Bernardo Luís da Silva Ferreira

Mestre em Engenharia Informática

Privacy-Preserving Efficient Searchable Encryption

Dissertação para obtenção do Grau de Doutor em

Engenharia Informática

Orientador: Henrique João Lopes Domingos, Prof. Auxiliar, Universidade Nova de Lisboa

Júri

Presidente:	José Augusto Legatheaux Martins
Arguentes:	Bruno Crispo
	Miguel Nuno Dias Alves Pupo Correia
Vogais:	Paulo Esteves Verissimo
	Rodrigo Seromenho Miragaia Rodrigues
	João Carlos Antunes Leitão

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Dezembro, 2016

Privacy-Preserving Efficient Searchable Encryption

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "unlthesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/unlthesis [2] http://www.di.fct.unl.pt

To all those that contributed to the work described in these pages, and to all those that will read this in the future and find it useful.

Acknowledgements

The work presented in this thesis would not have been possible without the collaboration of a considerable number of people to whom I would like to express my gratitude.

First and foremost I would like to deeply thank my advisor, Professor Henrique Domingos, for providing all the conditions and the environment for exploring new ideas, and for helping me overcome all challenges I had to face during my PhD research. I have learned a lot from him, and truly hope to be able to advise my students as well as he advised me.

I also have to thank my co-author Professor João Leitão, for the opportunity to collaborate and learn from him. His experience as a researcher proved a valuable asset and collaborating with him allowed me to grow as a scientist.

The Computer Systems group (CS) of NOVA LINCS was a great environment to work and explore ideas. Therefore I would like thank its senior researchers for helpful discussions, including Professors José Legatheaux, Nuno Preguiça, João Lourenço, Sérgio Duarte, Hervé Paulino, Cecília Gomes, Vitor Duarte, and Paulo Lopes. I would also like to thank Professors Luís Caires, José Alferes, and João Magalhães, for the teaching of the PhD courses *Scientific and Technical Communication* and *Web and Media Search*, as well as for helpful insights and discussions. Finally, I would like to thank my present and former PhD colleagues in the Computer Systems group, for setting an example, providing motivation, and helping brainstorm ideas: Ricardo Dias, João Soares, David Navalho, Valter Balegas, Tiago Vale, Filipe Freitas, João Rodrigues, Albert van der Linde, João Silva, Cheng Li, Daniel Porto, and Nancy Estrada.

Finally, my very heartful thanks to my family for their unconditional support. To my parents Luís and Graça, and my sister Inês, for motivating and encouraging me in starting a PhD. To my partner in life, Cláudia, for always supporting me and helping me wake up early for work everyday. To her family for the excellent meals and deserts that were crucial for the writing of this thesis.

I also would like to acknowledge the following institutions for their hosting and financial support: *Departamento de Informática* and *Faculdade de Ciências e*

Tecnologia of the *Universidade NOVA de Lisboa* (DI-FCT-UNL); the *NOVA Laboratory of Computer Science and Informatics* (NOVA LINCS); *Fundação para a Ciência e Tecnologia* (FCT/MCTES) through the project SITAN (PTDC/EIA/113729); *Agência Nacional de Investigação* (ANI) and *ByItNow Inc.* through the ByPhone Project (PO/FC 38875) under the QREN funding program; and *Amazon AWS* in the context of the research grant framed by the project MyPrivateEC2.

Abstract

Data storage and computation outsourcing to third-party managed data centers, in environments such as Cloud Computing, is increasingly being adopted by individuals, organizations, and governments. However, as cloud-based outsourcing models expand to society-critical data and services, the lack of effective and independent control over security and privacy conditions in such settings presents significant challenges.

An interesting solution to these issues is to perform computations on encrypted data, directly in the outsourcing servers. Such an approach benefits from not requiring major data transfers and decryptions, increasing performance and scalability of operations. Searching operations, an important application case when cloud-backed repositories increase in number and size, are good examples where security, efficiency, and precision are relevant requisites. Yet existing proposals for searching encrypted data are still limited from multiple perspectives, including usability, query expressiveness, and client-side performance and scalability.

This thesis focuses on the design and evaluation of mechanisms for searching encrypted data with improved efficiency, scalability, and usability. There are two particular concerns addressed in the thesis: on one hand, the thesis aims at supporting multiple media formats, especially text, images, and multimodal data (i.e. data with multiple media formats simultaneously); on the other hand the thesis addresses client-side overhead, and how it can be minimized in order to support client applications executing in both high-performance desktop devices and resource-constrained mobile devices.

From the research performed to address these issues, three core contributions were developed and are presented in the thesis: (*i*) CloudCryptoSearch, a middleware system for storing and searching text documents with privacy guarantees, while supporting multiple modes of deployment (user device, local proxy, or computational cloud) and exploring different tradeoffs between security, usability,

and performance; (*ii*) a novel framework for efficiently searching encrypted images based on IES-CBIR, an Image Encryption Scheme with Content-Based Image Retrieval properties that we also propose and evaluate; (*iii*) MIE, a Multimodal Indexable Encryption distributed middleware that allows storing, sharing, and searching encrypted multimodal data while minimizing client-side overhead and supporting both desktop and mobile devices.

Keywords: Cloud Computing; Data and Computation Outsourcing; Privacy; Dependability; Encrypted Data Processing; Searchable Encryption; Multimedia Data; Information Retrieval

Resumo

A externalização do armazenamento de dados e computações para Centros de Dados geridos por terceiros, em ambientes como Computação na Nuvem, é cada vez mais adotada por indivíduos, organizações e governos. No entanto, enquanto modelos de externalização baseados na nuvem expandem até dados e serviços críticos para a sociedade, a falta de controlo efetivo e independente sobre as condições de segurança e privacidade nestes casos apresenta desafios significativos.

Uma solução interessante para estes problemas consiste em efetuar computações sobre dados cifrados, diretamente nos servidores de terceiros. Esta solução beneficia de não requerer grandes transferências e decifras de dados, aumentando a performance e escalabilidade das operações. Operações de pesquisa em particular, importantes casos de uso quando repositórios baseados na nuvem aumentam em número e tamanho, são bons exemplos onde segurança, eficiência e precisão constituem requisitos relevantes. No entanto, as soluções existentes para efetuar operações de pesquisa sobre dados cifrados ainda hoje são limitadas em diferentes aspetos, incluindo usabilidade, expressividade das pesquisas, desempenho no lado do cliente e escalabilidade.

Esta tese foca-se no desenho e avaliação de mecanismos para pesquisa de dados cifrados com eficiência, escalabilidade e usabilidade aperfeiçoados. Mais especificamente, dois problemas particularmente importantes são endereçados na tese: por um lado, a tese tem como objetivo suportar operações sobre diferentes formatos medias, em especial texto, imagens e dados multimodais (ou seja, dados que suportem diferentes formatos media simultaneamente); por outro lado, a tese endereça custos extra de desempenho no lado do cliente, e como podem ser minimizados de formar a suportar tanto dispositivos fixos de alto desempenho como dispositivos móveis de baixos recursos.

Do trabalho de investigação realizado para endereçar estes problemas, três contribuições principais foram desenvolvidas e são apresentadas na tese: (*i*) CloudCryptoSearch, um sistema middleware para armazenamento e pesquisa de documentos de texto com garantias de privacidade, suportando diferentes modos de instalação (dispositivo do utilizador, proxy local, ou nuvem computacional) e explorando diferentes equilíbrios entre requisitos de segurança, usabilidade e desempenho; (*ii*) uma framework original para pesquisa eficiente de imagens cifradas baseada em IES-CBIR, um novo esquema de cifra de imagens com propriedades de pesquisa de imagens baseada no seu conteúdo, esquema esse também proposto e avaliado na tese; (*iii*) MIE, um middleware distribuído de cifra indexável multimodal, permitindo o armazenamento, partilha e pesquisa de dados multimodais cifrados e simultaneamente minimizando os custos extra de desempenho no lado do cliente, suportando assim tanto dipositivos fixos como móveis.

Palavras-chave: Computação na Nuvem; Externalização de Dados e Computações; Privacidade; Fiabilidade; Processamento de Dados Cifrados; Dados Multimédia; Pesquisa de Informação

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AES Advanced Encryption Standard.

CBC Cipher Block Chaining (mode of encryption).

CBIR Content-Based Image Retrieval.

CTR Counter (mode of encryption).

DET Deterministic Encryption.

DPE Distance Preserving Encoding.

FHE Fully Homomorphic Encryption.

HMAC Hash-based Message Authentication Code.

Hom-MSSE Homomorphic Multimodal Searchable Symmetric Encryption.

IES-CBIR Image Encryption Scheme with Content-Based Image Retrieval properties.

IND-CCA Indistinguishability Against Chosen Ciphertext Attacks.

LSS Linear Search Scheme.

mAP Mean Average Precision.

MIE Multimodal Indexable Encryption.

MSSE Multimodal Searchable Symmetric Encryption.

OPE Order Preserving Encryption.

ORAM Oblivious Random Access Memory.

PHE Partially Homomorphic Encryption.

PKHE Public-Key partially-Homomorphic Encryption.

SSE Searchable Symmetric Encryption.

TMS Trusted Mail System.

Chapter 1

Introduction

Cloud Computing has emerged as a viable alternative for organizations and individuals to overcome the overheads of managing their own storage and computation devices, servers, and data-centers. This new paradigm offers advantageous conditions for clients, including flexibility of configuration, pay-as-you-go charging models, and geo-replicated highly-available data (Armbrust et al. 2010). By avoiding management overloads, as well as operational and software licensing costs, cloud computing allows designing improved applications from both technical and economical viewpoints. Existing cloud solutions offer data storage and computation services with recognized reliability and availability guarantees, in ubiquitous access conditions independent of geographical location.

One key driving factor for cloud services has been the growth in production and sharing of multimedia data (Fung 2015). Both in corporate and personal use cases, multimedia data (including images, video, audio, and text) is responsible for one of the largest shares of global internet traffic (Meeker 2015). Such large collections of data benefit from specialized storage and processing solutions that only cloud technology can offer. An example of success are cloud services for storage and sharing of images, which have been reported to be amongst the largest growing internet services in recent years (Meeker 2016). In the next five years online video, for instance, is expected to be responsible for 80% of Internet traffic (Fung 2015). Given such large data sets, being able to search and retrieve relevant subsets of multimedia data in useful time also comes of increased importance.

As producers of most of today's multimedia data (Meeker 2016), and already

responsible for more than 70% of multimedia consumption (ComScore 2016), mobile devices play a central role in cloud computing and storage models. For these devices, cloud computing acts as a natural extension to their limited resources, expanding both storage capacity and computational power, as well as economizing battery life. Ubiquitous access conditions of cloud services are also a major advantage for mobile devices, allowing users to search, retrieve, and access their data from anywhere and at anytime.

1.1 Motivation and Context

Despite the advantages of cloud computing, outsourcing data and computations inherently leads to new privacy challenges that must be contemplated. This is especially true when dealing with sensitive data, and is a natural concern as outsourcing data and computations also entails outsourcing control (Chow et al. 2009). Recent news have proven that user's privacy is not protected when using cloud services (Rushe 2013). Governments impose increasing pressure on technological companies to disclose users' data and build insecure backdoors (Cook 2016; Greenwald and MacAskill 2013). Malicious or simply careless cloud system administrators have been responsible for critical data disclosures (Chen 2010; Frieden 2009; Halderman and Schoen 2009). Finally, one also has to consider internet hackers, exploiting software and hardware vulnerabilities in the cloud providers' infrastructures (National Vulnerability Database 2016) and possibly accessing the private data of millions of users (Lewis 2014).

A common approach for dealing with these privacy concerns is to rely on end-to-end encryption schemes, where users' devices are responsible for encrypting all data before storing it in the cloud (Asghar et al. 2016; Bessani et al. 2013; Mahajan et al. 2011; Shraer et al. 2010). However these schemes restrict functionalities available to users, including efficient data sharing and computations through the cloud infrastructure. While data sharing can easily be achieved by resorting to key distribution services and algorithms (Boneh et al. 2005a), efficient computing on encrypted data, including search and retrieval operations, is a challenging problem with non trivial solutions.

The literature has tried to address the challenges of computing on encrypted data with novel cryptographic primitives, ranging from homomorphic encryption (Gentry 2009; Gentry et al. 2012) to oblivious RAM (Goldreich and Ostrovsky 1996; Stefanov et al. 2013). However such cryptographic mechanisms, allowing

generic computations on encrypted data, are still too expensive for practical adoption. For the specific problem of searching encrypted data, a vast literature on more efficient algorithms has been published in the last years, under a research area known as Searchable Encryption (Song et al. 2000). Research on searchable encryption can be found on both asymmetric (Bellare et al. 2007) and symmetric (Curtmola et al. 2006) cryptographic models and key settings. However it is the Searchable Symmetric Encryption (SSE) field that is of particular interest, especially in the context of this thesis, due to its improved practical performance.

Originally designed for text documents, SSE schemes (Baldimtsi and Ohrimenko 2015; Cash et al. 2014; Curtmola et al. 2006; Hahn and Kerschbaum 2014; Kamara and Papamanthou 2013; Kamara et al. 2012; Kuzu et al. 2012; Naveed et al. 2014; Popa et al. 2014; Song et al. 2000; Stefanov et al. 2014) allow searching encrypted data in sub-linear time. In these schemes, clients index their data before storing it in the cloud (i.e. build a compact dictionary of the data with, for instance, the unique keywords in each text document). Encrypted index and data are then uploaded to the cloud for storage, allowing search operations to be remotely performed in the encrypted domain.

However indexing computations of SSE schemes are still too expensive for wide adoption, especially when client applications are to be executed in resource-constrained and mobile devices. This is particularly evident in multimedia data, including not only text documents but also richer media domains such as images, audio, and video, as indexing computations are even more complex and automated machine learning tasks (also known as training procedures) have to be performed before data can be indexed efficiently (Datta et al. 2008). Supporting dynamic multimedia datasets (where data can be added, updated, and removed at any time (Kamara et al. 2012)) and multiple user scenarios are also open research problems.

Although SSE schemes reveal no information to adversaries at initialization time, searching with sublinear search performance is only possible by leaking some information patterns with each query (Curtmola et al. 2006). These patterns reveal if the query has been performed before (known as *Search Pattern* and leaked by a deterministic hash of the query) and which documents are returned by each query (known as *Access Pattern* and leaked by deterministic identifiers of the documents). The first SSE schemes for dynamic collections of text documents (Kamara et al. 2012; Naveed et al. 2014) also revealed *Update Patterns*, i.e. if new/updated documents shared contents with other stored documents (leaked by deterministic hashes of the document's keywords). Finally, SSE schemes for richer queries and media types (Cao et al. 2014; Kuzu et al. 2012; Lu et al. 2009;

Wang et al. 2012; Weng et al. 2015; Yuan et al. 2014) also revealed *Frequency Patterns*, i.e. how many times each keyword (or the equivalent in other medias) appears in each document. Leaking these patterns to adversaries seems an undesirable property, and their implications are still being actively studied in the research community (Cash et al. 2015; Islam et al. 2012; Zhang et al. 2016). Nonetheless, the state of art has proven them to be unavoidable leakage in order to search encrypted data in sub-linear time (Naveed 2015). Considering these tradeoffs, an open question remains:

If revealing information patterns when performing operations is unavoidable, what are the benefits of protecting them at initialization time and at what cost does this protection come?

1.2 Problem Statement

In this PhD thesis we aim at answering the following question:

Can we improve the performance, scalability, and resource management of both mobile and desktop devices storing, sharing, and searching multimedia data in the cloud with strong security guarantees?

Answering this question involved understanding the previous open issue and its implications in the state of art SSE schemes. In truth, there is little benefit in protecting information patterns at initialization time, as their leakage is unavoidable especially in many real-world scenarios with multiple queries executed in concurrence. However this initial protection, which appears as a core design issue in the SSE literature, comes at a high cost, namely requiring users to index and train multimedia data in their local devices. In most cases, this high cost invalidates the applicability of existing SSE schemes for resource-constrained client devices.

By further studying these implications, we started a new research vector on cloud privacy and Searchable Symmetric Encryption (SSE). This research vector led to the development of new models and techniques with the objective of outsourcing to cloud servers the heaviest computations required by SSE schemes in a secure way. Outsourced computations include a combination of the following: feature extraction (i.e. calculating some form of descriptors for multimedia data, such as histograms of keyword frequencies in text documents); training tasks (i.e.

automated machine learning operations used to find homogeneous groups of objects in high-dimensional descriptors and build more compact representations of them); and indexing computations (i.e. building dictionary structures to improve query performance for some pre-extracted features). Our only requirement in terms of trusted computing base is to perform cryptographic operations in the clients' devices.

As a complementary research vector, we combine the privacy guarantees achieved in the first vector with other dependability criteria, including reliability and availability. This leads us to another question that is addressed in the thesis:

Can we complement the models and mechanisms developed in the first research vector with reliability and availability guarantees in a synergetic way, while efficiently supporting both mobile and desktop devices and their operations?

To answer this question, we explore state of art solutions for cloud dependability principles, including resilience through state-machine replication using multiple clouds (Verissimo et al. 2012), data fragmentation and erasure coding (Rodrigues and Liskov 2005), and cryptographic mechanisms for secret sharing (Shamir 1979) and threshold signatures (Shoup 2000). This approach allows us to provides a complete and integrated solution for cloud-backed applications with both privacy and dependability guarantees.

1.3 Main Contributions Summary

The main research vector of the thesis on cloud privacy materialized in the following three contributions:

- Encrypted Multi-Keyword Ranked Text Searching;
- Privacy-Preserving Content-Based Image Retrieval;
- Multimodal Indexable Encryption for Mobile Cloud-based Applications.

In the following we summarize each main contribution.

1.3.1 Encrypted Multi-Keyword Ranked Text Searching

The idea of outsourcing indexing computations in a secure way is proposed for the first time (Ferreira and Domingos 2012a; Ferreira and Domingos 2012b; Ferreira and Domingos 2013a; Ferreira and Domingos 2013b), in the context of a user

managing and searching text documents in the cloud. A novel framework implementing this new approach is presented, entitled CloudCryptoSearch, where queries can support multiple keywords simultaneously and search results are ranked by a relevance score. The framework explores different tradeoffs between security, usability, and performance, considering different deployment scenarios: (*i*) deployment in the user's trusted device, where performance and security are crucial (in this scenario only encrypted documents are stored in the cloud, i.e. index data and computations never leave the user's device); (*ii*) deployment in a non-trustable LAN proxy, when the user's device has lower storage capacity and latency of operations may be an issue (in this case the user's device indexes documents and encrypts the resulting index, storing it in the proxy); (*iii*) deployment as a non-trustable cloud service, where the user has very few computational resources and outsources both data storage and indexing computations. To support the last scenario, a new cryptographic scheme is proposed for text documents, which allows indexing and searching by the cloud with privacy guarantees. Additionally, to mitigate the problem of external hackers (which may not have continuous access to data like the cloud provider but may gain access to it for a limited time), in this last scenario we can optionally have the index encrypted by the cloud server with a partially-homomorphic scheme and execute search operations in the encrypted domain.

1.3.2 Privacy-Preserving Content-Based Image Retrieval

Moving towards the goal of supporting multimedia data, a new solution is proposed (Ferreira et al. 2014; Ferreira et al. 2015d; Ferreira et al. 2016b) for multiple users storing, sharing, and searching images in the cloud with privacy guarantees. The solution is based on a new encryption scheme specifically designed for images, called IES-CBIR: an Image Encryption Scheme with Content-Based Image Retrieval properties. The scheme follows the observation that in image processing, distinct feature types can be separated and encrypted with different cryptographic algorithms. In more detail, IES-CBIR separates color from texture information when encrypting images, and uses cryptographic primitives with different properties in the encryption of each. Pixel color values are encrypted through deterministic cryptography, hiding absolute values while preserving statistical correlations. Texture information is completely protected by rearranging pixel positions and transforming images into jigsaw puzzles through probabilistic encryption. This allows feature-extraction, training, and indexing computations, based on color features, to be outsourced to the cloud in a privacy-preserving

way. Formal analysis proves the security properties of the solution, while implementation and experimental comparison with the state of art reveals improved performance and scalability, with comparable retrieval precision.

1.3.3 Multimodal Indexable Encryption for Mobile Cloud-based Applications

The third main contribution of the thesis (Ferreira et al. 2015a; Ferreira et al. 2015b; Ferreira et al. 2015c; Ferreira et al. 2016a) is a secure framework for mobile applications storing, sharing, and searching multimodal data (i.e. data with multiple media formats simultaneously, possibly including text, image, audio, and video) in the cloud. The design of this framework is particularly relevant for supporting mobile and resource-constrained devices. The framework is called MIE - Multimodal Indexable Encryption, as it outsources indexing and training computations of multimodal data to the cloud with privacy guarantees. To support the framework's operations, a new family of encodings algorithms is proposed called DPE: Distance Preserving Encodings. These encodings preserve a controllable distance function between plaintexts, meaning that upon instantiation a security threshold is defined and distances are only preserved by the encoding if they are bellow the threshold's value. Two different implementations of DPE are designed, one suitable for dense media types including images, audio, and video, and another for sparse media types such as text. The framework was implemented and experimentally analysed on desktop and mobile (Android OS) deployments. As baseline comparison, we also implemented a recent Searchable Symmetric Encryption scheme from the literature (Cash et al. 2014), extending it to support multimodal data and queries (a feature not supported in its original design). Experimental results demonstrate MIE's improved performance, increased scalability, and optimized management of mobile resources (including battery life) when compared with the state of art.

1.4 Main Results Summary

Considering the contributions listed above, the main results present in the thesis are the following:

- Implementation of the CloudCryptoSearch middleware system and its evaluation through a combined use of real world datasets (English, Spanish, and

Portuguese Wikipedia dumps) and a prototype deployment over the Amazon EC2 and S3 cloud services. This prototype is open source and available at: `https://github.com/bernymac/CloudCryptoSearch`.

- Formal evaluation and proof of the security properties of the IES-CBIR encryption algorithm and the framework leveraging it for image storage, sharing, and retrieval. Prototype implementation of both IES-CBIR and the distributed framework. Experimental evaluation of the implemented prototypes through use of real world image datasets, deployment in Amazon EC2 cloud servers, and simulation of practical image storage and retrieval scenarios with multiple users. The implemented prototypes are open source and available at: `https://github.com/bernymac/IES-CBIR`.

- Formal evaluation of the security properties of the MIE distributed middleware, as well as prototype implementation and experimental evaluation through public commercial cloud deployment (Amazon EC2) and simulation of practical scenarios with real world multimodal datasets (Flickr photographs and respective user defined tags). The software prototype of MIE is open source and available at: `https://github.com/bernymac/MIE`.

1.5 Ramifications and Complementary Contributions

The complementary research question of the thesis, as stated in Section 1.2, led to a research line combining our main contributions and their novel foundations with reliability and availability criteria. In this research vector, three complementary contributions were achieved. These contributions are summarized next. For the sake of conciseness we discuss the first in more detail in Chapter 6, as a relevant and representative complementary contribution example.

- Trusted Cloud Storage for Email Repositories

  - A trusted and dependable framework for storing email repositories in untrusted public clouds is proposed (Rodrigues et al. 2013a; Rodrigues et al. 2013b). Named TMS (Trusted Mail System), the framework provides availability, integrity, and privacy guarantees, by exploring a cloud-of-clouds architecture complemented with threshold signatures and secret sharing.

  - Email searching with privacy guarantees is supported, based on exact-match queries on email header fields and ranked multi-keyword queries on email contents. This is achieved by further exploring the first main contribution of the thesis on encrypted text searching and studying its application to email data.
  - A prototype of TMS was developed, integrating multiple commercial storage clouds publicly available. Experimental evaluation showcases the good performance and scalability conditions offered by TMS, demonstrating that the impact introduced by the TMS middleware processing is modest and clearly compensates the additional dependability guarantees provided. The developed prototype is available at: `http://asc.di.fct.unl.pt/~bf/TMS.zip`.

- Oblivious Cloud-Based Collaborative Document Edition
  - A system is proposed for anonymous and oblivious collaborative edition of text documents (Rodrigues et al. 2014). The system provides read and write operations, indistinguishable from each other and from empty operations, and obfuscates document access positions.
  - The system was implemented and a software prototype was developed in the Java language. Using this prototype, the performance of the system was evaluated and compared to publicly available cloud-based collaborative document editing solutions. Obtained results showed that although the performance costs for completely obfuscating operations and access patterns can be high, practical and usable scenarios can still be designed through the proposed solution, including online collaborative document edition applications. The prototype is available at: `http://asc.di.fct.unl.pt/~bf/ObliviEdit.zip`.

- Multimodal Searchable Encryption for Dependable Multi-Cloud Storage
  - A solution is proposed exploring the synergy between design principles of dependable multi-cloud storage architectures and multimodal searchable encryption. In this contribution we reuse our core contributions in Multimodal Indexable Encryption (MIE), extended to the design of a searchable and dependable multi-cloud storage environment. In the proposed system model, each storage cloud replicates encrypted searchable fragments using conventional cloud storage services or in-memory cloud stores.

- The solution provides the necessary support for multimodal on-line searching operations over fragmented documents, with the fragments replicated and maintained always encrypted in a multi-cloud environment.
- We have implemented a software prototype and used it for experimental evaluation and validation. The obtained results show that the solution offers dependability properties with enhanced privacy guarantees, preserves precision and recall metrics of the original plaintext retrieval algorithms, and provides the efficiency and performance levels expected from state of art multi-cloud storage solutions. The prototype is available at: `https://github.com/khasm/seasky`.

1.6 Thesis Structure

The remaining of the thesis is organized in the following structure:

Chapter 2: introduces fundamental concepts necessary to clearly understand the following Chapters. It also presents the state of art techniques and tools related to the matters addressed by the thesis.

Chapter 3: presents and evaluates CloudCryptoSearch, the main contribution of the thesis on searchable encryption for text data.

Chapter 4: analyses the problem of how to search encrypted visual data and presents IES-CBIR and the results achieved in this second main contribution of the thesis.

Chapter 5: presents and discusses MIE and the results of the third core contribution of the thesis on multimodal searchable encryption for mobile and resource-constrained devices.

Chapter 6: presents TMS, a representative and relevant complementary contribution of the thesis that combines, in a synergetic way, our core contributions in privacy guarantees with reliability and availability conditions, for the specific application case of dependable cloud email repositories.

Chapter 7: concludes the thesis summarizing the results achieved and discussing several pointers for future work.

Chapter 2

Research Context

The thesis addresses new foundations, techniques, and mechanisms for the storage, sharing, and search of different multimedia data formats in the cloud. These mechanisms provide privacy guarantees and can be further combined with other techniques for dependability criteria. In this Chapter we present an overview of the fundamental concepts related to the thesis and discuss the state-of-art on its research field.

This Chapter is organized as follows: Section 2.1 discusses relevant cryptographic mechanisms for performing computations on encrypted data, with different guarantees in terms of performance, scalability, and security; Section 2.2 presents different information retrieval techniques used in the plaintext domain to efficiently search and retrieve relevant subsets of data in repositories of different media formats; the state of art in searching encrypted data is analysed in Section 2.3, with emphasis on different usability/performance tradeoffs achieved so far in the literature; and finally, we discuss mechanisms for reliability and availability of cloud services in Section 2.4.

2.1 Computing on Encrypted Data

Performing arbitrary computations on encrypted data has for long been considered the holy grail of modern cryptography (Ostrovsky 1990; Rivest et al. 1978b). With the rise of cloud computing and its related security concerns, computing on encrypted data has gained even greater interest. This has led to important advances in the field, especially in recent years (Cash et al. 2014; Gentry 2009;

Popa et al. 2013; Stefanov et al. 2013).

Mechanisms researched in this topic can be broadly divided by range of application and complexity. In a general manner, the broader the range of application of such a mechanism, the higher its computational and space complexity. In this sense, we can group the mechanisms discussed in this Section in two categories: mechanisms for general computations on encrypted data, which include Functional Encryption (Boneh et al. 2011; Goldwasser et al. 2013), Fully Homomorphic Encryption (Gentry 2009) and Oblivious RAM (Stefanov et al. 2013); and mechanisms for specific computations, including Partially Homomorphic Encryption (ElGamal 1984; Paillier and Pointcheval 1999), Attribute-Based Encryption (Goyal et al. 2006), Property-Preserving Encryption (Bellare et al. 2007; Boldyreva et al. 2009), and Searchable Encryption (Curtmola et al. 2006; Song et al. 2000).

2.1.1 Functional Encryption

Functional Encryption (Boneh et al. 2011; Goldwasser et al. 2013) stands as one of the most generic concepts inside the field of Computing on Encrypted Data, and can be seen as a generalization for most of the other cryptographic primitives, including Homomorphic and Attribute-Based Encryption. In summary, Functional Encryption is a public-key cryptographic paradigm with a public key, a secret master key, and multiple secret keys derived from the master key, where each allows its holder to learn a different function of the ciphertext.

So far, despite its generality and broad range of application, Functional Encryption still stands as a mostly theoretical concept. Few concrete instantiations have been found with adequate security and practicality, and these have been limited to specific computations (like inner-product predicates (Shen et al. 2009)). For achieving general functions, Functional Encryption has to be limited to single-key deployments (Goldwasser et al. 2013), effectively hindering its generality properties.

2.1.2 Homomorphic Encryption

Related in concept with Functional Encryption, Homomorphic Encryption also allows generic computations on encrypted data. More concretely, an encryption scheme is said to have homomorphic properties if it allows operations on the plaintext to be performed through its ciphertext, without requiring the respective decryption key. This is an apparently contradictory aspect regarding the security

properties of these schemes, meaning that an homomorphic encryption scheme must be able to combine the homomorphic properties required while preserving the expected security guarantees.

Fully Homomorphic Encryption Introduced for the first time in 1978 by Rivest et al. (Rivest et al. 1978b), shortly after the publication of RSA (Rivest et al. 1978a), Fully Homomorphic Encryption (FHE) allows arbitrary computations on encrypted data. Although some partial results were achieved in the meanwhile (Boneh et al. 2005b; ElGamal 1984; Goldwasser and Micali 1982; Paillier and Pointcheval 1999), only in 2009 the first plausible construction of a full encryption scheme was proposed by Gentry (Gentry 2009).

Gentry's FHE scheme is based on his bootstrapping theorem. This theorem states that given a somewhat homomorphic encryption scheme (SWHE), capable of evaluating low-degree polynomials homomorphically, one can transform it into a FHE scheme through a bootstrapping procedure (Gentry 2009). Gentry's original construction of a SWHE was based on the (worst-case, quantum) hardness of problems on ideal lattices (Lyubashevsky et al. 2013). In these schemes, the ciphertext is usually noisy, with a noise that grows exponentially with each homomorphic operation executed. When applied on the ideal lattices based scheme, the bootstrapping procedure resulted in reduced noise, thus leveling the SWHE into a working FHE. The bootstrapping step was achieved by running the decryption function on the ciphertext homomorphically. For this procedure to work, the SWHE needed to be able to evaluate its own decryption function. However, since this isn't possible in SWHE schemes based on ideal lattices, a final squashing step was required that transformed the scheme into one with the same homomorphic capacity but with a decryption function that was simple enough to allow boot-strapping. This step was considered the main caveat of Gentry's work, as it required an additional very strong hardness assumption, namely the hardness of the (average-case) sparse subset-sum problem (Gentry 2009).

Following Gentry's breakthrough work, an increased interest in FHE lead to additional research trying to improve on the performance of cryptographic homomorphic operations. Most of these works (Coron et al. 2011; Gentry and Halevi 2011; Smart and Vercauteren 2010) followed Gentry's original blueprint, based on ideal lattices, thus advancing little in its performance issues. Nonetheless, recent deviations (Brakerski and Vaikuntanathan 2014; Brakerski et al. 2012) have shown that it is possible to remove the squashing and bootstrapping steps

and base the SWHE on more general, well-known problems (such as general lattices (Brakerski et al. 2012) and the Learning With Errors (LWE) problem (Brakerski and Vaikuntanathan 2014)), resulting in better performance and in the looseness of some of the hard assumptions.

As the performance of FHE schemes improves, so does its interest and the hope of truly practical FHE schemes. However, despite the most recent advances, FHE constructions are still far away from the practical requirements of online cloud-based applications. As an example, a recent FHE scheme (Gentry et al. 2012) capable of evaluating AES-128 encryption circuits was implemented and published. However, the scheme is at least 10^9 times slower than the standard AES circuit and the presented tests required a machine with 256 GB of RAM (Popa et al. 2012).

Partially Homomorphic Encryption Schemes that allow a single operation (or a group of operations) to be performed over encrypted data are known as Partially Homomorphic Encryption (PHE) schemes (ElGamal 1984; Goldwasser and Micali 1982; Paillier and Pointcheval 1999). Compared to Fully Homomorphic Encryption, PHE schemes are more efficient an can be used in some practical scenarios. This is due to their properties usually being based on conventional cryptographic primitives, in particular public-key cryptography and modular arithmetic. The first cryptographic scheme to display partially homomorphic properties was basic (or unpadded) RSA (Rivest et al. 1978a). Basic RSA is multiplicatively homomorphic, i.e. it allows multiplications of the plaintexts through their ciphertexts (more specifically, through modular multiplication of the ciphertexts). However basic RSA also has another property, which is determinism, i.e. the encryption of a plaintext X will always yield the same ciphertext Y. In public-key cryptography, deterministic encryption is particularly troublesome as adversaries have access to the encryption key (i.e. the public-key) and can trivially perform Chosen Plaintext Attacks and try to reveal encrypted contents through dictionary attacks .

After RSA, Goldwasser and Micalli (Goldwasser and Micali 1982) published a public-key scheme that also displayed homomorphic properties. This scheme encrypted data at the granularity of bits, and allowed performing exclusive ORs (XORs) between plaintexts through their ciphertexts. Furthermore this scheme was the first public-key scheme to be proven secure under the standard cryptographic model (where adversaries are only limited by the amount of time and computational power available (Katz and Lindell 2007)) and is secure under Chosen-Plaintext attacks (its encryption algorithm is probabilistic, instead of deterministic). The downside of this scheme however is its ciphertext expansion,

which is much higher than in other public-key algorithms (increasing plaintext size by a factor of 1024 or 2048 bits, i.e. the size of the cryptographic key).

Proposed in 1985, the public-key scheme by ElGamal (ElGamal 1984), like basic RSA, has the property of being multiplicatively homomorphic. However it offers stronger security guarantees, as its encryption algorithm is probabilistic and it resists against Chosen-Plaintext Attacks. In privacy-preserving tools for data analytics, where multiplications in the encrypted domain are an essential step, this scheme offers the necessary functionality with acceptable performance (Rane and Boufounos 2013).

The last scheme that should be referred in this Section was proposed by Paillier in 1999 (Paillier and Pointcheval 1999). Along with ElGamal, the Paillier scheme is one the most widely used public-key schemes with partially homomorphic properties. The scheme is additively homomorphic, meaning that it allows additions between plaintexts through their ciphertexts. Since multiplications can also be expressed as additions (e.g. $3 * 2 = 3 + 3$), it is also possible to multiply Paillier ciphertexts with public (non-encrypted) values. Furthermore, this scheme also has the desirable security guarantee of resisting against chosen plaintext attacks, as its encryption is probabilistic.

