MuSE: Multimodal Searchable Encryption for Cloud Applications

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 tackle the practical challenges of searching encrypted multimodal data (i.e. data containing multiple media formats simultaneously), stored in public cloud servers, with reduced information leakage. To this end we propose MuSE, a Multimodal Searchable Encryption scheme that, by combining only standard cryptographic primitives and symmetric-key block ciphers, allows cloud-backed applications to dynamically store, update, and search multimodal datasets with privacy and efficiency guarantees. As searching encrypted data requires a tradeoff between privacy and efficiency, we also propose a variant of MuSE that resorts to partially homomorphic encryption to further reduce information leakage, but at the cost of additional computational overhead. Both schemes are formally proven secure and experimentally evaluated regarding performance and search precision. Experiments with realistic datasets show that our contributions achieve interesting levels of efficiency and privacy, making them suitable for practical application scenarios.**

I. Introduction

Applications nowadays manage increasingly larger data collections [36], including data that simultaneously contains different media formats (also known as multimodal data[1]) [2]. This dataset growth has led to the popularity of cloud services for data and computation outsourcing [1]. In the referred cloud services, applications outsource the storage and computations of their data to third-party managed infrastructures, decreasing operational costs with flexible charging models and leveraging from highly available geo-replicated servers. Moreover, as datasets increase in size, so does the importance of supporting efficient search operations that can return relevant subsets of data in response to multimodal queries [35].

Despite the clear advantages cloud services bring, they also lead to new security and privacy challenges that must be addressed, as outsourcing data and computations also means outsourcing control over them [13]. Recent incidents have shown that privacy is not preserved by cloud providers when using their services [43]. Governmental agencies impose increasing pressure on cloud companies to disclose users' data and build insecure backdoors [14], [23]. Malicious or simply careless cloud administrators have been responsible for critical data disclosures [12], [20]. Last but not least, internet hackers exploiting software vulnerabilities in cloud infrastructures must also be considered, as they may gain remote access to users data even if only for a limited time window [33].

[1]An example of multimodal applications are those for medical center management, where patient records can contain both text (written by the medical doctor) and visual data (images obtained from medical equipment).

The conventional approach for addressing such privacy issues is to have applications encrypt all data in transit and at rest [6]. However this leads to expensive computation and communication overheads, as large sets of multimodal data have to be downloaded (and possibly re-uploaded) when performing operations, especially in applications with frequent search operations. Performing computations over encrypted data directly in the cloud servers is possible, with recent advances in Fully Homomorphic Encryption [22] and Oblivious RAM [48]. However existing schemes still impose too much computation, storage, and/or communication overheads for enabling practical adoption [39].

Nonetheless more fine-grained cryptographic protocols, specifically designed for supporting search over encrypted data, can be used in practice with good privacy-efficiency tradeoffs. These protocols are known as Searchable Symmetric Encryption (SSE) schemes [7], [11], [15] and were originally designed for text data [46], with a few recent schemes also studying how to search encrypted visual data (i.e. images) [19], [34], [50]. In this paper we study a more broad topic: how to support applications dynamically storing and searching encrypted multimodal data, i.e. data that combines different media formats, including text, images, audio, and video[2].

We call our proposal MuSE - Multimodal Searchable Encryption, and base it solely on standard cryptographic primitives, including Pseudorandom Functions (PRFs) and Symmetric-Key Block Ciphers [29]. At its core MuSE relies on inverted index structures [35] and algorithms that represent different media formats through these structures. Multimodal queries (i.e. queries also composed of different media formats) can then be answered by searching in each format's index and combining results through an appropriate merging function. Using these techniques, the research challenge that must be addressed is how to securely protect indexing structures while allowing their privacy-preserving and efficient operation during both multimodal data updating and searching.

Since having both full security (i.e. leaking zero information) and practical efficiency has been shown to be impossible for SSE schemes [39], MuSE is required to reveal some minimal information patterns when performing operations (namely search, access, and frequency patterns [10]). This leakage is common in SSE schemes [7] and results from a tradeoff between security and efficiency that is required to achieve sub-

[2]A solution to this problem can also be fine-tuned to support only one media format at a time, offering the same functionality as existing schemes.

linear search performance. Nonetheless, further exploring this tradeoff we propose a variant of MuSE, called PHom-MuSE, that employs Partially Homomorphic Encryption [42] when encrypting index entries. This second scheme exhibits further reduced leakage by protecting frequency patterns, but at the cost of additional computational overhead. We formally prove the security properties of both schemes, implement them, and experimentally evaluate their performance and scalability with a real world multimodal dataset.

In summary, this work provides the following contributions:

- We start by revising the state of art on SSE, followed by an empirical analysis of existing schemes and their leakage. From this analysis we propose a new framework that will aid both researchers and developers in the characterization of SSE schemes through their leakage (Section II);
- We propose MuSE, an efficient dynamic multimodal searchable encryption scheme that allows cloud applications to securely store, update, and search multimodal datasets, by resorting only to standard and efficient cryptographic primitives. Compared to previous SSE schemes, MuSE provides additional functionality (multimodal ranked searching) while displaying similar efficiency and security (Section IV);
- We propose PHom-MuSE, a variant of MuSE that further reduces its leakage, namely the leakage of frequency patterns, at the cost of additional computational overhead by resorting to Partially Homomorphic Encryption (Section IV-A);
- We formally prove the security properties of our schemes and implement them. Our prototype implementations focus on text and image media formats, nonetheless we explain how to extend them to other medias. Using these prototypes we experimentally evaluate the performance and scalability of our schemes. Real world datasets and publicly-available commercial clouds are used in these experiments (Section VI).

II. Related Work

With the increasing popularity of cloud services and its associated security issues, the topic of searching encrypted data has quickly become an important area of research in recent years. In this field, Searchable Symmetric Encryption (SSE) schemes strive for a practical balance between efficiency and security.

First proposed by Song et al. [46], searching encrypted text documents initially required search time linear in the dataset size. Curtmola et al. [15] used an inverted index to achieve sub-linear search performance, while also providing the first security definitions for SSE. While these works were confined to static datasets, Kamara et al. [27], [28] proposed the first dynamic SSE schemes, where documents could be added, removed, or updated. Naveed et al. [40] designed a dynamic SSE scheme that only required storage services from the cloud, instead of storage and computation as in previous schemes. Cash et al. [11] proposed the most efficient dynamic SSE scheme to date. Stefanov et al. [47] presented the first forward-private dynamic SSE scheme, where updates reveal no information even when combined with previously issued query tokens. Raphael Bost [7] revisited the topic, proposing a more efficient scheme that achieved the same security notion.

The SSE schemes referred so far focused on exact-match searching of text documents, where all documents containing a keyword are returned when the keyword is searched. Ranked searching, where documents are returned in a sorted order of relevance to the query, was addressed by Wang et al. [49] with single keyword queries and Cao et al. [9] with multi-keyword (conjunctive) queries. However these works lacked a formal security analysis. Baldimtsi and Ohrimenko [3] proposed the first ranked SSE scheme with a formal security analysis, however their scheme required a cryptographic co-processor to be deployed in the cloud, under the client's control. Additionally, so far these ranked schemes have been limited 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.

