile device battery when creating a cloud-based repository and loading it with $1,000$, $2,000$, or $3,000$ multimodal objects. We also report the energy required to train the repository using machine learning techniques, which is required by the MSSE and Hom-MSSE solutions. For improved readability, the results for training and adding the three datasets are shown in separate. The measured energy capacity of the battery in the mobile device used in these experiments was $3,448mAh$. Figure 6 reports the obtained results, which were measured through Android's Operating System Power Profiles Framework [25]. This framework allows users to verify in a precise and hardware-backed way how much energy is consumed in a given period of time by the different applications running in the system.

Results show that MIE significantly outperforms the remaining schemes. This is a reflection of the results shown in the previous sub-sections, and further proves that MIE is more lightweight and better suited for mobile adoption than the state of the art alternatives. For the $2,000$ and $3,000$ dataset sizes, the Hom-MSSE scheme surpassed the available energy capacity, causing the mobile device to shutdown before completion of the test. Furthermore, as shown in Figure 6, MIE is also able to avoid the train operation which almost depletes the energy of the mobile device on its own. These results show that MIE is effectively the solution which is best tailored for operation on mobile devices with limited energy life.

VIII. Conclusion

In this paper we have tackled the practical challenges of efficient and dynamic storage and search of encrypted multimodal data on public clouds, while supporting resource constrained mobile devices. Our main contribution, named *Multmimodal Indexable Encryption* (MIE), is the first approach to address this problem, and is particularly suited for practical contexts and mobile devices. At the core of MIE lies a novel family of encoding algorithms, called *Distance Preserving Encoding* (DPE), which preserve a controllable distance function between plaintexts after encoding. By leveraging DPE, MIE is able to outsource indexing and training computations (shown to be the core of heaviest computations) from the mobile devices to the cloud servers in a secure way. We have implemented a prototype of MIE, operating both on desktop computers and Android mobile devices. Our prototype supports both textual and image modalities. We have experimentally shown that MIE is more adequate than other approaches for storing and searching encrypted multimodal data, especially when client applications are executed in resource constrained mobile devices.

References

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. In *SIGMOD'98*, pages 94–105, 1998.

[2] F. Baldimtsi and O. Ohrimenko. Sorting and Searching Behind the Curtain. In *FC'15*, 2015.

[3] H. Bay, T. Tuytelaars, and L. V. Gool. SURF: Speeded Up Robust Features. In *ECCV'06*, pages 404–417. Springer, 2006.

[4] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa. DepSKY: Dependable and Secure Storage in a Cloud-of-Clouds. *ACM Transactions on Storage*, 9(4), 2013.

[5] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Eurocrypt 2004*, pages 506–522. Springer, 2004.

[6] R. Bost. Sophos - Forward Secure Searchable Encryption. In *CCS'16*. ACM, 2016.

[7] P. Boufounos and S. Rane. Secure binary embeddings for privacy preserving nearest neighbors. In *IEEE International Workshop on Information Forensics and Security*, pages 1–6. Ieee, nov 2011.

[8] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. *Journal of the ACM*, 51(4):38, 2004.

[9] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou. Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data. *IEEE Transactions on Parallel and Distribibuted Systems*, 25(1):222–233, 2014.

[10] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-Abuse Attacks Against Searchable Encryption. In *CCS'15*, pages 668–679. ACM, 2015.

[11] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS'14*, volume 14, 2014.

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

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

[14] Cisco. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016 – 2021. https://tinyurl.com/zzo6766, 2017.

[15] ComScore. The 2016 U.S. Mobile App Report. Technical report, 2016.

[16] T. Cook. A Message to Our Customers. Apple. https://www.apple.com/customer-letter/, 2016.

[17] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In *CCS'06*, pages 79–88, 2006.

[18] R. Datta, D. Joshi, J. Li, and J. Z. Wang. Image retrieval. *ACM Computing Surveys*, 40(2):1–60, 2008.

[19] H. T. Dinh, C. Lee, D. Niyato, and P. Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611, 2013.

[20] B. Ferreira, J. Rodrigues, J. Leitão, and H. Domingos. Privacy-Preserving Content-Based Image Retrieval in the Cloud. In *SRDS'15*. IEEE, 2015.

[21] T. Frieden. VA will pay $20 million to settle lawsuit over stolen laptop's data. http://tinyurl.com/lg4os9m, 2009.