2.1.3 Oblivious RAM and Storage

Proposed in 1996 by Goldreich and Ostrovsky (Goldreich 1987; Goldreich and Ostrovsky 1996; Ostrovsky 1990), Oblivious RAM (ORAM) aims at providing a fully secure storage environment in untrusted remote servers (such as the cloud). The motivation behind ORAM is that encryption alone is not enough to protect privacy, as data accesses and related patterns may be used in statistical attacks and disclose sensitive information. As such, ORAM schemes try to conceal a user's access patterns by continuously shuffling and re-encrypting data as it is accessed.

Since its proposal, the research community has struggled to find a practical implementation of ORAM. Nonetheless, important advances have been achieved recently (Apon et al. 2014; Dautrich et al. 2014; Devadas et al. 2016; Mayberry et al. 2014; Stefanov and Shi 2013a; Stefanov and Shi 2013b; Stefanov et al. 2013). These advances were made possible, in some part, by the growth of the cloud computing paradigm and by considering not only a remote server that can store data, but also one that can perform computations. Stefanov et al. (Stefanov and Shi 2013a), for instance, were able to reduce client-server communication overhead

by considering a second non-colluding cloud and transforming part of that overhead to inter-cloud communication. Another approach is complementing ORAM with Homomorphic Encryption (Fully (Apon et al. 2014) or Partial (Devadas et al. 2016; Mayberry et al. 2014)) in order to reduce client-server communication overhead while maintaining the same security guarantees.

Since generic Oblivious RAM can support arbitrary access patterns, it is powerful enough for oblivious simulation of any program. Nonetheless despite the latest advances, bandwidth and storage overheads are still high, especially when applications require small block-sizes. Wang et al. (Wang et al. 2014) proposed oblivious data structures to more efficiently support applications with smaller block sizes and sparse access patterns. However when applied to practical problems, even these techniques are still far from practical. An example is the application of oblivious data structures in searching encrypted data (Naveed 2015), which has higher overhead than downloading the entire encrypted database with each search.

Other techniques similar to ORAM have also been proposed, improving efficiency at the expense of revealing some information patterns deemed public. An example is blind storage (Naveed et al. 2014), which allows a client to remotely store a set of files while hiding their contents, number, and individual sizes, but revealing their access patterns. Revealing such patterns may seem an unwanted property, however in many cases it is necessary in order to perform more complex computations with good security and performance guarantees (Naveed 2015).

2.1.4 Attribute-Based Encryption

Attribute-Based Encryption (Goyal et al. 2006) is a public-key cryptographic paradigm where data can only be decrypted when a set of particular conditions is met. These conditions are defined at key generation time, by parameterization of secret keys. Properties defined for decryption criteria can be, for instance, the attributes of a particular virtual machine (meaning data can only be decrypted inside that machine (Santos et al. 2012)) or of a specific user (also known as Identity-Based Encryption (Boneh and Franklin 2001)).

We include Attribute-Based Encryption as a part of the research field on Computing on Encrypted Data, since it allows specifying a particular computation (decryption) to only be possible in certain conditions. Attribute-Based Encryption can also be seen as a special case of Functional Encryption, where the function made possible by the cryptographic scheme is decryption of specific messages.

However, despite its large spectrum of interesting applications, including multi-user cloud storage and data sharing (Kamara and Lauter 2010), performance issues have to be considered when deploying Attribute-Based Encryption, as not to make existing applications unpractical.

2.1.5 Property-Preserving Encryption

Although Homomorphic Encryption allows performing arithmetic computations on encrypted data, other types of computations may also be required by applications. Prominent examples are determining equality and order relations between plaintexts through their ciphertexts.

Deterministic Encryption Cryptographic schemes that allow equality testing after encryption are called deterministic schemes (DET). Determinism is usually an unwanted property in terms of security, as it allows performing statistical attacks. However it has important applications in many domains, from relational databases (Popa et al. 2011) to searching encrypted data (Curtmola et al. 2006). The secret to a secure employment of deterministic encryption lies in analyzing each particular context of application.

The perfect domain of application for deterministic encryption is one with high entropy. If plaintext values never repeat, deterministic encryption is as secure as a probabilistic encryption scheme (Bellare et al. 2007). In domains where that is not the case, one can limit the adverse effects of determinism through complementary techniques. In Searchable Symmetric Encryption schemes (Curtmola et al. 2006), for instance, deterministic encryption is used in the protection of index keys, which are not repeated in the index. Encrypted index entries are thus indistinguishable between each other and deterministic ciphertexts are only sent again to the server when queries are performed. Hence information leakage is confined to search operations and the queried keywords.

Order-Preserving Encryption Another interesting form of property preserving encryption is OPE - Order Preserving Encryption. First proposed in the database community (Agrawal et al. 2004), OPE is very useful when performing computations on domains with natural order relations. Examples are numeric columns in relational databases and range queries.

The challenge when proposing an OPE scheme is how to reveal order relations without revealing plaintext values themselves. After its proposal, OPE quickly

caught the interest of the cryptographic community (Boldyreva et al. 2009). However the first OPE scheme with rigorous security treatment revealed more than order relations: it leaked at least half of the plaintext bits (Boldyreva et al. 2011). Subsequent proposals were able to achieve ideal security, but at the cost of extra space and time overheads (Kerschbaum and Schröpfer 2014; Popa et al. 2013).

Even if a scheme only reveals order relations between plaintexts, this leakage can still have important implications. The more plaintexts that are encrypted, the more impactful this leakage is. An extreme case is when all distinct plaintexts in an application domain are encrypted, as an adversary can then simply build a one to one mapping of plaintexts through their sorted ciphertexts. Kerschbaum (Kerschbaum 2015) recently proposed a frequency-hiding OPE, which preserves order relations but not equality. However this approach requires additional client storage and may introduce some error when performing queries.

2.1.6 Summary

Cryptographic mechanisms like Oblivious RAM (Devadas et al. 2016) and Fully Homomorphic Encryption (Brakerski and Vaikuntanathan 2014) allow performing arbitrary computations on encrypted data with good security guarantees. However, despite recent advances these mechanisms still remain largely unpractical, in terms of performance (for Fully Homomorphic Encryption (Popa et al. 2012)) and bandwidth overhead (for Oblivious RAM (Naveed 2015)). Achieving better performance is possible, either by limiting functionality as in Partially Homomorphic Encryption (Paillier and Pointcheval 1999), or by revealing some information patterns to adversaries as in Deterministic (Popa et al. 2011) and Order Preserving Encryption (Kerschbaum 2015). The implication of revealing these patterns depends on the application domain and, in many cases, it is not yet fully understood. The conclusion taken from this Section of the research context is that performing a thorough analysis of the security, usability, and performance requirements of a cloud-backed application is a very important step in its design, as multiple primitives can be used to support its operations and the same primitives can have different implications in different domains of application.

2.2 Information Retrieval

Information retrieval techniques allow us to retrieve relevant subsets of data from repositories of different media formats, by specifying some form of query. In this Section we focus on the state-of-art in information retrieval for plaintext domains.

How can these techniques be performed in encrypted domains is studied in Section 2.3, as well as in the contributions of the thesis.

2.2.1 Searching Text Documents

Queries in text retrieval can have varying length. Typical queries will contain from one to ten keywords. However, even larger queries can be specified and querying by example can also be supported (i.e. when the query is a text document itself, containing many keywords).

When a query is specified, different metrics can be used to answer it and retrieve relevant documents from a repository. The most basic metric is keyword (in)existence in documents (also known as exact-match searching), where all documents containing a queried keyword are returned. If multiple keywords are issued, results can be disjunctive (i.e. only documents containing all query keywords are returned) or conjunctive (i.e. all documents containing at least one of the keywords).

Relevance Metrics and Scoring Functions Since exact-match searching can result in large query results without any relevant ordering, other statistics are usually interesting to use. Such metrics may be document specific, including keyword frequency (i.e. how many times a query keyword appears in a document) and document length (i.e. how many keywords a document has). Nonetheless other interesting statistics may also be employed, including repository-wide metrics such as the total number of documents, average document length, and document frequency (i.e. in how many documents does a query keyword appear in).

When a query is processed, relevance metrics can be combined in different ways under scoring functions. One of the most simple and widely used scoring function is called TF-IDF (Sparck Jones 1972), which combines keyword and document frequencies in the following way:

$$TF-IDF(q,d) = \sum_{t \in q} tf_t^d \times log_{10} \frac{N}{df_t} \tag{2.1}$$

where t is a keyword of query q and d is a document, tf_t^d is the frequency of t in d, N is the number of documents in the repository, and df_t is the document frequency of t.

Another relevant example of a scoring function, which is also widely used and combines even more relevance metrics for higher retrieval precision, is BM25 (Jones

et al. 2000):

$$BM25(q,d) = \sum_{t \in q} log_{10}(\frac{N}{df_t}) \times \frac{(k_1+1) \times tf_t^d}{k_1 \times ((1-b) + b \times (L_d/L_{avg})) + tf_t^d} \tag{2.2}$$

where L_d is the length of document d, L_{avg} is average length of documents in the repository, and both b and k_1 are tunable parameters (usually with values 1.2 and 0.75 respectively). Many more scoring functions exist in the literature, slightly modifying or extending TF-IDF and BM25, but also considering completely different approaches. For further reference we point to Manning et al. (Manning et al. 2009), which provides an interesting and complete study on the topic.

Indexing Techniques for Text Documents The relevance metrics discussed so far are combined when a query is performed. However, if they are only extracted at query time, query performance will be at best linear with the number of documents and their sizes. Thus improving query performance to sub-linear levels is of utmost importance for usability's sake. Indexing solves this issue, by extracting relevance metrics from documents when they are created/updated and storing them in specialized indexing structures, which can be efficiently accessed at query time.

Indexing of text data can be performed in different ways, exploring tradeoffs between memory requirements and indexing time. In small to medium repositories, text data can typically be indexed by only resorting to main memory, speeding up indexing time. In large repositories however, secondary memory is required as intermediate indexing results will grow too large. When using secondary memory, indexing algorithms have to be adapted as secondary memory access times (i.e. access to hard disks or solid state drives) are always slower than in primary memory. The Single-Pass In Memory Indexing (SPIMI) (Heinz and Zobel 2003) algorithm is a good example, which keeps processing data while there is space in main memory, writing partial indexing structures to disk when memory is exhausted, and merging index partitions in the end.

Once extracted, relevance metrics are stored in specialized indexing structures. One of the most popular index type for textual data is the inverted list index (Zobel and Moffat 2006), which is a dictionary-like structure that maps keywords to the documents that contain them. This basic structure allows exact-match querying with sub-linear performance, and can be further augmented by, for instance, storing keyword frequencies along with the documents' ids. To support ranked queries with complex scoring functions (e.g. BM25), other necessary metrics can usually be inferred from these structures. For instance, document frequencies are

given by the size of the respective index entry lists. Nonetheless, some auxiliary structures may still be needed, such as a dictionary storing document lengths.

In large repositories even final indexing structures, which are usually a fraction of the repository size, may be too large for storage in main memory. Different techniques can be used to improve scalability in these cases (Manning et al. 2009). From the state-of-art we highlight a techniques called Champion Posting Lists (Brin and Page 1998), which consists in storing the full index in persistent storage and only keeping in main memory the top ranking documents for each keyword, with scoring functions already calculated and sorted. With this technique, updates have to be stored in an auxiliary index and queries have to access both indexing structures, with periodic merging of indexing structures for efficiency issues.

2.2.2 Content-Based Image Retrieval

Image retrieval can be performed by annotating images with textual references and then resorting to text retrieval techniques for accessing similarity (Jeon et al. 2003). This is usually called annotation-based image retrieval. A more interesting type of image retrieval, called Content-Based Image Retrieval (CBIR) uses image contents to retrieve relevant images. In this case queries are made by specifying a query image (i.e. query by example).

Similarity between images in content-based image retrieval can be accessed by extracting image features and calculating distance functions between them (Datta et al. 2008). Multiple types of image features can be used in image retrieval, based on different image characteristics, including (among others): color data, texture information, and visual points of interest.

In general terms, image features can be divided in two categories: global features and local features. Global features try to characterize an image with a single vector. An example are global color histograms (Swain and Ballard 1991), which count the number of pixels of an image at each pixel color value. In clear contrast, local features categorize only a portion of an image, and multiple are used in conjunction to represent a whole image. An example are local color histograms (Jeong et al. 2004), where each counts the number of pixels at each color value for a different region of an image. In general local features allow retrieving images in a more precise way, however require additional memory (and persistent storage) capacity while exhibiting worst indexing and searching performance.

Among local features, those that combine both texture and color information

usually outperform other options in retrieval precision. Some of the best examples are the SIFT (Lowe 2004) and SURF (Bay et al. 2006) features, both exhibiting high retrieval precision by detecting keypoints of interest in images, through combination of texture and color information.

Indexing Structures for Images After extracting features from image repositories, search operations can be performed by calculating distance functions between these and query features. As in textual data, searching can be performed by linear comparison with all repository features, or by accessing an indexing structure with sub-linear performance.

Different indexing structures can be used for global and local features (Hjaltason and Samet 2003). Approaches for global features, such as M-Trees (Hjaltason and Samet 2003), divide feature-vectors by regions according to a distance function and build a dictionary for efficiently transversing regions when processing queries. In local features, due to the large number of feature-vectors extracted per image, a more scalable approach is required. The Bag Of Visual Words (Nistér et al. 2006) model is a popular solution. In this model a set of feature-vectors are clustered in a training phase, building a tree of representative feature-vectors. At indexing time, feature-vectors from new images are compared with this tree and each is represented as its most close tree leaf node. Finally an histogram of leaf nodes is built for each image, counting their frequencies. Given an histogram of frequencies, traditional indexing structures used for textual data can be employed for improved efficiency and scalability (including the inverted list index (Zobel and Moffat 2006) discussed in the previous Section).

2.2.3 Multimodal Retrieval

Multimodal retrieval allows one to query multiple information sources and combine search results in a relevant way. An obvious application is in repositories of data combining multiple media formats, such as annotated or tagged photographs, videos with audio included, and emails with media attachments.

Different techniques can be used to combine search results from multiple sources. One approach, called early fusion, extracts feature-vectors from different modalities and combines them in a single feature-vector (Atrey et al. 2010). However this requires a common representation between modalities to be found. In contrast, late fusion processes and indexes each modality in separate, combining search results when queries are performed (Mourão et al. 2014). Fusion of search results can be performed based on scoring function results (Shaw and

Fox 1995) or based on document ranking (Mourão et al. 2013). Scoring based fusion requires weighting results, as each is calculated in a different way and may have varying impact on retrieval precision. Ranking based fusion allows merging search results in a transparent way, by only considering the order that documents are returned in search results.

2.2.4 Retrieval Evaluation Metrics

When evaluating retrieval systems, two basic metrics are usually considered: precision and recall rates (Manning et al. 2009). To access these metrics a relevance set is required, mapping queries to lists of relevant documents for each. Precision is specified by the number of relevant documents retrieved from all returned documents. More precisely:

$$P = \frac{RelevantDocumentsRetrieved}{TotalRetrievedDocuments} \tag{2.3}$$

Recall is determined by the number of relevant documents retrieved given all relevant documents:

$$R = \frac{RelevantDocumentsRetrieved}{TotalRelevantDocuments} \tag{2.4}$$

Other relevant evaluation metrics can be computed from precision and recall. For a group of queries and their relevance set, an interpolated precision-recall graph can be constructed by measuring average precisions and recalls at different points of precision (e.g. with intervals of 10% precision). Average Precision (AP) and Mean Average Precision (mAP) (Manning et al. 2009) are other useful examples widely used. Given a ranked set of results for a query, AP is calculated by summing precision values at each relevant document rank:

$$AP = \frac{\sum_{k=1}^{n} (P(k) \times rel(k))}{RelevantDocuments} \tag{2.5}$$

where $P(k)$ is the precision at rank k and $rel(k)$ is a boolean flag indicating if the document at that rank is relevant (1) or not (0). Given a group of queries, mAP calculates the mean of APs for all queries.

$$mAP = \frac{\sum_{q=1}^{Q} (AP(q)}{Q} \tag{2.6}$$

where Q is the number of queries.

2.2.5 Summary

When searching data repositories, different techniques can be used by addressing a tradeoff between retrieval time, precision, and computational resources required. Depending on the media formats represented and searched for in the repositories, more processing resources and extra retrieval time may be required to achieve meaningful retrieval precision. Nonetheless, indexing data before performing queries is always an important operation, as it improves search performance from linear to sub-linear levels.

2.3 Searchable Encryption

In the last decades, Searchable Encryption has emerged as an important problem at the intersection of security and storage/computation outsourcing (Bösch et al. 2015). First proposed by Song et al. (Song et al. 2000) in 2000, Searchable Encryption deals with the problem of how to efficiently search encrypted data stored in a remote server. The emergence of cloud computing gave a new relevance to the problem, as more and more data is outsourced to remote, untrusted servers.

There are two main types of Searchable Encryption schemes, those based on public-key cryptography and those based on symmetric cryptography. Independently of subcategorization, all Searchable Encryption schemes depend in some degree on deterministic cryptography, as it is the use of deterministic primitives that allows searching encrypted data efficiently. A secondary effect of using determinism is revealing information patterns to adversaries when some operations are executed (e.g. search operations).

In Searchable Encryption, the most common information patterns usually leaked are *search* and *access patterns*. A search pattern uniquely identifies a query and reveals if it has been performed before. This is the most basic pattern that can be leaked when searching encrypted data, and it has been proven unavoidable in order to search encrypted data efficiently (i.e. in sub-linear time) (Naveed 2015). An access pattern reveals the list of documents that are returned by a query, through their deterministic identifiers. Preventing access pattern leakage can be achieved, for instance, by using two non-colluding servers where one stores documents and the other performs search operations. However this leakage is often dismissed as an orthogonal issue in the literature (Islam et al. 2012).

Revealing information patterns with operations is problematic because leakage will accumulate as more and more operations are performed. Ultimately these patterns will be revealed for the domain of all operational values. Consequently,

even if a Searchable Encryption scheme initially revealed nothing to adversaries, i.e. offered semantic security guarantees (Katz and Lindell 2007), as operations are performed these guarantees will quickly degrade to deterministic guarantees. Periodic re-encryption of all data solves this problem, but in most cases will be too expensive to perform, especially in large datasets.

2.3.1 Public-Key Searchable Encryption

Searchable Encryption in the public-key setting (Abdalla et al. 2008; Bellare et al. 2007; Boneh et al. 2004; Dong et al. 2011; Popa et al. 2014) has the advantage of naturally supporting multiple users. First proposed by Boneh et al. (Boneh et al. 2004), this kind of schemes has since been called *Public-key Encryption with Keyword Search* (PEKS). PEKS with a single private-public key pair allows multiple clients writing and encrypting data (with the public key) while one client searches the encrypted data (with the private key). Different public-key techniques have been explored so far to achieve this setting, including Identity Based Encryption (IBE) (Boneh et al. 2004), Anonymous IBE (Abdalla et al. 2008), and Hierarchical IBE (Abdalla et al. 2008). In these schemes, query keywords are used as identities for key generation and encryption of a public token. When the plaintext domain has high minimum entropy, PEKS can also be efficiently and securely achieved through deterministic encryption of all data (Bellare et al. 2007).

PEKS is also possible with multiple writers and multiple searchers (Dong et al. 2011; Popa et al. 2014). This usually implies the combination of public-key techniques with a key distribution or user authentication mechanism (Bösch et al. 2015). Dong et al. (Dong et al. 2011) achieves this multiple writer / multiple reader setting by employing a trusted third party which re-encrypts data from the different users under a common server key. More recently Popa et al. (Popa et al. 2014) proposed to use bilinear maps on elliptic curves. The approach allows multiple clients, each with her own cryptographic key, to write data and search hers and other clients data. This is done by computing a delta between her cryptographic key and every other clients' keys.

PEKS appears to be an optimal solution to the problem of searching encrypted data with multiple users. However there are two main issues that hinder its practicality. The first is performance. Public-key cryptography is an order of magnitude slower than symmetric key schemes. PEKS schemes usually require multiple elliptic-curve pairing operations (Abdalla et al. 2008; Boneh et al. 2004; Popa et al. 2014) or modular exponentiations (Bellare et al. 2007; Dong et al.

2011) per query. Furthermore, with the exception of (Abdalla et al. 2008), which is only secure in high min-entropy plaintext domains, PEKS schemes have search complexity linear with the number of documents.

The second main issue with PEKS is that since encryption keys are public and encryption of queries is deterministic, performing dictionary attacks to obtain the contents of encrypted queries is trivial. When using a PEKS scheme, clients should assume that there is no query privacy. The exception to this is the scheme by Popa et al. (Popa et al. 2014), which encrypts data with private keys but shares these between clients, making the problem of client revocation much harder to solve.

2.3.2 Searchable Symmetric Encryption

The seminal work by Song et al. in 2000 (Song et al. 2000), which started the research field on Searchable Encryption, was based on symmetric key primitives. Given the limitations of Public-Key Searchable Encryption regarding performance and query privacy (due to encryption keys being public), it is only natural that the symmetric-key setting received higher research interest and focus along the years (Baldimtsi and Ohrimenko 2015; Cash et al. 2014; Curtmola et al. 2006; Goh 2003; Hahn and Kerschbaum 2014; Kamara and Papamanthou 2013; Kamara et al. 2012; Kuzu et al. 2012; Naveed et al. 2014; Song et al. 2000; Stefanov et al. 2014).

The first Searchable Symmetric Encryption (SSE) scheme by Song et al. (Song et al. 2000) encrypted a document's keywords separately and XORed the encryption with an HMAC of a random value. Although encryption was deterministic, ciphertexts were probabilistic because of the random value. However searching for a query keyword required verifying HMAC signatures, which meant linear search complexity with the total number of keywords. Goh (Goh 2003) identified the need for indexing structures, which would improve search performance, and proposed using a bloom filter per document. However this still meant linear search complexity, this time with the number of documents.

Curtmola et al. (Curtmola et al. 2006) in 2006 started a new chapter in SSE research, for two main reasons: (*i*) they provided the first security notions for SSE, namely Indistinguishability against (non)adaptive Chosen Keyword Attacks (IND-CKA1 and IND-CKA2); (*ii*) they proposed using an index per keyword, achieving sub-linear search complexity for the first time. These security notions and indexing approach have been used henceforth in the SSE literature until this day (Baldimtsi and Ohrimenko 2015; Cash et al. 2014; Hahn and Kerschbaum

2014; Kamara and Papamanthou 2013; Kamara et al. 2012; Kuzu et al. 2012; Naveed et al. 2014; Stefanov et al. 2014). However, this work had two main issues: on one hand, searching operations were hardly parallelizable, as they required accessing index positions and performing decryptions in chain. On the other hand, only static document collections were supported, i.e. documents couldn't be added, removed, or updated dynamically after instantiation of the scheme with an initial dataset.

Kamara et al. (Kamara et al. 2012) proposed a variation of the previous scheme that allowed dynamic updates. However it also introduced a new form of information pattern leakage, called update leakage. This new leakage function meant that newly added (or updated) documents leaked deterministic identifiers of their keywords at update time. Static SSE schemes only revealed this information through search patterns. Consequently, adversaries would now learn if the new documents shared contents with other dynamically added documents or with documents already returned by a search operation. The same authors (Kamara and Papamanthou 2013) proposed a new scheme, in the following year, which prevented the leakage of *update patterns*. However their new approach had higher search and update complexities and required a larger index size than previous works.

2014 once again marked a new chapter in SSE research, as four new schemes were proposed in relevant venues in the same year (Cash et al. 2014; Hahn and Kerschbaum 2014; Naveed et al. 2014; Stefanov et al. 2014). Naveed et al. (Naveed et al. 2014) proposed an alternative SSE design that only required storage clouds, whose operational cost is (according to the authors) lower than computational clouds. Their approach was based on the idea of blind storage (discussed in Section 2.1.3). However, the scheme still leaked update patterns.

Stefanov et al. (Stefanov et al. 2014) combined SSE with Oblivious RAM techniques. Their work addressed update pattern leakage and achieved, for the first time, forward privacy, i.e. previously issued queries couldn't be reused by adversaries to infer search patterns on newly added documents. However it incurs some of the overheads of Oblivious RAM, including logarithmic search time, and requires clients to keep some local storage.

Hahn et al. (Hahn and Kerschbaum 2014) proposed outsourcing indexing computations to the cloud, however search complexity for new queries was linear, amortizing to sub-linear in repeated queries. This was achieved by leveraging on search pattern leakage: the cloud performed new search operations linearly and then, using the search pattern leaked by the query, incrementally built an index of the data. When a previous query was repeated by the client, search performance

would now be sub-linear by accessing the index. This approach also required the client to keep some local state for each unique keyword in the system.

Finally, Cash et al. (Cash et al. 2014) proposed a new dynamic scheme without update leakage and more efficient than previous alternatives. The scheme follows the traditional SSE methodology, displaying sub-linear search performance at all times by requiring clients to build and encrypt an index of the documents and to keep some local storage linear with the number of unique keywords.

Ranked Searching with SSE All SSE schemes described so far only allow exact-match searching of text documents through single-keyword queries. In other words, when a query is performed they return all documents containing the queried keyword, without any relevant ordering. Retrieving relevant subsets of a collection of documents is also an important problem, especially as applications datasets increase in size.

Extending SSE to support richer query expressiveness, particularly ranked searching, has been harder to achieve. The first attempt at ranked searching by Wang et al. (Wang et al. 2012) return ranked results but only allowed searching through a single keyword. To support efficient ranked searching the authors appended a new metric to index entries encrypted with Order Preserving Encryption, namely how many times a keyword appeared in each document. The encryption of this metric with deterministic encryption has been referred in later works as *frequency pattern* (Cash et al. 2015). Although the authors did not present a formal security treatment, the Order Preserving Encryption of this metric not only reveals frequency patterns (i.e. equality relations), but significantly more: it reveals reveals order relations between plaintexts.

Cao et al. (Cao et al. 2014) extended the previous scheme to support queries with multiple keywords. However their performance overhead was much higher: queries had length N and searching complexity $O(N^2)$ for the client and $O(N \times M^2)$ for the server, where N is the number of unique keywords and M is the number of documents stored in the system. Although no rigorous security treatment was provided, the scheme leaked frequency patterns.

Kuzu et al. (Kuzu et al. 2012) proposed a similarity SSE scheme that supported fuzzy keyword searching, i.e. searching by proximity with possible grammatical errors. Their approach could also be extended to other media domains (such as images), given an adequate translation function. However it required heavy client processing and multiple rounds of client-server communication, revealed frequency patterns with each query, and didn't support dynamic updates.

Baldimtsi et al. (Baldimtsi and Ohrimenko 2015) proposed a new solution that prevented frequency patterns from being leaked. However their approach required a secure coprocessor, controllable by the client, to be deployed in the cloud infrastructure. This secure co-processor performed a multiparty computation protocol with the cloud server, so that neither would learn frequency values while performing search operations, while the client encrypted the index with a partially homomorphic encryption scheme. Although this approach is very appealing from a security perspective, it is hard to deploy in practice.

A big limitation of the schemes developed so far for ranked text retrieval (Baldimtsi and Ohrimenko 2015; Cao et al. 2014; Kuzu et al. 2012; Wang et al. 2012) is that they only support static collections of documents. Dynamic collections, where documents are updated, inserted and deleted at any time have not yet been supported. Extending these works to support dynamic updates isn't trivial as they depend on pre-calculated ranking scores that need to be refreshed and re-encrypted with each new document insertion, update or removal.

SSE Techniques for Visual Data Some schemes for searching encrypted data in other media domains, such as images, have also been proposed in recent years (Lu et al. 2009; Xia et al. 2015; Yuan et al. 2014). These works usually resort to techniques similar to SSE, although adapted to the requirements of their application context. Lu et al. (Lu et al. 2009) proposed using techniques similar to SSE for searching encrypted image databases. However the proposed system was limited to color searching and did not provide a complete security treatment. Yuan et al. (Yuan et al. 2014) proposed a privacy-preserving social discovery framework which, through techniques similar to SSE, made friendship recommendations based on the similarity of users' images. However it required the deployment of trusted proxies. Xia et al. (Xia et al. 2015) proposed using local SIFT features and the earth mover's distance, combined with Locality Sensitive Hashing for security. Their approach exhibits good retrieval precision results, but requires multiple rounds of communication for performing search operations and requires the client to train and index her data before outsourcing it to the cloud.

The biggest issue with using SSE to search encrypted data in richer media domains is the performance overhead imposed on clients. Processing and indexing data in rich media types, such as images, audio, and video is a computationally intensive operation. This is in part due to the necessity of performing training tasks. Since indexing and searching high-dimensional data (characteristic of rich media t ypes) is computationally complex, training tasks can be used to find homogeneous groups of objects (Agrawal et al. 1998) and build compact representations

of that data, thus reducing complexity. However, training tasks (performed by machine learning operations such as k-means (Hartigan 1975)) are still expensive procedures and increase the performance overhead induced in clients by SSE techniques. Text documents, on the contrary, usually only contain a small subset of their domain (the english vocabulary, or any other language), and are more easy to index and search.

2.3.3 Attacks on Searchable Encryption

Revealing information patterns with operations, although necessary for efficiency issues, is in some sense a security vulnerability. Recent works (Cash et al. 2015; Islam et al. 2012) have analyzed the implications of revealing these information patterns to adversaries, and what attacks can be achieved by exploring them. Islam et al. (Islam et al. 2012) initiated this research by proposing a query recovery attack. Their attack used simulated annealing (a probabilisitic technique (Islam et al. 2012)) to try and match queries with keywords based on the pattern of which documents were returned. This approach was able to achieve near perfect query retrieval rate, but it required full knowledge of the stored documents and their contents. Moreover its success rate depended on the number of unique keywords stored in the system being small, and decreased strongly as this number increased (Cash et al. 2015).

The previous attack, besides requiring the strong assumption of full document-set knowledge and having scalability problems, only considered query recovery attacks and passive adversaries. Cash et al. (Cash et al. 2015) performed a more complete study of the problem, proposing: (i) a more simple and scalable attack for query recovery under full document-set knowledge; (ii) a similar attack for query recovery under partial document-set knowledge; (iii) a passive attack for plaintext recovery under document subset knowledge; (iv) and an active attack for plaintext recovery under chosen document knowledge. Attacks for query recovery were able to achieve near perfect results, even under large number of keywords, but still required full document-set knowledge (95% of document knowledge achieved only around 58% query recovery rate, and 75% knowledge achieved near 0% recovery). Plaintext recovery attacks either required knowing a large fraction of the document set, assuming documents had their structure (i.e. order of keyword appearance) preserved and their contents were deterministically encrypted, or required an adversary to be able to plant documents of his choice (active adversary).

In other media domains, attacks on searchable encryption may not be so easy

to perform, even under strong assumptions like full document knowledge. As far as we know, no rigorous security analysis has been done yet regarding this topic. Nonetheless we argue that while in text documents keywords have a straight semantical meaning, in other domains keypoints alone may not have a semantical meaning, and additional background information may be required in order to recover queries and plaintext data.

2.3.4 Summary

A vast literature on Searchable Encryption has been published in the last few years. The largest body of work dedicated itself to improving security and performance under little query expressiveness, namely exact-match single keyword searching of text documents (Cash et al. 2014; Curtmola et al. 2006; Goh 2003; Hahn and Kerschbaum 2014; Kamara and Papamanthou 2013; Kamara et al. 2012; Naveed et al. 2014; Song et al. 2000; Stefanov et al. 2014). Only a few recent approaches have tried to achieve higher query expressiveness in text document retrieval (Baldimtsi and Ohrimenko 2015), or searching in other media domains (Lu et al. 2009). Furthermore it is clear that the state of art on encrypted search imposes a prohibitive computational burden on clients (Cash et al. 2014), even more when Homomorphic Encryption (Baldimtsi and Ohrimenko 2015) or Oblivious RAM (Naveed 2015) techniques are used. This burden increases with richer query expressiveness and richer media domains of application, and is one of the main limitations hindering the wide adoption of Searchable Encryption techniques.

Attacks on searchable encryption, exploring the leakage of information patterns with operations, have been explored in the literature. Their effectiveness depends on strong assumptions including full document-set knowledge or the ability to perform chosen-document attacks. Nonetheless they still pose a security risk and applications should mitigate their threat by using padding (Cash et al. 2015) and limiting the amount of background information disclosed to adversaries through different channels.

The literature on Searchable Encryption is particularly relevant in the context of this thesis, as it aims at tackling the problem of how to efficiently search encrypted data. From the analysis of the relevant works in the field we come to the conclusion that existing solutions, on one hand, impose too much overhead for client applications and, on the other hand, are still very limited in terms of usability, query expressiveness, and media domains supported.

2.4 Cloud Reliability and Availability

Applications based on the cloud are vulnerable not only regarding privacy, but also dependability. Dependability issues include reliability, availability, data integrity, and cloud vendor lock-in. This Section of the research context analyses the current state of the art on mechanisms for addressing these issues.

2.4.1 Byzantine Fault Tolerance and Replication through a Cloud-of-Clouds

Byzantine faults are arbitrary failures possibly induced by adversaries during a distributed system's processing. Example faults include processing requests incorrectly, corrupting the local state of a server, or sending incorrect responses to clients. Hence, Byzantine fault tolerance is an important tool for guaranteeing both data integrity and availability. Byzantine faults were introduced by Lamport et al. in 1982, under the name of Byzantine Generals Problem (Lamport et al. 1982). The solution presented by the authors, however, was considered impractical, and only in 1999 a first practical approach was achieved by Castro et al. (Castro et al. 1999). Their solution, based on state machine replication, was able to tolerate faults on f servers if a total of $3f + 1$ replicas were available. Synchronization between the replicas was achieved by election of a primary replica, responsible for sequencing requests and providing total order. The remaining replicas monitored the client's requests and primary's behavior, electing a new one if it was found misbehaving.

Despite it's practicality, the work by Castro et al. (Castro et al. 1999) was still considered costly, especially in terms of scalability and of the number of messages required for synchronization. To address these issues other works have since been proposed, including quorum based Byzantine protocols (Cowling et al. 2006), fault-scalable Byzantine services (Abd-El-Malek et al. 2005), and speculative Byzantine fault tolerance (Kotla et al. 2009), among others. Some of these works require more replicas (e.g. (Abd-El-Malek et al. 2005) requires $5f + 1$) or depend on hard assumptions (Kotla et al. 2009) initially assumes primary servers to be non-faulty).

In the cloud computing domain, data may be replicated through multiple cloud providers, in a cloud-of-clouds fashion (Verissimo et al. 2012). A Byzantine fault tolerance protocol, executed between the client and the different cloud replicas, guarantees data integrity and availability. Such approaches have been used in the literature for dependable cloud storage services (Bessani et al. 2013).

2.4.2 Erasure Coding

An issue of replicating data through multiple clouds is that the extra storage cost may be too expensive for generalized adoption. As a complementary mechanism, erasure coding schemes (Rabin 1989) can reduce cloud storage costs while increasing data availability. An erasure coding scheme fragments data in such a way that it can be restored even if only a subset of the fragments are available. If a data-object is fragmented into n shares, m will be required to restore it and the size of each share is $|F|/m$ ($|F|$ is the size of the data object; both n and m are configurable, as long as $n > m$). In a cloud of clouds setting, if we distribute one share per cloud then n is equal to the number of clouds. Increasing m towards n reduces the size of each share and thus cloud storage costs, but decreases availability as more shares are required to restore the object. Reducing m (to a minimum of 1) increases availability, at the cost of extra cloud storage (Rodrigues and Liskov 2005). Either way, erasure coding as a complementary technique for Byzantine fault tolerance and cloud-of-clouds replication allows exploring a tradeoff between availability and storage requirements, which will always be beneficial compared to linear replication of data (Bessani et al. 2013). Fragmentation and code correction overheads may be problematic, nonetheless important advances in these issues have been achieved throughout the years (Mitra et al. 2016).