Searching encrypted data has also been designed for other media formats, including visual data (i.e. images). Lu et al. [34] presented the first scheme for encrypted image search. Xia et al. [50] presented a more recent approach to the problem. However these works lack a formal security treatment and do not support dynamic updates. Ferreira et al. [19] presented the first dynamic SSE scheme for images with a formal security analysis, however it leaked more information than previous schemes for text data: it leaked frequency and update patterns for all stored data, including the initial dataset (these patterns will be fully detailed in the next Section). The problem of encrypted multimodal searching was addressed for the first time by Ferreira et al. [17]. Their work supported dynamic updates and provided a formal security analysis, however it also leaked update and frequency patterns for all stored data. Hence in this work we present the first dynamic, efficient, and provably-secure multimodal SSE schemes achieving similar security and leakage guarantees as the state of art literature on SSE for text data.

A. SSE Leakage Analysis

As an extension to the related work analysis performed so far, we now present an empirical study of the leakage of SSE schemes for different media formats. This study was initiated by Cash et al. [10], who focused on the leakage of exact-match queries on text data. In contrast, we also consider the leakage when supporting ranked queries on text data and queries on other media formats.

The efficiency guarantees provided by SSE schemes are only possible by leaking some information patterns with the execution of operations [39]. The most commonly leaked patterns are *search* and *access patterns* [15], both leaked by search operations. Search patterns reveal the history of a query, i.e. how many times it has been performed so far. This information is leaked by deterministic query tokens submitted at search time. Access patterns reveal which documents

Level	Leakage Name	Patterns Leaked	E.g. Schemes
L2	Fully-Revealed Frequency	Search, Access, Frequency & Update	[17], [19]
L1	Fully-Revealed Occurrence	Search, Access & Update	[28], [30], [40]
L0=>L2	Query-Revealed Frequency	Search, Access & Frequency	MuSE, [9], [50]
L0=>L1	Query-Revealed Occurrence	Search & Access	PHom-MuSE, [3], [7]
L0	Blind (Leakage)	–	[21]

TABLE I: Characterization of SSE schemes according to their leakage.

are returned by a query, which is leaked by deterministic identifiers of the documents accessed. These patterns have been revealed by all SSE schemes to date [7], and have been shown to be necessary leakage for achieving practical efficiency [39]. The first dynamic SSE schemes [28], [40] additionally leaked *update patterns* with the update operation: they resorted to deterministic update tokens, revealing if updates shared contents with previous updates and queried documents. Nonetheless, update leakage has been solved in more recent dynamic schemes [7], [11], [27], [47], by making updates non-deterministic. If additionally updates leak no information at all, even when combined with previously issued queries, SSE schemes are said to be *forward-private* [7], [47].

The leakage described so far is characteristic of the most simple type of queries: exact-match searching. As we move to more complex queries, including ranked search of text documents, images, and multimodal data in general, there is an additional data leakage that must be considered: *frequency patterns*, i.e. how many times a keyword (or a similar concept in other formats, e.g. a keypoint or a feature in images) appears in a document. This is a basic metric required for supporting most forms of ranked search [35], and may be leaked by update or search operations. As such, it should also be modeled in the formal treatment of ranked SSE schemes.

Given the previous patterns, Table I provides a new framework that helps characterizing SSE schemes according to their leakage. The framework is divided in different levels[3], with the top level being the least secure (i.e. leaks more data) and the bottom the more secure (i.e. leaks less). L0 reveals nothing except basic information like the dataset size; it represents O-RAM based schemes. L0=>L1 represents typical exact-match SSE schemes (on text data) as a transitory level: at initialization nothing is revealed (as in L0), but with each search some patterns are leaked (more precisely, search and access patterns), eventually leading to the equivalent fully-revealed level (L1). This level also represents the leakage of our PHom-MuSE scheme (which additionally supports multimodal ranked searching). L0=>L2 represents ranked SSE schemes (as is the case of our MuSE scheme) that additionally reveal frequency patterns with queries. L1 represents exact-match schemes (on text data) that leak update patterns, fully revealing the occurrence of keywords even if no queries are performed (we assume databases can start empty, with all data being added through updates, possibly in batches). L2 represents schemes that also reveal frequency patterns with updates and queries.

[3]Comparing to [10] we omit the leakage of document's structure for simplicity, since (as far as we know) there are no known SSE schemes in the literature that reveal it.

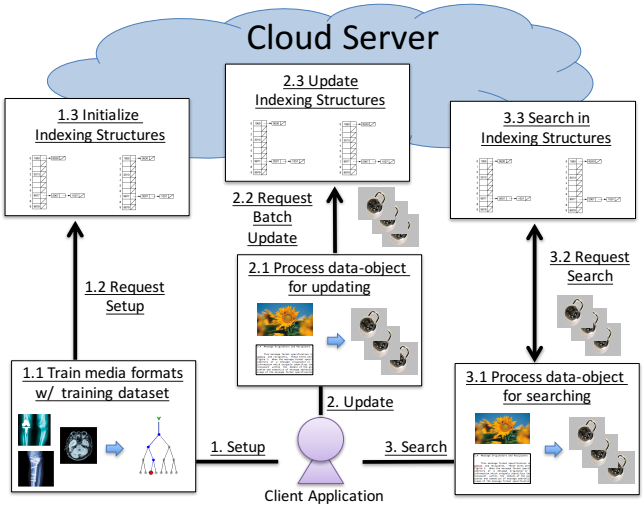

Fig. 1: System model with the interactions between client and cloud server.

III. Technical Overview

This section initiates the technical description of our paper. We start with some notations and concepts, following with an overview presentation of our system and adversary models. We call *multimodal object* to a data object combining multiple media formats. A *multimodal dataset* is a collection of multimodal objects. *Multimodal features* are distinctive characterizations of a data object in its different media formats: e.g. a document's keywords compose its text features, while an image's visual keypoints compose its visual features.

Multimodal searching is the operation used to search a multimodal dataset with a query, where the query is itself a multimodal object. Results of a multimodal search are returned ordered by relevance (or similarity) to the query, and are usually obtained for each media format in separate and aggregated through a merging function [38].

Multimodal indexing consists in building dictionary-like structures, one for each media format, that compactly describe a dataset and where each entry represents a feature (keyword or similar concept in other formats) and its frequency in a data object. Indexing structures allow searching in time sub-linear with the dataset size.

Multimodal training is an operation that is usually required in rich, highly dimensional media formats, including images, audio, and video. It consists in performing clustering operations (e.g. k-means [24]) on the dataset, particularly on the referred formats, reducing the data's dimensionality and allowing it to be more efficiently indexed. The result of training is a codebook structure [41] that assists in this more efficient indexing.

A. System Model

Figure 1 represents the system model employed in this work. We consider a client application and one cloud server, where the client is outsourcing the storage (and some computations) of his multimodal dataset to the server. We assume three main operations between the client and the server: Setup, Update, and Search. The cryptographic protocols subjacent to these operations will be formally specified and detailed in the next Section, while for now we focus on describing a high level overview of the possible interactions between client and server.

The Setup operation initiates the system. The client starts by generating the system's cryptographic keys. Then, for each rich media format where training is required (images, audio, and video) the client trains an appropriate training dataset, storing the resulting codebooks on his side. These will be used in Update and Search operations to allow an efficient indexing and retrieval of multimodal data, respectively. Finally, the client tells the server to initialize the system's indexing structures, one for each media format supported.

As implied by our minimalistic Setup, the client's dataset is initially empty. This means that all data can be added dynamically through the Update operation. When processing a new multimodal object for storage (or an existing one for update), the client starts by processing and extracting its relevant features in each media format. In formats where training is required, these features are additionally clustered through the respective codebook. Then each feature is encrypted and sent to the server through a batch Update request. When the server receives an Update request, he stores the encrypted features in the respective indexing structures.