[22] B. Fung. In 5 years, 80 percent of the whole Internet will be online video. http://tinyurl.com/jxod8kf, 2015.
[23] C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO'12*, pages 850–867. Springer, 2012.
[24] S. Goldwasser and Y. T. Kalai. Cryptographic Assumptions: A Position Paper. Cryptology ePrint Archive, Report 2015/907, 2015.
[25] Google. Power Profiles for Android. https://source.android.com/devices/tech/power/index.html.
[26] G. Greenwald and E. MacAskill. NSA Prism program taps in to user data of Apple, Google and others. http://tinyurl.com/oea3g8t, 2013.
[27] F. Hahn and F. Kerschbaum. Searchable Encryption with Secure and Efficient Updates. In *CCS'14*, pages 310–320. ACM, 2014.
[28] J. A. Hartigan. *Clustering algorithms*. Wiley, 1975.
[29] M. J. Huiskes and M. S. Lew. The MIR Flickr Retrieval Evaluation. In *MIR '08*, New York, NY, USA, 2008. ACM.
[30] M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS*, 2012.
[31] Itseez. OpenCV: Open Source Computer Vision. http://opencv.org.
[32] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *Computer Vision-ECCV*, pages 304–317. Springer, 2008.
[33] A. Juels and M. Wattenberg. A Fuzzy Commitment Scheme. *CCS '99*, pages 28–36, 1999.
[34] S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. *FC'13*, pages 1–15, 2013.
[35] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *CCS'12*, pages 965–976. ACM, 2012.
[36] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. CRC PRESS, 2007.
[37] S. Khalaf. Health and Fitness Apps Finally Take Off, Fueled by Fitness Fanatics. http://tinyurl.com/q4wyl7j, 2014.
[38] B. H. Kim and D. Lie. Caelus: Verifying the Consistency of Cloud Services with Battery-Powered Devices. *S&P'15*, 2015.
[39] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR'06*, volume 2, pages 2169–2178. IEEE, 2006.
[40] D. Lewis. iCloud Data Breach: Hacking And Celebrity Photos. https://tinyurl.com/nohznmr, 2014.
[41] W. Lu, A. Swaminathan, A. L. Varna, and M. Wu. Enabling Search over Encrypted Multimedia Databases. In *IS&T/SPIE Electronic Imaging*, pages 725418–725418–11, feb 2009.
[42] P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish. Depot: Cloud Storage with Minimal Trust. *ACM Transactions on Computer Systems*, 29(4):1–38, dec 2011.
[43] C. D. Manning, P. Raghavan, and H. Schütze. *An Introduction to Information Retrieval*, volume 1. Cambridge University Press, 2009.
[44] M. Meeker. Internet Trends 2015. In *Code Conference*, 2015.
[45] Microsoft. HealthVault. https://www.healthvault.com/, 2016.
[46] A. Mourão, F. Martins, and J. Magalhães. NovaSearch at TREC 2013 Federated Web Search Track : Experiments with rank fusion. In *TREC*, number Task 1, pages 1–8, 2013.
[47] A. Mourão, F. Martins, and J. Magalhães. Multimodal medical information retrieval with unsupervised rank fusion. *Computerized Medical Imaging and Graphics*, may 2014.
[48] M. Naveed. The Fallacy of Composition of Oblivious RAM and Searchable Encryption. Technical report, Cryptology ePrint Archive, Report 2015/668, 2015.
[49] M. Naveed, M. Prabhakaran, and C. A. Gunter. Dynamic Searchable Encryption via Blind Storage. In *IEEE S&P*, 2014.
[50] D. Nistér, H. Stewénius, D. Nister, and H. Stewenius. Scalable recognition with a vocabulary tree. In *IEEE CVPR'06*, volume 2, pages 2161–2168. IEEE, 2006.
[51] OpenSSL Software Foundation. OpenSSL: Cryptography and SSL/TLS Toolkit. https://www.openssl.org.
[52] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT'99*, pages 223–238, 1999.
[53] R. A. Popa, E. Stark, J. Helfer, and S. Valdez. Building web applications on top of encrypted data using Mylar. In *NSDI'14*, 2014.
[54] D. Rushe. Google: don't expect privacy when sending to Gmail. http://tinyurl.com/kjga34x, 2013.
[55] E. Stefanov, C. Papamanthou, and E. Shi. Practical Dynamic Searchable Encryption with Small Leakage. In *NDSS'14*, 2014.
[56] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, S. Devadas, M. V. Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path oram: An extremely simple oblivious ram protocol. *CCS'13*, 2013.
[57] C. Wang, N. Cao, K. Ren, and W. Lou. Enabling Secure and Efficient Ranked Keyword Search over Outsourced Cloud Data. *IEEE Transactions on Parallel and Distributed Systems*, 23(8):1467–1479, aug 2012.
[58] Y. Zhang, J. Katz, and C. Papamanthou. All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption. In *Usenix Security '16*, 2016.