2.4.3 Secret Sharing and Threshold Cryptography

Similarly to erasure coding, secret sharing (Asmuth and Bloom 1983; Blakley 1979; Shamir 1979) allows fragmenting data (in this case a secret) into n shares, in such a way that at least m are required to recover the data. However, secret sharing schemes provide cryptographic guarantees that nothing is revealed about the secret by $m-1$ shares, and shares have a constant size equal to the size of the secret. Due to its storage cost and its performance overhead increasing with the number of shares, the main application of secret sharing is for safeguarding small secrets, such as cryptographic keys. In cloud computing scenarios with replication through multiple clouds, secret sharing can be used to split cryptographic keys into multiple shares and store each share on a different (non-colluding) cloud (Bessani et al. 2013).

Based on the same principles, a Threshold Signature scheme (Boldyreva 2003; Desmedt 1987; Shoup 2000) allows data to be authenticated by multiple parties at the same time. More specifically, a threshold signature is a distributed protocol where any subset of m out of n participants can generate a signature but no valid signature can be generated if only $m-1$ participants are available. Compared to

secret sharing, threshold signatures benefit from reduced share sizes and usually involve a public key setting, where a private key is required to generate a signature share and a public key is required to validate an aggregated signature. Threshold signatures can be used in cloud replication scenarios, where a client validates the availability and integrity of a replicated object by requesting each replica's signature share and applying the corresponding validation function.

2.4.4 Summary

Different mechanisms can be conjugated to address data integrity and availability in an efficient way. From the state of the art, Byzantine fault tolerance, erasure coding, secret sharing, and threshold signatures are mechanisms that make an interesting combination as all explore some form of threshold. This combination of dependable mechanisms and their thresholds, applied in a cloud computing model with multiple non-colluding cloud providers joined in a cloud-of-clouds architecture, shows potential for various dependability and efficiency benefits. Some dependable systems in the state of the art, like DepSky (Bessani et al. 2013), have explored these mechanisms in the scope of dependable cloud storage. What can be achieved by exploring these mechanism in applications that require both cloud storage and computation, as is the case of systems searching encrypted data, is an interesting open question that has been explored in this thesis as a complementary research line.

Chapter 3

Encrypted Multi-Keyword Ranked Text Searching

In this Chapter we present CloudCryptoSearch, a middleware system for storing and searching text documents with privacy guarantees.

We start by providing the motivation and goals of the contribution. We follow with a survey of the related work, discussing its limitations regarding our goals. We follow with a discussion of our system model, adversary model, and an applicational case study. The main technical details of our proposal are presented next, followed by analysis of experimental results.

3.1 Motivation and Goals

The security guarantees of operations and data in storage cloud solutions are primary concerns in their generalized adoption. These solutions present interesting characteristics, including remote data access, on-demand storage configuration, pay-per-use charging models, good quality of service, and reliability (Mell and Grance 2011). At first glance they are advantageous solutions, not only from technical and operational viewpoints, but also from an economical analysis. Storage clouds avoid management and software administration overloads, as well as software licensing costs. The existing cloud solutions offer a data storage services with availability guarantees and ubiquitous access conditions with independence of geographical location (Armbrust et al. 2010). However, when effective and independent control guarantees are required by the users over the availability,

security, and privacy of outsourced data-storage and operations, the adoption of those solutions can be more problematic.

The undue access or unauthorized disclosure of private data kept in storage clouds has been referred as a critical problem (Privacy Rights Clearinghouse 2009), not only in the use case of secure data backup solutions but also to preserve privacy guarantees of sensitive data accessed by online applications. This is the case of applications managing and searching medical records, financial data, and public administration documents (Kamara and Lauter 2010). The dependency of third party trust in outsourced data does not allow its control and complete auditing by end-users, allowing the data to be targeted by possible illicit actions or unattended operation by technical and administrative staff. Incidents have been verified due to software vulnerabilities or that explore physical access to the computational and communicational resources used by the providers, including network and communications equipments, software systems, or computational infrastructure devices: memory, hard drives, or internal backup solutions (Halderman and Schoen 2009).

To address the above problems and take the advantages of storage and computational clouds, a solution is needed conjugating two fundamental dimensions:

- Data privacy management, preserving privacy conditions during data searches or other possible operations;

- Scalability and performance guarantees, to manage of big data sets and support large number of operations.

The solutions proposed in recent years are mostly based in the use of cryptographic techniques to encrypt the stored data (e.g. (Feldman et al. 2010; Li et al. 2004)). In its majority they advocate the protection of data stored encrypted in the servers, with encryption done before data outsourcing (Brunette et al. 2009). In order to be operated, encrypted data must be transferred to clients for decryption. These solutions are limited when data processing at the server side is required, for efficiency and low-latency requirements. This is the case of applications using storage clouds as remote data-storage backends. Those solutions impose serious overheads in applications that have to process and search big data volumes organized in key-value stores, as is the case of many cloud data processing algorithms.

In this Chapter we propose CloudCryptoSearch, a middleware solution that has in sight the conjugation of the two fundamental dimensions described previously. CloudCryptoSearch addresses security and privacy concerns, maintaining

independent control of data-privacy by the end-user, promoting a trustable environment for data storage and data management on Internet storage clouds, and reducing the role of cloud providers as trust-entities. Based on a middleware architecture, CloudCryptoSearch is supported by searchable encryption techniques (Song et al. 2000) combined with dynamic indexing mechanisms (Manning et al. 2009). The cryptographic mechanisms proposed preserve data privacy without need to transfer and decrypt data during search operations in the cloud. The middleware system is neutral to cloud providers and may be adopted to operate with different clouds.

3.2 Related Work

One of the most common types of queries over files or text documents is based on the use of keywords or subsets of searchable metadata. Usually the keywords and the queries are supported over the original documents in plaintext. The need to execute these queries over encrypted data raises new problems. In general, operations on the encrypted domain require some properties of the plaintext data to be preserved after encryption. This is an apparently contradictory aspect regarding the security characteristics of the cryptographic algorithms, and property-preserving schemes must be able to preserve the properties required while enforcing the security characteristics necessary as considered for conventional cryptography.

Different properties may be needed for different purposes (Popa et al. 2012; Song et al. 2000). A cryptographic transformation presents pure and complete homomorphism when any operation on the plaintext data can be transformed into an equivalent operation on the ciphertext data. As detailed in Chapter 2, a fully homomorphic scheme supporting additions and multiplication in encrypted data (Gentry et al. 2012) does not have today a generic and practical solution. However, partially homomorphic (Paillier and Pointcheval 1999) or property preserving (Popa et al. 2012) schemes can be addressed for practical applications. As will be detailed in the following Sections, in our solution we adopt the Paillier scheme (Paillier and Pointcheval 1999) for additively homomorphic operations, and design a new equality-preserving scheme (LSS scheme, detailed in Section 3.5.1) for supporting privacy-preserving data indexing and searching functionalities.

Although some conventional encryption schemes (Boneh et al. 2004; Curtmola et al. 2006; Song et al. 2000) allow searching over encrypted data, a large group

of the current practical approaches only address exact-match queries, i.e. search operations that verify the (in)existence of one or more text keywords. Such solutions do not allow the capture of complementary indicators, including relevance scores and related metrics, needed for multi-keyword ranked queries. In this case, the common solutions present limitations, possibly not supporting: (*i*) queries without complete or partial data transfer to a user's trust base, where they can be decrypted during the search process; (*ii*) queries requiring low processing latency and avoiding network traffic, also affecting the "pay-per-use models" common in most cloud repositories.

Recent works have demonstrated the practicality of property preserving schemes in other forms of searching. A good example comes from the encrypted relational database world, where the CryptDB system (Popa et al. 2012) allows most SQL queries to be executed over the encrypted data. This is achieved by combining different property-preserving schemes. Another example comes from private information retrieval on public plaintext cloud repositories (Liu et al. 2012), where a property-preserving scheme is used to allow searching plaintext databases while protecting the privacy of query contents, relevance metrics, and search patterns.

More related to our work is (Wang et al. 2012), which aims at supporting single keyword ranked searching over encrypted cloud data. The work uses a common information retrieval approach, namely TF-IDF scoring function (Sparck Jones 1972) and inverted list indexes (Zobel and Moffat 2006). It also achieves good security and efficiency conditions, through the use of an order-preserving encryption scheme that allows ordering rank scores while encrypted and stored in the cloud. However it only allows searching through single keyword queries. In (Cao et al. 2014), a multi-keyword ranked searching solution is proposed, based on a secure k-nearest neighbor (kNN) technique combined with a *coordinate matching* ranking function. However the encryption scheme used requires sequential scanning of the index each time a search is requested. Moreover both works do not foresee a dynamic scenario where data can be updated, and both require the index to be built locally, encrypted, and only then it can be stored in the cloud.

In this Chapter we have followed, for the first time, an approach that allows the ranking and indexing of encrypted documents in the cloud while preserving their privacy. This approach requires considerably less computational power in the client and fully explores the potential of the cloud, with good performance and security conditions as proved by our experimental results (detailed in Section 3.7).

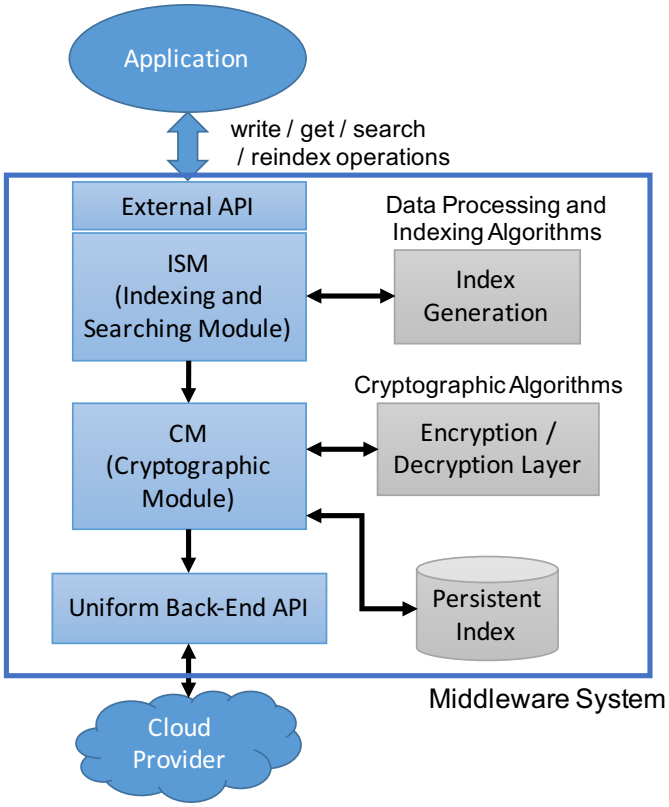

Figure 3.1: Reference architecture of the CloudCryptoSearch middleware solution.

3.3 System Overview

CloudCryptoSearch aims to manage data privacy by using a security approach, which allows the storage of sensitive data on storage clouds under control of the users who own the data. For this purpose the proposed solution addresses the following requirements:

- Privacy of the data, with independent control and auditing by the users;
- Extension of the previous guarantees to the search operations over the encrypted data, namely supporting secure ranked multi-keyword searches;
- Operation under full control of the users, independently of trustability services or guarantees offered by cloud providers;

3.3.1 System Model and Architecture

Figure 3.1 shows the reference architecture of the CloudCryptoSearch middleware. The architecture has in sight the use of property-preserving and homomorphic encryption techniques (CM module) combined with data indexing and

dynamic updates (ISM module). In this model, the users execute applications that interact in a secure way with the middleware. We assume users are previously authenticated through an orthogonal authentication mechanism, with communications protected by secure remote channels. User data (typically sets of text documents) are processed and indexed by the middleware in order to be securely transferred to the cloud. Data or documents stored in the cloud are maintained encrypted even when remote operations are executed. This way, the middleware offers data storage outsourcing while preserving privacy control over that data. A dynamic system is considered (Kamara et al. 2012), where documents can be inserted and removed at anytime of the middleware execution. Documents are also considered mutable, i.e. new versions of already existing documents can be inserted, with repercussion on the metrics stored in the index.

3.3.2 Adversary Model

In this contribution we consider the cloud provider as our main adversary, trying to compromise privacy of data and operations. As such, our trusted computing base is restricted to the client application. cloud servers are considered not trustable, admitting they may be subject to passive attacks. It is assumed, that the cloud provider infrastructure is always dependable and available, executing the storage operations correctly and according to specification. Nonetheless, attacks on the communications or on the servers aiming at breaking data privacy, are admitted.

3.3.3 Middleware External API

CloudCryptoSearch provides a web service security API (SSL supported) described in Table 3.1. The API implements data management operations as in key-value storage clouds. Additionally, a search and reindex operations are supported. The former provides encrypted ranked multi-keyword searching functionalities, while the latter provides an auxiliary method of controlling the precision of index entries when they grow too large to fit in main memory (more details in Section 3.5.2).

Using the API described in Table 3.1, applications can access the middleware services in a transparent way, since the provided *put* and *get* operations are similar to the usual operations offered by current cloud service providers.

Operation	Description
DocumentId put (Document)	Stores and indexes a document to be transferred to the cloud.
Document get (DocumentId)	Retrieves a document from the cloud using it's identifier.
Set<DocumentId> search (Set<Keyword>)	Searches the cloud repository for a given set of keywords, returning a set of document identifiers.
reindex()	Recalculates index entries and relevance metrics from persistent storage.

Table 3.1: CloudCryptoSearch external middleware API.

3.3.4 Middleware Processing

Internally, the different modules implement the required algorithms for indexing, data encryption, and searching. When the user writes documents through the external API, the indexing module builds a searchable secure index from all relevant keywords. This index is then encrypted and stored persistently. As discussed in the next Section, the index may be stored locally or remotely in the cloud. At the same time, documents are encrypted and remotely stored in the cloud.

3.4 Deployment Options

The middleware components and algorithms described so far allow a generic and flexible solution that can fit different tradeoffs between security level requirements, performance of operations, and client resources available. Exploring these tradeoffs, three implementation scenarios were conceived.

3.4.1 Middleware in User's Trustable Device

This scenario is based on high security and performance requirements. The user wishes to have the best available security and chooses to keep the middleware running inside his trustable device, in an attempt to minimize system exposure. Documents are encrypted with standard symmetric cryptography (Random scheme, presented in Section 3.5.1) and uploaded to the cloud, however they are first indexed by the user before being uploaded. Since the user's device is assumed to be trustable, and to increase performance even further, the index is not encrypted in this scenario. Search operations start by accessing the local index and then retrieving the relevant documents from the cloud. As such, this scenario can work with

storage-only clouds. This scenario has high computational power requirements, as the whole system will be running in the user's device.

3.4.2 Middleware as a Proxy Service

As in the previous scenario, in this use case the user has high security and performance requirements. However, she wishes to interact with the middleware through a device with low resources (e.g. mobile device, smartphone, etc.). To this end, the middleware system is moved to an auditable proxy service running in a local network. In this case we assume the proxy service can be targeted by passive attacks. As such, the index is encrypted with a symmetric encryption algorithm (Random scheme) after being built and in-between re-indexing processes that may occur. Documents are still encrypted with the Random scheme and uploaded for storage to the cloud. Searching operations decrypt relevant index entries in the proxy and return ranked results to the user's application.

3.4.3 Middleware as a Service in the Cloud

In this scenario, the user has very limited resources and cannot afford the deployment and management of a local proxy. This is the case of a user running an application in a resource-constrained mobile device. As such the idea is to move the middleware system to a computational cloud. In order to securely index private data in the cloud, a special cryptographic scheme is designed and applied to all document keywords (Linear Search Scheme (LSS) as discussed in Section 3.5.1). However, document structure is not used in searching operations and keywords within a document are sorted in a random order. Through the properties of the LSS scheme, the cloud can process documents for relevance ranking while preserving their privacy. After the index is built (or updated with dynamic operations), it is encrypted with either conventional symmetric-key encryption (Random scheme) or with a partially-homomorphic scheme (*Partially-Homomorphic Scheme*, presented in Section 3.5.1). The former requires search operations to be interactive, forcing clients to decrypt index entries and rank search results. The later allows searching operations to be executed by the cloud in a secure way, but at the cost of increased cryptographic overhead due to partially-homomorphic encryption.

3.5 Components of the Middleware Architecture

This Section discusses in more detail the main components of the middleware system. These are the Cryptographic Module (CM), which is responsible for encrypting data and operations with different cryptographic schemes, and the Indexing and Searching Module, which is responsible for performing information retrieval operations in the encrypted domain.

3.5.1 Cryptographic Module

The Cryptographic Module (CM) is a software-based library implementing three different cryptographic schemes, each displaying different properties: Random Scheme, Homomorphic Scheme and Linear Search Scheme.

Random Scheme The most secure encryption scheme used in our solution and with best performance is the Random scheme. In this scheme, two equal plaintexts originate different ciphertexts with overwhelming (pseudo-random) probability, protecting data against Adaptive Chosen Ciphertext Attacks (IND-CCA). However due to its randomness properties, the scheme does not allow computations over encrypted data. These properties make it suitable for encrypting data that will not be subject of further operations, including text documents or the index when they are being encrypted for persistent storage. In our solution the random scheme is implemented using a symmetric block-cipher algorithm with a secure encryption mode, such as AES in CBC mode (Katz and Lindell 2007).

Partially Homomorphic Scheme The middleware uses a partially homomorphic encryption scheme (PHE) that allows additions on the encrypted domain. This scheme, which implements the Paillier cryptosystem (Paillier and Pointcheval 1999), is based on modular arithmetic properties where the multiplication of ciphertexts $E(X1)$ and $E(X2)$ is equivalent to the encryption of the modular addition of their plaintexts $X1$ and $X2$:

$$E(X1) \times E(X2) = (g^{X1} r1^n) \times (g^{X2} r2^n) = g^{X1+X2} (r1 \times r2)^n = E(X1 + X2 \ mod \ n) \quad (3.1)$$

where g, r and n are prime numbers with at least 1024 bits length each (for achieving reasonable security regarding nowadays computational power) (Paillier and Pointcheval 1999). Ciphertexts will have double the length of the prime numbers, in this case 2048 bits. It is well known that modular exponentiations have high computational complexity, meaning that Paillier encryptions should be

kept to a minimum required for achieving the desired properties (e.g. only used in the encryption of a few metrics, like those stored in the index).

Linear Search Scheme Linear Search Scheme (LSS) is a scheme that we propose, based on the literature on searchable encryption (Song et al. 2000). This scheme has deterministic properties in relation to text data. In more detail, it allows linear scans and queries for patterns on the encrypted domain. The ciphertext is calculated using hash based authentication codes (HMAC) and a secret as a master key:

$$E(keyword) := Hash(HMAC_{MasterKey}(keyword)) \tag{3.2}$$

The HMAC of a keyword is further hashed as a way to increase ciphertext distribution and limit its length. The proposed scheme is deterministic, as two equal plaintexts always generate the same ciphertext. Using this scheme, a client can encode the various keywords of a document and later check if it contains a particular keyword by generating its encoding (with the same master key) and comparing it with all encoded keywords. The cloud server, given access to different documents and their encoded keywords, can index the documents by counting the number of times each distinct keyword appears in each document. The scheme's security is inherited by the underlying security of the HMAC function (namely the cryptographic hash function used in its construction) and by the protection of the $MasterKey$ (which should be stored in the user device). The performance of the scheme is also guaranteed by only using hash functions.

3.5.2 Indexing and Searching Module

A key component in the middleware architecture is the Indexing and Searching Module, which supports searching and indexing operations in the encrypted domain.

Indexing and Scoring The indexing structures built by this module are used to speed up query execution time and to guarantee sub-linear search performance. An inverted index (Manning et al. 2009) (the type of index we use in this work) is a dictionary structure that maps keywords to a list of documents containing them. Additional metrics may be included in index entries, including frequency values (i.e. how many times the keyword appears in the document), or more complex pre-computed ranking metrics that try to assess the value of a document in a whole repository. Figure 3.2 represents an example inverted index, implemented as an hash map.

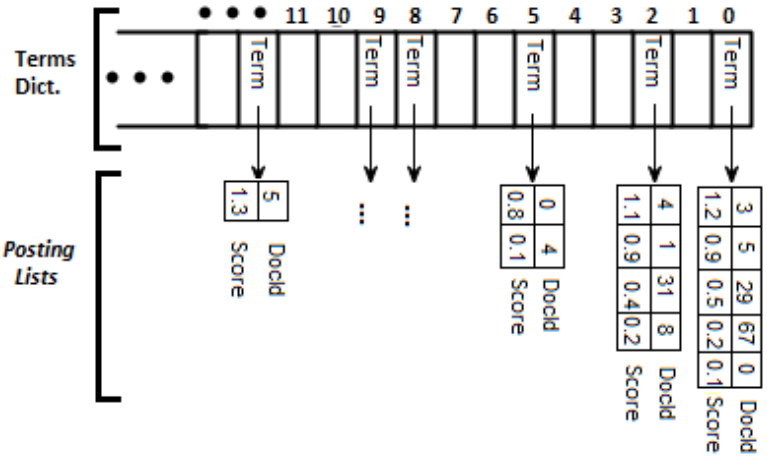

Figure 3.2: Inverted list index implemented with an HashMap.

When a search is done, indexing statistics are grouped in a scoring function that returns the final relevant results for the query. An example scoring function vastly used in information retrieval, as it cleverly combines multiple repository-wide metrics, is BM25 (Manning et al. 2009):

$$BM25(q,d) = \sum_{t \in q} log_{10}(\frac{N}{df_t}) \times \frac{(k_1+1) \times tf_t^d}{k_1 \times ((1-b) + b \times (L_d/L_{avg})) + tf_t^d} \tag{3.3}$$

where q is a query with multiple keywords, d is a document, N is the total number of documents in the collection, df_t is the document frequency of keyword t (i.e. the number of documents that a keyword appears on), tf_t^d is the frequency of t in document d, L_d is the length of the document, L_{avg} is the average length of all documents, and k_1 and b are configurable parameters (usually with values 1.2 and 0.75, respectively).

An index can be constructed using different approaches. A scalable example, that can be used for indexing datasets larger than a system's main memory capacity, is the Single Pass In Memory Indexing (SPIMI) (Manning et al. 2009). After its construction, if the index does not fit entirely in main memory we use a technique called ChampionLists (Manning et al. 2009), which consists in keeping the whole index in persistent storage and leaving in memory only the top scoring documents for each keyword, ordered by their relevance. This way search operations can be performed without the overhead of accessing persistent storage. With this technique some precision may be lost, however it is not expected to be substantial for the final result.

Dynamic Insertions and Removals The proposed middleware offers a fully dynamic system where documents can be inserted, removed, and updated at anytime (Kamara et al. 2012). This means that the Indexing and Searching Module must be able to deal with document insertions and removals while minimizing: (*i*) costs in the searching functionality and its precision; (*ii*) and threats to the privacy of documents and related metrics.

Dynamic changes can occur in two circumstances: when the index still fits in memory or when it has been written to disk and only ChampionLists are maintained in memory. In the first case, updates are trivial. New documents are processed as usual and resulting index entries are immediately inserted in the index, while removed documents have their references deleted from the index. In the second case, the possible retrieval to memory of various partitions of the index in order to update them becomes too expensive. As such, an auxiliary index is created in memory that will hold new index entries and the identifiers of removed documents. Search operations are performed by consulting both in-memory indexes and filtering removed documents from the results. Periodically a merging operation can be performed, which will merge the auxiliary index with the full index in disc and fully remove deleted documents from the main index. After merging, the index is persisted again and ChampionLists are calculated and stored in main memory. This operation can be executed in background without interrupting other middleware functionalities.

3.6 Middleware Operations

In this Section we describe the main operations in the middleware API, namely the *put* and *search* operations. In this case we consider the middleware system deployed as a service in the cloud (described in Section 3.4), as it is the most interesting use case in terms of usability. Nonetheless, operations can easily be adapted for the other use cases.

3.6.1 Indexing and Writing

When using the middleware as a service in the cloud, some local processing needs to be executed by the client application before documents can be uploaded to the cloud. These operations are presented in Algorithm 3.1.

Algorithm 3.1 is executed when the client wishes to store a document in the cloud. The algorithm starts by encrypting the document with the Random scheme for storage (line 2). Then the plaintext document is processed for indexing, i.e.

Algorithm 3.1 Client processing for writing/indexing documents in the cloud.

```
1: function PUT(document)
2:     encrypted_document ← encrypt_Random (document)
3:     searchable_keywords ← process_document (document)
4:     encrypted_keywords ← encrypt_LSS (searchable_keywords)
5:     document_id ← write (encrypted_document, encrypted_keywords)
6:     reindex()
7:     return document_id
```

Algorithm 3.2 Service in the cloud: support for writing and indexing.

```
1: function WRITE(encrypted_document, encrypted_keywords)
2:     document_id ← store_object (encrypted_document)
3:     relevance_metrics ← process_keywords (encrypted_keywords)
4:     insert_index (relevance_metrics)
5:     if memory_full then
6:         ciphered_index_partition ← encrypt_Random (index)
7:         store_object (encrypted_index_partition)
8:         clean_index()
9:     return document_id
10: procedure REINDEX( )
11:     clean_index()
12:     for all index_partition_id do
13:         encrypted_index_partition ← get (index_partition_id)
14:         index_partition ← decrypt_Random (encrypted_index_partition)
15:         ChampionList ← build_champion_list (index_partition)
16:         merge_ChampionList_with_index(ChampionList)
17:     encrypt_PHE (index)
```

stop words are filtered, searchable words are stemmed, and punctuation is removed (line 3). After this pre-processing, the resulting keywords are encoded with the LSS scheme, one at a time, and the resulting set is transferred to the cloud for indexing, along with the encrypted full-document for storage (lines 4 and 5). Next the client can issue a *reindex* operation in order to merge all index partitions in persistent storage and update the in-memory index of the cloud (line 6). In the presented algorithm, the *put* and *reindex* operations are remote procedures supported by the middleware service running in the cloud, processed as presented in Algorithm 3.2.

In the cloud, the procedure used for persistent storage and indexing of a document is the *write* operation. The set of encrypted keywords of the document is processed for relevance metrics extraction (lines 3 and 4) and periodically, if main memory is full, an index partition is encrypted and persistently stored

Algorithm 3.3 Client searching by keywords.

```
1: procedure SEARCH(keywords)
2:     query_keywords ← process_keywords (keywords)
3:     encrypted_keywords ← encrypt_LSS (query_keywords)
4:     encrypted_ranked_scores ← search (encrypted_keywords)
5:     ranked_scores ← decrypt_PHE (encrypted_ranked_scores)
6:     ordered_ranked_scores ← order_ranked_scores (ranked_scores)
```

Algorithm 3.4 Service in the cloud: support for the searching operation.

```
1: function SEARCH(encrypted_keywords)
2:     relevant_index_entries ← access_index (encrypted_keywords)
3:     encrypted_ranked_scores ← calculate_ranking_scores (relevant_index_entries)
4:     return encrypted_ranked_scores
```

(lines 5 to 8). After multiple documents of a collection have been processed, stored, and indexed the client can issue the *reindex* operation (line 11) to merge all the index partitions and build the ChampionLists from the top ranking documents (lines 13 to 16). This operation ends with the encryption of the index with the Partially-Homomorphic scheme (PHE, in line 17).

According to the above explanation, the *put* operation (performed by Algorithm 3.1 on the client side and Algorithm 3.2 in the cloud side) builds an index from an initial collection of documents. It should be noted however that the middleware can also start operating from an empty index and accept the same group of documents dynamically, although the costs in performance will be higher.

3.6.2 Searching

After the storage of multiple documents, ranked searches can be executed by issuing a set of query keywords. Algorithm 3.3 exemplifies the operation executed in the client side for searching a collection of documents.

The client starts by processing and encrypting the query keywords with the LSS scheme (lines 2 and 3), preserving their privacy. Then the encrypted keywords are sent to the middleware service running in the cloud (line 4), in order to perform the search. The cloud service returns encrypted ranked scores, which the client decrypts and sorts to obtain the ids of the top-ranking document for the issued query (lines 5 and 6). Algorithm 3.4 represents the search process as executed by the middleware solution running in the cloud.

To execute a client search operation, the middleware service at the cloud consults the index to find the relevant entries (line 2), calculates the final scores from

the encrypted entries and returns them to the Client (lines 3 and 4).

3.7 Implementation and Experimental Evaluation

We have implemented a prototype of the proposed solution for experimental evaluation. The prototype is an open source software project and can be found at: `https://github.com/bernymac/CloudCryptoSearch`. The evaluation we now present is focused primarily on performance criteria and latency considerations.

The implementation of the prototype is based on the Java language and libraries. The results were obtained using a OpenJDK 6 JVM, with 6 GB of primary memory, Concurrent Garbage Collector activated and executing in a PC with Intel Core i3 3.4 GHz processor. The chosen cloud provider was Amazon (AWS), Ireland data-center. Amazon S3 is used as storage service and Amazon EC2 as computational cloud service, configured with a m3.large instance. The version of the Amazon Java SDK used is 1.3.2. The connection to the cloud was limited to 30 Mbps for downloads and 3 Mbps for upload. We evaluated the system with a dataset comprised of different versions of Wikipedia documents. The versions used were the English, Spanish, and Portuguese Wikipedia dumps (Wikimedia Foundation 2016) with uncompressed sizes of 37.9, 7.2, and 4.4 GB respectively.

In order to experimentally validate the designed solution and the implemented prototype, different load tests were carried. Figure 3.3 shows the first group of tests. This first group of tests aimed at measuring the performance overhead of the middleware processing when compared to simple storage in the cloud. To increase the evaluation interest of the comparison, a secure solution from Amazon, based on Client-Side Encryption (Services 2011), was used as baseline comparison (referred to in the graphs as AWS). The solution is based on encrypting the data at the client-side with AES before sending it to Amazon cloud servers, while the master encryption key is stored and managed by the client.

In Figure 3.3, results are divided between datasets (Portuguese (PT) Wiki, Spanish (ES) Wiki, and English (EN) Wiki) and between implementation scenarios. These implementation scenarios correspond to the different settings described in Section 3.4, with the last setting (middleware as service in the cloud) divided in two use cases with a small variation: index encrypted with the Random scheme (implemented with AES) or with the Partially-Homomorphic scheme (PHE, implemented with Paillier). Although small this variation has high impact on obtained results, as will be seen next.

The results presented in Figure 3.3 show that the overhead introduced by

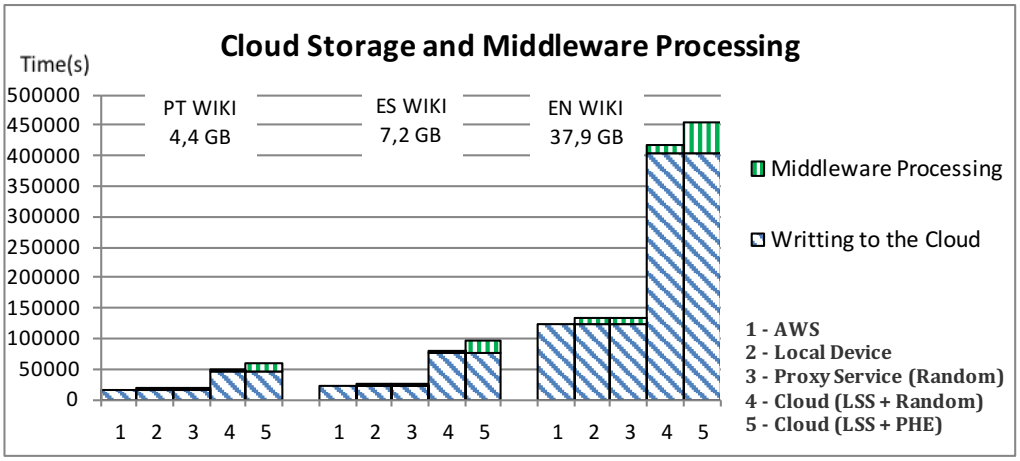

Figure 3.3: Performance of writing and indexing in the different scenarios.

the middleware is minimal when compared to the cost of transferring the same collection of documents to the cloud. On the other hand, if we analyze in more detail the two use cases entirely based on the cloud (the rightmost bars in each of the 3 datasets), we can conclude that they require writing more data to the cloud than the other tests. This requirement comes from the need to upload encrypted documents (Random scheme, for storage purposes) and encrypted keywords (LSS scheme, for indexing and searching purposes) in a separated way. This is due to the LSS scheme being one-way, as it is based on HMAC techniques. An LSS scheme based on symmetric encryption could be designed in the future (e.g. AES in ECB mode (Katz and Lindell 2007)), allowing decryption and hence reducing network traffic by only requiring data to be uploaded once.

Considering the middleware processing in more detail, we can analyze and compare the performance of its main processes. Figure 3.4 represents these tests. In the figure, document processing refers to the act of processing a document and extracting its keywords; indexing refers to the re-indexing operation and building of ChampionLists from persistently stored index partitions; and security refers to all processes of encryption and decryption that need to be performed with the different schemes.

Analyzing Figure 3.4 in detail and the performance of the middleware processes, we conclude that using conventional symmetric encryption (Random scheme) to protect the privacy of the index adds relatively little latency to overall performance. This is shown by the cryptographic overhead in the *Proxy Service* scenario. On both cloud scenarios, cryptographic overhead is slightly increased due to requiring encryption of each keyword independently with the LSS scheme. Additionally, as expected from partially homomorphic encryption, using the PHE scheme to encrypt the index has a very high penalty on performance. For instance,

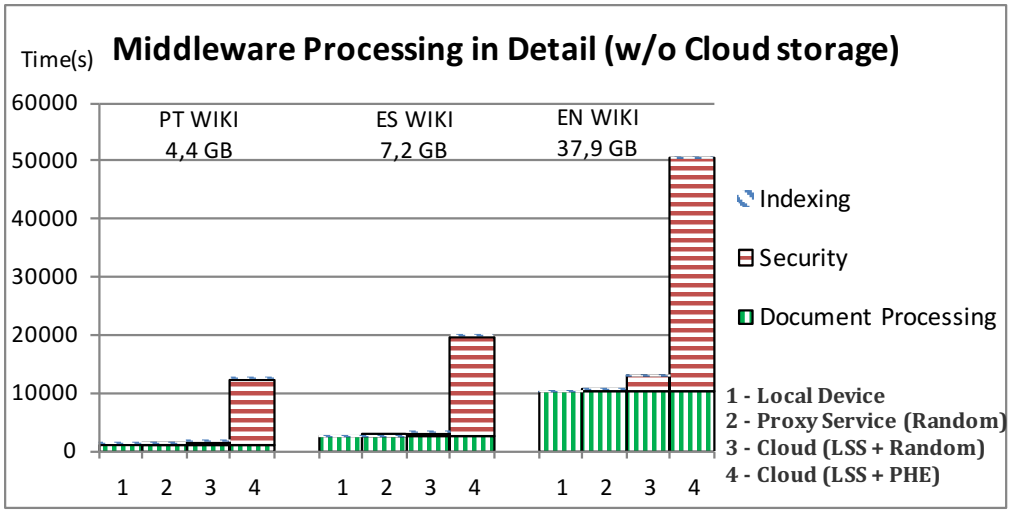

Figure 3.4: Performance of the middleware processes in the different scenarios.

the Portuguese dataset test in the cloud (LSS+Random) scenario takes around 19 minutes, while the cloud (LSS+PHE) scenario takes around 3 hours. However the cost of using the PHE scheme seems to gradually reduce with the increase in data size, as index size is kept to a minimum with ChampionLists. As an example, from the English to the Spanish Wikipedia there is a data size increase of approximately 5 times but the performance cost in this scenario only increases around 2.3 times. The reader should also note that the cost of the indexing operation only has impact when a collection of documents are stored.