The Search operation is performed in a similar fashion as the Update. Given a multimodal object as query, the client extracts its features in each media format, clustering them with the respective training codebooks if required. Then each feature is encrypted and the resulting query tokens are sent to the server through a Search request. Query tokens are then used by the server to accesses its indexing structures and calculate search results, which are returned to the client as a single set of object identifiers ordered by relevance to the query.

All communications between the client and the server must be done through secure channels (e.g. TLS/SSL [29]), nonetheless we consider these details to be easily implementable and orthogonal to the main scope of the paper.

B. Adversary Model

In this work we consider as main adversary the cloud server, i.e. the cloud provider company and any system administrators working for it that may have access to the client's data and computations. As in previous works [7], we assume the cloud server to operate in an honest but curious fashion: it is expected to fulfill its contract agreements and not destroy or temper with data and computations, but may eavesdrop on their contents at will without detection by the client. In more detail, the cloud server keeps a log of all operations done and all information leaked by them, and may resort to any other background information available in order to learn the contents of both the dataset stored and the performed queries.

A second important adversary that should also be considered is the *snapshot attacker*, i.e. an adversary that does not have continuous access to the server but may gain that access for a limited time window and may perform a snapshot copy of all stored data. This adversary represents the typical Internet hacker. We informally argue that by addressing the cloud server adversary, our approach is also implicitly addressing this second adversary, since his capabilities are a subset of the first. Hence, we focus our security analysis on the honest-but-curious cloud adversary.

Data integrity, availability, and verifiability are also important issues in cloud-based applications. However these issues can be easily addressed by combining existing techniques in the literature [8], [45], and hence we also find them to be orthogonal to the main scope of the paper.

IV. Designing a Multimodal SSE Scheme

In this section we detail MuSE, our efficient multimodal SSE scheme, and analyse its security properties. We start by defining what is a Dynamic Multimodal Searchable Encryption scheme.

Definition 1 (Dynamic Multimodal Searchable Encryption)**.** *A Dynamic Multimodal Searchable Encryption scheme consists of three protocols* Setup, Update, and Search *executed between a client and a server, such that:*

- $\textsc{Setup}(\textsc{SetupC}(1^\lambda, m, \{\{w_i^j, f_i^j\}_{i=0}^n\}_{j=0}^m), \textsc{SetupS}(m))$ *is the protocol used to initiate the scheme. The client takes as input the security parameter λ, the number of modalities m, and a training dataset $\{\{w_i^j, f_i^j\}_{i=0}^n\}_{j=0}^m$ (where w_i^j is a feature and f_i^j is its frequency in the object). It trains the modalities that require training and generates the cryptographic keys of the scheme, outputting these keys and the codebooks resulting from the training step. The server also takes m as input and initializes the indexing structures of each modality as empty, returning no outputs.*
- $\textsc{Update}(\textsc{UpdateC}(\{\{w_i^j, id_i^j, f_i^j\}_{i=0}^n\}_{j=0}^m), \textsc{UpdateS}(\{\{ut_i^j\}_{i=0}^n\}_{j=0}^m))$ *is a protocol between the client with input a group of n features w_i^j, object identifiers id_i^j, and frequencies f_i^j in m different modalities, and the server with input n update tokens ut_i^j in the same m modalities. The client builds each ut_i^j as a function of w_i^j, id_i^j, and f_i^j, while the server uses ut_i^j to update its indexing structures accordingly. This protocol reflects a batch update of multiple features n in different modalities m, where each sub-update can represent the addition of a new feature w to a (also possibly new) object id, an update to the frequency f of an existing w in id, or the deletion of w from id (in which case f is zero).*
- $\textsc{Search}(\textsc{SearchC}(\{\{w_i^j, f_i^j\}_{i=0}^n\}_{j=0}^m), \textsc{SearchS}(\{\{st_i^j\}_{i=0}^n\}_{j=0}^m))$ *is the protocol used to perform a multimodal search. The client takes as input a query object, represented as a collection of n features and their frequencies in m different modalities ($\{\{w_i^j, f_i^j\}_{i=0}^n\}_{j=0}^m$). The server receives the respective search tokens $\{\{st_i^j\}_{i=0}^n\}_{j=0}^m$ and returns a set of object identifiers ordered by relevance to the query.*

We now detail the operations of MuSE. We begin by designing a scheme that only supports exact-match searching in text documents, expanding its usability by steps until we achieve full multimodal ranked searching.

An Exact-Match Text Searching Scheme. Exact-match searching in text documents has been extensively researched in the literature [7], [11], [28]. From the previous works, we found the methodology of Cash et al. [11] for dynamic SSE to be one of the most efficient and promising for extension to richer queries. In this approach the client stores D, a dictionary of counters where each unique keyword in the dataset is mapped to a counter initiated at 0. Each counter value represents a new object where the keyword is being added to, and counter values are used during updates/searches to determine where to store/find keyword-document occurrences in the server's index. Index positions (i.e. the counters) are encrypted with a Pseudo-Random Function (PRF) [29] and a key derived from the respective keyword, while index values (the documents' ids) are encrypted with a RCPA block-cipher encryption scheme (i.e. a block cipher scheme resistant to Chosen-Plaintext Attacks [29]) and a second key derived from the keyword. Encrypted index positions and values combined form an update token.

When searching with a query keyword the client derives its two keys, as in the update protocol, and sends them to the server. By applying a PRF (with the first key) to an incrementing counter value c, initiated at zero and stopping when an empty index position is found, the server is able to efficiently find all relevant index positions. These are then decrypted with the second key and returned to the client.

From Exact-Match to Ranked Searching. In ranked text searching we need to store not only keyword-document occurrences, but also their frequencies. Frequency is the basis for most ranked scoring functions, including the popular TF-IDF [35] (which we will be using in MuSE). Since both informations (occurrence and frequency) are closely related, we design our extended scheme to concatenate frequencies with document ids and store their RCPA encryption as index values. For calculating ranking scores, other repository wide metrics may still be required, including the dataset size (i.e. number of objects) and keyword dataset size (i.e. number of objects containing the keyword), nonetheless these are usually general information that the server already has access to.

Dynamic Updates and Deletions. The scheme described so far efficiently supports new additions of keywords to documents. However supporting updates of existing keyword-document occurrences, including frequency updates and deletions, is still challenging. This is a side effect of the counters approach, since when performing updates there is no way for the client or server to know if the specified keyword already exists in the object and where in the index is this information stored. Searching for the keyword before updating the index would solve this problem, however it would also lead to additional unnecessary leakage.

We foresee two solutions to this problem. The first consists in incrementing keyword counters with all updates. When searching, only the most recent frequencies for each document id (given by higher counter values) will be used. This solution works better for applications with few updates, as it will make index size grow significantly. Since we expect dynamic SSE schemes to receive many update operations, we devise a second solution that requires larger server storage at setup time, but no additional storage will be required as updates are performed.

Our solution consists in dividing index storage in two data structures. In the first index, which we call I^A, we map PRFs on keyword counters to encrypted document ids. I^A represents our previous index and allows efficient searching through the counters approach. In the second index, called I^U, we map PRFs on document ids to encrypted frequencies. I^U allows efficient updates to keyword frequencies, as well as keyword deletions, without requiring knowledge of the respective index positions in I^A.