APPENDIX

A MULTIMODAL SSE SCHEME

In this section we detail how we implemented and extended a recent SSE scheme from the literature [11] to support multimodal querying. This scheme, which we call MSSE, was used in §VII as a baseline comparison for the experimental evaluation of our work.

An Exact-Match Text Searching Scheme. From the recent literature on SSE schemes [11], [27], [34], [35], [49], [53], [55], whose authors have been focusing on single keyword exact-match search on text documents, we found the approach by Cash et al. [11] to be the most promising for supporting multimodal queries. The scheme originally requires users to store, in their devices, a counter for each unique keyword found in the repository of documents. These counters are incremented each time a new document with that keyword is added. Counter values are used to determine where to store keyword/document occurrences in the index of the repository. Index positions (i.e. the counters) are encrypted with a Pseudo-Random Function (PRF) and a key derived from its respective keyword, while index values (i.e. document identifiers) are encrypted with a IND-CPA block-cipher encryption scheme (such as AES in CTR mode [36]) and a second key derived from the keyword. To search with a query keyword (in the Random Oracle Model [8]) the user derives its two keys and sends them to the server, which finds index positions by applying the PRF to an incrementing value starting at zero and stopping when an empty index position is found.

From Exact-Match to Ranked Searching. Due to its simplicity, it's straightforward to extend the previous methodology to support richer query expressiveness and multimodal searching. For ease of exposition we start by discussing how to perform ranked text searching. In this case we need to store frequency information along with keyword-document occurrence. This will be the basis for all scoring functions, including the popular TF-IDF [43]. Since both informations are closely related, we can concatenate frequencies to document ids and store their IND-CPA encryption in the index. For calculating ranking functions other repository wide metrics may still be required, however these are usually easy to infer from general information that the server already has access to. In the case of TF-IDF these include the total number of stored documents, which is general information usually leaked to the server, and document frequency, i.e. number of documents that the keyword appears in, which the server already has access to when searching.

Supporting Multimodality. Extending this methodology to search over other modalities is also straightforward, given some index representation of the features in each modality. For example, image features (of any kind, from facial recognition to colored key-point detection) can be represented as visual words and indexed the same way as text (i.e. through an inverted index) [50]. Similar approaches can be used for indexing audio and video features. Searching in multiple modalities simultaneously can be achieved by merging search results of each separate modality. We achieve this by using an unsupervised late rank fusion approach such as the one proposed in [46].

Updates and Removals. One of the main limitations of the approach by Cash et al. [11] is that it requires server storage for supporting the removal of data-objects. This server storage grows linearly with the number of removed keywords. This is a consequence of hiding the full document structure through the use of counters. When documents are removed neither the user nor the server have enough information to assert which index entries can be removed. In multimodal ranked retrieval this is further aggravated as an update dictionary will be required instead, keeping track of all updates to keyword frequencies (removals can be seen as a frequency update to zero). Furthermore in the original scheme [11] the revocation list can fluctuate in size, if removed keywords are later re-added to their documents however, in our case the updated dictionary would only grow in size (up to a maximum bound of the main index size) since future updates could have any frequency value.

We remark that the only benefit of this approach is in being able to hide document lengths. However document lengths were actually being implicitly revealed when adding new documents, as users need to store not only index positions but also the documents themselves, hence when adding a single document the server could link its id to the index positions added (adding multiple documents in batch would still give lower and higher bounds on their sizes to the server).