The third and final group of tests aimed at comparing the performance of searching in the different implementation and operation scenarios. Figure 3.5 presents the results. The results shown were obtained using the 37,9 GB collection of English Wikipedia documents. In all scenarios search operations are very fast and always performed under 1 second (a threshold for usability in online applications). However, there are visible differences between the scenarios. Comparing the first two use cases, we can see that direct access to the index is very fast and that traditional security has very low cost on search performance. Also, in both scenarios queries are performed under the 100 milliseconds mark. When comparing these results to the scenarios that use the cloud for index storage and computation, latency is increased as access to the middleware has to be done through the Internet. This increase is due to Internet latency and network traffic, and should be expected in any implementation scenario where a search has to request information from a remote site.

Analysis of the cloud (LSS+PHE) scenario shows that the increase in cryptographic overheard is very high. This is due to the natural cost of decrypting

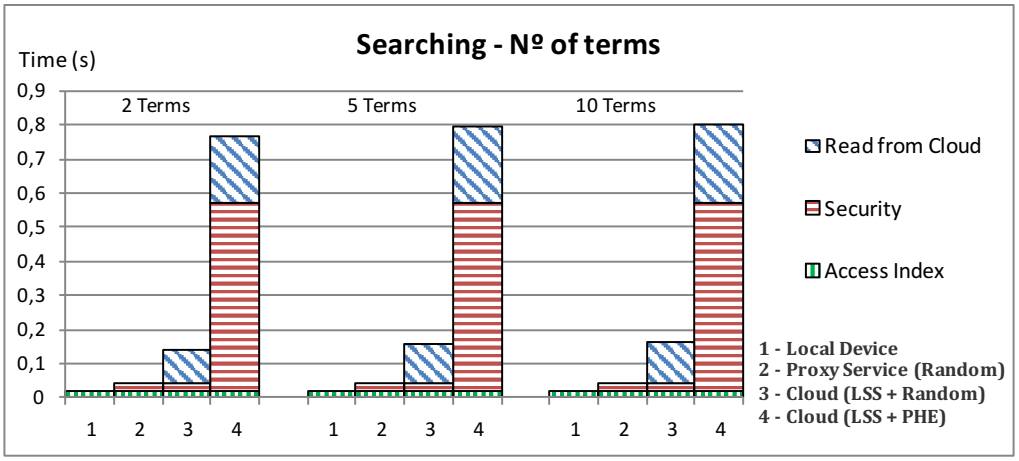

Figure 3.5: Search performance in the different scenarios.

index entries encrypted with the PHE scheme. The time taken to retrieve relevant results from the cloud-stored index also increases as the ciphertext size of this scheme is much larger than the ciphertext size of the Random scheme. Finally results show that performance is preserved when both queries and datasets increase in size, due to the use of scalable techniques including ChampionLists.

3.8 Summary

In this Chapter we presented a solution that has in sight the conjugation of security and privacy requirements of data stored in Internet storage clouds. The solution is designed as a middleware system for intermediation of secure storage services for private data in storage clouds. The presented system supports the management and storage of private data, under full control of the user, and allows searching over the encrypted data, independently of different clouds that may be used. A relevant contribution of the proposed solution focuses on the support of secure searching operations over the data, proposing a solution for ranked multi-keyword queries. The solution achieved allows the use of effective mechanisms for searching over the private documents with multiple keywords and accessing the encrypted information in the cloud, based on ranking operations on the relevance of the data. During the search operations, privacy conditions are preserved under full control of the users. The presented approach uses cryptographic schemes that explore property-preserving and partially-homomorphic encryption techniques combined with dynamic indexing mechanisms. The implementation of the proposed middleware system and its evaluation shows that the

solution is viable, offers more security and greater user control (compared to a secure solution promoted by Amazon AWS (Services 2011)), and does not aggravate conditions of access latency and data availability. Finally, comparing with state of art approaches, our solution performs an improved tradeoff management between conditions of performance, client overhead, and privacy conditions, while preserving retrieval precision.

Publications The contribution presented in this Chapter has been published in the following venues:

- **Gestão e Pesquisa de Dados Privados em Nuvens de Armazenamento**. Bernardo Ferreira and Henrique Domingos. In proceedings of the 4th Simpósio de Informática (INFORUM'12). Caparica, Portugal, September 2012.
- **Management and search of private data on storage clouds**. Bernardo Ferreira and Henrique Domingos. In proceedings of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management (SDMCMM'12). Montreal, Canada, December 2012.
- **Searching Private Data in a Cloud Encrypted Domain**. Bernardo Ferreira and Henrique Domingos. In proceedings of the 10th Conference on Open Areas in Information Retrieval (OAIR'13). Lisbon, Portugal, May 2013.
- **CloudCryptoSearch: a prototype for secure searching of private data in cloud encrypted domains**. Bernardo Ferreira and Henrique Domingos. In proceedings of the 10th Conference on Open Areas in Information Retrieval (OAIR'13). Lisbon, Portugal, May 2013.

Prototypes A software prototype of the contribution is also available at:

- `https://github.com/bernymac/CloudCryptoSearch`

Chapter 4

Privacy-Preserving Content-Based Image Retrieval

In this Chapter we present a second main contribution of the thesis on how to search encrypted visual data. The core of this contribution is a novel searchable encryption framework to support content-based image retrieval in the encrypted domain, while outsourcing heavy training and indexing computations from client applications to the cloud. This framework is based on IES-CBIR, a new Image Encryption Scheme with Content Based Image Retrieval properties that we also propose and evaluate.

We start by providing the motivation and goals of this contribution. Following the identified goals we survey the state of art, discussing drawbacks of existing solutions regarding our objectives. The Chapter follows with a definition of its system and adversary model, as well as with a description of an applicational case study. The main technical details of the proposed contribution are presented next, followed by discussion of developed prototypes and experimental results.

4.1 Motivation and Goals

Nowadays visual data is responsible for one of the largest shares of global Internet traffic in both corporate and personal use scenarios (Meeker 2015). The amount of images, graphics, and photos being generated and shared everyday, especially by mobile devices, is growing at an ever increasing rate. The storage needs for such large amounts of data in resource-constrained mobile devices has been a driving

factor for data outsourcing services such as the ones leveraging cloud storage and computing solutions. Such services (*e.g.* Instagram and Flickr) have been reported to be among the largest growing internet services (Global Web Index 2013). Additionally, the availability of large amounts of images in public and private repositories also leads to the need for content-based search and retrieval solutions (CBIR) (Manning et al. 2009).

Despite the fact that data outsourcing, especially to cloud computing infrastructures, seems a natural solution to support large scale image storage and retrieval systems, it also raises new challenges in terms of data privacy control. This is a consequence of outsourcing data, which usually implies releasing control (and some times even effective ownership) over it (Chow et al. 2009). Recent incidents have provided clear proofs that privacy should not be expected to be preserved by cloud providers (Greenwald and MacAskill 2013; Rushe 2013). Furthermore, malicious or simply careless system administrators working for the providers have full access to data on the hosting cloud machines (Chen 2010; Halderman and Schoen 2009). Finally, external hackers can exploit software vulnerabilities to gain unauthorized access to servers (National Vulnerability Database 2016). The recent incident with the iCloud image storage service and celebrity photo leakage (Lewis 2014) illustrates the danger these threats pose for cloud-based visual data stores.

The conventional approach to address privacy in this context is to encrypt sensitive data before outsourcing it and run all computations on the client side (Mahajan et al. 2011). However this imposes too much client-overhead, as data must continuously be downloaded, decrypted, processed, and securely re-uploaded. Many applications cannot cope with this overhead, particularly online and mobile applications operating over very large datasets such as image repositories with CBIR services. A more viable approach would be to outsource computations and perform operations over the encrypted data on the server side. Existing proposals in this domain remain largely unpractical, namely those requiring fully homomorphic encryption, which is still computationally too expensive (Gentry et al. 2012). Nonetheless, partially homomorphic encryption schemes (ElGamal 1984; Hsu et al. 2012; Paillier 1999; Zheng and Huang 2013) and symmetric-key solutions (or property-preserving schemes) supporting specific search patterns (Lu et al. 2009; Weng et al. 2015; Yuan et al. 2014) are interesting alternatives, yielding more practical results while providing a good tradeoff between security,

privacy, and usability. Unfortunately, even these solutions are too computationally complex for wide adoption, particularly regarding the support of privacy-preserving CBIR over large-scale, dynamically updated[1] image repositories. This prohibitive complexity is even further exacerbated if we consider mobile (resource constrained) clients, which are already responsible for more than 30% of internet traffic (Meeker 2015).

To address these challenges we propose a new secure framework for privacy preserving outsourced storage, search, and retrieval of large-scale, dynamically updated image repositories. We base our proposal on IES-CBIR, a novel Image Encryption Scheme (IES) with Content-Based Image Retrieval (CBIR) properties. Key to the design of IES-CBIR is the observation that in image processing, distinct feature types can be separated and encrypted with different cryptographic algorithms. As an example, image color and texture data can be separated in such a way that CBIR in the encrypted domain can be performed on one feature type while the other remains fully randomized and protected with semantically-secure cryptography. Following this observation, and considering that texture is usually more relevant than color in object recognition (Wang et al. 2001), in IES-CBIR we make the following security-oriented tradeoff: we choose to privilege the protection of image contents, by encrypting texture information with probabilistic (semantically-secure) encryption (Katz and Lindell 2007); then we controllably relax the security on color features, by using deterministic encryption on image color information. This methodology allows privacy-preserving CBIR based on color information to be performed directly on the outsourced servers with high security guarantees[2]. Notably, our solution allows outsourcing servers to generate and update an index used to efficiently process and reply to queries, a task that in many state of art solutions must be managed by client devices. As we show further ahead in the Chapter, our new methodology leads to optimized computation and communication overheads with non-negligible impact on system performance and mobile battery consumption. In summary, we provide the following contributions:

- We formally define IES-CBIR, a novel Image Encryption Scheme with Content-Based Image Retrieval properties, and propose an efficient construction that achieves its functionality;

[1] As in the previous Chapter, by dynamically updated we mean a repository where clients continually add, update, and remove images.

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

- We show how to design an outsourced image storage, search, and retrieval framework by leveraging IES-CBIR to avoid most heavy computations to be performed by the client (i.e. indexing of dynamically added/updated images), hence circumventing performance pitfalls that exist in current state-of.the-art proposals (Hsu et al. 2012; Lu et al. 2009; Weng et al. 2015; Yuan et al. 2014; Zheng and Huang 2013);

- We formally prove the security of our framework and IES-CBIR;

- We experimentally show that when compared with competing alternatives (Hsu et al. 2012; Lu et al. 2009), our framework provides increased scalability, performance (from user's perspective), and lower bandwidth consumption, allowing client applications to be increasingly lightweight and mobile;

- And finally we show that the retrieval precision and recall of the proposed solution is on par with the current state-of-art (Hsu et al. 2012; Lu et al. 2009).

4.2 Related Work

Previous proposals for supporting outsourced storage, search, and retrieval of images in the encrypted domain can be broadly divided in two classes: those based on Searchable Symmetric Encryption (SSE) techniques and those based on Public-Key partially-Homomorphic Encryption (PKHE). SSE has been widely used in the past by the research community, both for text (Curtmola et al. 2006; Hahn and Kerschbaum 2014; Kuzu et al. 2012) and image (Lu et al. 2009; Weng et al. 2015; Xia et al. 2015; Yuan et al. 2014) search/retrieval. In SSE-based solutions, clients process their data before encrypting and outsourcing it to the cloud. From this processing, an index is created, encrypted, and stored in the outsourced infrastructure, which allows clients to search their data efficiently and in a secure way. Data is typically encrypted with probabilistic symmetric-key encryption schemes, while the index is protected through a combination of probabilistic and deterministic (or even order-preserving (Popa et al. 2013)) encryption. Unfortunately, SSE-based approaches in general share the following limitations:

- Clients either require a trusted proxy (Yuan et al. 2014) or have to index their images (and encrypt that index) locally (Lu et al. 2009; Xia et al. 2015), which entails the use of additional computational power on their side and

limits the practicality of such solutions for resource-constrained and mobile devices. This effect is further exacerbated when considering dynamic scenarios, where images are constantly being added, updated, and removed. In such dynamic scenarios, SSE solutions usually require multiple rounds of communication for updating image repositories and their indexes. For instance, a previous approach by Lu et al. (Lu et al. 2009) uses repository-wide statistics (e.g. inverse-document frequencies), which change as the repositories are updated and thus force the re-construction and re-encryption of the index, requiring clients to download and decrypt the full contents of the repository. Additionally index values are encrypted with an order-preserving encryption scheme that depends on plaintext domain distribution. With multiple updates this distribution changes, again requiring the re-construction and re-encryption of the index. This is an important issue from a security viewpoint. Other approaches from the literature require multiple rounds of communication for performing such operations (Kuzu et al. 2012; Xia et al. 2015; Yuan et al. 2014);

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

- As SSE works use deterministic tokens to provide their functionality with practical performance. Deterministic tokens include unique document identifiers and deterministic encryptions of keywords. Consequently they leak what are known as *search*, *access*, *similarity*, and *update* patterns (Curtmola et al. 2006; Hahn and Kerschbaum 2014; Kuzu et al. 2012; Yuan et al. 2014), i.e. they reveal respectively: if a query has been submitted before; which images are returned for each query; which images are similar to a given query image (in case of similarity/ranked search); and which images (previously searched) are similar to a new image being inserted. These leakage patterns result in exposing as much information as a fully deterministic encryption scheme, albeit with much higher computational overhead. This is demonstrated in (Islam et al. 2012) and is particularly evident in long-lived system with many queries being executed concurrently and all index entries being accessed. Nonetheless, the reader should note that deterministic schemes (and SSE-based schemes with the referred leakages) can still be provably-secure, as long as the higher-level applications leveraging them control the

amount of background information leaked to adversaries (including plaintext distribution knowledge) (Islam et al. 2012).

The alternatives to SSE that can be found in the literature (Hsu et al. 2012; Zheng and Huang 2013) are based on public-key partially-homomorphic encryption (PKHE) schemes such as Paillier (Paillier 1999) or ElGammal (ElGamal 1984), which allow additions and multiplications on the encrypted domain, respectively. In these approaches, clients encrypt images pixel by pixel with a PKHE scheme, allowing the cloud to process and index encrypted images on their behalf and thus avoiding many of the practical issues of SSE-based solutions. Unfortunately, PKHE works present much higher time and space complexities. For instance, Hsu et al. (Hsu et al. 2012) proposed a high-precision CBIR algorithm in the encrypted domain, by resorting to the Paillier cryptosystem (Paillier 1999). However, their approach results in significant ciphertext expansion (for a secure key size of 1024 bits, each pixel is transformed from its traditional 24 bits representation into 2048 ciphertext bits), slow encryption and decryption times (as we will experimentally demonstrate in Evaluation Section 4.5.1), and in limited scalability (the "ciphertext blowup" problem (Troncoso-Pastoriza and Perez-Gonzalez 2013), i.e. when ciphertext values reach their arithmetic group limits through multiple multiplications). Furthermore their work was later shown to be either insecure or computationally intractable for a typical cloud server (Schneider and Schneider 2014). Zheng et al. (Zheng and Huang 2013) proposed a variant of that work, overcoming some of its drawbacks by replacing Paillier ciphertexts with pointers to a ciphertext table with all possible ciphertext pixel values. This approach can potentially reduce the number of encryption operations and minimize ciphertext expansion in some use cases. However Paillier encryptions still present a significant computational overhead, limiting the practicality of the approach (Zheng and Huang 2013).

Aside from the SSE and PKHE research directions, there have been other works following similar approaches to what we propose in this Chapter, although for different purposes. An example is the work by Nourian et al. (Nourian and Maheswaran 2013) which aims at providing privacy-preserving single image template matching performed by third-party clouds. The work doesn't support large-scale repositories however, as it only allows linear searching, requires the template being matched to be re-encrypted for comparison with each different image in a repository, and requires the availability of public images as noise for encryption which can be easily found by an attacker using popular high-availability repositories for dictionary attacks (or by tracking users' traffic). Another example is the

more theoretical work by Chase et al. (Chase and Kamara 2010), which proposes a set of algorithms for the encryption of several data structures (including matrix-based datatypes such as images), while enabling queries to be performed over the ciphertext. Their main motivation is to extract partial information about a single encrypted data object (such as the color of a given pixel in an image). In our proposal we focus on allowing the generation of indexes over large collections of encrypted images by an untrusted third party and the efficient and precise resolution of user queries over these large collections.

The reader should also note that most research works on privacy-preserving image retrieval are not actually proven secure. In some works the underlaying cryptographic primitives used are well known and are used as intended (Lu et al. 2009), meaning that their security correctness may be easy to infer despite the lack of a formal security analysis. However, other approaches rely on small modifications that can actually compromise security. Such is the case of (Hsu et al. 2012) and (Nourian and Maheswaran 2013), as discussed in the previous paragraphs, and (Weng et al. 2015) which bases its security solely on hashing functions which can be compromised through dictionary attacks (Katz and Lindell 2007).

Table 4.1 summarizes key aspects of the state of art, by comparing our work with the most relevant approaches from SSE (Lu et al. 2009) and the PKHE (Hsu et al. 2012) research contexts in terms of information leakage and computational complexity for clients. We also implemented these works and will experimentally compare them with a prototype of our framework in Section 4.5. In the Table, the *Information Leakage* column represents the leakage of all system operations (particularly the update, search, and remove operations) as a whole; *Local Index Size* represents a maximum bound on the possible index size on the clients' side; and the CBIR Algorithm column represents the CBIR algorithms used in each work: local color histograms (Lu et al. 2009), SIFT (Lowe 2004), and global color histograms (Swain and Ballard 1991).

In the Table 4.1, ID_I is a deterministic identifier of an image I being stored/updated or being searched for as query image; put(x) and get(x) represent the complexity of respectively, sending and retrieving data item x to/from the server; FE_I is the *Feature Extraction* of I and fv is the extracted *Feature-Vector*; vw_I are the visual words of I, resulting from its clustering (more details on this operation in Section 4.4.2); E_S represents encryption with scheme S and C_P is the resulting ciphertext when applied to plaintext P; D is the decryption operation; Idx is the index; $|vw|$ is the total number of visual words in the repository; $|Rep|$ is the number of images in the repository; and $|CB|$ is the size of the clustering codebook.

Scheme	Information Leakage	Search Time	Update Time	Local Index Size	CBIR Alg.
SSE (Lu et al. 2009)	$ID_I + ID_{vw_I}$	$O(FE_I+Cluster_{fv_I}+\text{put}(vw_I))$	$O(E_{AES}(I)+\text{put}(C_I)$ + $\text{get}(Idx)$ + $D_{OPE}(Idx)+FE_I+Cluster_{fv_I}+$ $Update_{vw_I}(Idx)+E_{OPE}(Idx)+\text{put}(Idx))$	$O(\lvert CB\rvert+\lvert vw\rvert\times\lvert Rep\rvert)$	Local Color
PKHE (Hsu et al. 2012)	$ID_I+size_I+ID_{vw_I}$	$O(E_{Paillier}(I)+\text{put}(C_I))$	$(O(E_{Paillier}(\text{I})$ + $\text{put}(C_I))$	–	SIFT
This Work	$ID_I+size_I+ID_{vw_I}$	$O(E_{IES-CBIR}(I)+\text{put}(C_I))$	$O(E_{IES-CBIR}(I)$ + $\text{put}(C_I))$	–	Global Color

Table 4.1: Overview of information leakage and average complexities (on the client-side) for the most relevant privacy-preserving CBIR approaches and IES-CBIR.

4.3 System Overview

Overcoming the limitations of the state of art, we propose a framework for privacy-preserving outsourced storage, search, and retrieval of images in large-scale, dynamically updated repositories. Our framework is composed of two main components: an image encryption component, executed on client devices; and a storage, indexing, and searching component (in the encrypted domain), executed in the outsourcing server (e.g. a cloud provider). We base this framework on a new encryption scheme specifically designed for images, called IES-CBIR, which allows us to design outsourced image repository systems that support content-based image retrieval (CBIR) based on color features, while protecting the privacy of both image owners and other users issuing queries. Regarding the state-of-art, IES-CBIR shows comparable retrieval precision and higher computational performance than previous approaches as perceived by clients, since it securely moves indexing computations to the cloud provider's infrastructure and avoids public-key and homomorphic cryptography. IES-CBIR also minimizes ciphertext expansion and consequently bandwidth and outsourced space requirements, reinforcing the positive impact on user-perceived latency. These benefits are further illustrated in our experimental analysis in Section 4.5, where the performance of a IES-CBIR system is compared against the state-of-art SSE (Lu et al. 2009) and PKHE (Hsu et al. 2012) based approaches.

For the remainder of the Chapter we use the following terminology: a *repository* is a collection of images which is stored in the infrastructure of a cloud provider; the *cloud server*, or just *cloud*, is the outsourcing infrastructure that acts as a server both for storage and computation over images; *users* are the clients of our system, possibly using lightweight mobile devices, where each user accesses one or more repositories to search, add, and update images at any time; *repository keys* are secret cryptographic keys that are used to search, add, and update images in the repositories (each repository has its own repository key); *image keys* are secret keys used for encrypting and decrypting images in the repositories, in conjunction with the respective repository keys.

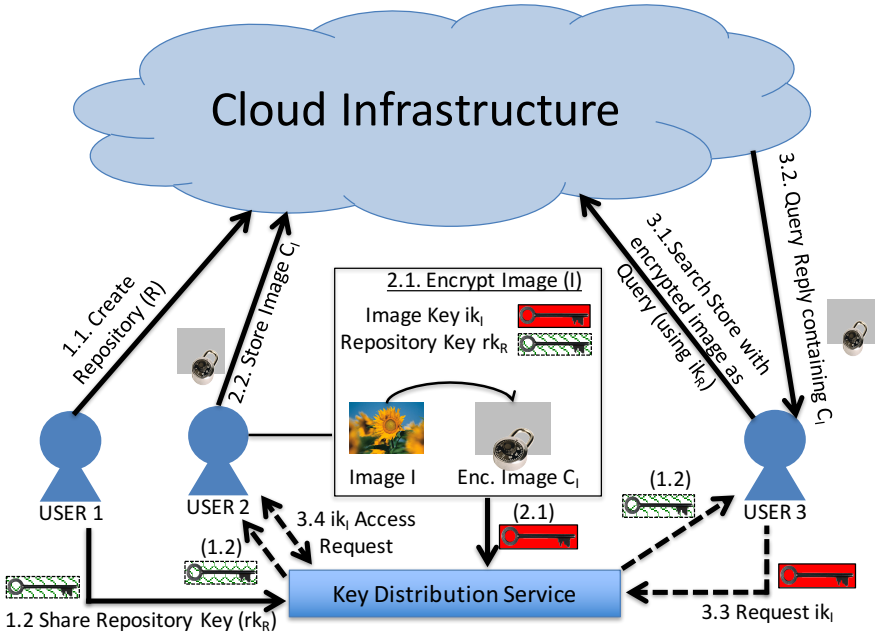

Figure 4.1: System model overview of the proposed framework.

In the following subsections we present the system model for our proposed framework (Sections 4.3.1), followed by its adversary model and security assumptions (Sections 4.3.2) and by some relevant use cases we envision for the application of our proposal (Section 4.3.3).

4.3.1 System Model and Architecture

We now describe the system model and architecture envisioned for using our framework and IES-CBIR. In this model, we consider two main entities: the *cloud* and (multiple) *users* (Figure 4.1). Images are outsourced to *repositories* that reside in the cloud. Each repository is used by multiples *Users*, where they can both add their own images and/or search using a query image. Users can also request access to stored images from their creators/owners. Our objective is to ensure the privacy of users, hence all data sent to the cloud is encrypted.

Each repository is created by a single user. Upon the creation of a repository, a new *repository key* is generated by that user and then shared with other trusted users, allowing them to search on the repository and add/update images. To add/update images (but not search), a user further needs an *image key* generated for that image. Image keys are kept secret by their users, meaning that even users capable of searching in a repository (i.e. with access to the repository key) will need to ask the owners of specific images for access to them. Note that using

specific keys per-image should be seen as an option in our framework, i.e. if the users of a repository prefer to avoid further key management overhead and are willing to sacrifice fine-grained access control, they can use the same image key for all images in a repository.

When the cloud receives an encrypted image for storage it extracts its relevant features (in our framework, we use global color features (Swain and Ballard 1991)) and indexes the image based on these features. The same action is performed for a query image, which after being encrypted by a user with a repository key, is then processed by the cloud and has its features extracted and matched with the repository's index. The reply to a query will contain k (a tunable system parameter) number of encrypted images and respective metadata, which include each image's id and the id of the user that owns each of the images. To fully decrypt and access the contents of an image, besides the repository key, the querying user will further require the image key for that specific image.

It should be noted that all key sharing interactions can be done by resorting to an orthogonal key distribution service, implemented either in a centralized way (using protocols such as Kerberos (Neuman and Ts'o 1994) or in a distributed fashion (through asynchronous communications or protocols such as Diffie-Hellman (Katz and Lindell 2007)). User authorization and revocation can also be easily achieved, for instance, through the sharing (and refreshment when user revocations are issued) of repository-specific tokens between trusted users, and its request in the framework operations. Nonetheless, we find these discussions to be orthogonal to the main focus of this contribution, as the mechanisms involved can be easily integrated into our framework.

4.3.2 Adversary Model

In this work we focus at protecting the privacy of users' images and queries. The main adversary we consider is the *cloud administrator*, who operates the cloud's infrastructure and servers. Similar to many previous works found in the literature (Curtmola et al. 2006; Hahn and Kerschbaum 2014; Kuzu et al. 2012; Lu et al. 2009; Popa et al. 2013; Yuan et al. 2014), we assume an *honest-but-curious* adversary (Chow et al. 2009), that is, the cloud is seen as a passive adversary that is expected to correctly perform operations when asked (i.e. fulfill its contract agreements), but may eavesdrop and disclosure users' data. We assume that a malicious cloud administrator has access to all data stored on disk or in RAM on any device located at the cloud infrastructure, and passing through the network from or to the cloud. In Section 4.4.4, we formally prove the security of our framework

against such an adversary.

A stronger adversary that should also be considered is the *malicious user*, i.e. a user of the system who deviates from his expected behavior. Malicious users are an open problem for any multi-user application, as they may be given access to multiple repository and image keys before being discovered, and can more easily eavesdrop on other users' images. In this work we focus on protecting our proposals and proving their security against the *honest-but-curious* cloud administrator, and leave the malicious users challenge as a future research direction. Nonetheless, we acknowledge that different orthogonal mechanisms can be deployed to mitigate the challenges introduced by malicious users, including access control techniques, repository access revocation, and periodic key refreshment (Curtmola et al. 2006). Furthermore, we do not consider integrity or availability threats, as these can be handled by different mechanisms orthogonal to our contributions (Brandenburger et al. 2015; Kim and Lie 2015; Shraer et al. 2010).

4.3.3 Relevant Use Cases

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

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

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

effort. Doctors can then perform search operations on these repositories, and indirectly request the image keys to PHRs that might be of their interest, through the physician following the patient to whom those records belong to and with her authorization.

Storage for Mobile Users Existing studies have shown that Internet users are increasingly mobile (Meeker 2015). Since mobile clients usually have limited computational and storage resources, they tend to rely on cloud services for storing and processing bulky data such as images. In this scenario, mobile clients (*users*) want to delegate their private image repositories storage to a *cloud provider*, while copping with the limitations of their device's storage capability, computational power, and battery life. Additionally, clients might be interested in allowing their images to be searched (and eventually accessed) by other users (either friends, family, or co-workers). Privacy can be relevant for instance when a user has access to sensitive material. Additionally, one might imagine that some companies could have interest in accessing the images owned by a given user, for instance when performing background checks on prospective new employees, among other cases.

4.4 A Privacy-Preserving CBIR Framework

In this Section we present the design and details of our proposed framework. We start by formally defining IES-CBIR and its algorithms (Section 4.4.1). Then we explain how the framework's component in the cloud side leverages IES-CBIR properties to store, index, and search images while preserving their privacy (Section 4.4.2). Finally we detail the system protocols of our framework (Section 4.4.3) and formally prove its security (Section 4.4.4).

4.4.1 IES-CBIR Design and Implementation

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

Informally, we define image privacy as the ability to keep the contents of an image secret to public (or simply unauthorized) disclosure (Danezis and Gürses 2010). Generally speaking, image contents are characterized by the combination of its color and texture information. These two components form what one can

readily identify in an image: objects, people, etc. As such, to safeguard image privacy entails preventing unauthorized entities from recognizing objects in those images. We further remark that image color and texture informations can be separated from each other. Indeed, color information is given from pixel color values in the different channels of some color models; while texture information is given by the (relative) position of pixels and strong color changes across neighboring pixels. We also remark that texture information is usually more relevant in images for object recognition (Wang et al. 2001). Finally, we conclude that no sub-component alone (i.e. color or texture information) can be used to infer the precise contents of an image, as color information on itself is usually ambiguous (e.g. strong blue can translate into sky, ocean, etc.) and texture information depends not only on pixel positions but also on their color values. These observations are further supported by the most recent works in image reconstruction (Weinzaepfel et al. 2011), which not only depend on local features extracted from sub-parts of the images (in this work we focus on global features extracted from each image as a whole), but also on those local features not being encrypted.

Leveraging the previous definition and remarks we design IES-CBIR, an image encryption scheme that separates color from texture information, applying different encryption techniques for protecting each. Emphasizing that texture is usually more relevant than color for object recognition (Wang et al. 2001), we design IES-CBIR to protect image texture with probabilistic encryption and color information with deterministic encryption. This way, content-based image indexing and retrieval, based on color information, can be performed on the cloud servers in a privacy-preserving way and without intervention of users, while texture information remains protected with the highest level of security (we provide a detailed and formal security evaluation in Section 4.4.4). We define IES-CBIR as:

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

• GenRK*(sp_{rk}): is a probabilistic algorithm that takes as input the security parameter $sp_{rk} \in \mathbb{N}$ and generates a repository key rk;*

• GenIK*(sp_{ik}): is a probabilistic algorithm that takes as input the security parameter $sp_{ik} \in \mathbb{N}$ and generates an image key ik;*

• Enc*(I, rk, ik): takes as input an image I and the cryptographic keys $\{rk, ik\}$, returning an encrypted image C_I;*

• Dec(C_I, rk, ik): *takes as input an encrypted image* C_I *and keys* $\{rk, ik\}$*, returning the decrypted image* I*;*

• TrpGen(Q, rk): *takes as input a query image* Q *and a repository key* rk*, returning a searching trapdoor* C_Q*;*

Key Generation As already discussed (in Section 4.3.1) IES-CBIR works with two different types of cryptographic keys, repository keys (rk) and image keys (ik), which are generated by the GenRK and GenIK algorithms respectively. Repository keys deterministically map a pixel's color value in a color channel to some new random value[4]. To prevent images from increasing in size after encryption (i.e. prevent ciphertext expansion), encrypted pixels should be in the same range of values as their original plaintexts (usually 8 bits per color channel). As such, we build repository keys in IES-CBIR by performing random permutations of all possible pixel color values in each color channel. Leveraging the HSV color space ((H) hue, (S) saturation, (V) value/brightness), we perform three independent random permutations of the values in range [0..100]. This range represents all possible color values in the HSV color space, and each permutation is used for a different color channel, resulting in 3 repository sub-keys: rk_H, rk_S, rk_V. Permutations are performed by a Pseudo-Random Generator (PRG) (Katz and Lindell 2007) $\mathcal{G}$ parameterized with the security parameter sp_{rk} as random seed (in our implementation we use an AES-based PRG (Katz and Lindell 2007) for $\mathcal{G}$ and 128 bits for sp_{rk}). Besides limiting ciphertext expansion, this approach allows image processing operations to be executed in the encrypted domain without alterations, including image indexing, searching, and compressing operations.

$$rk_z \leftarrow RandPerm(\mathcal{G}_{sp_{rk}}, [0..100]) : \ \forall z \in (H, S, V),\ rk = \{rk_H, rk_S, rk_V\} \tag{4.1}$$

Equation 4.1 formalizes the algorithm for generating repository keys. In contrast, image keys are generated by requesting a number of pseudorandom bits to $\mathcal{G}$ (initialized with some random seed) equal to sp_{ik} (we use 128 bits in our implementation). Image keys will be used as a cryptographic seeds for the probabilistic encryption step of IES-CBIR.

Encryption Image encryption in IES-CBIR is achieved through two main steps and a final (optional) step: i) pixel color values encryption, ii) pixel positions permutation, and iii) image compression. The goal of the first step is to protect

[4]Instead of encrypting pixel color values with deterministic encryption, we could also use Order-Preserving Encryption (Popa et al. 2013), slightly increasing image retrieval precision at the expense of greater information leakage.

image color features, through the application of a Pseudo-Random Permutation (PRP) (Katz and Lindell 2007) $\mathcal{P}$ on all pixel color values. Although we could use a standard PRP construction to instantiate $\mathcal{P}$ (such as an AES-based PRP (Katz and Lindell 2007)), we chose to conceive a specific color-domain PRP, allowing us to preserve the format of encrypted images. Our construction encrypts pixel color values by deterministically replacing them, in each color channel, using repository key $rk = \{rk_H, rk_S, rk_V\}$. Equation 4.2 represents this operation, where $\mathcal{P}_{rk}(p_z)$ is the encryption of pixel p in color component z through $\mathcal{P}$ and key rk_z, and c_{p_z} is the resulting ciphertext.

$$c_{p_z} \leftarrow \mathcal{P}_{rk_z}(p_z) : \forall z \in (H, S, V),\ \forall p \in I,\ \forall c_p \in C_I \tag{4.2}$$

This step of encryption securely hides color values of encrypted pixels. However, due to the deterministic properties of $\mathcal{P}$ (a requirement to enable CBIR in the encrypted domain), patterns present in the original image (which denote its texture) will remain visible. To fully protect image contents, we rely on a second probabilistic step in our encryption algorithm: (pseudo)random pixel position permutation, through pixel rows and columns shifting. In this step a PRG $\mathcal{G}$ is instantiated with a previously generated image key ik (operation GenIK above) as cryptographic seed. Then for each pixel column we request from $\mathcal{G}$ a new pseudorandom value r between 1 and the image height, shifting that column r positions downward, overflowing to its beginning. After all columns have been randomly shifted, we repeat the procedure for the rows (with random values ranging between 1 and the image width). Equations 4.3 and 4.4 formally describe this step, where w and h are, respectively, the width and height of image I. Note that this encryption algorithm has no ciphertext expansion (i.e, after encryption the image has the same width and height as before).

$$C_I(x, y) \leftarrow C_I(x, (y + r)\ mod\ h) : \forall x \in \{1, .., w\}, \forall y \in \{1, .., h\} \tag{4.3}$$

$$C_I(x, y) \leftarrow C_I((x + r)\ mod\ w, y) : \forall x \in \{1, .., w\}, \forall y \in \{1, .., h\} \tag{4.4}$$

The above step is probabilistic, as each new image will have a new pseudorandomly generated ik, even if the same image is stored multiple times with different names (if the same image key is used for all images, then a random iv must also be used as input to $\mathcal{G}$). Moreover, this step effectively hides existing texture patterns in the image, making it computationally unfeasible to extrapolate correlations between plaintext and ciphertext. We choose to shift rows and columns, instead of pseudorandomly permuting all single pixel positions, because its more efficient (only $w + h$ pseudorandom values are required instead of $w \times h$) and we obtain

similar robustness against cryptanalysis even for images as small as 16×16 pixels (see security analysis in Section 4.4.4).