In more detail our update protocol will now give the server two update tokens (per feature) as input, $ut^A = (l^A, d^A)$ and $ut^U = (l^U, d^U)$, where the first represents our old tokens and is used on index I^A, and the second represents our new tokens (mapping ids to frequencies) and is used on I^U. The server starts by accessing I^U with (l^U, d^U). If there already exists an entry for it (meaning that this is an update or deletion of an existing frequency) then it stores the new encrypted frequency (which will be 0 for deletions) and discards ut^A. Otherwise, besides storing d^U in $I^U[l^U]$, it also stores d^A in $I^A[l^A]$. Finally, the server outputs to the client a bit r, where value 0 means that this operation was a new addition and value 1 means it was an update to an existing frequency. The client now waits for this response before incrementing c, and only updates it if r is 0.

The search protocol now also needs a second search token for each feature, and accesses both indexing structures: first the server accesses I^A with the old query token and then, after decrypting the document id fetched from I^A, it accesses I^U with it and decrypts the corresponding frequency.

Supporting Multimodality. So far we have an index approach that efficiently supports the storage, update, and ranked searching of text data. If we can find similar index representations in other media formats, extending our approach to multimodal searching will be straightforward to achieve.

Image features of any kind, (e.g. from facial recognition to keypoint detection [16]) can be clustered and represented as visual words [41], allowing their efficient indexing in dictionary structures as performed for text features. Similar approaches can be used for indexing audio [32] and video features [44]. In these approaches, a training phase is usually required before indexing is possible, which takes a training dataset as input and builds a codebook cb_j for each modality that requires training. Hence we change the Setup operation to have the client perform this training and storing $cb = \{cb_j\}_{j=0}^{m}$. Moreover,

Setup($1^\lambda, m, \{\{w_i^j, f_i^j\}_{i=0}^n\}_{j=0}^m$)

Client:

for $j = 0..m$ **do**
  $K_j^A, K_j^U \xleftarrow{\$} \{0,1\}^\lambda$
  $D_j \leftarrow \mathsf{Init}()$
$cb \leftarrow$ Train($\{\{w_i^j, f_i^j\}_{i=0}^n\}_{j=0}^m$)
Send m to the server.

Server:

for $j = 0..m$ **do**
  $I_j^A, I_j^U \leftarrow \mathsf{Init}()$

Update($\{\{w_i^j, id_i^j, f_i^j\}_{i=0}^n\}_{j=0}^m$)

Client:

$\{\{cw_i^j, id_i^j, cf_i^j\}_{i=0}^n\}_{j=0}^m \leftarrow$ Cluster($cb, \{\{w_i^j, id_i^j, f_i^j\}_{i=0}^n\}_{j=0}^m$)
$ut \leftarrow []$
for $j = 0..m$ **do**
  $ut_j \leftarrow []$
  for $i = 0..n$ **do**
    $K1^A \leftarrow F(K_j^A, cw_i^j||1)$; $K2^A \leftarrow F(K_j^A, cw_i^j||2)$
    $K1^U \leftarrow F(K_j^U, cw_i^j||1)$; $K2^U \leftarrow F(K_j^U, cw_i^j||2)$
    $c \leftarrow D_j[cw_i^j]$
    if $c = \perp$ **then** $c \leftarrow 0$
    $l^A \leftarrow F(K1^A, c)$; $d^A \leftarrow Enc(K2^A, id_i^j)$
    $l^U \leftarrow F(K1^U, id_i^j)$; $d^U \leftarrow Enc(K2^U, cf_i^j)$
    $ut_j \leftarrow \{l^A, d^A, l^U, d^U\} : ut_j$
  $ut \leftarrow ut_j : ut$
Send ut to the server.

Server:

$R \leftarrow []$
for all $ut_j \in ut$ **do**
  $R_j \leftarrow []$
  for all $\{l^A, d^A, l^U, d^U\} \in ut_j$ **do**
    if $I_j^U[l^U] = \perp$ **then**
      $I_j^A[l^A] \leftarrow d^A$; $r \leftarrow 0$
    else
      $r \leftarrow 1$
    $I_j^U[l^U] \leftarrow d^U$; $R_j \leftarrow r : R_j$
  $R \leftarrow R_j : R$
Send (R) to the client.

Client:

for all $R_j \in R$ **do**
  for all $r \in R_j$ **do**
    if $r = 0$ **then** $D_j[w] \leftarrow D_j[w] + 1$

Search($\{\{w_i^j, f_i^j\}_{i=0}^n\}_{j=0}^m$)

Client:

$\{\{cw_i^j, cf_i^j\}_{i=0}^n\}_{j=0}^m \leftarrow$ Cluster($cb, \{\{w_i^j, f_i^j\}_{i=0}^n\}_{j=0}^m$)
$st \leftarrow []$
for $j = 0..m$ **do**
  $st_j \leftarrow []$
  for $i = 0..n$ **do**
    $K1^A \leftarrow F(K_j^A, cw_i^j||1)$
    $K2^A \leftarrow F(K_j^A, cw_i^j||2)$
    $K1^U \leftarrow F(K_j^U, cw_i^j||1)$
    $K2^U \leftarrow F(K_j^U, cw_i^j||2)$
    $st_j \leftarrow \{cf_i^j, K1^A, K2^A, K1^U, K2^U\} : st_j$
  $st \leftarrow st_j : st$
Send st, N to the server ▷ N is the current dataset size

Server:

$R \leftarrow []$
for all $st_j \in st$ **do**
  $R_j \leftarrow \mathsf{Init}()$
  for all $\{f_q, K1^A, K2^A, K1^U, K2^U\} \in st_j$ **do**
    $L \leftarrow []$
    $c \leftarrow 0$
    $l^A \leftarrow F(K1^A, c)$
    $d^A \leftarrow I_j^A[l^A]$
    while $d^A \neq \perp$ **do**
      $id \leftarrow Dec(K2^A, d^A)$
      $l^U \leftarrow F(K1^U, id)$
      $d^U \leftarrow I_j^U[l^U]$
      $f \leftarrow Dec(K2^U, d^U)$
      $L \leftarrow \{id, f\} : L$
      $c \leftarrow c + 1$
      $l^A \leftarrow F(K1^A, c)$
      $d^A \leftarrow I^A[l^A]$
    $idf \leftarrow \log(\frac{N}{|L|})$
    for all $\{id, f\} \in L$ **do**
      tf-idf $\leftarrow f \times idf \times f_q$
      if $R_j[id] = \perp$ **then** $R_j[id] \leftarrow 0$
      $R_j[id] \leftarrow R_j[id] +$ tf-idf
  $R_j \leftarrow \mathsf{Sort}(R_j)$
  $R \leftarrow R_j : R$
$S \leftarrow \mathsf{ISR}(R)$
Send S to the client

Fig. 2: The MuSE scheme, based on PRF F and RCPA scheme (Enc,Dec).

in the Update and Search operations the client now starts by using cb to cluster the inputed features (in medias requiring this step) before indexing/searching them.

Finally, multimodal searching (i.e. search in multiple formats simultaneously) can be achieved by searching in each format in separate and merging results with a multimodal merging function, such as logarithmic Inverse Square Rank (ISR) rank-fusion [37]. Figure 2 presents MuSE, our final efficient dynamic multimodal scheme.