In MSSE we remove revocation and update storage, keeping only one index at the server which stores document ids in plaintext. To remove a document the server either goes through the index and deletes all of its occurrences or alternatively, in background the server builds a structure mapping document ids to their positions in the index, which speads up removals. Updates are performed by first removing the document and then adding its new version. Index positions are still encrypted counter values and frequency values are still IND-CPA encrypted, both only being revealed at search time. The consequence of this approach is that document lengths (i.e. the number of unique keywords per document) will be revealed, as the server can count how many times each document id appears. Nonetheless this can still be hidden through index padding as previously proposed in [10].

Multiple Clients and Client Storage. The methodology proposed by Cash et al. [11] requires clients to be stateful, i.e. they must store in their devices (or in the server and retrieve them with each operation) the counters for each unique keyword. In fact, all of the most recent SSE schemes with smallest information leakage and practical performance require client-storage [11], [27]. In settings with multiple clients the negative aspects of such design becomes further exacerbated, as now clients using the same repository must share the same client storage and make sure their replicas are consistent. To solve this issue we propose a centralized consistency preservation mechanism. In this mechanism, counters are stored encrypted in the server and are requested for each update and search operation. Since updates need to increment counter values and the server can not perform this operation without learning their value, it must be the users that retrieve and decrypt all counters, increment the relevant ones, and upload all back to the server after encryption. To make sure users do not override counter increments, and consequently index positions, the server locks write accesses to this counter dictionary (searching can proceed as normal as such operations can use a (eventual not update) snapshot view of the index that was valid when the operation is first received by the cloud infrastructure).

Figure 7 presents a formalization of scheme MSSE.

MSSE Without Frequency Patterns

One issue with MSSE is that besides revealing search and access information patterns as in previous SSE schemes, it further reveals frequency patterns with each query. We now propose a second multimodal SSE scheme, which we call Hom-MSSE, that is able to hide frequency information patterns at the cost of increased cryptographic overhead. Hom-MSSE was also used in §VII as a baseline comparison for the experimental evaluation of our work.

Our proposal is based on partially homomorphic cryptography. In MSSE if we encrypt index keyword frequencies with an Additively Homomorphic IND-CPA scheme, such as Paillier [52], the server can calculate search scores without knowing their values, through encrypted frequency additions and multiplications with public parameters. For instance, in the TF-IDF function frequencies will be homomorphically encrypted and added, while inverse document frequencies are public parameters (that were already revealed as discussed in the previous section) that will be multiplied. One limitation of this approach however is that now it must be the user to sort search results in each modality and merge them to obtain the final search scores. In [2] an approach for privacy-preserving sorting by the cloud server is proposed, however a cryptographic co-processor is also required in the cloud infrastructure, which is not available in most of nowadays publicly available clouds and as thus we don't consider it a practical assumption.

We can further extend the use of partially homomorphic cryptography to solve another main issue of MSSE, which is the need for coordination between users when updating repositories. More concretely, if we encrypt counter values with Paillier we can have the server update them without knowing nor learning their values. This way when a user is adding/updating a document, he will request for the required counters current values and at the same time tell the server to increment each by a given encrypted amount. Since adding a single document at a time means that counter increments will always be by one value (and the server can track this), the user can either make updates in batch or pad his requests by requiring additional counters and telling the server to increment them by zero (according to [10], padding by 1.6x of the request size would be enough to stop keyword-retrieval attacks). Figure 8 formalizes the Hom-MSSE scheme, as a modification of MSSE.

Create Repository Operation

```
1: procedure USER($U$).CREATEREPOSITORY($ID_R$, $sp_R$)
2:     $rk1_R \leftarrow PRG(sp_R)$
3:     $rk2_R \leftarrow PRG(sp_R)$
4:     CLOUD.CreateRepository($ID_R$)
5:     RepUsers.ShareKey($\{rk1_R, rk2_R\}$)
6:     return $rk_R = \{rk1_R, rk2_R\}$
7: procedure CLOUD.CREATEREPOSITORY($ID_R$)
8:     $Rep[ID_R] \leftarrow$ InitializeRepository()
9:     $Fvs[ID_R] \leftarrow$ InitializeFeatureVectorsList()
```