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

Decryption The decryption algorithm applies the different steps of encryption in the inverse order, or more formally, through the ordered application of the transformations denoted by Equations. 4.5, 4.6, and 4.7 (after decompressing the ciphertext if required). Note that the r random values must be generated in the same order as in the encryption.

$$C_I((x+r)\ mod\ w, y) \leftarrow C_I(x,y) : \forall x \in \{1,..,w\}, \forall y \in \{1,..,h\} \tag{4.5}$$

$$C_I(x, (y+r)\ mod\ h) \leftarrow C_I(x,y) : \forall x \in \{1,..,w\}, \forall y \in \{1,..,h\} \tag{4.6}$$

$$p_z \leftarrow \mathscr{P}_{rk_z}(c_{p_z}) : \forall z \in (H,S,V),\ \forall p \in I,\ \forall c_p \in C_I \tag{4.7}$$

Searching Trapdoor Generation The TrpGen algorithm generates searching trapdoors that users can leverage to search over image repositories. Trapdoor generation requires a query image Q as input, as well as the repository key rk. This means that users with access to rk will be able to access color values of all images stored in that repository. However, users can't access texture information (and hence full image contents) without the corresponding image keys, and can't use rk to search other repositories. Given rk, the TrpGen algorithm operates in a similar fashion to the Enc algorithm (Equation 4.8, where the image key is substituted by a new ik randomly generated for the query). This means that searching trapdoors are also decryptable, and can be stored in the repositories as new images as long as users locally save the image keys generated for the queries.

$$Trp(Q, rk) \leftarrow Enc(Q, rk, ik) \tag{4.8}$$

4.4.2 CBIR in the Encrypted Domain

On the cloud's side, the received encrypted images are processed and indexed for CBIR before being persistently stored. IES-CBIR enables these operations (for

color features) to be performed over their ciphertexts, using algorithms that operate on non-encrypted images and without requiring any modifications. Encrypted image processing has two main steps: feature extraction and feature indexing.

Feature extraction consists in processing an image and extracting a reduced set of feature vectors that describe it. In this work we focus on color features in the HSV color model and their representation as color histograms. For each encrypted image and each HSV color channel, the cloud server builds a color histogram by counting the number of pixels in each intensity level. This yields 3 color histograms with 101 entries each.

Upon extracting these features, the cloud can perform feature indexing to speedup query execution. In this work, we use the Bag-Of-Visual-Words (BOVW) representation (Nistér et al. 2006) to build a *vocabulary tree* and an *inverted list index* for each repository. We choose this approach for indexing as it shows good search performance and scalability properties. In the BOVW model, feature-vectors are hierarchically clustered (for instance, using the *k-means* algorithm (Nistér et al. 2006)) into a vocabulary tree (also known as *codebook*), where each node denotes a representative feature-vector in the collection and leaf nodes are selected as the most representative nodes (called *visual words*). This clustering step requires a training dataset, so in the prototype implementation of our framework based on IES-CBIR, we request an initial image collection from users when creating a new repository. After the creation of the codebook, additional images can be stored dynamically by hierarchically stemming them against it. This stemming returns the closest visual words to the image, according to some distance function (in our prototype we use the Hamming Distance). Finally, the cloud server builds an inverted list index, with all visual words as keys and the list of images most close to them (plus a frequency score) as values. This type of list is known as a *Posting List* (Manning et al. 2009).

After processing and indexing encrypted images, the cloud server can receive search requests from users, through the submission of search trapdoors for some query images of their choice. When a new search trapdoor is received, the cloud server extracts its color feature-vectors and finds their closest visual words by stemming them against the codebook. The query's visual words are used to access the repository's index, obtaining the corresponding posting lists in the process. Then, for each image referenced in at least one posting list, a search score is calculated for that image (in our implementation we use a "scaled tf-idf" scoring function (Lu et al. 2009)). Finally, the cloud returns the top k images to the user, according to their scores (k is a configurable parameter). The BOVW approach guarantees that only the most relevant images (a fraction of the repository) have

Algorithm 4.1 Operation Create New Repository.

procedure User(ID_U).CreateRepository(ID_R, sp_{rk}, sp_{ik}, n, m, $\{ID_{I_i}, I_i\}_{i=0}^{d}$)
  $rk_R \leftarrow$ IES-CBIR.GenRK(sp_{rk})
  for all $\{ID_{I_i}, I_i\}_{i=0}^{d}$ **do**
    $ik_{I_i} \leftarrow$ IES-CBIR.GenIK(sp_{ik})
    $C_{I_i} \leftarrow$ IES-CBIR.Enc(I_i, rk_R, ik_{I_i})
  Cloud.CreateRepository($ID_R, n, m, ID_U, \{ID_{I_i}, C_{I_i}\}_{i=0}^{d}$)
  return $\{rk_R, \{ik_{I_i}\}_{i=0}^{d}\}$

procedure Cloud.CreateRepository(ID_R, n, m, ID_U, $\{ID_{I_i}, C_{I_i}\}_{i=0}^{d}$)
  $Rep_R = \{ID_{I_i}, \{C_{I_i}, ID_{U_i}\}\}_{i=0}^{*} \leftarrow$ InitiateRepository()
  $Idx_R = \{ID_{vw_i}, \{ID_{I_j}, freq_{vw_i}^{I_j}\}_{j=0}^{*}\}_{i=0}^{n} \leftarrow$ InitiateIndex(n)
  for all $\{C_{I_i}\}_{i=0}^{d}$ **do**
    $fv_{C_{I_i}} = \{hist_H, hist_S, hist_V\} \leftarrow$ ExtractFeatures(C_{I_i})
  $CB_R \leftarrow$ ClusterFeaturesIntoCodebook($n, m, \{fv_{C_{I_i}}\}_{i=0}^{d}$)
  for all $\{ID_{I_i}, C_{I_i}, fv_{C_{I_i}}\}_{i=0}^{d}$ **do**
    $vw_{C_{I_i}} = \{ID_{vw_j}, freq_{vw_j}^{C_{I_i}}\}_{j=0}^{|vw_{C_{I_i}}|} \leftarrow CB_R$.Stem($fv_{C_{I_i}}$)
    for all $\{ID_{vw_j}, freq_{vw_j}^{C_{I_i}}\}_{j=0}^{|vw_{C_{I_i}}|}$ **do**
      $Idx_R[ID_{vw_j}]$.add($\{ID_{I_i}, freq_{vw_j}^{C_{I_i}}\}$)
    $Rep_R[ID_{I_i}] \leftarrow \{C_{I_i}, ID_U\}$

to be compared in the scoring step (key to ensuring scalability). After receiving search results, users can explicitly request full access to images by requesting the corresponding image keys from their owners.

4.4.3 Framework Protocols

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

Instantiate a new Repository. We start by describing the operation used by a user U to create a new repository R (Algorithm 4.1). On the user's side, the protocol takes as input the repository id (ID_R), the security parameters for the required keys (sp_{rk}, sp_{ik}), some initialization parameters (height m and leaf width n of the clustering codebook), and an initial collection of d images for the repository along with their user-defined ids ($\{ID_{I_i}, I_i\}_{i=0}^{d}$). In the protocol, the user starts by locally

Algorithm 4.2 Operation Store/Update Image.

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

generating a repository key rk_R for the repository, through the IES-CBIR.GenRK algorithm (line 2). Then, for each image I in the initial group of images, the user generates a new image key ik_I and encrypts the image with ik_I and rk_R (lines 3-5). The user then sends the initialization parameters, pseudorandom ids (including his own id) and encrypted images to the cloud server (line 6). The cloud starts by initializing the storage space Rep_R and index Idx_R for R (lines 9-10), and then extracts the color feature-vectors (histograms) of all the d initial images (lines 11-12). Then it hierarchically clusters these d feature-vectors, building codebook CB_R (line 13). Finally, it stems the feature-vectors against CB_R to determine their visual words representations, stores these and their frequencies in Idx_R, and stores each image (with its user id) in Rep_R (lines 14-18).

Store/Update Image. Algorithm 4.2 illustrates the procedure followed by a user U to store a new image I in repository R, or update it if it already exists. U is assumed to have access to R and rk_R. U starts with inputs ID_R, rk_R, image I and security parameter sp_{ik}. The algorithm is straightforward and basically consists in a sub-group of Algorithm 4.1's instructions (since Algorithm 4.1 also stores a group of initial images), where the only difference is the creation of codebook CB_{ID_R} in Algorithm 4.1. We point to the presentation of Algorithm 4.1 for a detailed explanation of each instruction.

Search with an Image as Query. Algorithm 4.3 sketches the procedure to search in a repository R with query image Q. The input for this operation on the user

Algorithm 4.3 Operation Search with Image as Query.

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

Algorithm 4.4 Operation Access Image.

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

side is ID_R, Q, repository key rk_R, and parameter k (the number of most similar results to be returned). User U starts by generating Q's searching trapdoor C_Q, through IES-CBIR.GenTrp algorithm (line 2). Then she sends it to the cloud server, along with k and ID_R, as parameters for the Search remote invocation (line 3). The cloud starts by extracting C_Q's feature-vector, stems it against CB_R to determine its visual words vw_{C_Q}, and accesses Idx_R with them to retrieve the respective posting lists PL_{vw} (lines 7-10). Then, for each image referenced in each of the posting lists retrieved, the cloud calculates its *scaled tf-idf score* (Lu et al. 2009) and adds it to the set of results for the query (lines 11-14). In this set, scores for the same image but different visual word are summed. Finally, the cloud sorts this set by descending score and returns the top k to the user (lines 15 to 17).

Access an Image. Algorithm 4.4 illustrates the protocol to access an encrypted image C_I previously returned by a search. This algorithm can be executed by a

Algorithm 4.5 Operation Remove Image.

```
1: procedure Cloud.Remove(ID_R, ID_I)
2:     Rep_R[ID_I] = {}
3:     for all PL_vw ∈ Idx_R do
4:         PL_vw.Remove(ID_I)
```

user after he has been given access to the image key ik_I by its owner, user ID_{U_I}. The protocol is a straightforward application of IES-CBIR.Dec algorithm, with inputs C_I, rk_R, and ik_I.

Remove an Image. Algorithm 4.5 shows the protocol for removing an image I from repository R. Since the algorithm is very simple, we only show the cloud computation part. The cloud server takes as input pseudorandom ids ID_R and ID_I, and starts by removing C_I from Rep_R (line 2). Then, for each posting list in Idx_R, it removes the reference and frequency for image I, if they exist (lines 3-4). In this protocol we assume again the presence of an authorization mechanism, which enforces that users can only remove their own images.

4.4.4 Security Analysis and Proofs

In this subsection we prove the security properties of our work by specifying an *idealized functionality* that our framework based on IES-CBIR should securely fulfill. Our security proofs follow the *real/ideal* paradigm that is conventional in secure multi-party computations (Canetti 2001).

Algorithm 4.6 formalizes the *ideal functionality* $\mathcal{F}$ of our framework. In $\mathcal{F}$ we consider as adversary the honest-but-curious cloud administrator (Section 4.3.2), which corrupts the cloud server passively. As stated in the Related Work Section 4.2, the leakage functions specified in Algorithm 4.6 are equivalent to the *search*, *access*, *similarity*, and *update* leakages of SSE-based works (Curtmola et al. 2006; Hahn and Kerschbaum 2014; Kuzu et al. 2012; Yuan et al. 2014), particularly for any long-lived system with many queries being executed as expected in real-world application scenarios. Furthermore applications using our framework can ensure that the information leaked will not compromise security, by limiting the amount of background information made available to adversaries (Islam et al. 2012).

The proof that our framework securely realizes $\mathcal{F}$ involves showing that a simulator $\mathcal{S}$, interacting with a user only through $\mathcal{F}$ (the ideal experiment), can simulate the view of the cloud server (i.e. the adversary) in a real interaction with

Algorithm 4.6 The ideal functionality of our framework, $\mathcal{F}$; all information leaked is specified here.

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

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

the user through an instance of our framework (the real experiment), and that the two experiments would be indistinguishable (apart from a negligible probability (Katz and Lindell 2007)), even when combined with the *adaptively* influenced inputs of the client. The essential rational that justifies our security properties is as follows: In the ideal functionality $\mathcal{F}$, when the user stores an image or sends it as a query to a repository, the server basically learns the frequency of its visual

words (and consequently its similarity to other images in the repository), but nothing more. In the real experiment, the user will invoke (through our framework protocols) the algorithms of IES-CBIR to achieve the same functionality. Thus, the crucial point in proving security is to show that IES-CBIR leaks no additional information to the cloud server beyond what is specified in $\mathcal{F}$.

Formally $\mathcal{S}$ can simulate the view of the cloud server randomly, based only on the size (number of images) of the repository. The only difference between this simulation and the real execution is the following: in the real execution there is a limitation on the size (in terms of pixel width and height) of the images being stored and searched. In fact, IES-CBIR algorithms (at least without padding) can only be proven computationally secure for images with at least 16×16 pixels of width and height, respectively. For images smaller than that, a Probabilistic Polynomial-Time (PPT) bounded adversary can compromise the probabilistic counterpart of IES-CBIR encryption in useful time. In the simulation, such limitation does not exist. As such, the proof must show that if this requirement on the size of images is respected by the user, security properties will hold and the real and ideal experiments will be indistinguishable.

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

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

- When it receives the CreateRepository message from $\mathcal{F}$ with its *Setup Leakage* $= \{ID_R, ID_U, d, n, m, \{ID_{I_i}, w_{I_i}, h_{I_i}, \{ID_{vw_j}, freq^{ID_{I_i}}_{ID_{vw_j}}\}^{|vw_{I_i}|}_{j=0}\}^{d}_{i=0}\}$, $\mathcal{S}$ initializes some data structures: *(i)* A simulated repository $Rep'_R = \{ID_{I_i}\{I'_i, ID_{U_{I_i}}\}\}^{*}_{i=0}$, which will simulate the contents of images as they are stored in the server; *(ii)* A simulated codebook CB'_R, with height m and leafs $\{ID_{vw_i}, vw'_i\}^{n}_{i=0}$, where vw'_i is a simulated visual word; *(iii)* a simulated index $Idx'_R = \{ID_{vw_i}, PL'_{vw_i}\}^{n}_{i=0}$, where $PL'_{vw} = \{ID_{I_i}, freq^{ID_{I_i}}_{ID_{vw}}\}^{*}_{i=0}$ is a simulated posting list of the images that contain vw' and respective frequencies; *(iv)* a simulated search list $Search'_R = \{ID_{Q_i}, Q'_i, \{ID_{vw_j}, freq^{ID_{Q_i}}_{ID_{vw_j}}\}^{|vw_{Q_i}|}_{j=0}\}^{*}_{i=0}$ that stores all performed queries; *(v)* a simulated removal list $Rem'_R = \{ID_I\}$ that stores the ids of removed images. Then, $\mathcal{S}$ creates an initial group of d simulated images. For each image I' in this group, $\mathcal{S}$ creates $w_I \times h_I$ uniformly randomly sampled pixels from the HSV color range ($H, S, V \in [0..100]$), fills I' with these, and

sets $R'[ID_I] = \{I', ID_U\}$. $\mathcal{S}$ also creates a simulated color feature-vector fv'_I, by extracting the color features of I'. Then $\mathcal{S}$ takes all simulated feature-vectors and performs hierarchical clustering, resulting in simulated codebook CB'_R with height m and leaf width n. CB'_R leaf nodes $\{ID_{vw_i}, vw'_i\}_{i=0}^{n}$ are used to fill Idx'_R keys. Finally, $\mathcal{S}$ stems each fv'_I against CB'_R, yielding $vw'_I = \{vw'_i, ID_{vw_i}, freq_{ID_{vw_i}}^{ID_I}\}_{i=0}^{|vw'_I|}$, and inserts $\{ID_I, freq_{ID_{vw}}^{ID_I}\}$ in $PL'_{vw} = Idx'_R[ID_{vw}]$, $\forall vw' \in vw'_I$.

Since the encryption algorithm in IES-CBIR has the pixels encrypted through two steps, first by a pseudorandom permutation (PRP) (Katz and Lindell 2007) of their color values and then by a random swapping of their pixel positions through a pseudorandom generator (PRG) (Katz and Lindell 2007), and due to the properties of these cryptographic primitives (PRPs and PRGs), I and I' will be computationally indistinguishable. Consequently, fv_I and fv'_I will also be indistinguishable, as well as vw_I and vw'_I, CB_R and CB'_R, and Idx_R and Idx'_R. However, due to the use of PRG in randomly shifting pixel rows and columns, the computational indistinguishability of I and I' will not only depend on sp_{ik} (security parameter of the *image key*), but also on the size of I in pixels width and height. More specifically, the computational complexity of a distinguisher $\mathcal{D}$, executed by $\mathcal{S}$, in distinguishing I from I' will be $w^h \times h^w$, as $\mathcal{D}$ has to resolve w random values, each with a possible value range of $[0..h]$, and h additional random values with a possible value range of $[0..w]$. If we consider 128 as minimum security bound for sp_{ik} (as recommended for AES encryption (Katz and Lindell 2007)), then $w^h \times h^w$ should be at least 2^{128}. Since an image of 16×16 pixels of width and height leads to $16^{16} \times 16^{16} = 2^{128}$, this represents the minimum security bound for I to be indistinguishable from I'.

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

respectively.

- When a query image Q is searched for with *Search Leakage* = $\{ID_R, ID_Q, k, w_Q, h_Q, \{ID_{vw_j}, freq_{ID_{vw_j}}^{ID_Q}\}_{j=0}^{|vw_Q|}\}$, $\mathcal{S}$ creates a simulated query image Q' with size $w_Q \times h_Q$. Then Q' is filled with pixels uniformly randomly sampled from the HSV color range ($H, S, V \in [0..100]$), and its simulated color feature-vector fv'_Q is extracted. $\mathcal{S}$ stems fv'_Q with CB'_R, getting visual words $vw'_Q = \{ID_{vw_i}, vw'_i, freq_{ID_{vw_i}}^{ID_Q}\}_{i=0}^{|vw'_Q|}$. Then $\mathcal{S}$ accesses $Search'_R$ and stores $\{Q', \{ID_{vw_j}, freq_{ID_{vw_j}}^{ID_Q}\}_{j=0}^{|vw_Q|}\}$ in position $[ID_Q]$. Since the search algorithm in IES-CBIR is based on IES-CBIR's Encryption algorithm, whose output was already proven indistinguishable from the simulated output, Q and Q' will also be indistinguishable as long as $w_Q, h_Q \geq 16$. Consequently, fv_Q and vw_Q will also be indistinguishable from fv'_Q and vw'_Q, respectively.

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

□

4.5 Implementation and Experimental Evaluation

In this Section we experimentally evaluate our proposal, comparing it with some of the most recent and relevant competing alternatives in the literature. To this end we have implemented, in the Java language, a prototype of our IES-CBIR based framework (as described in Section 4.4) and prototypes of the competing relevant alternatives: i) the SSE solution proposed in (Lu et al. 2009), based on Bag of Visual Words indexing and Order-Preserving Encryption (labelled *SSE* in the graphics presented in this Section); and (ii) a system leveraging the Paillier cryptosystem described in (Hsu et al. 2012) (labeled *PKHE*). The code of these prototypes is open source and available at: `https://github.com/bernymac/IES-CBIR`.

Using these prototypes we conducted an experimental evaluation of the performance and precision of our solution and the competing works. All experimental assessments were carried out using Amazon EC2 instances, both for user and cloud computations. To simulate geographic distance, user processes were executed in Oregon's data-center instances, while the cloud component was deployed

Figure 4.2: Example image from the Holidays dataset and its encryption with IES-CBIR.

in a North-Virginia's data-center instance. User instances, in our framework's testing scenario, were of the *general-purpose m3.medium* type and the cloud server was of the *m3.large* type (Amazon Web Services (AWS) 2016a). In the competing works testing scenarios user instances had to be increased to the *m3.large* type, as they have to perform heavier computations. For testing purposes, we used two image datasets: the Wang dataset (Wang et al. 2001), containing 1000 low-resolution images with a JPEG compressed size of 29.8 MB; and the Inria Holidays Dataset (Jegou et al. 2008), containing 1491 high-resolution images with total JPEG compressed size of 2.85 GB.

Figure 4.2 shows an example of an image from the Holidays Dataset and the result of its IES-CBIR encryption. We present our results in the following order: first we discuss the performance and scalability of our solution when storing and searching images, comparing it with the alternative approaches; then we study the achieved retrieval precision; finally we analyse the statistical entropy generated by IES-CBIR encryption algorithm.

4.5.1 Store/Update Performance

In these experiments we used the larger-sized Holidays dataset to analyse the performance of our system, with and without image compression (labeled *IES-CBIR w/ compression* and *IES-CBIR no compression* respectively) with the SSE and PKHE alternatives. To this end we have measured the time taken by the Update Image operation considering two distinct workloads: one where the 1491 (2.85 GB) images from the Holidays Dataset are used to create and populate a new image repository in the cloud, and another where after the initial upload of 1491 images, another 150 users try to upload 10 new images each concurrently (total

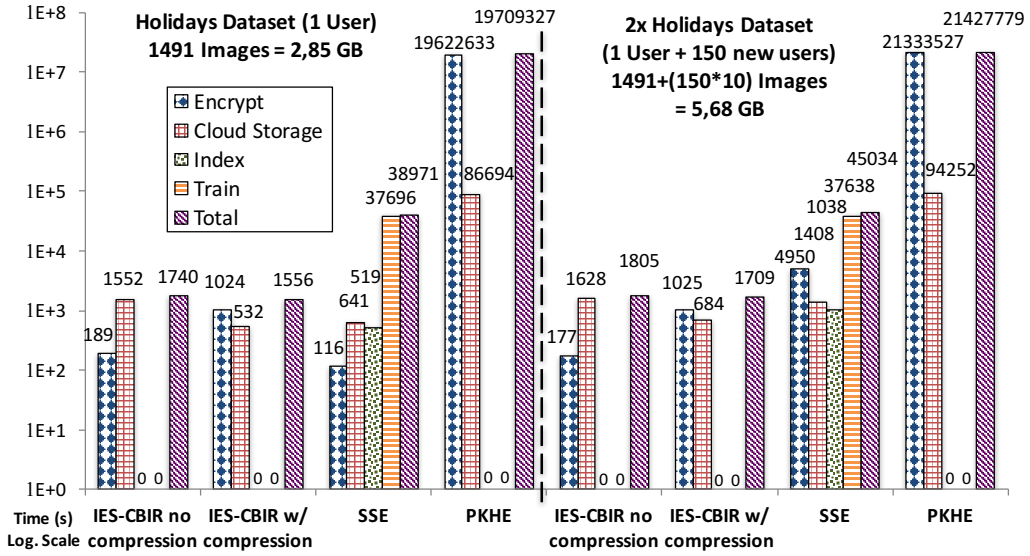

Figure 4.3: Performance for the *Store/Update Image* operation (log scale).

5.68 GB). This last workload allow us to show hidden overheads that emerge when updating repositories in some alternatives described in the literature, especially when multiple users try to store new images concurrently. Figure 4.3 summarizes the results for each system, in terms of time required for each sub-operation: *Encryption*, *Indexing*, *Training* (i.e. the hierarchical k-means clustering necessary for each repository, as described in Section 4.4.2) and *Cloud Storage*, as well as a whole (*Total*). The left part of the figure represents the first (static) workload, and the right part represents the second (dynamic) workload. Results are presented in a logarithmic scale and capture the elapsed time for each operation from the users' perspective (without considering cloud computations, which are performed asynchronously). Each experiment shows the average of 10 independent runs.

The results show that *IES-CBIR* with image compression offers overall better performance when compared with the remaining competing alternatives. This is a consequence of the very high cryptographic processing throughput presented by IES-CBIR, combined with the fact that users only have to encrypt images. In contrast, other alternatives either present very slow cryptographic throughput (*PKHE*), or additionally require indexing operations from the users (*SSE*). In more detail: IES-CBIR without compression presents slightly higher cryptographic throughput, but the performance gained there is somehow lost when the user has to upload larger decompressed images; *PKHE* shows prohibitive overheads both due to the low cryptographic throughput of the public-key Paillier cryptosystem and to its high ciphertext expansion (which then has to be uploaded to the cloud server); and *SSE* requires expensive initial training performed by the user that creates the repository, in order to build the clustering codebook required

for indexing, and also requires the user to locally index his images before uploading them. Results for the *PKHE* system had to be simulated from the results with a smaller subset (of 10 images, which took approximately 36 hours to encrypt and upload), as experiments would take approximately 228 days in the first experiment and 248 days in the second.

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

4.5.2 Search Performance

Figure 4.4 shows the experimental results for the Search Operation, comparing all approaches (IES-CBIR with and without compression, *SSE*, and *PKHE*) in a logarithmic scale (necessary because of the high overhead of the *PKHE* solution in comparison with the remaining approaches). The results showed here represent the performance for searching in the Inria Holidays dataset (Jegou et al. 2008) with a random image chosen from the collection as query (the results represent the average of 100 random runs each). The *Encrypt* and *Index* columns represent local processing done by the querying user, while the *Cloud* column represents not only the network time for transmitting the query and receiving its results, but also the time elapsed by the server in processing and calculating those results.

The results obtained show that IES-CBIR with compression achieves the overall best performance of all alternatives. Compared to the competing alternatives *SSE* and *PKHE*, IES-CBIR obtains an efficiency of 65% and 99,97% respectively. In the *SSE* case, the increased overhead is mainly due to the decryption of the search results, which are encrypted with an Order-Preserving Encryption scheme (Lu

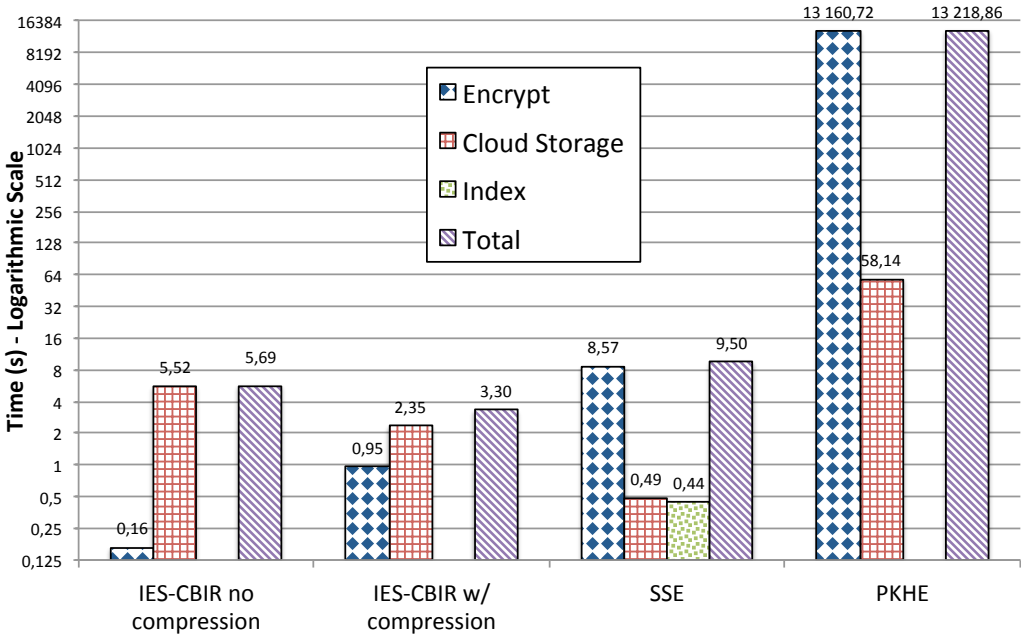

Figure 4.4: Performance of the *Search with Query Image* operation, for all analyzed alternatives (log_2 scale).

et al. 2009). Although IES-CBIR (in both variants) has higher cloud computation time, as most of the work involved in this operation is done by the cloud server instead of the user device (as in the *SSE* approach), that overhead is still smaller than the difference between the encryption overheads in the two systems. Moreover, overhead can be further reduced with (the less expensive) scaling of the cloud server's resources. In the *PKHE* approach, the high overhead is once again consequence of the low throughput and ciphertext expansion of the Paillier cryptosystem (Paillier 1999).

4.5.3 Retrieval Precision and Recall

We start by defining the metrics used herein (Müller et al. 2001): when a search is done, precision is the number of relevant images retrieved across all returned results; recall is the number of relevant images retrieved from all the relevant results for the query; average precision (AP) is the average of the precision measured each time a new relevant image is retrieved; and mean average precision (mAP) is the mean of APs for a group of queries.

To evaluate the retrieval precision that can be achieved with IES-CBIR, we extracted two metrics: an interpolated recall-precision graph, built with the Wang

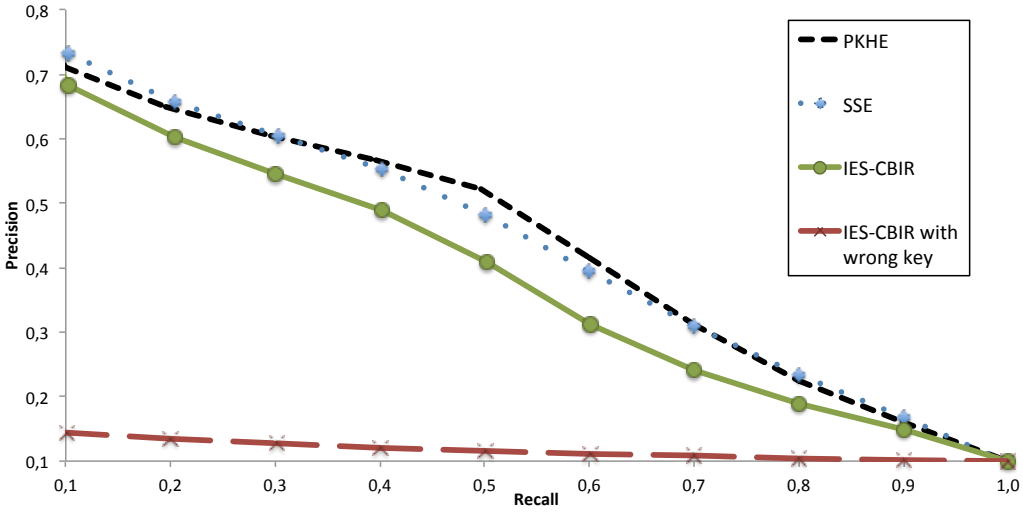

Figure 4.5: Precision vs Recall graph for the Wang dataset.

dataset (Wang et al. 2001) and all its images as queries; and the mAP of the Holidays dataset, for a group of queries pre-defined by the authors of the dataset (Jegou et al. 2008). Regarding the first experiment, we used a workload where each image in the dataset is used as query over all others in the repository. We then computed the average precision and recall, for all possible response sizes ($[1 \ldots 1000]$). Similarly to the previous Section, we compared the precision of IES-CBIR with it's competing alternatives, *SSE* and *PKHE*. We also assessed the precision that an adversary would achieve if he was to search in the repository with a randomly chosen repository key.

Figure 4.5 summarizes the results. Our framework shows similar precision and recall as the compared alternatives, with a small variation of about 6%. This small difference is the advantage gained by these alternatives through the sacrifice of performance and scalability. Nonetheless, the reader should note that our approach can be extended to also consider texture information in its CBIR algorithm, increasing retrieval precision at the expense of increased information leakage. Regarding the *IES-CBIR with wrong key* baseline, results show that a malicious user using the framework to search repositories with an incorrect repository key would achieve similar precision as if he was picking random images from those repositories.

The second precision test consisted in using the evaluation package of the Holidays dataset (available online (Jegou et al. 2008)) for calculating the mAP of a group of 500 pre-defined queries. Table 4.2 shows the results. In this experiment, *PKHE* achieved the best result, as expected due to the use of the SIFT retrieval algorithm (Lowe 2004). *IES-CBIR* achieved the second highest results, followed

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

Table 4.2: Mean Average Precision (mAP) for the Holidays dataset.

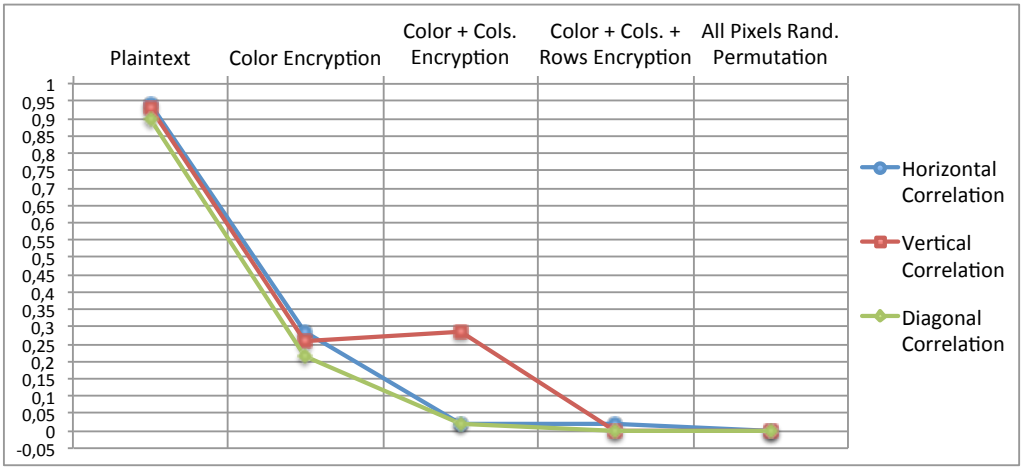

Figure 4.6: Average vertical, horizontal and diagonal correlation between all pixels of all images in the Wang dataset.

by *SSE*. Retrieval precision results for the *PKHE* system (in both experiments) were not substantially different from the other systems, even though it uses strong texture-based image features (in particular, SIFT). SIFT features were originally designed for object-recognition, and we believe that their use to search by example in image repositories (such as the ones used in our experiments and in the literature) does not leverage its full potential.

Comparing the results of the two experiments, we conclude that in some datasets IES-CBIR can actually achieve better precision than some of the competing alternatives, including personal photos and holidays datasets as in the Inria dataset (Jegou et al. 2008). Based on all results presented so far, we conclude that a framework leveraging IES-CBIR can achieve a good trade-off between precision/recall and performance/scalability.