Security and Leakage Analysis. We now sketch a proof of security for MuSE. A full proof of security can be found in the Appendix Section of this paper. Our security analysis follows the real/ideal simulation paradigm that is standard in secure multi-party computations [29]. We define $\mathcal{L}$=($\mathcal{L}^{Stp}$,$\mathcal{L}^{Upd}$,$\mathcal{L}^{Srch}$) as a leakage function that captures all information MuSE is ideally allowed to leak. Intuitively $\mathcal{L}$ outputs the following:

- The setup protocol leaks m, the number of distinct modalities supported in an instantiation of MuSE (i.e. $\mathcal{L}^{Stp} = m$).
- An update leaks the type of each of its sub-updates (new addition or a frequency update, with deletions indistinguishable from other updates). Additionally, if the added/updated features have already been searched for, it also leaks the corresponding object identifiers and

frequencies (i.e. $\mathcal{L}^{Upd} = \{op_i \in \{add, upd\}, \{id_i, f_i\} : id_i \in \mathcal{L}^{Srch}\}_{i=0}^{n}$).

- Search protocols leak the size N of the dataset and, for all features w contained in a query object, they also leak search, access, and frequency patterns. Search patterns, i.e. if queries are being repeated, are due to the deterministic nature of the search tokens used. Access patterns correspond to the set of object ids that contain each feature queried for. Frequency patterns means that access patterns not only include occurrences, but also frequencies (i.e. $\mathcal{L}^{Srch} = N, ID_w, \{id_i, f_i\}_{i=0}^{|R|}$).

These leakage components, particularly search and access patterns, are unavoidable in efficient SSE and considered minimal leakage [39]. Frequency patterns are additional leakage characteristic of ranked SSE schemes [19], nonetheless we will address them next in our PHom-MuSE scheme at the cost of additional cryptographic overhead. Forward privacy (i.e. preventing updates from revealing if they match contents with previous queries) can be orthogonally addressed, as in [7], by introducing a public-key scheme in the encryption of keyword counters (we leave this as future work).

Non-adaptive security [15] follows if we can prove that MuSE leaks nothing beyond what is specified in $\mathcal{L}$. This proof relies on *F* being a secure PRF and (Enc,Dec) being a RCPA-secure encryption scheme. Additionally if *F* is modeled as a random oracle [5], adaptive security can also be proven and we can state that:

Theorem 1. *MuSE is correct and $\mathcal{L}$-secure against adaptive attacks.*

The proof of this theorem can be found in the Appendix Section of this paper.

A. Multimodal SSE without Frequency Leakage

An issue with MuSE, that was not present in previous exact-match SSE schemes, is the leakage of frequency patterns with search operations. To solve this problem we propose a variant of MuSE that addresses this leakage, at the cost of increased cryptographic overhead. Our proposal, called PHom-MuSE, is based on Partially Homomorphic cryptography, more concretely on an additively homomorphic scheme such as the Paillier cryptosystem [42].

We design PHom-MuSE through simple modifications to MuSE. In the Setup operation the client now additionally generates a private/public key pair for the Paillier scheme. Then, in update operations, we replace the RCPA encryption of keyword frequencies ($d^U \leftarrow Enc(K2^U, f)$, line 11 in the update protocol, Figure 2) with their public-key Paillier encryption. Only the client, who has the private key, can decrypt these values.

Given the use of homomorphic encryption, when responding to search operations the server can calculate search scores through encrypted frequency additions (and multiplications with public parameters, which can be seen as a series of homomorphic additions). The result is the protection of both frequency values and final search scores. In more detail, in the TF-IDF function frequencies f will be encrypted and multiplied by public parameters *idf* and f_q (line 32 in the Search protocol) and the resulting scores for the same object *id* will be homomorphically added (line 34). However it must now be the client to sort search results and perform multimodal merging, since order is not preserved by homomorphic encryption (line 35). The client performs this after receiving encrypted results from the server and decrypting them with the Paillier private key.

We now define $\mathcal{L}_{PHom}$, the leakage that PHom-MuSE is ideally allowed to reveal, as an iteration of our previous leakage function $\mathcal{L}$ for MuSE. In more detail, the only difference between $\mathcal{L}_{PHom}$ and $\mathcal{L}$ is that frequency patterns are not revealed when performing search operations, nor when adding/updating a feature that has already been searched. Furthermore, we can prove that:

Theorem 2. *PHom-MuSE is correct and $\mathcal{L}_{PHom}$-secure against adaptive attacks.*

The proof for this theorem is straightforward to sketch by extending the proof of Theorem 1. The Paillier cryptosystem is used as a black-box component, and PHom-MuSE involves no additional security protocols. Hence, a simulator $\mathcal{S}$ can simulate all the interactions in the protocol using the information it obtains from $\mathcal{L}_{PHom}$. Correctness and security against adaptive attacks follows in the random oracle model and assuming Paillier is a correct and RCPA Additively-Homomorphic scheme. Details are straightforward and thus omitted.

V. Implementation

We implemented prototype versions of our MuSE and PHom-MuSE schemes. These prototypes will be used for experimental evaluation in the next section, while for now we focus on describing their implementation. We focused our prototypes on supporting multimodal data with text and image media formats. All code was developed in C++, with little over 2000 lines of original code. Cryptographic computations were implemented using the OpenSSL 1.0.2 library[4]. PRFs were implemented with an HMAC function, using SHA1 as the underlying cryptographic hash function. The (Enc,Dec) RCPA encryption scheme was implemented with AES in CTR mode, using a 256-bit key. For the Paillier scheme, we used the LIBPAILLIER library from the ACSC project[5].

Algorithms for processing and indexing text data were implemented by us. Text feature extraction was performed first by keyword stemming (Porter Stemming algorithm) and stopwords removal [35]. Indexing was done through the Single-Pass in Memory Indexing (SPIMI) algorithm and as indexing structures we used the inverted list index approach [35]. For processing and indexing images we used the OpenCV 2.4.10 library[6]. For feature extraction, we used the *SURF* keypoint detection [4] and *Dense Pyramid* descriptor extraction [31] algorithms. As a rich media format, images need to be trained

[4]https://www.openssl.org/

[5]http://acsc.cs.utexas.edu/libpaillier/

[6]http://opencv.org/

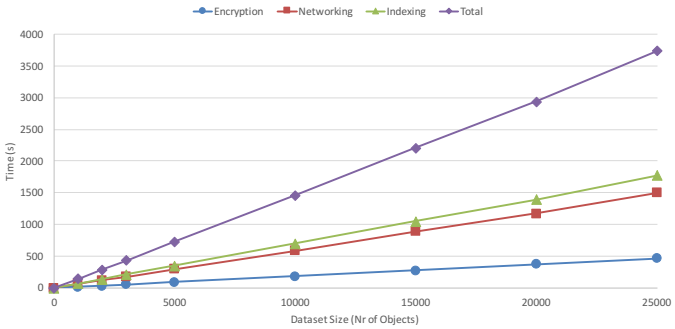

Fig. 3: Performance of MuSE in the Update operation.

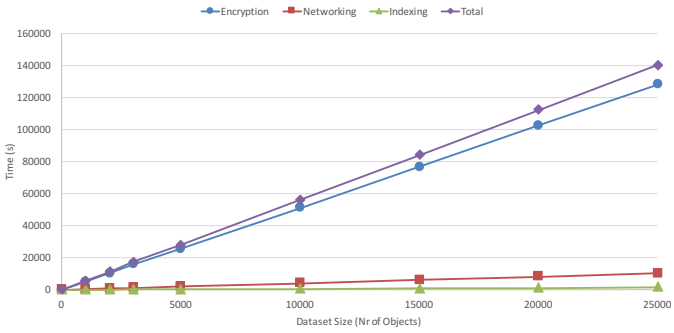

Fig. 4: Performance of PHom-MuSE in the Update operation.

before they can be efficiently indexed. We used hierarchical k-means and the Bag of Visual Words model for this [41]. For this model we trained a codebook tree with height three and leaf width ten, resulting in 10.000 clusters.