Update Operation

```
1: procedure USER($U$).UPDATE($ID_R$, $ID_p$, $p$, $dk_p$, $\{rk1_R, rk2_R\}$, $\{ID_{m_i}\}_{i=0}^{n}$)
2:     $e \leftarrow Enc(dk_p, p)$
3:     for all $\{ID_{m_i}\}_{i=0}^{n}$ do
4:         $fvs^p_{m_i} \leftarrow$ ExtractFeatureVectors($p, ID_{m_i}$)
5:         $efvs^p_{m_i} \leftarrow ENC(rk1_R, fvs^p_{m_i})$
6:     if !IsTrained($ID_R$) then
7:         CLOUD.UntrainedUpdate($ID_R$, $ID_p$, $e$, $\{efvs^p_{m_i}\}_{i=0}^{n}$)
8:     else
9:         $\{ectrs_{m_i}\}_{i=0}^{n} \leftarrow$ CLOUD.GetCtrs($ID_R, \{ID_{m_i}\}_{i=0}^{n}$)
10:        for all $\{ID_{m_i}\}_{i=0}^{n}$ do
11:            $ctrs_{m_i} \leftarrow DEC(rk1_R, ectrs_{m_i})$
12:            $hist^p_{m_i} \leftarrow$ ClusterizeAndSort($fvs^p_{m_i}$)
13:            $L_{m_i} \leftarrow$ InitializeList($|hist^p_{m_i}|$)
14:            for all $\{fv_j, freq_{fv_j}\}_{j=0}^{|hist^p_{m_i}|}$ do
15:                if $ctrs_{m_i}[fv_j] == \{\}$ then
16:                    $ctrs_{m_i}[fv_j] \leftarrow 0$
17:                $k1 \leftarrow PRF(rk2_R, fv_j||1)$
18:                $k2 \leftarrow PRF(rk2_R, fv_j||2)$
19:                $l \leftarrow PRF(k1, ctrs_{m_i}[fv_j])$
20:                $ctrs_{m_i}[fv_j] + +$
21:                $d \leftarrow ID_p||ENC(k2, freq_{fv_j})$
22:                $L_{m_i}$.Add($\{l, d\}$)
23:            $ectrs_{m_i} \leftarrow ENC(rk1_R, ctrs_{m_i})$
24:        CLOUD.TrainedUpdate($ID_R$, $ID_p$, $e$, $\{ID_{m_i}$, $L_{m_i}$, $efvs^p_{m_i}$, $ectrs_{m_i}\}_{i=0}^{n}$)
25: procedure CLOUD.GETCTRS($ID_R, \{ID_{m_i}\}_{i=0}^{n}$)
26:     for all $\{ID_{m_i}\}_{i=0}^{n}$ do
27:         $ectrs_{m_i} \leftarrow Ctrs[ID_R][ID_{m_i}]$
28:         LockCounterAccess($Ctrs[[ID_R][ID_{m_i}]$)
29:     return $\{ectrs_{m_i}\}_{i=0}^{n}$
30: procedure CLOUD.UNTRAINEDUPDATE($ID_R$, $ID_p$, $e$, $\{efvs^p_{m_i}\}_{i=0}^{n}$)
31:     $Rep[ID_R][ID_p] \leftarrow e$
32:     $Fvs[ID_R][ID_p] \leftarrow \{efvs^p_{m_i}\}_{i=0}^{n}$
33: procedure CLOUD.TRAINEDUPDATE($ID_R$, $ID_p$, $e$, $\{ID_{m_i}$, $L_{m_i}$, $efvs^p_{m_i}$, $ectrs_{m_i}\}_{i=0}^{n}$)
34:     for all $\{ID_{m_i}, ectrs_{m_i}\}_{i=0}^{n}$ do
35:         $Ctrs[ID_{m_i}] \leftarrow ectrs_{m_i}$
36:         UnLockCounterAccess($Ctrs[ID_{m_i}]$)
37:     CLOUD.Remove($ID_R, ID_p$)
38:     $Rep[ID_R][ID_p] \leftarrow e$;  $Fvs[ID_R][ID_p] \leftarrow \{efvs^p_{m_i}\}_{i=0}^{n}$
39:     for all $\{ID_{m_i}, L_{m_i}\}_{i=0}^{n}$ do
40:         for all $\{l, d\} \in L_{m_i}$ do
41:             $Idx[ID_R][ID_{m_i}][l] \leftarrow d$
```