4.5.4 Experimental Security Evaluation

To finalize the experimental Section of our work, we made a statistical analysis to experimentally assess the entropy level in IES-CBIR encrypted images. This experiment consisted in analyzing the correlation level between all horizontally, vertically, and diagonally adjacent pixels, for: original plaintext images, at different steps of IES-CBIR encryption process, and for a complete random permutation of all pixel positions. For this experiment we leveraged the correlation

function of (Nourian and Maheswaran 2013), where the obtained values range between $[-1 \ldots 1]$, and the images with higher entropy get closer to 0. We used the low-resolution Wang Dataset (Wang et al. 2001), proving that IES-CBIR can achieve high levels of entropy even for smaller images. All pixels of all images in the dataset were considered, being the average results presented in Figure 4.6. The first set of points in the figure represents the plaintext images; the second represents IES-CBIR color encryption only; the third is color encryption plus columns shifting; the fourth is color encryption plus columns and rows shifting (i.e. full IES-CBIR encryption); and the last point is random permutation of all pixel positions between each others.

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

4.6 Summary

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

Publications The work presented in this Chapter resulted in the following publications:

- **Towards an Image Encryption Scheme with Content-Based Image Retrieval Properties**. Bernardo Ferreira, João Rodrigues, João Leitão, and Henrique Domingos. In proceedings of the 9th Workshop on Data Privacy Management (DPM'14). Wroclaw, Poland, September 2014.
- **Privacy-Preserving Content-Based Image Retrieval in the Cloud**. Bernardo Ferreira, João Rodrigues, João Leitão, and Henrique Domingos. In proceedings of the 34th IEEE Symposium on Reliable Distributed Systems (SRDS'15). Montreal, Canada, September 2015.
- **Practical Privacy-Preserving Content-Based Retrieval in Cloud Image Repositories** (Technical Report). Bernardo Ferreira, João Rodrigues, João Leitão, and Henrique Domingos. Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Caparica, Portugal, December 2015 (Available in `http://asc.di.fct.unl.pt/%7Ebf/IES-CBIR`).

Prototypes A software prototype of the contribution is also available at:

- `https://github.com/bernymac/IES-CBIR`

Chapter 5

Multimodal Indexable Encryption for Mobile Cloud-based Applications

In this Chapter we present MIE, a Multimodal Indexable Encryption distributed middleware that allows searching encrypted multimodal data with both desktop and mobile devices. MIE provides privacy, efficiency, and scalability guarantees for client applications, by supporting multiple media formats simultaneously (i.e. multimodal data) and by outsourcing heavy indexing and training computations to the cloud in a secure way. MIE is based on a new cryptographic family of Distance Preserving Encodings (DPE) that we also propose, which securely encode data while preserving a controllable distance function between plaintexts. Distance functions are described as controllable since they are only revealed for a range of plaintext distance values defined by the user.

We start by providing the motivation and goals of the contribution. Then we survey the related work and discuss its limitations regarding our goals. The Chapter follows with the discussion of its system model, adversary model, and an application as case study. The main technical details are presented next, followed by discussion of experimental results.

5.1 Motivation and Goals

Mobile devices currently permeate everyday life, being responsible for more than 50% Internet traffic in some countries and surpassing the sales of PCs and Laptops

by six times (Meeker 2015). 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) using their mobile devices (Fung 2015). On the other hand, users expect to access and share their data in an ubiquitous way (Cisco 2016).

Due to resource limitations (computational power, battery life, and storage capacity) and to increasingly larger multimodal datasets produced and accessed by users (e.g. in cloud-backed multimedia storage apps (Apple 2016b; Google 2016d)), mobile devices have been a key driving factor for the outsourcing of data storage and processing, through solutions such as the ones provided by cloud computing (Meeker 2015). Such solutions effectively operate as a natural extension to the 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 privacy challenges, especially when data and computations are sensitive. This is a natural observation as outsourcing data and computations also entails outsourcing control over them (Chow et al. 2009). Recent news have proven that user's privacy is not protected when using cloud services (Rushe 2013). Governments impose increasing pressure on technological companies to disclose users' data and build insecure backdoors (Cook 2016; Greenwald and MacAskill 2013). Malicious or simply careless cloud system administrators have been responsible for critical data disclosures (Chen 2010; Frieden 2009; Halderman and Schoen 2009). Finally, one also has to consider internet hackers exploiting possible software and hardware vulnerabilities in the cloud providers' infrastructures (Lewis 2014; National Vulnerability Database 2016).

A common approach for dealing with these privacy concerns is to rely on end-to-end encryption schemes, where users' devices are responsible for encrypting all data before sending it to the cloud (Bessani et al. 2013; Mahajan et al. 2011; Shraer et al. 2010). 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 algorithms (Boneh et al. 2005a), searching encrypted data is a non trivial challenge.

Previous approaches in the literature have tried to address this challenge by proposing what are called Searchable Symmetric Encryption (SSE) schemes (Baldimtsi

and Ohrimenko 2015; Cash et al. 2014; Curtmola et al. 2006; Hahn and Kerschbaum 2014; Kamara and Papamanthou 2013; Kamara et al. 2012; Kuzu et al. 2012; Naveed et al. 2014; Popa et al. 2014; Song et al. 2000; Stefanov et al. 2014). Originally designed for text documents, SSE schemes allow searching encrypted data in sub-linear time, by having users index their data (i.e. build a compact dictionary of the data; e.g. with the unique keywords of each text document) and uploading both encrypted index and data to the cloud for storage. However indexing computations are still too expensive for mobile devices, especially for multimodal data and rich media types such as images, audio, and video where training (i.e. machine learning) tasks also have to be performed before data can be efficiently indexed (Datta et al. 2008). Furthermore, searching 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 (Curtmola et al. 2006), respectively). Finally, extending SSE to richer queries (Baldimtsi and Ohrimenko 2015; Cao et al. 2014; Kuzu et al. 2012; Wang et al. 2012) and other media domains (Lu et al. 2009; Weng et al. 2015; Yuan et al. 2014) has proven to be challenging. Existing works are limited to static collections (i.e. data-objects can't be added, updated, or removed dynamically after deployment and initial load of a data repository) and require heavier client processing, while leaking additional information patterns such as frequency (e.g. how many times a keyword appears in a text document) (Kuzu et al. 2012).

In this Chapter we propose a novel distributed middleware architecture to tackle head on the practical challenges of supporting mobile applications 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 when searching encrypted data in sublinear time has been proven unavoidable (Naveed 2015); 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 search and access patterns with each update/create operation, instead of each query. This will allow users to securely outsource indexing and training computations to the cloud, which we later show in our experimental results (Section 5.7) to be the heaviest computations and more unsuitable for mobile applications.

To support MIE's operations and enable the cloud to train and index multimodal data in a privacy-preserving way, 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. 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 architectures and protocols.

We implemented both an Android and Desktop applications on top of our MIE middleware prototype 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, extended, and evaluated a recent SSE scheme from the literature (Cash et al. 2014) to support multimodal searching and compare MIE's performance with it.

In summary, in this Chapter we present the following contributions:

- We propose an alternative design to searching encrypted data that allows the secure outsourcing of machine learning and indexing computations. We call our proposal MIE - Multimodal Indexable Encryption (Section 5.5).

- 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 (Section 5.4);

- We formally prove the correctness and security of our proposals under the standard security model, i.e. without resorting to heuristic models which may not have secure implementations in practice (Canetti et al. 2004; Goldwasser and Kalai 2015) (Section 5.4 and Section 5.5);

- We implement MIE, both for Desktop and Mobile (Android) devices (Section 5.6), and a multimodal SSE scheme based on a recent proposal (Cash et al. 2014), evaluating and comparing both in terms of performance and scalability across different operations (Section 5.7). Real-world datasets and public commercial clouds (Amazon EC2) are used in these experiments.

5.2 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 (Gentry 2009; Rivest et al. 1978b) and Oblivious RAM (Goldreich and Ostrovsky 1996; Stefanov et al. 2013). However such techniques still remain too expensive to be practical: for instance, computing an AES decryption circuit through fully homomorphic encryption is at least 10^9 times slower (Gentry et al. 2012); while developing a Searchable Symmetric Encryption (SSE) scheme on top of Oblivious-RAM, to protect search and access patterns, increases query data-transfer overhead by at least 128 times in comparison with recent SSE schemes and by at least 1.75 times in comparison to simply downloading the entire database with each search (Naveed 2015).

Searchable Symmetric Encryption (SSE) (Song et al. 2000) strives for a practical balance between efficiency and security. Originally designed for exact-match search over static collections of text documents with a single user, SSE schemes are able to achieve sub-linear search performance by not revealing any information regarding the encrypted data initially and then gradually revealing some information patterns with each search (Curtmola et al. 2006). 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 (Kamara et al. 2012; Naveed et al. 2014) (i.e. if new documents share contents with other documents, leaked by deterministic hashes of the document's keywords).

Recent dynamic SSE schemes were able to overcome the update leakage issue by increasing operational overhead (Kamara and Papamanthou 2013; Stefanov et al. 2014) and/or requiring client storage that grows linearly with the number of unique keywords (Cash et al. 2014; Hahn and Kerschbaum 2014). A recent work also introduced the concept of forward privacy (Stefanov et al. 2014), where old queries can not be reused by adversaries to immediately infer search and access patterns in dynamically added documents. 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 (Naveed et al.

2014), dynamic SSE schemes described so far depend on heuristic models which may not have secure implementations in practice (e.g. Random Oracles (Bellare and Rogaway 1993)) and that have been highly criticized in recent years (Canetti et al. 2004; Goldwasser and Kalai 2015). Making them secure under standard security assumptions requires further client processing and multiple communication rounds (Cash et al. 2014; Stefanov et al. 2014), 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 (Wang et al. 2012), or required heavy client processing (Kuzu et al. 2012) and search time linear with the index size (Cao et al. 2014). 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, simultaneously, the existence of a user-controlled cryptographic module in the cloud server, performing multi-party computation between that module and the cloud server, and encrypting the index with an additively-homomorphic encryption scheme (Baldimtsi and Ohrimenko 2015). 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 (Baldimtsi and Ohrimenko 2015; Hahn and Kerschbaum 2014; Yuan et al. 2014). Some schemes extend this model to support multiple searchers, however it must be a single writer to generate searching tokens for all other users (Cao et al. 2014; Kuzu et al. 2012; Wang et al. 2012). In (Popa et al. 2014), the first multi-key SSE 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.

Beside text documents, SSE-based schemes have also been designed for other media domains such as images (Lu et al. 2009; Weng et al. 2015; Yuan et al. 2014). 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. Furthermore, both training and indexing of dense media data are computationally intensive operations. We addressed some

Scheme	Search Time	Update Time	Client Storage	Index Size	Rev. Size	Query Type	Search Leakage	Update Leakage
Kamara et al. 2012	$O(m/n)$	$O(m/n)$	$O(1)$	$O(m+n)$	–	Text Match	$ID(w), ID(d)$	$ID(w)$
Kamara and Papamanthou 2013	$O(log\|F\|.m/n)$	$O(log\|F\|.n)$	$O(1)$	$O(\|F\|.n)$	–	Text Match	$ID(w), ID(d)$	–
Cash et al. 2014	$O(m/n)$	$O(m/n)$	$O(n)$	$O(m+n)$	$O(m)$	Text Match	$ID(w), ID(d)$	–
Cao et al. 2014	$O(n^2)$	$O(n^2)$	$O(1)$	$O(m+n)$	–	Text Ranked	$ID(w), ID(d)$	$ID(w), freq(w)$
Chapter 4	$O(m/n)$	$O(m/n)$	$O(1)$	$O(m+n)$	–	Image Ranked	$ID(w), ID(d)$	$ID(w), freq(w)$
MSSE	$O(m/n)$	$O(m/n)$	$O(n)$	$O(m+n)$	–	Multimodal	$ID(w), ID(d), freq(w)$	–
Hom-MSSE	$O(m/n)$	$O(m/n)$	$O(n)$	$O(m+n)$	–	Multimodal	$ID(w), ID(d)$	–
MIE	$O(m/n)$	$O(m/n)$	$O(1)$	$O(m+n)$	–	Multimodal	$ID(w), ID(d)$	$ID(w), freq(w)$

Table 5.1: Overview of average complexities for MIE and competing alternatives.

of these performance issues in Chapter 4, however our previous work was limited to color features in the image domain. Hence, and to the best of our knowledge, this contribution 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 5.1 provides a summary review of the recent literature on SSE and comparison with our approach and two multimodal SSE schemes (MSSE and Hom-MSSE) designed for baseline experimental comparison (by extension of a recent SSE scheme (Cash et al. 2014); more details in the Evaluation Section 5.7). In the Table, 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 keywords, $|F|$ 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. In this analysis we consider that repositories can start empty, i.e. all data-objects may be added dynamically.

5.3 System 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*, 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 (Mourão et al. 2014)); 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 (Manning et al. 2009), while the image modality by a set of visual points of interest (Bay et al. 2006)); *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 consists in separately searching data in different media formats, aggregating the multiple results through a merging function. This search is performed using a multimodal object as a query (Mourão et al. 2014).

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) (Manning et al. 2009). 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 (Hartigan 1975)) used to find homogeneous groups of objects in dense, high-dimensional data (Agrawal et al. 1998). These groups are later used to build more compact representations of high-dimensional data-objects (e.g. an object-recognition algorithm (Lowe 2004) will find multiple points of interest in an image). Training a collection of such keypoints from different images will yield a group of distinctive keypoints (Nistér et al. 2006). Representing the different keypoints of an image can then be achieved in a compact way by finding the most similar distinctive keypoint of each and building an histogram with their frequencies.

5.3.1 System Model and Architecture

We focus on the challenges inherent to building practical, secure, and searchable cloud-backed multimodal data repositories especially tailored for mobile devices. More generally, we consider a system with multiple readers and writers (which we call *Users*) who store, share, and search data through multiple independent repositories hosted by a *Cloud Server* (or simply *Server*). We assume that 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 5.1 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 store/update objects in that particular

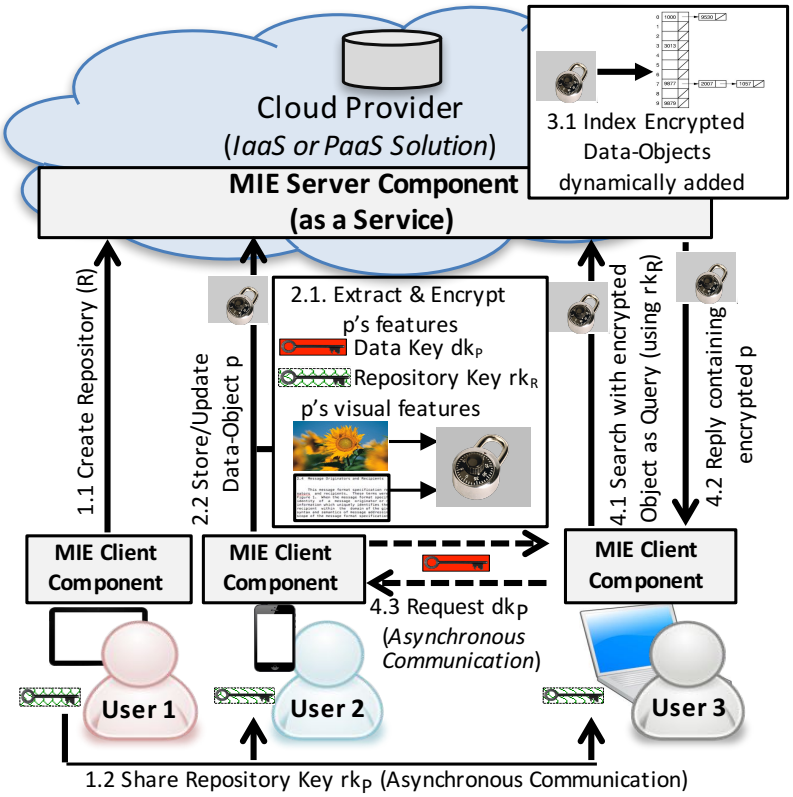

Figure 5.1: System model with example interactions between users and the cloud infrastructure, considering image and text media domains.

repository, being used in the indexing of new/updated objects and 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 (Katz and Lindell 2007)). 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 Section 5.4) 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, the querying user will still need to ask the object's owner for its data key, in order to fully access its contents).

All remote communications between users and the server should be encrypted and authenticated through secure communication protocols (TLS/SSL (Katz and Lindell 2007)). Key sharing interactions can be done asynchronously and out-of-band by resorting to broadcast encryption techniques (Boneh et al. 2005a) or a trusted key-sharing protocol based on public-key authentication such as Needham-Schroeder-Lowe (Katz and Lindell 2007). User authentication and access control can be achieved through different mechanisms found in the literature, such as sharing authorization tokens between trusted users (Curtmola et al. 2006). This discussion, however, is orthogonal to the main focus of the contribution as these mechanisms can easily be integrated into our solution.

5.3.2 Adversary Model

In this work we aim at protecting the privacy of users' data and queries. Similar to many previous approaches found in the literature (Cao et al. 2014; Curtmola et al. 2006; Hahn and Kerschbaum 2014; Kuzu et al. 2012; Lu et al. 2009; Naveed et al. 2014; Wang et al. 2012; Yuan et al. 2014), we consider as main adversary the *honest-but-curious cloud administrator*, who operates the cloud's infrastructure and servers and may eavesdrop on users' data, but 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 Chapter 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 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 this contribution (Brandenburger et al. 2015; Kim and Lie 2015; Shraer et al. 2010). Finally, we assume that the higher-level applications using our scheme can control the amount of background information they reveal, as this may be sensitive and can be leveraged by adversaries for breaking security (Cash et al. 2015). In Section 5.5.2 we discuss possible attack vectors on our work and how to mitigate their effectiveness.

5.3.3 Application Use Case

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

Personal Health Records. The number of mobile applications leveraging sensorial data for personal health tracking is growing by a large faction (Khalaf 2014). 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) (Microsoft 2016b). PHR 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 (i.e. the Users), or medical doctors on their behalf, outsource their PHR directly from their mobile 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. On the other hand, Repository Keys can be shared between medical doctors and centers, organized in alliance based and medical-specialty based repositories between cooperating professionals. Doctors can then perform search operations on these repositories, and request the data keys to PHRs that might be of their interest directly to the respective patients or indirectly through the medical doctors following them.

5.4 Distance-Preserving Encoding

In this Section we propose a new family of encoding algorithms, called Distance Preserving Encodings (DPE)[1]. 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 (e.g. Chapter 3 and (Boufounos and Rane 2011)). DPE is the basis of this work and our new approach to searching multimodal encrypted data. Nonetheless its abstract 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 providing an *idealized functionality* for DPE schemes, that captures the functionality and information leakage of each of its operations. Then we present two efficient instantiations of the DPE functionality, one applied to dense media types (e.g. images), and another for sparse media (e.g. text), which we use in Section 5.6 to implement an efficient Multimodal Indexable Encryption prototype. Finally, we formally prove that the two proposed implementations are secure realizations of the idealized DPE functionality. Our security proofs follow the *real/ideal* or *simulation-based model* that is standard in secure multiparty protocols (Katz and Lindell 2007), under the requirement of Universally Composable Security (Canetti 2001) (restricted to our passive adversary model).

5.4.1 DPE Definition and Functionality

Informally, we define Distance Preserving Encoding (DPE) as a family of encoding schemes that preserve a controllable distance function between the 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. This threshold 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 the definition of 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 all plaintexts in the application domain. More formally:

[1] DPE could also be defined as an cryptographic scheme, by including a decryption function in its algorithms. However in the context of this work we only require a one-way function, and as such define DPE as an encoding scheme.

Algorithm 5.1 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 $D_{e_1}^{e_2}$.

Definition 2 (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 5.1 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 Section 5.3.2), which can only corrupt 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 this information is not sensitive. In the following we present two implementations of DPE and formally prove that they are secure realizations of $\mathcal{F}_{\text{DPE}}$.

5.4.2 A DPE Implementation for Dense Data

Rich media types, including images, audio, and video are characterized by their high-dimensionality and high-density (Böhm et al. 2001). 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 SIFT (Lowe 2004) and SURF (Bay et al. 2006) feature extraction algorithms for images, which compute feature-vectors of 128 and 64 dimensions respectively. High density means that in all dimensions necessary 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 (Manning et al. 2009) (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. (Boufounos and Rane 2011) for privacy-preserving nearest neighbors. This encoding provides information-theoretic security (Katz and Lindell 2007) by transforming feature vectors through universal scalar quantization (Boufounos and Rane 2011). Moreover, it preserves Euclidean (Manning et al. 2009) distances between plaintext feature-vectors, through the normalized Hamming (Manning et al. 2009)) 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:

$$e(x) = Q(\Delta^{-1}(Ax + w)) \tag{5.1}$$

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: the secret

Algorithm 5.2 Dense-DPE Implementation.

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

key $\{A, w\}$ has size proportional to the input and output sizes (N and M respectively). On the one hand, this will lead to large key sizes; on the other hand, this limits the flexibility of deployment of the scheme, 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) $\mathcal{G}$ (Katz and Lindell 2007) into 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 (Katz and Lindell 2007). Although this approach restricts the scheme to computationally-bounded security (Katz and Lindell 2007), it does not limit the applicability of our work as we are considering PPT adversaries in this contribution for practical reasons.

Algorithm 5.2 describes our implementation in detail, which we call *Dense-DPE*. Consistent with our security 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 2. *Dense-DPE securely realizes functionality $\mathcal{F}_{\mathrm{DPE}}$ against honest-but-curious PPT adversaries.*

Proof. The proof involves showing that a simulator $\mathcal{S}$, interacting with the client only through $\mathcal{F}_{\mathrm{DPE}}$ (the ideal experiment), can simulate the view of the server in a real interaction with the client through an instance of Dense-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 (Katz and Lindell 2007)).

$\mathcal{S}$ 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 $\mathcal{F}_{\mathrm{DPE}}$ with its *Distance Leakage*=$\{ID_{e_1}, ID_{e_2}, D^{e_2}_{e_1}\}$, 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 $D^{e_2}_{e_1} < t$. If that is the case, $\mathcal{S}$ knows that the distance between the plaintexts has been preserved and adds $\{ID_{e_2}, D^{e_2}_{e_1}\}$ to $M'[ID_{e_1}]$ and $\{ID_{e_1}, D^{e_2}_{e_1}\}$ 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 (Boufounos and Rane 2011) and of the Pseudo-Random Generator $\mathcal{G}$ (Katz and Lindell 2007), 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 (Boufounos and Rane 2011). Moreover, if $D^{e_2}_{e_1} \geq t$, it will also be indistinguishable from the simulated distance D', hence concluding the proof. □

5.4.3 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) (Katz and Lindell 2007). More concretely,

[2]Edit distance (Kuzu et al. 2012) and cryptographic schemes such as (Juels and Wattenberg 1999; Kerschbaum 2007) 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 wields similar search precision in ranked text retrieval (Manning et al. 2009).

Algorithm 5.3 Sparse-DPE Implementation.

```
1: function KEYGEN(k)
2:     K ← G(k)
3:     t ← 0
4:     return K, t
5: function ENCODE(p, K)
6:     e ← P_K(p)
7:     return e
8: function DISTANCE(e_1, e_2)
9:     if e_1 == e_2 then
10:        D ← 0
11:    else
12:        D ← 1
13:    return D
```

given a feature-vector from a sparse data-object (i.e. a text document), we apply:

$$f(x) = \mathcal{P}_K(x) \tag{5.2}$$

where x is a single keyword and $\mathcal{P}$ is a PRF, instantiated with secret key K. Algorithm 5.3 provides the full details of our implementation, which we call *Sparse-DPE*. Furthermore, we can prove that:

Theorem 3. *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 2. 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 $\{ID_{e_1}, ID_{e_2}, D_{e_1}^{e_2}\}$, it checks if $D_{e_1}^{e_2} = 0$. 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) (Katz and Lindell 2007), 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 $D_{e_1}^{e_2}$ will only be zero if and only if $p_1 == p_2$, thus proving the security and correctness of the implementation. □

Table 5.2 presents a summary evaluation of the entropy generated by DPE encodings, by analyzing encoded (i.e. normalized Hamming) distance functions 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)

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 5.2: Encoded (i.e. normalized Hamming) distances between different DPE encodings and plaintexts.

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.

5.5 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 particularly heavy operations 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 is that instead of revealing information patterns as queries are performed, like SSE schemes do, we reveal the same information pattern at data creation/update time (namely search, access, and frequency patterns (Kuzu et al. 2012)). However this allows to effectively support mobile devices, with increased performance and scalability (see Section 5.7 for experimental results).

From a systems perspective, MIE is defined as a distributed middleware with two main components: one running in the mobile 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 3 (Multimodal Indexable Encryption)**.** *A Multimodal Indexable Encryption middleware is a collection of five polynomial time algorithms (*CreateRepository, Train, Update, Remove, Search*) executed collaboratively between a user and*

a server, such that:

• $\mathbf{rk_R} \leftarrow \mathbf{CreateRepository}(\mathbf{ID_R}, \mathbf{sp_R}, \{\mathbf{ID_{m_i}}\}_{\mathbf{i=0}}^{\mathbf{n}})$*: 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 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 *for the repository.*

• $\mathbf{Train}(\mathbf{ID_R}, \mathbf{rk_R}, \{\mathbf{ID_{m_i}}, \mathbf{ip_{m_i}}\}_{\mathbf{i=0}}^{\mathbf{n}})$*: 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 *(Manning et al. 2009), more details are provided in Section 5.6). 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.*

• $\mathbf{Update}(\mathbf{ID_R}, \mathbf{ID_p}, \mathbf{p}, \mathbf{dk_p}, \mathbf{rk_R}, \{\mathbf{ID_{m_i}}\}_{\mathbf{i=0}}^{\mathbf{n}})$*: 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.*

• $\mathbf{Remove}(\mathbf{ID_R}, \mathbf{ID_p})$*: is an operation that allows a user to fully remove a data-object p that he created from repository R and its indexing structures.*

• $\{\mathbf{ID_{p_i}}, \mathbf{p_i}, \mathbf{score_{p_i}^{q}}\}_{\mathbf{i=0}}^{\mathbf{k}} \leftarrow \mathbf{Search}(\mathbf{ID_R}, \mathbf{q}, \mathbf{rk_R}, \{\mathbf{ID_{m_i}}\}_{\mathbf{i=0}}^{\mathbf{n}}, \mathbf{k})$*: 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 (*$\{ID_{m_i}\}_{i=0}^{n}$*). 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.*

5.5.1 Provable Security Properties

Algorithm 5.4 presents an idealized functionality for MIE ($\mathcal{F}_{\text{MIE}}$), while Algorithms 5.5 through 5.9 detail our MIE's implementation based on DPE, more concretely on DPE-Sparse and DPE-Dense. The main difference between our MIE implementation and functionality $\mathcal{F}_{\text{MIE}}$ is the use of DPE. Hence, the main argument in proving security lies in showing that by using DPE's algorithms, our

Algorithm 5.4 The ideal functionality $\mathcal{F}_{\text{MIE}}$.

- 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).

MIE implementation doesn't leak anything further to the server beyond what is specified in $\mathcal{F}_{\text{MIE}}$. Furthermore, we can prove that:

Theorem 4. *The DPE-based MIE implementation presented in Algorithms 5.5-5.9 securely realizes functionality $\mathcal{F}_{\text{MIE}}$ 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 $\mathcal{F}_{\text{MIE}}$. As such, simulator $\mathcal{S}$ can simulate all the interactions in the protocol using the information it obtains from $\mathcal{F}_{\text{MIE}}$. The details are straightforward and hence omitted. □

5.5.2 Additional Security Considerations

Applications using MIE have provable security guarantees, equivalent to the ones of previous SSE schemes in practical deployments (Cash et al. 2014; Curtmola et al. 2006; Naveed et al. 2014), of the information leaked by each operation. Nonetheless, the impact of this information leakage and to what extent it can be leveraged by adversaries in inference and statistical attacks is not yet fully understood. Recent advances have been achieved in this field, with query recovery and plaintext recovery attacks being proposed in the text domain (Cash et al. 2015; Islam et al. 2012). However the efficiency of these attacks depends on very strong assumptions. Query recovery attacks, for instance, require almost complete document set knowledge, i.e. adversaries must know the contents of all encrypted data or at least a large subset. The best known example attack requires 95% document knowledge to achieve 58% query recovery rate. With 75% document knowledge, query recovery drops to values close to 0% (Cash et al. 2015). Plaintext recovery attacks can have strong consequences but require a malicious adversary capable of encrypting and planting documents of his choice, i.e. performing chosen plaintext-attacks (Cash et al. 2015). Users can prevent such attacks by controlling the source of their documents, and by protecting their devices from external hacking. Furthermore, while keywords in the text domain usually have a straight semantical meaning, the same may not hold for similar concepts in richer media domains (including audio, images, and video). Although, as far as we know, attacks over these media domains are still an open area of research, we argue that additional background information (controllable by users) may be required for adversaries to achieve acceptable recovery rates.

5.6 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 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

Algorithm 5.5 MIE's Create New Repository Operation.

function USER(U).CREATEREPOSITORY(ID_R, sp_R)
  $rk1_R \leftarrow$ DENSE-DPE.Keygen(sp_{m_i})
  $rk2_R \leftarrow$ SPARSE-DPE.Keygen(sp_{m_i})
  CLOUD.CreateRepository(ID_R)
  RepUsers.ShareKey($\{rk1_R, rk2_R\}$)
  return $\{rk1_R, rk2_R\}$
procedure CLOUD.CREATEREPOSITORY(ID_R)
  $Rep[ID_R] \leftarrow$ InitializeRepository()
  $Fvs[ID_R] \leftarrow$ InitializeFeatureVectorsList()

Algorithm 5.6 MIE's Train Repository Operation.

procedure USER(U).TRAIN($ID_R, \{rk1_R, rk2_R\}, ID_{m_i}, ip_{m_i}\}_{i=0}^{n}$)
  CLOUD.Train($ID_R, \{ID_{m_i}, ip_{m_i}\}_{i=0}^{n}$)
procedure CLOUD.TRAIN(ID_R, $\{ID_{m_i}, ip_{m_i}\}_{i=0}^{n}$)
  for all $\{ID_{m_i}, ip_{m_i}\}_{i=0}^{n}$ **do**
    $Idx[ID_R][ID_{m_i}] \leftarrow$ InitializeIndex(ID_{m_i}, ip_{m_i})
    if DenseMediaType(ID_{m_i}) **then**
      $CB_R^{m_i} \leftarrow$ TrainIndex($Idx[ID_R][ID_{m_i}], ip_{m_i}, Fvs[ID_R]$)
    IndexData($Idx[ID_R][ID_{m_i}], Fvs[ID_R]$)

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 Section 5.7, 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 (Google 2016a). The cloud server component was fully developed in C++.

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 (Manning et al. 2009), 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 (binary descriptors (Heinly et al. 2012) could also be used if a different Dense-DPE implementation preserving Hamming distances was designed). As such, we use the *SURF* descriptor extraction algorithm (Bay et al. 2006) and *Dense Pyramid* feature detection (Lazebnik et al. 2006) for our prototype implementation. Dense-DPE was instantiated with threshold

Algorithm 5.7 MIE's Add/Update Object in Repository Operation.

procedure User(U).Update(ID_R, ID_p, p, dk_p, $\{rk1_R, rk2_R\}$, $\{ID_{m_i}\}_{i=0}^{n}$)
  for all $\{ID_{m_i}\}_{i=0}^{n}$ **do**
    $fvs_{m_i}^{p} \leftarrow$ ExtractFeatureVectors(p, ID_{m_i})
    if Dense-Media(ID_{m_i}) **then**
      $efvs_{m_i}^{p} \leftarrow$ Dense-DPE.Encode($fvs_{m_i}^{p}, rk1_R$)
    else
      $efvs_{m_i}^{p} \leftarrow$ Sparse-DPE.Encode($fvs_{m_i}^{p}, rk2_R$)
  $e \leftarrow Enc(dk_p, p)$
  Cloud.Update($ID_R, ID_p, e, \{ID_{m_i}, efvs_{m_i}^{p}\}_{i=0}^{n}$)

procedure Cloud.Update($ID_R, ID_p, e, \{ID_{m_i}, efvs_{m_i}^{p}\}_{i=0}^{n}$)
  Cloud.Remove(ID_R, ID_p)
  $Rep[ID_R][ID_p] \leftarrow e$
  $Fvs[ID_R][ID_p] \leftarrow \{efvs_{m_i}^{p}\}_{i=0}^{n}$
  if IsTrained(ID_R) **then**
    for all $\{ID_{m_i}\}_{i=0}^{n}$ **do**
      for all $fv \in efvs_{m_i}^{p}$ **do**
        if $Idx[ID_R][ID_{m_i}][fv][ID_p] == \{\}$ **then** $Idx[ID_R][ID_{m_i}][fv][ID_p] \leftarrow 0$
        $Idx[ID_R][ID_{m_i}][fv][ID_p]++$

Algorithm 5.8 MIE's Remove Object from Repository Operation.

procedure User(U).Remove(ID_R, ID_p)
  Cloud.Remove(ID_R, ID_p)

procedure Cloud.Remove(ID_R, ID_p)
  if $Rep[ID_R][ID_p] != \{\}$ **then**
    $Rep[ID_R][ID_p] \leftarrow \{\}$; $Fvs[ID_R][ID_p] \leftarrow \{\}$
    if IsTrained(ID_R) **then**
      for all $\{ID_{m_i}\}_{i=0}^{n}$ **do**
        for all $fv \in Idx[ID_R][ID_{m_i}]$ **do**
          $Idx[ID_R][ID_{m_i}][fv]$.Remove(ID_p)

$t = 0.5$ 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 v1.0.2 (OpenSSL Software Foundation 2016) and OpenCV v2.4.10 (Itseez 2016), which are open source libraries implementing different cryptographic and image processing algorithms respectively, were compiled for Android integration and support MIE's user-side computations. All remaining computations, including text feature-extraction,

Algorithm 5.9 MIE's Search Repository with Object as Query Operation.

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** $\{ID_{p_i}, Rep[ID_{p_i}], Res[ID_{p_i}]\}_{i=0}^{k}$

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 (Manning et al. 2009) approach is used, where each index key represents a keyword and index values are 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 (Nistér et al. 2006). 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 (Nistér et al. 2006). 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 with an image, the different feature-vectors of the 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 (Nistér et al. 2006), 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 memory, champion posting lists (Manning et al. 2009) are used to ensure that only the top ranked data-objects of each index entry are kept in memory (which doesn't impact search/retrieval precision), while the full index is stored in disk and periodically merged with updated/newly added index entries. 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 as in its original design).

To rank search results, the TF-IDF (Manning et al. 2009) weighting function is used both for images and text. Nonetheless more complex functions could be used without loss of generality, including BM25 (Manning et al. 2009) and its variants (Lv and Zhai 2011). Finally, to enable multimodal querying (simultaneous search with multiple media query formats) we use the logarithmic inverse square rank fusion approach (Mourão et al. 2013). 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 middleware, we also implemented simple Android and desktop applications which exercise all operations provided by MIE.

5.7 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 (Cash et al. 2014) 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 (Paillier 1999), protecting frequency

patterns when performing queries. We refer to these schemes as MSSE and Hom-MSSE, respectively, and full implementation details can be found in Appendix A.

Experimental Test-Bench In the following we will present performance 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 Mbps). As dataset we used the MIR-Flickr dataset (Huiskes and Lew 2008), which contains one million images and their user defined textual tags extracted from the Flickr social network.