Ranking of search results in each media format was done using the TF-IDF function, as described in Figure 2. Ranked results were then merged into the final multimodal search results through rank fusion, more concretely the logarithmic Inverse Square Rank (ISR) rank-fusion algorithm [37]. Finally, we remark that our MuSE and PHom-Muse schemes display a high flexibility of deployment and configuration, meaning that the implementation described is just one possibility and our schemes can easily be implemented using other algorithms from the state of art in cryptography and information retrieval.

VI. Experimental Evaluation

In this section we perform an experimental evaluation of the performance and search precision of our MuSE and PHom-MuSE schemes, comparing them with the stare of art on encrypted multimodal search [17]. We conducted experiments as follows: client implementations were executed in a Macbook Pro with Mac OS X 10.13.1, 16GB of RAM, 500GB SSD disk, and 2.3Ghz quad-core Core i7 CPU; server implementations were deployed in the Amazon AWS cloud, using an EC2 m5.large instance. Communications were performed on a 5MB/s connection, with 49.932ms round-trip time. We used the MIR-Flickr dataset [25] as a multimodal dataset with 25000 objects composed of both image (users' photos) and text (photo tags) media formats.

A. Update Performance

In our first experiment we measured the performance of MuSE and PHom-MuSE when executing the Update operation. To this end, we performed batch updates with increasing numbers of objects, ranging from 1 to 25000 objects, and measured the performance cost of each sub-operation involved: Encryption (i.e. cryptographic computations), Networking (communication between the client and server), and Indexing (feature extraction, data-structure accesses, and other indexing related computations). For convenience, Total results for each scheme are also exhibited. Figure 3 shows the results for MuSE and Figure 4 for PHom-MuSE. Table II further compares the two schemes with MIE [17], a similar approach from the literature. Results represent an average of five independent executions.

Scheme	Encryption	Networking	Indexing	Total
MIE [17]	16.38	292.18	349.71	658.27
MuSE	80.86	295.36	347.83	724.05
Phom-MuSE	25596.22	2076.69	344.66	28017.57

TABLE II: Update performance comparison between MuSE, PHom-MuSE, and MIE [17], for a batch update of 5000 objects.

Starting with MuSE, Figure 3 shows that its Update operation can be very efficient. An update for a single object exhibits a total performance cost of around 0.18 seconds, while a batch update for 25000 objects can be performed in under 3700 seconds (a linear increase). Analysing the sub-operations involved, we can observe that MuSE's cryptographic computations (Encryption in Figure 3) are very efficient. Moreover, Indexing computations are the biggest bottleneck in MuSE's update, followed by Networking. Networking overheads can further be improved with a faster network connection. Regarding indexing computations, these include typical feature extraction and data-structure accesses that are also required in plaintext multimodal retrieval systems with no security guarantees. This means that MuSE's security properties actually add very little overhead to overall system performance.

Analysing Figure 4, however, we can conclude that PHom-MuSE has a much higher cryptographic overhead. For a batch update of 25000 objects, PHom-MuSE has a cryptographic performance cost of around 128000 seconds (35.5 hours), an increase of around 277 times in comparison with MuSE's cryptographic cost of 461 seconds. This is due to the use of partially homomorphic encryption to prevent the leakage of frequency patterns, namely the Paillier scheme. Furthermore, networking in PHom-MuSE also exhibits an increased performance cost in comparison with MuSE, due to the higher ciphertext expansion of the Paillier scheme. For an update of 250000 objects MuSE has a networking cost of around 1500 seconds, while PHom-MuSE exhibits a cost of 10383 seconds.

Table II also shows us that MuSE exhibits similar performance as the state of art on encrypted multimodal retrieval [17] (MIE in Table II). The major difference between MuSE and the competing solution MIE lies in the encryption overhead, which can be explained by the better security guarantees offered by MuSE (see Table I in Section II-A, where MuSE has leakage level L0=>L2 and MIE has level L2).

B. Search Performance

In our second experiment we measured the performance of MuSE and PHom-MuSE when executing Search operations, while also comparing with the competing alternative MIE

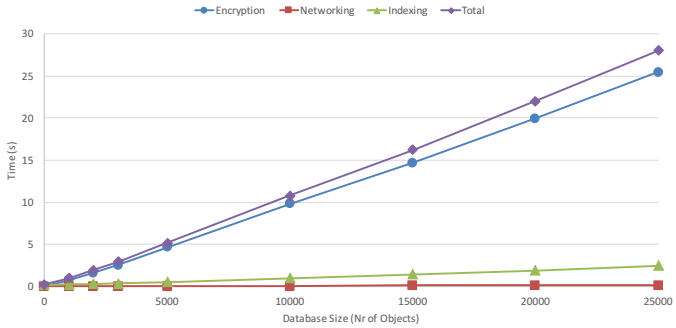

Fig. 5: Performance of MuSE in the Search operation.

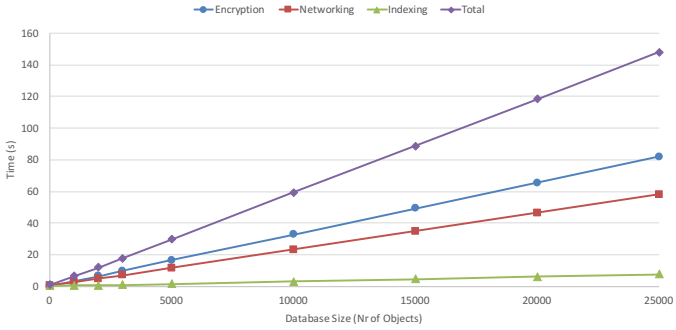

Fig. 6: Performance of PHom-MuSE in the Search operation.

[17]. To achieve this goal, we performed queries with a sample query object (chosen at random from the dataset) and measured the performance of both schemes as we scaled the dataset size, from 1 to 25000 objects. Again we show total performance costs for each scheme, as well as for each sub-operation (Encryption, Networking, and Indexing). Figure 5 shows the results for MuSE, Figure 6 for PHom-MuSE, and Table III compares the two with MIE. Results represent an average of 50 independent executions.

Comparing Total results from the two Figures, we can observe that MuSE is once again more efficient than PHom-MuSE (notice the difference in y-axis scale). For a dataset size of 25000 objects, MuSE exhibits a total search performance cost of around 28 seconds while PHom-MuSE requires around 147 seconds to complete the same operation. This is mostly due to the higher Encryption and Networking overheads of PHom-MuSE, and is observable at all dataset sizes.

Analysing sub-operations in detail, we can observe that in both schemes Encryption overhead has the highest impact on total performance cost. Moreover, this cost increases as we scale the dataset size. This is a natural observation, as increasing the dataset size means that increasingly more entries have to be accessed to resolve the same query. Indexing computations, in contrast, exhibit constant performance as we increase the dataset size in both schemes, since most overhead here comes from processing the query. The same effect can be observed for Networking performance in MuSE, however in PHom-MuSE this performance cost increases with the dataset size. This can be explained by the fact that while in MuSE the server has access to the decrypted relevance scores of the query and is able to sort them and return only the top k (in our experiments, we set k to 20), in PHom-MuSE the server only sees homomorphically encrypted scores. Since homomorphic

Scheme	Encryption	Networking	Indexing	Total
MIE [17]	0.008	0.072	0.6	0.68
MuSE	4.64	0.001	0.55	5.191
Phom-MuSE	16.38	11.63	1.56	29.57

TABLE III: Search performance comparison between MuSE, PHom-MuSE, and MIE [17]), for a query on a repository with 5000 objects.