Remove Operation

```
1: procedure USER($U$).REMOVE($ID_R, ID_p$)
2:     CLOUD.Remove($ID_R, ID_p$)
3: procedure CLOUD.REMOVE($ID_R, ID_p$)
4:     if $Rep[ID_R][ID_p]! = \{\}$ then
5:         $Rep[ID_R][ID_p] \leftarrow \{\}$;  $Fvs[ID_R][ID_p] \leftarrow \{\}$
6:         if IsTrained($ID_R$) then
7:             for all $\{Idx[ID_R][ID_{m_i}]\}_{i=0}^{n}$ do
8:                 for all $\{Idx[ID_R][ID_{m_i}][l_j]\}_{j=0}^{|Idx[ID_R][ID_{m_i}]|}$ do
9:                     if $Idx[ID_R][ID_{m_i}][l_j].ID == ID_p$ then
10:                        $Idx[ID_R][ID_{m_i}][l_j] \leftarrow \{\}$
```

Train Operation

```
1: procedure USER($U$).TRAIN($ID_R$, $\{rk1_R, rk2_R\}$, $ID_{m_i}, ip_{m_i}\}_{i=0}^{n}$)
2:     $efvs \leftarrow$ CLOUD.GetFeatures($ID_R$)
3:     $fvs \leftarrow DEC(rk1_R, efvs)$
4:     for all $\{ID_{m_i}, ip_{m_i}\}_{i=0}^{n}$ do
5:         $D[ID_{m_i}] \leftarrow$ InitializeIndex($ip_{m_i}$)
6:         if DenseMediaType($ID_{m_i}$) then
7:             $CB^{m_i}_R \leftarrow$ TrainIndex($D[ID_{m_i}], ip_{m_i}, fvs_{m_i}$)
8:             RepUsers.ShareCodebook($CB_R$)
9:         IndexData($D[ID_{m_i}], fvs_{m_i}$)
10:    CLOUD.StoreIndex($ID_R, \{ID_{m_i}, D[ID_{m_i}]\}_{i=0}^{n}$)
11: procedure CLOUD.GETFEATURES($ID_R$)
12:     return $Fvs[ID_R]$
13: procedure CLOUD.STOREINDEX($ID_R, \{ID_{m_i}, D[ID_{m_i}]\}_{i=0}^{n}$)
14:     for all $\{D[ID_{m_i}]\}_{i=0}^{n}$ do
15:         $Idx[ID_R][ID_{m_i}] \leftarrow D[ID_{m_i}]$
```