Experimental Evaluation Roadmap The goals of our experimental work is to answer the following questions: **i**) what are the implications on user perceived performance (i.e, time consumed by the user device) to process and upload multimodal data to a cloud infrastructure, considering different devices (mobile device and desktop computer) and how performance evolves as we scale the size of the data set in a scenario where a single user is accessing the repository (Section 5.7.1)? **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 (Section 5.7.2)? **iii**) What is the user perceived performance associated with searching a repository using MIE and the concurrent schemes (Section 5.7.3)? **iv**) What is the retrieval precision obtained by MIE, in comparison with the concurrent schemes and with plaintext retrieval (Section 5.7.4)? And finally, **v**) what are the implications of the different schemes on the battery life of mobile devices when users upload new multimodal content to the repository, and how this varies as the size of data sets manipulated by clients grows (Section 5.7.5)?

5.7.1 Single User Scenario

Figures 5.2 and 5.3 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

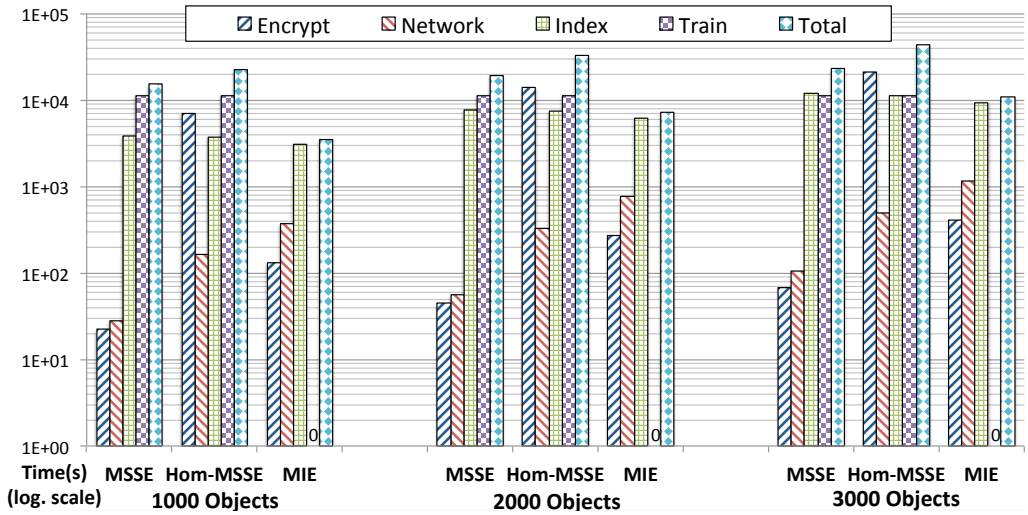

Figure 5.2: Performance of the update operation in a mobile device.

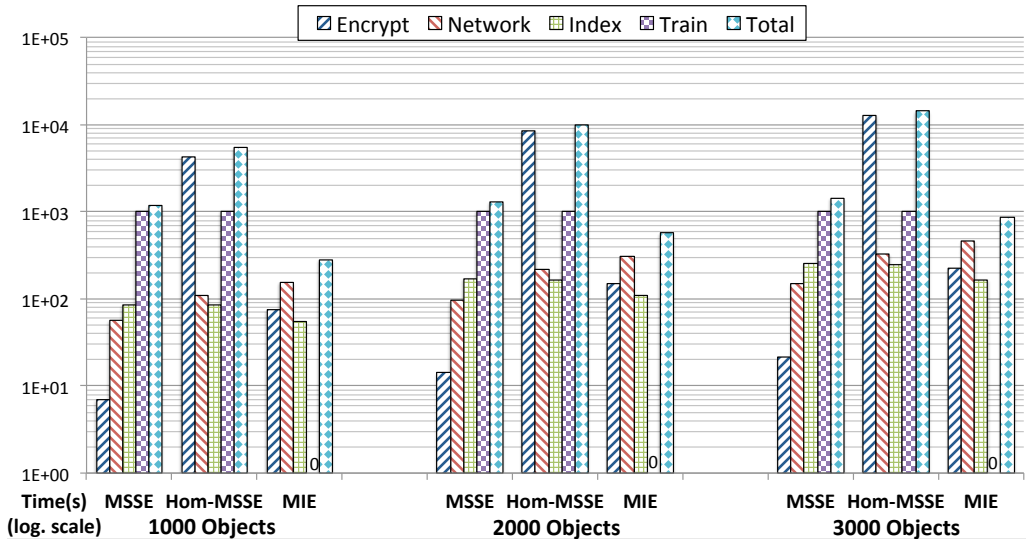

Figure 5.3: Performance of the update operation in a desktop device.

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 servers; *Index* is the time spent 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 the all sub-operations.

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 Encoding (DPE) schemes, all other computations are securely offload to the cloud server. In contrast, MSSE and Hom-MSSE clients have to perform those operations in their devices, which besides extracting feature-vectors, also include: clustering them against the training data-structures obtained during the training step (for dense media types); and indexing those feature-vectors (or their clustered versions, in dense media types), storing the results in 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 (although not as prohibitively slow as fully homomorphic encryption, it is still based in modular arithmetic and asymmetric cryptography, which is slower then the symmetric-key primitives used in MSSE and MIE). 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 worst 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 training operation, MIE clients still show lower total execution time than MSEE and Hom-MSSE clients. The average 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 execute in much less time 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 users to execute the initialization and loading of a cloud-based secure 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.

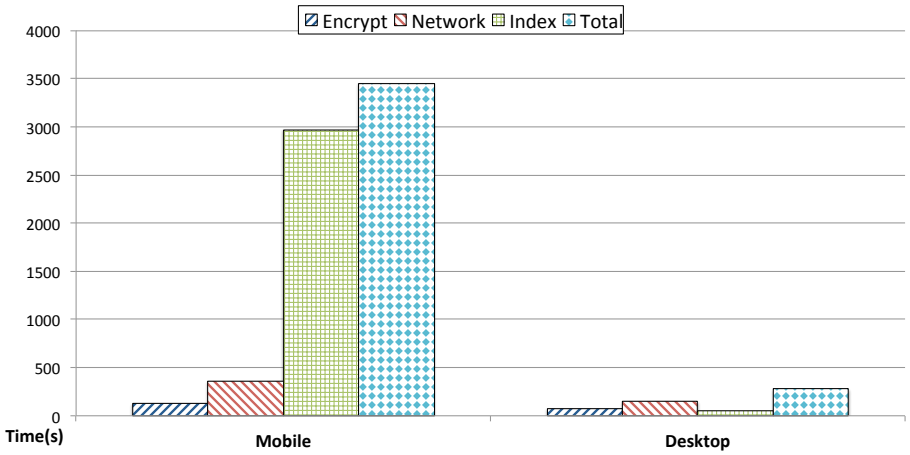

Figure 5.4: Multi-client update performance, with 1 mobile and 1 desktop client where each upload 1,000 data-objects.

5.7.2 Multi-User Scenario

We next conducted an experiment where two clients, one executing in a desktop computer and the other executing on the mobile device, both process and upload 1,000 multimodal data objects to a single cloud-based repository. In this experiment we only evaluated the MIE approach, since recent SSE schemes in the literature (Cash et al. 2014; Hahn and Kerschbaum 2014; Stefanov et al. 2014) require clients to keep some local storage, which in a multi-client scenario must be kept in a consistent state between the different users of a repository.

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 5.4 summarizes the results for both clients. The figure shows that when compared with the results presented above, both clients are able to make independent progress, and that both consume the same amount of time when processing and uploading a dataset composed of 1,000 multimodal objects.

5.7.3 Query Performance

Figure 5.5 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

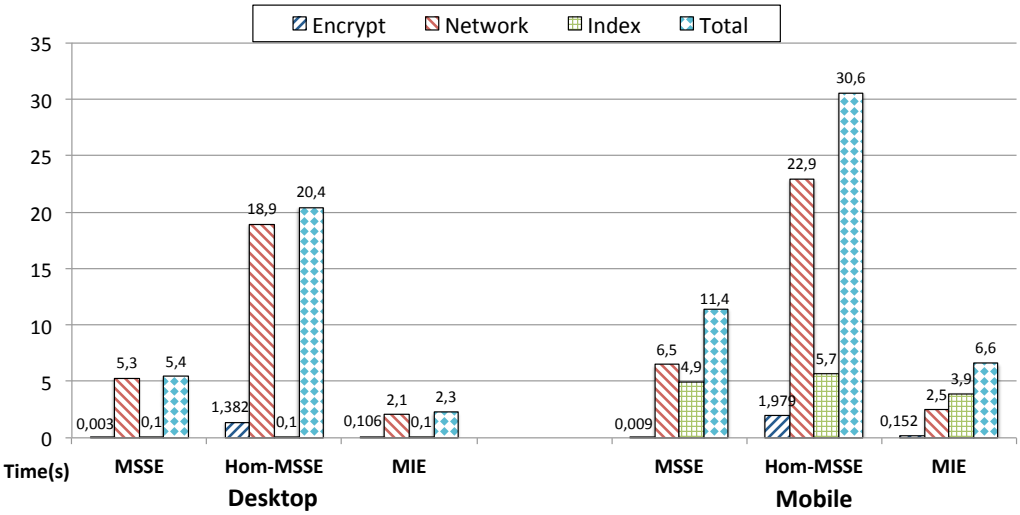

Figure 5.5: Performance of the search operation for Mobile and Desktop.

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

Table 5.3: Mean Average Precision (mAP) for the Holidays dataset.

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.

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 information on a public cloud infrastructure and when performing queries to retrieve data objects.

5.7.4 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

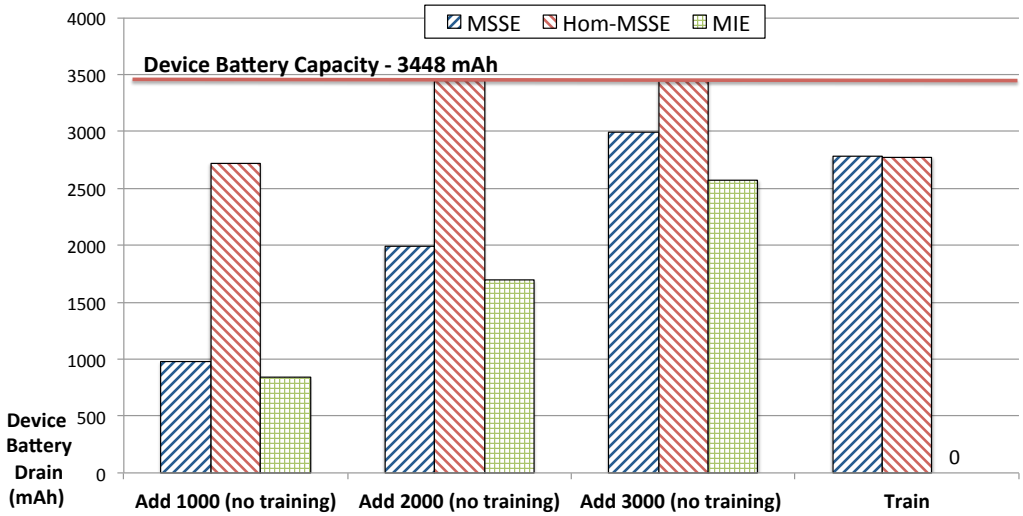

Figure 5.6: Mobile energy consumption for the different operations.

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 (Jegou et al. 2008), measuring the mean average precision (mAP) of 500 queries over a repository of 1491 photos. Table 5.3 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 middleware architecture, as clustering is a NP-Hard problem and only an approximated solution can be found (Hartigan 1975).

5.7.5 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 5.6 reports the obtained results, which were measured through Android's Operating System Power Profiles Framework (Google 2016e). This framework allows users to verify in a precise way how much energy is consumed in a given period of time by the different applications running in the system.

The results shows that MIE significantly outperforms the remaining schemes. This is a reflection of the results shown in the previous subsections, 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 5.6, 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.

5.8 Summary

In this Chapter we have tackled the practical challenges of efficient 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.

Publications The results presented in this Chapter were published in:

- **Multimodal Indexable Encryption for Mobile Cloud-based Applications**

(Conference Poster). Bernardo Ferreira, João Leitão, and Henrique Domingos. In the 10th ACM European Conference on Computer Systems (EuroSys'15). Bordeaux, France, April 2015.

- **Cifra Multimodal Indexável para Aplicações Móveis baseadas na Nuvem**. Bernardo Ferreira, João Leitão, and Henrique Domingos. In proceedings of the 7th Simpósio de Informática (INFORUM'15). Covilhã, Portugal, September 2015.
- **Indexable Encryption: Searching Cloud-Stored Multimodal Data on Mobile Devices** (Oral Communication). Bernardo Ferreira, João Leitão, and Henrique Domingos. In the PhD Forum of the 34th IEEE Symposium on Reliable Distributed Systems (SRDS'15). Montreal, Canada, September 2015.
- **Multimodal Indexable Encryption for Mobile Cloud-based Applications** (Technical Report). Bernardo Ferreira, João Leitão, and Henrique Domingos. Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Caparica, Portugal, February 2016 (Available in `http://asc.di.fct.unl.pt/%7Ebf/MIE`).

Prototypes A software prototype of the contribution is also available at:

- `https://github.com/bernymac/MIE`

Chapter 6

Complementary Results - Trusted Cloud Storage for Email Repositories

Given the core research vector of the thesis and the main contributions presented so far, we now present a relevant and representative complementary contribution accomplished in the thesis. This contribution, entitled TMS (Trusted Mail System) combines in a synergetic way our main contributions and their novel privacy foundations with dependability criteria, including reliability and availability guarantees. In more detail, TMS allows storing and searching sensitive email repositories in untrusted public clouds. TMS provides availability, integrity, and privacy guarantees, by exploring a cloud-of-clouds architecture complemented with threshold signatures and secret sharing.

6.1 Motivation and Goals

Most companies consider email to be a mission critical application (MediaBuzz 2010) and considerable information related with intellectual property is processed via email services and applications. Email messages are commonly used for strategic commercial information, confirmation of business transactions, or even to recover authentication credentials that give full access to other critical services. As such, email repositories are examples of systems where reliability and security concerns must be carefully addressed.

Despite their critical nature, email services for individual, enterprise, or institutional use, are among the most popular contexts for data outsourcing in public Internet cloud storage providers. Examples of such services are Gmail (Google 2016b), Hotmail (Microsoft 2016c), and iCloud Mail (Apple 2016a). In these cases, it is common to observe contradictory approaches in the way how cloud-based email outsourcing services are adopted. In one hand, cloud storage services provide no dependability guarantees under the control of end-users, with poor or very limited liability and Service Level Agreement (SLA) conditions (Amazon Web Services (AWS) 2016c). On the other hand, many studies have rated security and privacy to be major areas of concern and obstacles to adopt cloud-based solutions (Catteddu and Hogben 2009; Ion et al. 2011).

In an attempt to improve the reliability, availability, and security conditions of dependable cloud storage services, the use of multiple storage clouds offers an innovative, yet challenging research direction (Verissimo et al. 2012). Such approach allows the materialization of a transparent and dependable cloud-of-clouds data repository architecture. Solutions following this approach (Bessani et al. 2013) benefit from the resilience conditions established by the diversity of multiple clouds, as well as, from the security controls that can be provided by integrated cryptographic methods and data replication or fragmentation techniques, under the control of end-users (Verissimo et al. 2012). It is also an interesting design option in addressing intrusion-tolerance, leveraging from hardware/software heterogeneity and independent failure/attack models in each individual cloud (Bessani et al. 2013).

Inspired from the relevant work on dependability services in the design of cloud-of-clouds data-storage architectures (Bessani et al. 2013; Verissimo et al. 2012) we addressed, in the context of the thesis, the design and implementation of TMS (Trusted Mail System), a searchable email repository service based on a storage backend built on top of a cloud of internet storage clouds. These storage clouds are used as conventional repository components offered by current cloud providers (e.g. Dropbox (Dropbox 2016), Nirvanix Cloud Storage (Nirvanix 2013), Amazon Web Services (Amazon Web Services (AWS) 2016c), and Rackspace (Rackspace 2016)). TMS offers security, privacy, availability, and reliability guarantees for email repositories, under control of the users. TMS is designed to run as a middleware service that can be deployed as a local proxy (in a client machine) or as a remote proxy (used as a trusted cloud service). The system provides SMTP and POP standard operations to support Mail User Agents (MUAs) and Webmail applications, as well as an alternative API which translates read/write/search operations on mailboxes to the equivalent operations in the

backend encrypted storage clouds.

In summary, the contribution of this work lies in the proposal of a novel architecture for email data outsourcing, enforcing:

- **Security and Privacy** of mailbox data by combining conventional cryptography with homomorphic encryption algorithms and state-of-the-art threshold cryptography mechanisms;
- **Privacy-Preserving Search Operations**, providing similar functionality as supported in conventional email services. TMS offers both secure exact-match search of email header fields and ranked search of email contents and attachments, by combining information retrieval techniques with partially-homomorphic and property-preserving encryption algorithms.
- **Availability**, by using multiple cloud repositories as a transparent storage support;
- **Confidentiality, Integrity, and Authenticity** without Key Management overhead by using secret sharing and threshold based signature schemes, enabling the secure storage of cryptographic keys and data signatures in untrusted storage clouds, without the need of managing verification keys in centralized key-stores;

6.2 Related Work

As will be detailed in the next Sections, TMS follows a cloud-of-clouds architecture in which diverse untrusted cloud storage services are used as a storage backend in a secure, reliable, and dependable email repository solution. Traditionally, data outsourcing systems are addressed by network file systems (Howard 1988; Shepler et al. 2003). Authentication and access-control services allow correct clients to locally mount file systems stored at the server, accessing remote files for transparent use. In these systems, the server is a trusted computing base, supporting authentication functions and enforcing access control policies over the user's stored data.

Cryptographic file systems (Goh et al. 2003; Kallahalla et al. 2003; Wright et al. 2003) improve security guarantees, under the assumption that remote storage services are not necessarily trustable to provide confidentiality, privacy, data-authentication, or data-integrity properties to the clients. In cryptographic file systems all data operations are done at the client side, where encryption/decryption takes place. Some cryptographic file systems (Goh et al. 2003; Kallahalla

et al. 2003) also add file-sharing facilities, provided by means of an authenticated key distribution service.

In comparison, TMS adopts a cloud-of-clouds architecture, with email messages replicated on diverse untrusted storage clouds that can fail or may be attacked arbitrarily. In the TMS system, security mechanisms implement threshold-signatures (Shoup 2000) and secret-sharing techniques (Ilker Nadi Bozkurt et al. 2008; Kaya and Selçuk 2007; Shamir 1979) combined as built-in middleware components, preserving guarantees of authenticity, confidentiality, and integrity of private mail messages. TMS is also particularly focused in building a middleware solution for email repository services, allowing the transparent integration of email user agents implementing SMTP and POP protocols and allowing read, write, and search operations of applications over mailboxes and email messages.

Some data outsourcing models are based on the use of remote databases as cloud services (or DbaaS) (Hacigümüş et al. 2002). These systems are focused in using remote SQL databases, not necessarily trusted. To support security and privacy guarantees, it is necessary to provide support for client execution of SQL queries over remote encrypted data. The use of property-preserving and partially-homomorphic encryption schemes allows this solution. Some interesting approaches, such as CryptDB (Popa et al. 2012) show that the support for some SQL-based operations over encrypted databases running in untrusted servers is possible, with an interesting balance between security and performance. Comparing with database approaches, TMS is mainly focused in exploring property-preserving and partially-homomorphic schemes to provide the relevant operations provided by email-storage systems and allowing ranked queries over private mailboxes maintained in multiple key-value stores, as offered by Internet cloud-storage providers.

The use of multiple clouds and secret sharing for improved security, reliability, and availability has also been seen in recent approaches like DepSky (Bessani et al. 2013). However while the objective of DepSky is to store generic data blocks with security and dependability guarantees, in TMS we focus on email data and also have the requirement of performing efficient computations on encrypted data, particularly search operations over email contents and header fields.

6.3 System Overview

The TMS architectural model follows a “cloud-of-clouds” middleware layering solution to be used between Mail User Agents (MUAs) or Webmail applications,

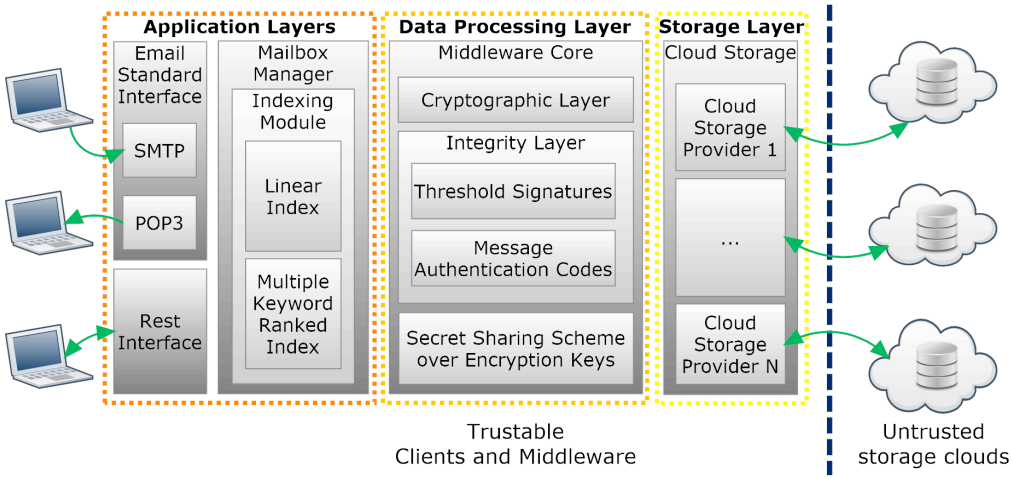

Figure 6.1: TMS architecture and its main components.

and a backend repository composed by multiple Internet storage clouds. More specifically, the system design consists in a three-layered architecture approach, as represented in Figure 6.1.

In the Figure, the Application Layer is composed by two sub-layers: Email Standard Interface and Mailbox Manager. The Email Standard Interface supports SMTP (Klensin 2008) and POP-based (Myers and Rose 1996) standard endpoints. This sub-layer externalizes the required support for smooth integration of any MUA. Alternatively, a restful/web-service oriented API (detailed in Table 6.1) is also provided for the integration of web-oriented applications, offering the same functionality as found on conventional object storage Internet cloud solutions. The Mailbox Manager is designed as a dependable functional mailbox management component providing indexing, searching, and management mechanisms as part of message storage and retrieval operations.

The Middleware Core layer provides confidentiality and integrity services, while the Cloud Storage Layer provides a data repository layer that materializes a transparent data storage backend, implemented by specialized connectors for different Internet storage clouds. These two layers will be described in more detail in Section 6.4, while next we follow with the system and adversary model definitions.

6.3.1 System Model

We define n as the total number of storage clouds used as backend in TMS, f as the number of clouds that can be attacked or fail and t as the threshold number of clouds required as a resilience factor to support dependability guarantees. t

Operation	Description
ObjectId **put** (MailObject)	Writes and stores email messages as Objects
MailObject **get** (ObjectId)	Retrieves a mail message
Set< *ObjectId* > **list** ()	Lists the mail message identifiers in user's mailboxes
Set< *ObjectId* > **searchContent** (Set< *Keyword* >)	Searches the user's mailbox and mail contents for a set of keywords, returning the set of ranked relevant mail messages
Set< *ObjectId* > **searchMetadata** (Set< *MetadataKeys* >)	Searches the user's mailbox for a set of mail metadata keys (ex: Sender=Alice AND CC=Bob)

Table 6.1: TMS REST API, providing complementary operations to SMTP and POP endpoints.

is closely related with the parameterization of the employed threshold cryptographic mechanisms, as well as the number of clouds used to store data in a replicated way (e.g. the parameterization factor used for threshold signatures or to recover a secret from n secret shares, or the number of data-replicas stored in multiple clouds for resilience purposes; more details in Section 6.4).

In TMS, external applications act as writers, readers, and searchers of mail messages in mailboxes. The middleware backend implements a transparent and uniform object access layer over the different Internet cloud storage providers, supporting a replication process in which mail messages are encapsulated as generic objects, (optionally) fragmented, encrypted and then replicated through the developed cloud connectors. In this backend level, reads and writes are supported "as is" by the cloud providers and TMS connectors adopt the same consistency models as offered by them. From the TMS middleware perspective, read operations can fail with a subjacent arbitrary failure model and write operations (for replicas of the same value) can arbitrarily fail at most f times, as long as $t=2f+1$ writes of the same replica are correct. Beyond this restriction only fail-stop faults are supported. Our failure model follows the same assumptions as in the relevant state of art (Bessani et al. 2013).

6.3.2 Adversary Model

In the system model described so far, the trust base for preserving conditions of dependability, availability, security, and privacy is restricted to the components of the TMS middleware system. The storage clouds are considered as potentially untrusted, admitting they may be subject to both active attacks on the clouds' infrastructure (done by External Hackers and possible accidental/careless maneuver by cloud provider employees), as well as passive attacks done from inside the cloud servers (the honest but curious cloud model, as has been described in

state-of-art (Bessani et al. 2013; Popa et al. 2012; Verissimo et al. 2012)). To support reliability and intrusion tolerance of up to f faulty (or attacked) clouds, we adopt a set of $n=3f+1$ untrusted storage clouds and $t=n-f$ threshold/secret shares, with n being the same for both storage of mail replicas and generated threshold shares. The rational for this is that minimizing the number of shares distributed through the different clouds reduces the security levels of the secret sharing and threshold signature mechanisms. On the other hand, increasing the number of shares requires the employment of more independent clouds. For specific scenarios, this tradeoff may be addressed by parameterization. For implementation purposes, we decided to have $t=n-f$, while also leaving room for a possible future employment of a Byzantine fault-tolerant protocol requiring $3f+1$ replicas to agree on a correct value.

6.4 TMS Components and Mechanisms

In this Section we first describe the TMS Data Model, followed by discussion of the mechanisms and algorithms used in the different components of the TMS middleware core services, as introduced in Sections 6.1.

6.4.1 Back-End Storage and Data Model

We start this subsection with a high-level data model description.

Data Model Overview. We can divide the TMS data model in two main levels: the middleware local storage level, where we store mailbox indexes; and the cloud storage level, where actual email data (contents and headers) is stored as data payloads of opaque object containers. Figure 6.2 represents these two levels. As seen in the upper part of the Figure (Middleware Local Storage), the middleware keeps a local version of the users' mailboxes, containing pointers to the actual email data. Each mailbox is composed by three main indexes, as represented in the Figure:

- Boolean Index, allowing fast search operations over email header fields (as defined in RFC 5322), including recipients, sender or subject.
- Multi-Keyword Ranked Index, which allows search operations over encrypted email message contents while preserving their privacy. A search in the index returns a set of unique message identifiers, translated to in-cloud references through the reference index.

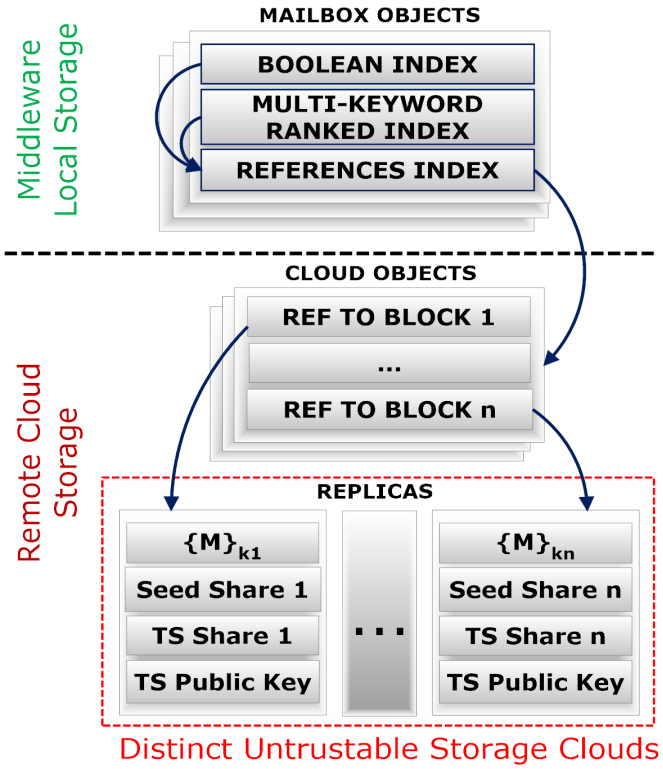

Figure 6.2: TMS data model and back-end storage, divided between TMS's local storage and cloud storage.

- Reference Index, which co-relates message ids with tokens composed by: a cloud object reference, pointing to the objects in the cloud repository; the cryptographic key used to encrypt the Cloud Object; and optionally a Message Authentication Code (based on secure hash-functions) for fast authenticity and integrity checks.

Both the ranked and boolean (or exact-match) indexing structures are built using a combination of property-preserving and partially-homomorphic encryption algorithms with information retrieval techniques. Additionally these structures can be periodically replicated to the clouds (to support backup and recovering of TMS middleware in any moment), using the TMS write operation to upload their state as a special mail message.

In the Remote Cloud Storage Layer (lower part of Figure 6.2), two main data structures are considered:

- Cloud Objects, representing email messages by referencing a set of data blocks. This objects are encrypted using a Password-Based Encryption scheme, where a password used as a seed for a symmetric encryption key is protected and stored on the middleware reference index. Although each

Cloud Object has a unique representation in the Figure 6.2, it can be replicated alongside the data blocks for increased availability.

- Replicated Data Blocks (Replicas), consisting in a set of data blocks which represent replicas of email messages. Each block stores encrypted data along with: a share of the seed used to generate the different cryptographic keys required; a share of the threshold signature used; and a public verification key for the threshold signature. The cryptographic constructions used will be detailed in Section 6.4.2.

Both data blocks and Cloud Object references are generated based on all object data and are used as unique identifiers in the key-value backend data storage clouds.

Indexing Structures. We now analyze in more detail the indexing structures that support the search functionality provided by TMS. The three indexing structures mentioned before (Reference, Multi-Keyword and Boolean) follow very similar structures. As an example, Figure 6.3 represents the multi-keyword ranked index in detail. The index maps each searchable term (existing in one or more email messages) to a posting list (Manning et al. 2009). Each posting list stores the unique identifiers of email messages, containing the term and a score for the email-term tuple. Scores are calculated through a scoring function which aggregates different term-email tuples and dataset wide statistics (e.g. BM25 (Manning et al. 2009)). Posting lists are then sorted by score, representing the relevance of each email message in relation to that term.

The Boolean index follows exactly the same structure except that posting lists only contain email references (no scores), and the existence of an email identifier on a posting list means that the email message contains that particular email header. The Reference index structure simply maps email identifiers to the respective Cloud Objects and cryptographic keys, as explained before.

6.4.2 Cryptographic Mechanisms

This Section presents the relevant cryptographic primitives used in TMS system model and architecture: secret sharing, threshold signatures, and property-preserving encryption schemes. In the core crypto provider component of the TMS middleware, these cryptographic mechanisms use conventional cryptographic hash functions, symmetric encryption algorithms, and public-key cryptographic

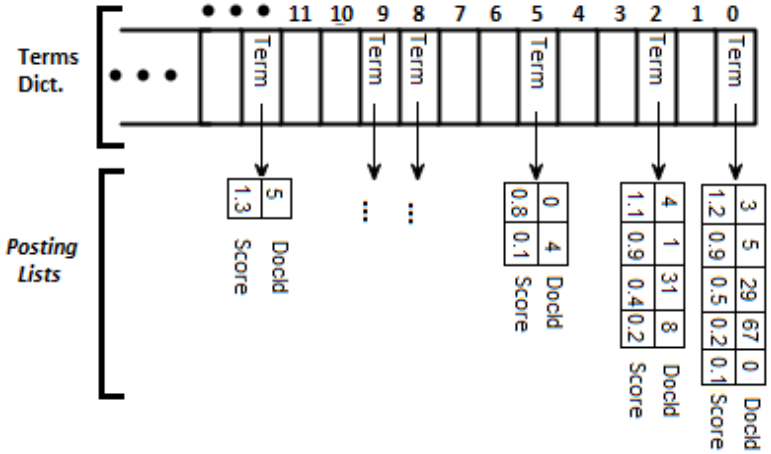

Figure 6.3: Multi-keyword ranked index in detail.

schemes, composed in ciphersuites. For this purpose, the ciphersuite composition to be used is a configuration parameter of the TMS system in each deployed instance.

Secret Sharing. In TMS, we employ a secret sharing scheme (Menezes et al. 1996) in order to safeguard cryptographic keys and securely store them in the multiple untrusted storage clouds. In TMS, three secret sharing schemes were implemented and experimentally evaluated: Shamir (Shamir 1979), Blakley (Ilker Nadi Bozkurt et al. 2008), and Asmuth-Bloom (Kaya and Selçuk 2007) schemes. Each of the evaluated secret sharing schemes is based on different mathematical principals. The Shamir scheme is based on the Lagrange interpolation (Shamir 1979) of a set of points (shares), the Blakley scheme is based on hyperplane intersection (where each plane is considered a share) (Ilker Nadi Bozkurt et al. 2008), while Asmuth-Bloom scheme is based on the Chinese Remainder Theorem (Kaya and Selçuk 2007) in which each congruence is considered a share part of the secret sharing scheme. From our experimental observations, the conducted evaluation revealed that the three schemes possess similar performance in practice. As such the Blakley Scheme was chosen as the secret sharing scheme to be used in the TMS prototype discussed and evaluated in Section 6.5.

Threshold Signatures. TMS employs Threshold Signatures (Shoup 2000) in order to obtain authenticity guarantees of replicated data blocks. For the TMS implementation, we use the threshold signature scheme proposed by Shoup (Shoup 2000), in such a way that each signature and public key could be wrapped in raw data sets in order to be distributed by multiple storage clouds. This scheme

explores RSA signatures combined with Lagrange interpolation for key generation, and so its security relies on the same security assumptions of RSA (discrete logarithms and factorization of large prime numbers (Schneier 1996)) and interpolation problems. Furthermore the proof of correctness of each share based on discrete logarithm problems avoids the poisoning of the signature verification process.

Property-Preserving and Partially-Homomorphic Encryption. In TMS, a requirement for secure and efficient search operations, allied with a desire for minimizing data exposure during operations, has led to the research and deployment of partially-homomorphic and property-preserving encryption schemes. More concretely, two schemes were considered: Search and Paillier.

Search Scheme (presented in Chapter 3) is a fast property-preserving encryption scheme, designed for text data, that allows equality comparison between encrypted keywords. Using the Search Scheme, TMS encrypts the index keys of the indexing structures previously discussed in Section 6.4.1) (Ranked and Boolean indexes). Afterwards, the existence of a keyword in the email repository can be verified by evaluating the encrypted keyword in face of the existing index entries, supporting efficient search operations over user's mailboxes while preserving the privacy of queries and index keys.

Paillier (Paillier 1999) In TMS, the Paillier cryptosystem is used in the encryption of the user's mailbox indexing structures (Ranked and Boolean indexes). Despite the use of Paillier encryption is sometimes referred as a possible source of cryptographic overhead (Popa et al. 2012), its use for encrypting indexing structures allows search operations to be performed remotely without requiring major data transfers and decryptions.

6.4.3 TMS Processing

This Section describes the different steps required to complete the most relevant operations in the TMS API: the *put*, *get*, and *search* operations.