	Plaintext	MIE	MuSE	PHom-MuSE
mAP (%)	57.938	57.562	57.965	57.881

TABLE IV: Mean Average Precision (mAP) for the Holidays dataset.

encryption does not preserve order, the server can not sort these encrypted scores and has to return all to the client.

Comparing with the competing alternative MIE [17] (Table III), we can observe that MuSE (and Phom-MuSE as well) has a higher total performance cost. As in the update operation, this is mostly due to Encryption overheads, and is the tradeoff for achieving better security guarantees. Nonetheless, since most of this overhead comes from server side computations and our prototype implementation is single-threaded, we believe it can be further reduced through parallelization of cryptographic tasks.

C. Search Precision

The final experiment we conducted assessed the search precision of our schemes, comparing it with a plaintext system without any security guarantees and with the competing alternative MIE from the literature [17]. Since the MIR-Flickr dataset, although a good choice for performance evaluation, did not contain a group of queries with relevance set that would allow us to assess precision, we used the Inria Holidays dataset [26] for this experiment. The Holidays dataset contains an online evaluation package, consisting of 500 pre-chosen queries and their expected results, allowing a transparent and independent evaluation of precision results. This is an image only dataset, showing that our schemes do not affect query precision for this media format. Nonetheless, similar results are expected for other formats and multimodal searching.

Table IV shows the results obtained, with an average of 50 independent executions. Results for the four approaches are very similar, which can be explained by the fact that both our schemes, as well as the competing alternative MIE, preserve the search precision of the indexing and searching algorithms used. Furthermore, these results fundamentally represent the precision of the indexing and searching algorithms used in our prototype implementations, meaning that they can possibly be further improved by exploring other algorithms from the state of art in information retrieval, without impacting the security guarantees of our schemes.

VII. Conclusion

In this paper we addressed the problem of multimodal searchable encryption, allowing client applications to store, update, and search their multimodal data in remote cloud servers with privacy guarantees. We started by providing a new framework, based on an empirical analysis of the literature on searchable encryption and its common leakage, that researchers and developers can use to characterize their schemes and better understand their security properties. Then we formally defined dynamic multimodal searchable encryption and

designed two schemes supporting its functionality: an efficient scheme, called MuSE, that exhibits similar performance as previous exact-match schemes for text data, but that leaks a new type of patterns which we call frequency patterns; and a less efficient scheme (although still practical) based on partially homomorphic encryption, called PHom-MuSE, that prevents the leakage of frequency patterns and provides the same security properties as previous text exact-match schemes, although greatly improving usability. We formally evaluated the security of our schemes and implemented them. Using our prototype implementations we conducted a thorough experimental evaluation of performance and search precision. Results showed that both MuSE and PHom-MuSE exhibit practical performance for real world deployments, making different tradeoffs between security and performance, and preserving the search precision of the multimodal retrieval algorithms used.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[2] P. K. Atrey, M. A. Hossain, A. El Saddik, and M. S. Kankanhalli. Multimodal fusion for multimedia analysis: A survey. *Multimedia Systems*, 16(6):345–379, 2010.

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

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

[5] M. Bellare and P. Rogaway. Random Oracles are Practical : A Paradigm for Designing Efficient Protocols. In *CCS'93*, pages 1–20. ACM, 1993.

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

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

[8] M. Brandenburger, C. Cachin, and N. Knezevic. Don't Trust the Cloud, Verify: Integrity and Consistency for Cloud Object Stores. In *SYSTOR'15*. ACM, 2015.

[9] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou. Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data. *IEEE TPDS*, 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*, 2014.

[12] A. Chen. GCreep: Google Engineer Stalked Teens, Spied on Chats. Gawker. 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] T. Cook. A Message to Our Customers. Apple. https://www.apple.com/customer-letter, 2016.

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

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

[17] B. Ferreira, J. Leitão, and H. Domingos. Multimodal Indexable Encryption for Mobile Cloud-based Applications. In *DSN'17*. IEEE, 2017.

[18] B. Ferreira, J. Leito, and H. Domingos. Muse: Multimodal searchable encryption for cloud applications. Cryptology ePrint Archive, Report 2017/661, 2017. https://eprint.iacr.org/2017/661.

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

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

[21] S. Garg, P. Mohassel, and C. Papamanthou. TWORAM: efficient oblivious RAM in two rounds with applications to searchable encryption. In *CRYPTO'16*, pages 563–592. Springer, 2016.

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

[23] G. Greenwald and E. MacAskill. NSA Prism program taps in to user data of Apple, Google and others. The Guardian, 2013.

[24] J. A. Hartigan. *Clustering algorithms*. Wiley, 1975.

[25] M. J. Huiskes and M. S. Lew. The MIR Flickr Retrieval Evaluation. In *MIR'08*, New York, NY, USA, 2008. ACM.

[26] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *ECCV'08*. Springer, 2008.

[27] S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography - FC'13*, 2013.

[28] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *CCS'12*, pages 965–976. ACM, 2012.

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

[30] B. Lau, S. P. Chung, C. Song, Y. Jang, W. Lee, and A. Boldyreva. Mimesis Aegis: A Mimicry Privacy Shield-A System's Approach to Data Privacy on Public Cloud. In *USENIX Security*, pages 33–48, 2014.

[31] 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.

[32] M. Levy and M. Sandler. Music information retrieval using social tags and audio. *IEEE Transactions on Multimedia*, 11(3):383–395, 2009.

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

[34] W. Lu, A. Swaminathan, A. L. Varna, and M. Wu. Enabling Search over Encrypted Multimedia Databases. *IS&T/SPIE Electronic Imaging*, 7254, 2009.

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

[36] M. Meeker. Internet Trends 2016. In *Code Conference*, 2016.

[37] A. Mourão, F. Martins, and J. Magalhães. NovaSearch at TREC 2013 Federated Web Search Track : Experiments with rank fusion. In *TREC'13*, 2013.

[38] A. Mourão, F. Martins, and J. Magalhães. Multimodal medical information retrieval with unsupervised rank fusion. *Computerized Medical Imaging and Graphics*, May 2014.

[39] M. Naveed. The Fallacy of Composition of Oblivious RAM and Searchable Encryption. Technical report, Cryptology ePrint Archive, Report 2015/668, 2015.

[40] M. Naveed, M. Prabhakaran, and C. A. Gunter. Dynamic Searchable Encryption via Blind Storage. In *S&P'14*. IEEE, 2014.

[41] D. Nistér, H. Stewénius, D. Nister, and H. Stewenius. Scalable recognition with a vocabulary tree. In *CVPR'06*. IEEE, 2006.

[42] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT'99*, pages 223–238. IACR, 1999.

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

[44] P. Scovanner, S. Ali, and M. Shah. A 3-dimensional sift descriptor and its application to action recognition. In *MM'07*. ACM, 2007.

[45] A. Shraer, C. Cachin, A. Cidon, I. Keidar, Y. Michalevsky, and D. Shaket. Venus: Verification for untrusted cloud storage. In *CCSW'10*. ACM, 2010.

[46] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *S&P'00*, pages 44–55. IEEE, 2000.

[47] E. Stefanov, C. Papamanthou, and E. Shi. Practical Dynamic Searchable Encryption with Small Leakage. In *NDSS'14*, 2014.