Search Operation

```
1: procedure USER($U$).SEARCH($ID_R, q, \{rk1_R, rk2_R\}, \{ID_{m_i}\}_{i=0}^{n}, k$)
2:     for all $\{ID_{m_i}\}_{i=0}^{n}$ do
3:         $fvs^q_{m_i} \leftarrow$ ExtractFeatureVectors($q, ID_{m_i}$)
4:     if !IsTrained($ID_R$) then
5:         $\{efvs, Rep\} \leftarrow$ CLOUD.GetFeaturesAndObjects($ID_R$)
6:         $fvs \leftarrow DEC(rk1_R, efvs)$
7:         for all $\{ID_{m_i}\}_{i=0}^{n}$ do
8:             $Res_{m_i} \leftarrow$ LinearRankedSearch($fvs^q_{m_i}, fvs_{m_i}$)
9:         $Res \leftarrow$ FusionRank($\{ID_{m_i}, Res_{m_i}\}_{i=0}^{n}, k$)
10:        return $\{ID_{p_i}, Rep[ID_{p_i}], Res[ID_{p_i}]\}_{i=0}^{k}$
11:    else
12:        $\{ectrs_{m_i}\}_{i=0}^{n} \leftarrow$ CLOUD.GetCounters($ID_R, \{ID_{m_i}\}_{i=0}^{n}$)
13:        for all $\{ID_{m_i}\}_{i=0}^{n}$ do
14:            $ctrs_{m_i} \leftarrow DEC(rk1_R, ectrs_{m_i})$
15:            $hist^q_{m_i} \leftarrow$ ClusterizeAndSort($CB^{m_i}_R, fvs^q_{m_i}$)
16:            $L_{m_i} \leftarrow$ InitializeList($|hist^q_{m_i}|$)
17:            for all $\{fv, freq_{fv}\} \in hist^q_{m_i}$ do
18:                $ll \leftarrow$ InitializeList($ctrs_{m_i}[fv] + 1$)
19:                $k1 \leftarrow PRF(rk2_R, fv||1)$
20:                $k2 \leftarrow PRF(rk2_R, fv||2)$
21:                for $ctr \leftarrow 0 \ldots ctrs_{m_i}[fv]$ do
22:                    $l \leftarrow PRF(k1, ctr)$
23:                    $ll$.Add($l$)
24:                $L_{m_i}$.Add($\{ll, k2, freq_{fv}\}$)
25:        $\{ID_{p_i}, p_i, score^q_{p_i}\}_{i=0}^{k} \leftarrow$ CLOUD.Search($ID_R, \{L_{m_i}\}_{i=0}^{n}, k$)
26:    return $\{ID_{p_i}, p_i, score^q_{p_i}\}_{i=0}^{k}$
27: procedure CLOUD.GETFEATURESANDOBJECTS($ID_R$)
28:     return $\{Fvs[ID_R], Rep[ID_R]\}$
29: procedure CLOUD.SEARCH($ID_R, \{ID_{m_i}, L_{m_i}\}_{i=0}^{n}, k$)
30:     for all $\{ID_{m_i}\}_{i=0}^{n}$ do
31:         $Res_{m_i} \leftarrow$ InitializeList($k$)
32:         for all $\{ll, k2, freq_q\} \in L_{m_i}$ do
33:             $tfs \leftarrow$ InitializeList($|ll|$)
34:             for all $l \in ll$ do
35:                 if $Idx[ID_R][ID_{m_i}][l]! = \{\}$ then
36:                     $\{ID_p, efreq\} \leftarrow Idx[ID_R][ID_{m_i}][l]$
37:                     $freq \leftarrow DEC(k2, efreq)$
38:                     $tfs$.Add($\{ID_p, freq\}$)
39:             for all $\{ID_p, freq\} \in tfs$ do
40:                 $idf \leftarrow log(|Rep[ID_R]|/|tfs|)$
41:                 $tfidf \leftarrow freq_q \times freq \times idf$
42:                 if $Res_{m_i}[ID_p] == \{\}$ then
43:                     $Res_{m_i}[ID_p] \leftarrow tfidf$
44:                 else
45:                     $Res_{m_i}[ID_p] \leftarrow Res[ID_p] + tfidf$
46:         $Res_{m_i} \leftarrow$ Sort($Res_{m_i}$)
47:     $Res \leftarrow$ FusionRank($\{ID_{m_i}, Res_{m_i}\}_{i=0}^{n}, k$)
48:     return $\{ID_{p_i}, Rep[ID_R][ID_{p_i}], Res[ID_{p_i}]\}_{i=0}^{k}$
```

Fig. 7: Scheme MSSE, without Random Oracles. ENC and DEC are the encryption and decryption algorithms of a IND-CPA block-cipher scheme, PRF is a Pseudo-Random Function, and PRG is Pseudo-Random Number Generator

Update Operation

```
1: procedure USER($U$).UPDATE($ID_R$, $ID_p$, $p$, $dk_p$, $\{rk1_R, rk2_R\}$, $\{ID_{m_i}\}_{i=0}^{n}$)
   ...
9:     for all $\{ID_{m_i}\}_{i=0}^{n}$ do
10:        for all $fv_j \in fvs_{m_i}^{p}$ do
11:            $inc_{fv_j} \leftarrow Hom.ENC(rk2.HomPub, 1)$
12:        for all $fv_j \in$ Padding($fvs_{m_i}^{p}$) do
13:            $inc_{fv_j} \leftarrow Hom.ENC(rk2.HomPub, 0)$
14:    CLOUD.GetAndIncCtrs($ID_R$, $\{ID_{m_i}$, $rk2_R.HomPub$, $\{ID_{fv_j^{m_i}}$, $inc_{fv_j^{m_i}}\}_{j=0}^{l}\}_{i=0}^{n}$)
   ...
11:    $ctrs_{m_i} \leftarrow Hom.DEC(rk2_R.HomPriv, ectrs_{m_i})$
   ...
15:    Removed Line
16:    Removed Line
   ...
18:    Removed Line
   ...
21:    $d \leftarrow ID_p || Hom.ENC(rk2_R.HomPub, freq_{fv_j})$
   ...
23:    Removed Line
24:    CLOUD.TrainedUpdate($ID_R$, $ID_p$, $e$, $\{ID_{m_i}$, $L_{m_i}$, $efvs_{m_i}^{p}\}_{i=0}^{n}$)
   ...
25: procedure CLOUD.GETANDINCCTRS($ID_R$, $\{ID_{m_i}$, $rk2_R.HomPub$, $\{ID_{fv_j^{m_i}}$, $inc_{fv_j^{m_i}}\}_{j=0}^{l}\}_{i=0}^{n}$)
26:    for all $\{ID_{m_i}\}_{i=0}^{n}$ do
27:        for all $\{ID_{fv_j}\}_{i=0}^{l}$ do
28:            $ectrs[ID_{m_i}][ID_{fv_j}] \leftarrow Ctrs[ID_R][ID_{m_i}][ID_{fv_j}]$
29:            if $Ctrs[ID_R][ID_{m_i}][ID_{fv_j}] == \{\}$ then
30:                $Ctrs[ID_R][ID_{m_i}][ID_{fv_j}] \leftarrow Hom.ENC(rk2_R.HomPub, 0)$
31:            $Ctrs[ID_R][ID_{m_i}][ID_{fv_j}] \leftarrow$ HomAdd($ectr[ID_{m_i}][ID_{fv_j}]$, $inc_{fv_j^{m_i}}$, $rk2_R.HomPub$)
32:    return $ectrs$
   ...
33: procedure CLOUD.TRAINEDUPDATE($ID_R$, $ID_p$, $e$, $\{ID_{m_i}$, $L_{m_i}$, $efvs_{m_i}^{p}\}_{i=0}^{n}$)
34:    Removed Line
35:    Removed Line
36:    Removed Line
```

Create Repository Operation

```
1: procedure USER($U$).CREATEREPOSITORY($ID_R$, $sp_R$)
   ...
3:     $rk2_R \leftarrow \{HomPub, HomPriv\} \leftarrow PRG(sp_R)$
   ...
```

Search Operation

```
1: procedure USER($U$).SEARCH($ID_R$, $q$, $\{rk1_R, rk2_R\}$, $\{ID_{m_i}\}_{i=0}^{n}$, $k$)
   ...
14:    $ctrs_{m_i} \leftarrow Hom.DEC(rk2_R.HomPriv, ectrs_{m_i})$
   ...
20:    Removed Line
   ...
24:    $L_{m_i}$.Add($ll$, $freq_{fv}$)
25:    $\{\{ID_{p_j}, Rep[ID_R][ID_{p_j}], Res_{m_i}[ID_{p_j}]\}_{j=0}^{|Rep[ID_R]|}\}_{i=0}^{n} \leftarrow$ CLOUD.Search($ID_R$, $\{ID_{m_i}, L_{m_i}\}_{i=0}^{n}$, $rk2_R.HomPub$)
26:    for all $\{ID_{m_i}\}_{i=0}^{n}$ do
27:        $Res_{m_i} \leftarrow Hom.DEC(rk2_R.HomPriv, Res_{m_i})$
28:        $Res_{m_i} \leftarrow$ Sort($Res_{m_i}$)
29:    $Res \leftarrow$ FusionRank($\{ID_{m_i}, Res_{m_i}\}_{i=0}^{n}, k$)
30:    return $\{ID_{p_i}, Rep[ID_R][ID_{p_i}], Res[ID_{p_i}]\}_{i=0}^{k}$
   ...
29: procedure CLOUD.SEARCH($ID_R$, $\{ID_{m_i}, L_{m_i}\}_{i=0}^{n}$, $rk2_R.HomPub$)
   ...
32:    for all $\{ll, freq_q\} \in L_{m_i}$ do
   ...
37:    Removed Line
38:    $tfs$.Add($\{ID_p, efreq\}$)
   ...
41:    $tfidf \leftarrow$ HomMult($efreq$, $freq_q \times idf$, $rk2_R.HomPub$)
   ...
46:    Removed Line
47:    Removed Line
48:    return $\{\{ID_{p_j}, Rep[ID_R][ID_{p_j}], Res_{m_i}[ID_{p_j}]\}_{j=0}^{|Rep[ID_R]|}\}_{i=0}^{n}$
```

Fig. 8: Scheme Hom-MSSE, without Random Oracles. The scheme is presented as an iteration over scheme MSSE from Figure 7. Ellipsis represent skipped lines from Figure 7, and each line after an ellipsis represents a re-written line from Figure 7 with the same line number. Lines marked with *Removed Line* are lines from Figure 7 that should be removed in Hom-MSSE.