[48] E. Stefanov, M. Van Dijk, E. Shi, and et al. Path oram: An extremely simple oblivious ram protocol. In *CCS'13*, pages 299–310. ACM, 2013.

[49] C. Wang, N. Cao, K. Ren, and W. Lou. Enabling Secure and Efficient Ranked Keyword Search over Outsourced Cloud Data. *IEEE TPDS*, 23(8):1467–1479, aug 2012.

[50] Z. Xia, X. Wang, L. Zhang, Z. Qin, X. Sun, and K. Ren. A privacy-preserving and copy-deterrence content-based image retrieval scheme in cloud computing. *IEEE TIFS*, 11(11):2594–2608, 2016.

APPENDIX - PROOF OF MUSE SECURITY

In this appendix we formally prove Theorem 1. Formally, $\mathcal{L}$=($\mathcal{L}^{Stp}$, $\mathcal{L}^{Upd}$, $\mathcal{L}^{Srch}$) is a stateful party in an ideal security game, defined as follows:

Definition 2. *Let $\prod$=(Setup,Update,Search) be a dynamic multimodal SSE scheme and $\mathcal{L}$ a leakage function. For algorithms $\mathcal{A}$ and $\mathcal{S}$, define the following games:*

$\mathbf{Real}^{\prod}_{\mathcal{A}}(\lambda)$*: The game runs $K \leftarrow$ Setup() and gives $\mathcal{A}(1^\lambda)$ a timestamp t. Then $\mathcal{A}$ repeatedly invokes Update and Search protocols, picking client inputs in. The game responds by running Search or Update protocols with client input (K,in) and server input EDB (the encrypted dataset), giving the transcript to $\mathcal{A}$ (the server is deterministic so this constitutes its entire view). Eventually $\mathcal{A}$ returns a bit used as the game's output.*

$\mathbf{Ideal}^{\prod}_{\mathcal{A},\mathcal{S}}(\lambda)$*: The game runs $\mathcal{S}(\mathcal{L}())$ and gives $\mathcal{A}(1^\lambda)$ a timestamp t. Then $\mathcal{A}$ repeatedly invokes Update and Search protocols, picking client inputs in. The game responds by giving the output of $\mathcal{L}(in)$ to $\mathcal{S}$, which outputs a simulated transcript that is given to $\mathcal{A}$. Eventually $\mathcal{A}$ returns a bit used by the game.*

$\prod$ *is $\mathcal{L}$-secure against adaptive attacks if for all adversaries $\mathcal{A}$ there is a simulator $\mathcal{S}$ such that:*

$$Pr\ [\mathbf{Real}^{\prod}_{\mathcal{A}}(\lambda) = 1]\ -\ Pr\ [\mathbf{Ideal}^{\prod}_{\mathcal{A},\mathcal{S}}(\lambda) = 1] \leq negl(\lambda)$$

Amongst its state $\mathcal{L}$ keeps: a set *ID* initialized to contain all object identifiers in the dataset; and a list Q describing all operations issued so far, where an entry takes the form (*i*,*op*,...), meaning an operation counter, an operation type, and then one or more inputs to the operation.

We define sp(*w*,Q), the search pattern of feature *w* with respect to Q, to be the indices of operations that searched for *w*: sp(*w*,Q) = $\{j : (j, \text{srch}, w) \in Q\}$.

For object *id*, feature *w*, and frequency *f*, the add pattern of *id*, *w*, *f* with respect to Q corresponds to the indices that added/updated (*w*,*f*) to *id*:
ap(*w*,*id*,*f*,Q) = $\{j : (j, \text{add}, w, id, f) \in Q, w \in W_{id}\} \cup \{j : (j, \text{updt}, w, id, f) \in Q, w \in W_{id}\}$.

Finally, the add pattern of *w* with respect to Q and *ID* is the set of all ids to which *w* was ever added, along with the respective frequencies and indices showing when it was added: AP(*w*,Q,*ID*) = $\{(id, f, \text{ap}(w,id,f,Q)) : id \in ID, \text{ap}(w,id,f,Q) \neq \emptyset\}$.

Intuitively, sp captures what we previously called search leakage, while AP captures what we called access and frequency leakage. Given a set of setup, update, and search operations, $\mathcal{L}$ produces outputs as follows:

- On initial setup, $\mathcal{L}$ initiates its state with $i \leftarrow 0$, empty list Q and empty set *ID*, outputting the number of modalities m.
- For a search operation on *w*, $\mathcal{L}$ appends (*i*,srch,*w*) to Q and increments *i*, outputting sp(*w*,Q), AP(*w*,Q,*ID*), and the current size of the dataset *N*.
- For an addition/update operation (*w*,*id*,*f*), $\mathcal{L}$ appends (*i*,add/updt,*w*,*id*,*f*) to Q, adds *id* to ID, and increments *i*. It outputs sp(*w*,Q) and, if this is non-empty, it also outputs *id* and *f*.

We are now ready to prove Theorem 1.

Proof. We begin by proving correctness and security against non-adaptive attacks. Correctness follows as collisions between the outputs of PRF *F* will only happen with negligible probability. Additionally when we model *F* as a random oracle *H* (for proving adaptive security), simulator $\mathcal{S}$ can program *H* so that its outputs are truly random and hence without collisions.

Proving non-adaptive security implies showing that $\mathcal{S}$, given only the leakage output of $\mathcal{L}$, can produce the view of the server and the two are indistinguishable except for a negligible probability in λ. Setup operations are easy to simulate. Since when Setup is performed $\mathcal{L}$ only outputs a timestamp of execution and the number of modalities m, $\mathcal{S}$ can be trivially shown to have the same view as the server.

To simulate search operations, $\mathcal{S}$ iterates over the log of queries choosing keys $K1_i^A$,$K2_i^A$,$K1_i^U$,$K2_i^U \xleftarrow{\$} \{0,1\}^\lambda$ for the *i*-th query. Then, for each *id* $\in$ DB(w_i), $\mathcal{S}$ computes l^A, d^A, l^U, and d^U as specified in the real experiment (but using the keys it chose instead), adding each group of labels to a list *L*. Additionally it creates a dataset γ with *N* entries, filling it with simulated objects picked uniformly at random (if γ already existed, $\mathcal{S}$ adjusts its size with the new *N*).

To simulate update operations, $\mathcal{S}$ iterates over the log of adds/updates and decides for each group of labels (l^A,d^A,l^U,d^U) sent if it is supposed to be random (and meaningless) or if the pair should be computed with one of the keys used for search operations. It does this by using both the add pattern leakage AP from the search queries and the leakage from update operations, which includes *id* and *f* if the keyword was previously searched. Finally $\mathcal{S}$ adds the computed labels to *L* and, after processing all operations, it outputs the simulated dataset EDB = Create(γ,*L*).

A simple hybrid argument shows that the simulator's output is indistinguishable from the real server view. The first hybrid shows that selecting each $K1_i^A$,$K2_i^A$,$K1_i^U$,$K2_i^U$ at random is indistinguishable from deriving them from K^A, K^U, by the PRF security of *F*. The next hybrid shows that the labels l^A, l^U and ciphertexts d^A, d^U for un-queried features are pseudorandom, by the RCPA security of (Enc,Dec). This proves non-adaptive security.

Finally, security against adaptive attacks can be proven by having $\mathcal{S}$ program a random oracle *H* to model the behavior of PRF *F*, outputting truly random labels in response to adaptive queries. The only defects in this new simulation occur when an adversary manages to query the random oracle with a key before it is revealed, which can be shown to happen with negligible probability in λ. □