Sending or Putting a Mail Message. Algorithm 6.1 describes the procedure for storing a new mail message in the TMS repository. This occurs when an SMTP server forwards messages to the TMS SMTP endpoint or when a user decides to store mail messages using TMS. When a message is sent trough the SMTP

Algorithm 6.1 Support for the TMS *put* operation.

```
 1: procedure PUT(Data)
 2:     newIndexEntries ← processAndIndex(Data)
 3:     HomEncrypt(newIndexEntries, PaillierKey, SearchKey)
 4:     Seed ← random()
 5:     KeyGenerator ← CreateKeyGenerator(Seed)
 6:     TSS ← generateThresholdSignatureShares(Data)
 7:     SSS ← generateSecretSharingShares(Seed)
 8:     cloudObject ← CloudObject()
 9:     for all c_i ∈ |C| do
10:         K_i ← keyGenerator.next()
11:         Data'_i ← encrypt(Data, K_i)
12:         replica_i ← Data'_i||TSS_i||SSS_i||TSS.PubKey
13:         RRef_i ← SHA1(replica_i)
14:         cloudObject.Add(RRef_i, i)
15:     masterKey ← keyGenerator.next()
16:     cloudObject' ← encrypt(cloudObject, masterKey)
17:     masterRef ← SHA1(cloudObject')
18:     for all c_i ∈ C do
19:         c_i.put(RRef_i, replica_i)
20:         c_i.put(masterRef, cloudObject')
21:     storeReferenceIndex(masterRef, masterKey)
```

endpoint or via the external *put* operation in the REST API, it is delivered to the Mailbox Manager (see Figure 6.1), which is responsible for processing the mail message, extracting attachments, metadata, and message contents. This information is then processed and indexed, storing the relevant metrics in TMS's indexing structures (line 2). To conclude this step, the new index entries are then encrypted with the schemes discussed in Section 6.4.2 (line 3). The cryptographic keys required in this step are only used to encrypt the indexing structures and are also replicated securely through the multiple storage clouds and using the secret sharing mechanism. Optionally, these keys can be refreshed periodically through a key refreshment mechanism. Once the indexing is done, the Mailbox Manager requests a data put internal operation to the layer below, with all the message data. This requires replicating the mail message and distributing the resulting replica blocks through any available c clouds (lines 4-22). The algorithm generates a set of keys from a cryptographic seed (line 10) and encrypts all replicas with a different key (line 11). The data is then attached to a share of the created threshold signature (line 6), a share of the seed used in the key generation process (line 7), and a copy of the threshold signature public key (line 12). Each replica built this way is referenced by a message digest (line 13) and this reference is stored

Algorithm 6.2 Support for the TMS *get* operation.

```
1:  procedure GET(masterRef, masterKey)
2:      cloudObject' ← c_x.get(masterRef)                          ▷ x ∈ C
3:      cloudObject ← decrypt(cloudObject', masterKey)
4:      for i ← 1...K do
5:          replica_i ← c_i.get(RRef_i)
6:          if SHA1(replica_i)! = RRef_i then
7:              //Corrupted replica, ignore
8:          else
9:              //Continue
10:         TSS_i ← replica_i.TSS
11:         SSS_i ← replica_i.SSS
12:         Data'_i ← replica_i.Data'
13:         TSS.PubK ← replica_i.TSS.PubKey
14:     Seed ← recoverSSSecret(SSS)
15:     keyGenerator ← KeyGenerator(Seed)
16:     for all Data'_i ∈ Data' do
17:         Data_i ← decrypt(Data'_i, keyGenerator.next())
18:         isValidData ← checkTSScheme(Data_i, TSS, TSS.PubK)
19:         if isValidData then
20:             return Data_i
21:         else
22:             //Continue
23:     //Unable to recover valid Data
```

in the new Cloud Object created (line 14). The Cloud Object is then encrypted with a master key (line 17) and replicated through the *c* clouds, along with the other replica blocks (lines 19-21). The algorithm concludes by storing the Cloud Object's reference and respective master key in the Reference Index (line 23). The index is also stored encrypted and the respective key is replicated through the secret sharing mechanism.

Receiving or Getting a Mail Message. When a message fetch is requested via the POP endpoint (or through the *get* operation in the REST API), the request is forwarded to the Mailbox Manager. Once the Manager obtains the relevant data (master reference and cryptographic key for the Cloud Object of the fetched mail) it invokes a GET request on TMS's core. The core layer then proceeds as described in Algorithm 6.2. Briefly, the algorithm starts by recovering and decrypting the Cloud Object of the requested mail from one of the available clouds (lines 2-3). Any cloud can be chosen, as this object is replicated through all. Then, TMS retrieves all the replicas referred in the Cloud Object, validating their integrity (lines 5-8) and extracting the stored signature and seed shares (lines

Algorithm 6.3 Support for the TMS *search* operation.

```
1: procedure SEARCH(Keywords, SearchKey)
2:     QueryTerms ← processKeywords(Keywords)
3:     QueryTerms' ← encrypt(QueryTerms, SearchKey)
4:     IndexEntries' ← accessIndex(queryTerms')
5:     SearchScores' ← calculateSearchScores(indexEntries')
6:     SearchScores ← decrypt(SearchScores', PaillierKey)
7:     OrderedSearchScores ← orderSearchScores(SearchScores)
```

9-12). From the different seed shares the original seed is reconstructed (line 14) and the different replica blocks are decrypted (line 17). To conclude, the replicas' integrity is validated through the threshold signatures recovered (lines 18-22). A non-poisoned replica (if any was possible to recover) is then passed to the Mailbox Manager, which returns it to the client through the POP endpoint or REST API.

Searching the Mailbox. The last TMS operation analysed in this Section is the search operation. Algorithm 6.3 shows the steps of a ranked search on email contents. Exact-match queries on email header fields follows the same procedure, without some specific ranking steps like sorting of search results. The user starts by specifying his query, which is processed by the Mailbox Manager (line 2) and converted into encrypted trapdoors, through the Search Scheme, in order to access the ranked index (lines 3-4). From the encrypted index entries, TMS calculates the final search scores (line 5), decrypts them (line 6) and presents the ordered results to the client (line 7). It should be noted that in the algorithm explained we are considering a native Mail User Agent (MUA) application that cannot execute user defined code, accessing TMS services directly through its external API. This forces TMS to encrypt/decrypt the search keywords and final search scores. If the application used by the client was able to run user defined code, such cryptographic primitives could be performed on the client side, thus reducing data exposure on TMS.

6.5 TMS Prototype and Evaluation

A TMS prototype was developed in Java. The implementation includes connectors for Amazon S3 (Amazon Web Services (AWS) 2016b), Nirvanix Cloud Storage (Nirvanix 2013), Rackspace Cloud Files (Rackspace 2016), Google Cloud Storage (Google 2016c), Dropbox (Dropbox 2016), Luna Cloud Storage (Luna Cloud 2016), and Microsoft Azure Storage service (Microsoft 2016a). Versions of the

secret sharing algorithms, threshold signatures, and homomorphic encryption schemes (discussed in Section 6.4.2) were also implemented in Java and using its standard JCE (Java Cryptographic Extension) library. The implemented prototype was then evaluated in two scenarios, with a focus on performance metrics. In a first testing environment we analyzed the performance of our proposal as a local middleware service, running in the same machine as the application leveraging it (Section 6.5.1). In the second environment we deployed TMS as a remote cloud service (Section 6.5.2). In both experiments, local computations (the webmail client and the middleware in the first experiment) were done in a 2.4GHz Intel Core i7-3630QM machine, with a 512MB JVM Heap and a network connection of 100 Mbps. As a dataset, we used the Enron email database (Klimt and Yang 2004) and extracted two subsets of emails: a smaller subset of 1.000 email messages, and a larger one of 10.000 email messages. As baseline comparison, we used Google Gmail service (Google 2016b). However, due to technical limitations in Gmail, it was impossible to experimentally evaluate the baseline with more than 1.000 messages. Nonetheless the performance of the baseline would be expected to grow linearly with the dataset increase, as occurred with our observed results, which we show next.

6.5.1 Performance of TMS as a Local Middleware

In this Section we evaluate the performance of TMS as a local service running in the webmail client machine. The setting of this experiment is represented in Figure 6.4, where the local machine was deployed in the faculty site, with the following storage clouds used: Amazon Ireland datacenter; Nirvanix US; Rackspace US; and Google US. In this evaluation we are particularly concerned in observing three main factors: (1) the overhead of TMS, when compared with the Gmail service through its SMTP and POP interfaces, (2) the impact of Threshold Signatures when compared with the use of the lightweight signatures adopting MAC schemes, and (3) the scalability factor and cloud overhead of the proposed solution with larger message batches.

Figure 6.5 shows the results for the *Message Send* and *Receive* operations analysed in terms of: overhead of communication between the clients and the middleware (*Client* in the Figure); core execution metrics, which include message processing time, indexing, and cryptographic operations (*Core* in the Figure); and the aggregated overhead of replicating data through the four clouds in parallel (*Clouds*). Additionally, two versions of middleware were prototyped and evaluated for performance comparison: a full version of the middleware as described

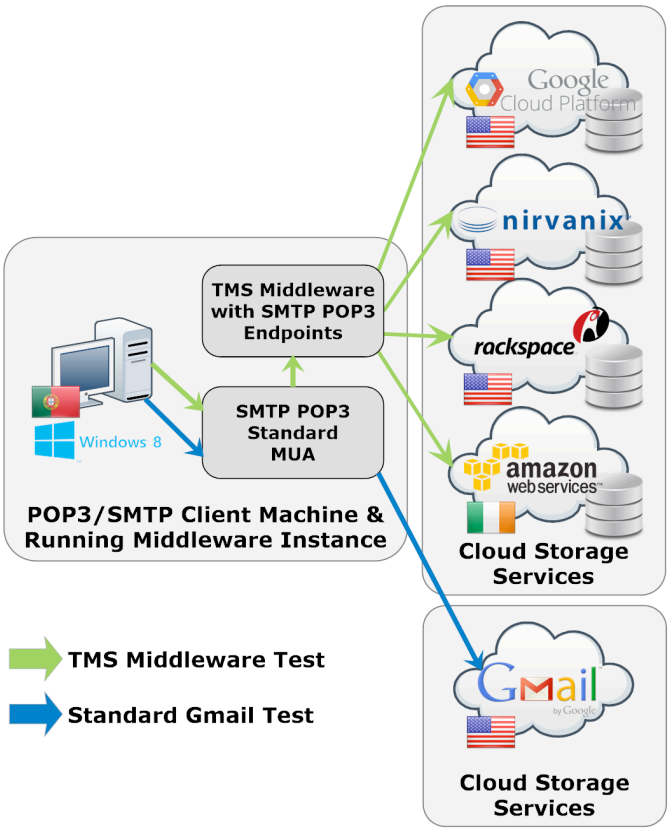

Figure 6.4: Initial test setting based on provider/location.

so far (TMS-TS in the Figure), and a more simple version where the threshold signatures were replaced by conventional MAC signatures (TMS-MAC).

Comparing our solution with the baseline Gmail service, we can see that there is some overhead introduced by TMS. However we find the observed overhead negligible especially considering the benefits gained in terms of security, reliability, and availability. Nonetheless, TMS outperforms the baseline in terms of client's perceived latency in the message send operation (*Client* in the Figure). These results can be explained by a synchronous message delivery of the Google service, contrary to the asynchronous nature of our proposal.

Comparing the three observed metrics in TMS (Client, Core, and Clouds), we can see that the heaviest overhead comes from cloud operations. This means that cloud latencies dictate the overall performance of our solution and that TMS algorithms themselves are very fast. When we compare TMS-MAC and TMS-TS approaches, we notice that as we increase the dataset size from 1k to 10k messages, there is a large overhead increase in the threshold signatures approach (represented by the *Core* metric in the figure). This is the tradeoff for gaining fault-tolerance guarantees in asynchronous Byzantine settings. Nonetheless this overhead is still very small comparing to the cloud replication overheads (*Clouds*

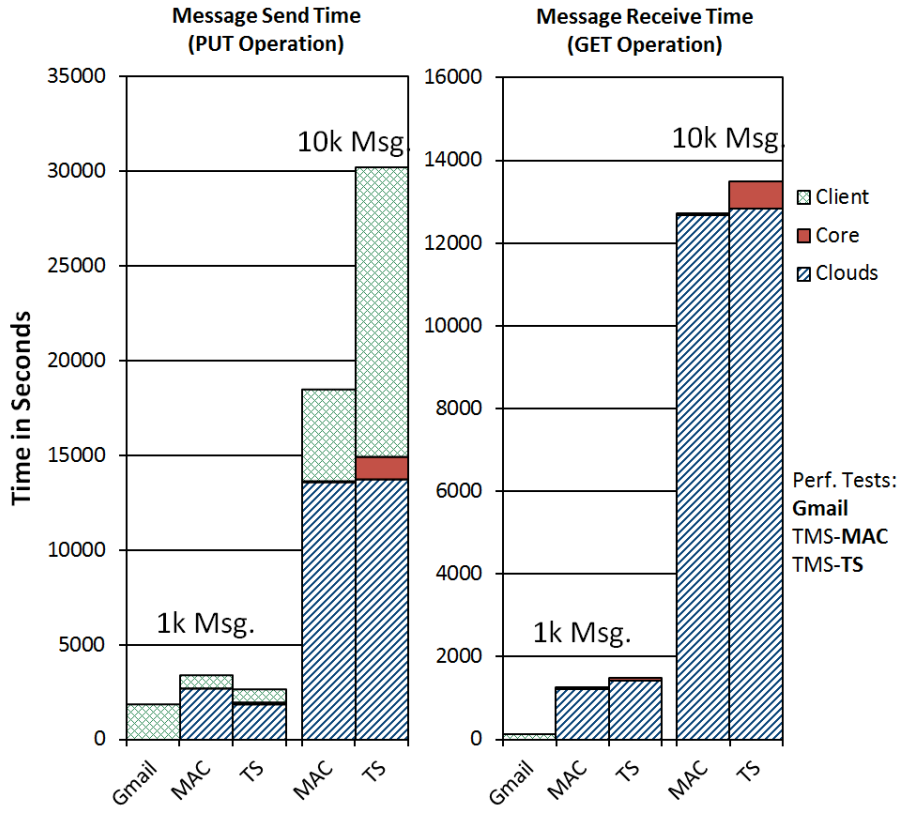

Figure 6.5: Performance impact of TMS (MAC and TS) comparing with Gmail service.

in the Figure).

6.5.2 Performance of TMS as a Cloud Service

In the previous experiments, the major overhead of TMS clearly came from replicating email data through multiple clouds. As such we have designed a new test setting where the middleware is deployed as a cloud service, running in a cloud provider's remote machine. In such setting we expect cloud latencies to be reduced, as the middleware executes now more closely the storage clouds and can benefit from higher network throughputs.

Figure 6.6 shows the test setting for this experiment, and Figure 6.7 shows the results obtained. To reason about having TMS running in different providers and locations, three different tests were performed: with the middleware deployed in a different cloud provider for each test; with an email client (communicating with the middleware) deployed in a fourth cloud datacenter; and with another four cloud providers serving as storage clouds for email replication (Test 1, Test 2 and Test 3 in the Figure). All remote clouds were deployed as ExtraSmall/Micro virtual machine instances. Additionally, the previous experiment (client and

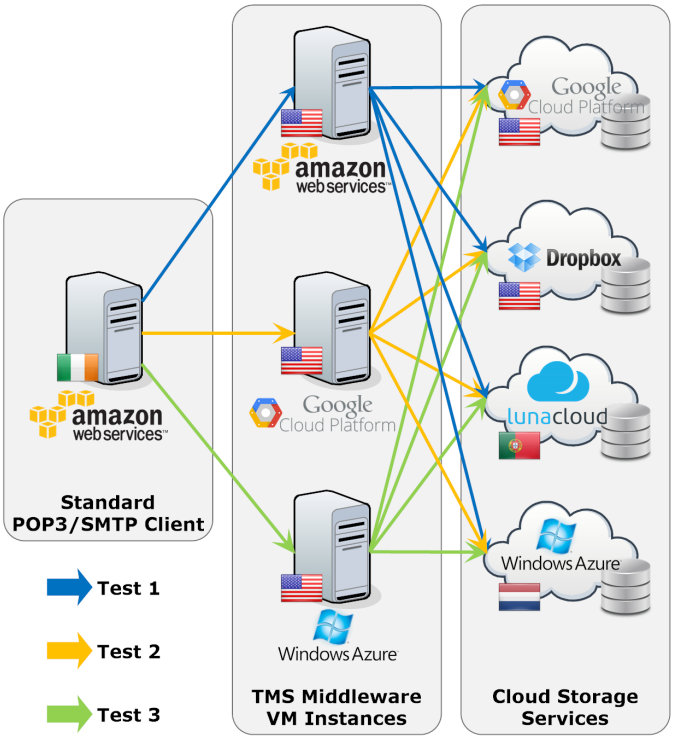

Figure 6.6: Performed test settings based on service/provider/location.

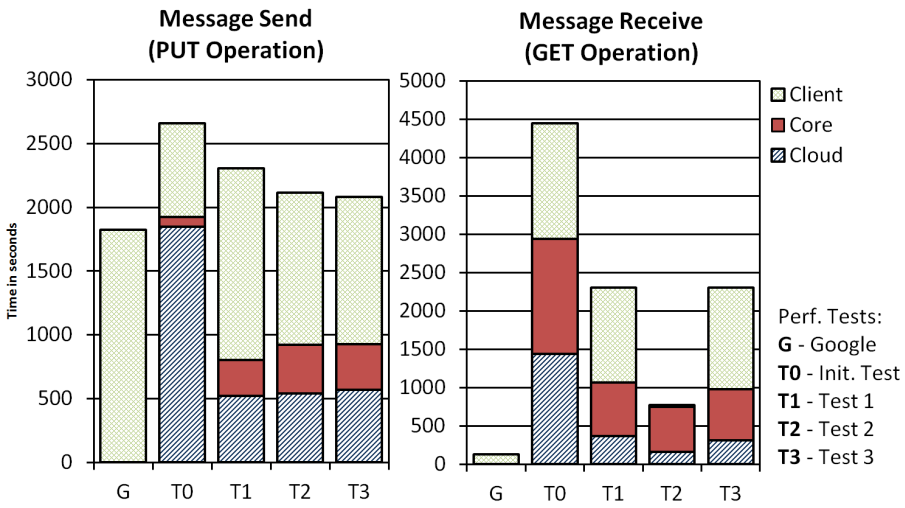

Figure 6.7: Performance comparison between Gmail webmail service and the performed tests.

middleware on the same machine) was repeated in this setting for comparison (T0 in Figure 6.7). In all these experiments only the one thousand (1k) email dataset was used.

For both send and receive operations, the TMS cloud deployment (Tests T1, T2 and T3) improves overall latency, comparing with running it in the same machine

as the email client (Test T0). In the message send operation, client perceived latency (*Client* in the Figure) had to be artificially augmented due to DoS protection mechanisms in cloud infrastructure and Gmail service. Despite this, we verify improved overheads for all middleware sub-processes, both in send and receive operations, which shows that a cloud deployment in a real case scenario would improve overall system performance. On the other hand, we found some dissimilarities between the different cloud deployments (T1, T2 and T3), especially in the message receive operation. This means that mechanisms for optimizing the overall solution by choosing clouds with optimal latency conditions is an interesting research direction. The Gmail service performed similarly to the previous experiment, as expected.

6.6 Summary

In this contribution we addressed the design and implementation of TMS, an interoperable middleware architecture providing a trusted email repository service on top of untrusted storage clouds. TMS offers security, privacy, availability, and reliability guarantees, using a storage backend implemented by multiple untrusted cloud solutions in a cloud-of-clouds architecture. The solution offers external services as provided by conventional email repositories, supporting Mail User Agents and Webmail applications implementing SMTP/POP standard operations (over SSL). TMS adds security, privacy, availability, and reliability guarantees, controlled by the user, and is designed to run as a local proxy in a client machine or as a trusted remote service. The TMS implementation shows the feasibility of its design options. The evaluation demonstrates interesting and promising results for latency and performance, revealing that the impact introduced by the TMS middleware processing is modest and clearly compensates the additional dependability guarantees offered to the users.

TMS is an example of a synergetic model combining dependability criteria with privacy conditions, showing that the main contributions of the thesis can be conjugated with reliability and availability services, providing a more complete solution to address the security issues of cloud services.

Publications The contribution presented in this Section was published in:

- **A Secure Email Repository Service using Public Untrusted Storage Clouds**. João Rodrigues, Bernardo Ferreira, and Henrique Domingos. In proceedings of the 5th Simpósio de Informática (INFORUM'13). Évora, Portugal,

September 2013.

- **TMS: A Trusted Mail Repository Service using Public Storage Clouds**. João Rodrigues, Bernardo Ferreira, and Henrique Domingos. In proceedings of the 8th Workshop on Middleware for Next Generation Internet Computing (MW4NG'13). Beijing, China, December 2013.

Prototypes A software prototype of this contribution is available at:

- `http://asc.di.fct.unl.pt/~bf/TMS.zip`

Chapter 7

Conclusions and Future Work

This Chapter closes the thesis. Section 7.1 summarizes the main contributions presented in the previous Chapters, while Section 7.2 discusses relevant future work directions.

7.1 Conclusions

In this thesis we have proposed, developed, and evaluated solutions for searching encrypted data in the cloud. Our goal was to improve efficiency, scalability, and usability of existing approaches in the literature. This was achieved with three main contributions, supporting various media types (with especial focus on text, images, and multimodal data) and offloading the heaviest computations for client applications to the cloud (namely indexing and training related computations).

The first main contribution of the thesis focused on text documents and how to efficiently manage and search them in the cloud. With this contribution we presented CloudCryptoSearch, an efficient middleware architecture to store text documents in the cloud and retrieve them through ranked multi-keyword queries. CloudCryptoSearch was designed as flexible architecture, possibly deployed in the users' trusted devices, in a local network proxy, or as a remote service in a computational cloud. To support these different modes of operation, partially-homomorphic and property-preserving encryption schemes were studied and employed, and a novel Linear Search Scheme was especially designed for text documents and their indexing and retrieval in the encrypted domain.

In our second main contribution we designed IES-CBIR, an Image Encryption

Scheme with Content Based Image Retrieval properties. To leverage IES-CBIR we also developed a new framework providing cloud-backed storing, sharing, and searching services of images with privacy guarantees. The proposed solution was able to improve efficiency and scalability for client applications retrieving images based on their color features, while displaying comparative retrieval precision results in real world datasets.

MIE, a Multimodal Indexable Encryption distributed middleware, was the result of the third core contribution of the thesis. Using this middleware, both desktop and mobile devices were able to efficiently store and retrieve multimodal data (i.e. data containing different media formats) in the cloud. MIE was able to securely outsource training and indexing computations to the cloud servers, largely improving client-side efficiency and scalability. This was achieved by conceiving a novel family of cryptographic encoding algorithms that preserved a controllable distance function between plaintexts.

Considering the work presented in the thesis and its core contributions, we believe we have positively answered the fundamental question addressed in the thesis: *Can we improve the performance, scalability, and resource management of both mobile and desktop devices storing, sharing, and searching multimedia data in the cloud with strong security guarantees?*

To answer this question different tradeoffs had to be explored in the design of the core techniques of the thesis, namely between security guarantees, efficiency of operations, and functionalities supported. The main insight of the thesis, which was materialized in all three core contributions, was how to outsource the most complex computations for client applications with privacy and security guarantees.

Implementation of the proposed cryptographic schemes and frameworks, as well as of key approaches from the state of art for baseline comparison, allowed obtaining detailed experimental results. These results demonstrated the improved efficiency and scalability offered by the contributions of the thesis, as well as optimized resource management. Retrieval precision evaluations revealed equal or comparable results regarding the literature on both encrypted and plaintext retrieval, while formal security analysis provided strong guarantees for the security properties of the contributions achieved.

Complementary Research Lines A complementary research vector in dependability issues proved that the main research vector of the thesis on cloud privacy

could be further complemented with reliability and availability guarantees in a synergetic way, offering a complete and integrated secure solution for mobile applications storing, sharing, and searching multimedia data in the cloud.

7.2 Future Work

A final conclusion of the thesis is that there still multiple open research directions that can be pursued in the field of Searchable Encryption.

One of the most important advances still to be achieved is an efficient Searchable Encryption solution that does not reveal information patterns with operations, in particular access patterns. As discussed in the previous Chapters, existing solutions that do not reveal these patterns exhibit, at best, linear query performance with the size of the database. Providing the same level of security while retaining sub-linear search performance is hence one of the main goals of the research field. From the research done in this thesis, we believe that designing such a solution entirely based on software will be very hard to achieve, if not impossible. However the employment of specialized hardware modules, including Trusted Platform Modules (TPM) (International Organization for Standardization 2015), has been seen in similar research fields (Arasu et al. 2013; Santos et al. 2012) and shows potential for improvements in the Searchable Encryption research area.

Another interesting research direction is in developing searchable encryption schemes that can efficiently support multiple users while fully addressing the security issues posed by malicious users. The thesis has improved the literature in this research vector by supporting multiple users both writing and searching data, and analyzing different mechanisms to minimize the impact of malicious users. However, as malicious users may be given access to multiple data objects and repositories before being discovered, more complete techniques may still be required for fully addressing this problem, including onion-layered encryption techniques for data at rest, authentication and revocation mechanisms, and key distribution protocols based on trusted third-party entities.

Further exploring the complementary research vector of the thesis constitutes another interesting open research direction. This complementary vector aimed on combining, in a synergetic way, the main contributions and their novel privacy foundations with reliability and availability guarantees. Some interesting results have already been achieved, as presented in Chapter 6, nonetheless the research effort in this vector is still ongoing and further interesting results can still be

achieved in this research vector.

Finally, we note that the mechanisms researched and developed in this thesis may have interesting applications in other critical and sensitive contexts. An interesting and relevant example is the storage and computation of scientific data, particularly biomedical and genomic data, where privacy and security are critical issues (Esteves-Verissimo and Decouchant 2016; Verissimo and Bessani 2013). In this context, we foresee the employment of techniques developed in this thesis with important benefits for achieving a balance between collaboration, data sharing, and privacy control.

7.3 Publications Summary

In the following we summarize the publications and results achieved in the context of the thesis:

- International Conference Papers
  - Bernardo Ferreira and Henrique Domingos. *Searching private data in a cloud encrypted domain*. In proceedings of the 10th Conference on Open Areas in Information Retrieval (OAIR'13). May 2013.
    - Full conference publication of the first main contribution of the thesis on searchable encryption for text data.
  - Bernardo Ferreira, João Rodrigues, João Leitão, and Henrique Domingos. Privacy-Preserving Content-Based Image Retrieval in the Cloud. In proceedings of the 34th IEEE Symposium on Reliable Distributed Systems (SRDS'15). September 2015.
    - Full conference publication of the second main contribution of the thesis on searchable encryption for visual data.
- International Workshop Papers
  - Bernardo Ferreira and Henrique Domingos. *Management and search of private data on storage clouds*. In proceedings of the 1st Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management (SDMCMM'12). December 2012.
    - Preliminary workshop publication of the first main contribution of the thesis on searchable encryption for text data.

ão Rodrigues, João Leitão, and Henrique Domingos. *Towards an Image Encryption Scheme with Content-Based Image Retrieval Properties*. In proceedings of the 9th Workshop on Data Privacy Management (DPM'14). September 2014.

- Preliminary workshop publication of the second main contribution of the thesis on searchable encryption for visual data.

- João Rodrigues, Bernardo Ferreira, and Henrique Domingos. *TMS: A Trusted Mail Repository Service using Public Storage Clouds*. In proceedings of the 8th Workshop on Middleware for Next Generation Internet Computing (MW4NG'13). December 2013.
  - Workshop publication of the first complementary contribution of the thesis on trustable and searchable cloud-based email repositories.

– National Conference Papers

- Bernardo Ferreira and Henrique Domingos. *Gestão e Pesquisa de Dados Privados em Nuvens de Armazenamento*. In proceedings of the 4th Simpósio de Informática (INFORUM'12). September 2012.
  - Preliminary national publication of the first main contribution of the thesis on searchable encryption for text data.
- Bernardo Ferreira, João Leitão, and Henrique Domingos. *Cifra Multimodal Indexável para Aplicações Móveis baseadas na Nuvem*. In proceedings of the 7th Simpósio de Informática (INFORUM'15). September 2015.
  - Preliminary national publication of the third main contribution of the thesis on searchable encryption for multimodal data.
- João Rodrigues, Bernardo Ferreira, and Henrique Domingos. *A Secure Email Repository Service using Public Untrusted Storage Clouds*. In proceedings of the 5th Simpósio de Informática (INFORUM'13). September 2013.
  - Preliminary national publication of the first complementary contribution of the thesis on trustable and searchable cloud-based email repositories.
- João Rodrigues, Bernardo Ferreira, João Leitão, and Henrique Domingos. *DocNuvem: Edição Colaborativa de Documentos na Nuvem com*

Garantias de Privacidade. In proceedings of the 6th Simpósio de Informática (INFORUM'14). September 2014.

- National publication of the second complementary contribution of the thesis on oblivious cloud-based collaborative document edition.

– International Conference Posters and Demos

- Bernardo Ferreira and Henrique Domingos. *CloudCryptoSearch: a prototype for secure searching of private data in cloud encrypted domains*. Conference Demo. In proceedings of the 10th Conference on Open Areas in Information Retrieval (OAIR'13). May 2013.
  - Conference demonstration of a prototype and presentation of a poster of the first main contribution of the thesis on searchable encryption for text data.
- Bernardo Ferreira, João Leitão, and Henrique Domingos. *Multimodal Indexable Encryption for Mobile Cloud-based Applications*. Conference Poster. In the 10th ACM European Conference on Computer Systems (EuroSys'15). April 2015.
  - Preliminary conference poster presentation of the third main contribution of the thesis on searchable encryption for multimodal data.

– International Oral Presentations and PhD Workshops

- Bernardo Ferreira and Henrique Domingos. *Security and Dependability in Cloud based Critical Online Applications*. Oral Communication and Poster. In the 7th EuroSys Doctoral Workshop (EuroDW'13). April 2013.
  - Oral and poster presentations of the topic of the thesis in an international PhD Workshop.
- Bernardo Ferreira, João Leitão, and Henrique Domingos. *Indexable Encryption: Searching Cloud-Stored Multimodal Data on Mobile Devices*. Oral Communication. In the PhD Forum of the 34th IEEE Symposium on Reliable Distributed Systems. September 2015.
  - Oral presentation in a PhD Workshop of the third main contribution of the thesis on searchable encryption for multimodal data.

– Technical Reports and Works in Submission

- Bernardo Ferreira, João Rodrigues, João Leitão, and Henrique Domingos. *Practical Privacy-Preserving Content-Based Retrieval in Cloud Image Repositories*. Technical Report. Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa. December 2015.
    - Extended journal version of the second main contribution of the thesis on searchable encryption for visual data. In submission to an international journal.
- Bernardo Ferreira, João Leitão, and Henrique Domingos. *Multimodal Indexable Encryption for Mobile Cloud-based Applications*. Technical Report. Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa. February 2016.
    - Full conference version of the third main contribution of the thesis on searchable encryption for multimodal data. In submission to an international conference.

– Software Prototypes

- CloudCryptoSearch – This prototype presents a searchable encryption middleware, allowing the storage, update, and ranked multi-keyword searching of encrypted text documents in a remote server or cloud.
    - Prototype page: `https://github.com/bernymac/CloudCryptoSearch`
- IES-CBIR – In this prototype we present a searchable encryption framework for visual data (i.e. images). It allows the secure storage, update, and retrieval of images based on their contents.
    - Prototype page: `https://github.com/bernymac/IES-CBIR`
- MIE – MIE is a searchable encryption framework for multimodal data (i.e. data containing multiple media formats simultaneously) especially designed for supporting mobile devices and optimizing their storage, computation, and battery resources. It allows multiple users to securely store, update, and retrieve multimodal data in commercial clouds.
    - Prototype page: `https://github.com/bernymac/MIE`

Bibliography

Abd-El-Malek, M., G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie (2005). "Fault-scalable Byzantine fault-tolerant services". In: *Proceedings of the 20th ACM Symposium on Operating System Principles - SOSP'05*, p. 59.

Abdalla, M., M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi (2008). "Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions". In: *Journal of Cryptology* 21.3, pp. 350–391.

Agrawal, R., J. Gehrke, D. Gunopulos, and P. Raghavan (1998). "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications". In: *Proceedings of the 1998 ACM SIGMOD international Conference on Management of data*. ACM, pp. 94–105.

Agrawal, R., J. Kiernan, R. Srikant, and Y. Xu (2004). "Order preserving encryption for numeric data". In: *Proceedings of the 2004 ACM SIGMOD international Conference on Management of Data*. ACM, pp. 563–574.

Amazon Web Services (AWS) (2016a). *Amazon EC2 Instance Types*. URL: https://aws.amazon.com/pt/ec2/instance-types/.

— (2016b). *S3 Cloud Storage Service*. URL: https://aws.amazon.com.

— (2016c). *S3 Service Level Agreement*. URL: https://aws.amazon.com/en/s3/sla/.

Apon, D., J. Katz, E. Shi, and A. Thiruvengadam (2014). "Verifiable oblivious storage". In: *Proceedings of the 17th International Conference on Practice and Theory in Public-Key Cryptography - PKC'14*. Springer, pp. 131–148.

Apple (2016a). *Apple iCloud Email Service*. URL: https://www.icloud.com/.

— (2016b). *iCloud Photo Library*. URL: http://www.apple.com/icloud/photos.

Arasu, A., S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan (2013). "Orthogonal Security with Cipherbase". In: *Proceedings of the 6th Biennial Conference on Innovative Data Systems Research (CIDR'13)*.

Armbrust, M., A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia (2010). "A view of cloud computing". In: *Communications of the ACM (CACM)* 53.4, pp. 50–58.

Asghar, M. R., C. Bernardini, and B. Crispo (2016). "PROTECTOR: Privacy-preserving information lookup in content-centric networks". In: *Proceedings of the IEEE International Conference on Communications - ICC'16*.

Asmuth, C. and J. Bloom (1983). "A modular approach to key safeguarding". In: *IEEE Transactions on Information Theory* 30.2, pp. 208–210.

Atrey, P. K., M. A. Hossain, A. El Saddik, and M. S. Kankanhalli (2010). "Multimodal fusion for multimedia analysis: A survey". In: *Multimedia Systems* 16.6, pp. 345–379.

Baldimtsi, F. and O. Ohrimenko (2015). "Sorting and Searching Behind the Curtain". In: *Proceedings of the 9th International Conference on Financial Cryptography and Data Security*.

Bay, H., T. Tuytelaars, and L. V. Gool (2006). "SURF: Speeded Up Robust Features". In: *Proceedings of the 9th European Conference on Computer Vision - ECCV'06*. Springer, pp. 404–417.

Bellare, M. and P. Rogaway (1993). "Random Oracles are Practical : A Paradigm for Designing Efficient Protocols". In: *Proceedings of the 1st ACM conference on Computer and communications security*. November 1993, pp. 1–20.

Bellare, M., A. Boldyreva, and A. O. Neill (2007). "Deterministic and Efficiently Searchable Encryption". In: *Proceedings of the 27th International Cryptology Conference - CRYPTO'07*, pp. 535–552.

Bessani, A., M. Correia, B. Quaresma, F. André, and P. Sousa (2013). "DepSKY: Dependable and Secure Storage in a Cloud-of-Clouds". In: *ACM Transactions on Storage* 9.4.

Blakley, G. R. (1979). "Safeguarding cryptographic keys". In: *National Computer Conference - NCC'79*, p. 313.

Böhm, C., S. Berchtold, and D. a. Keim (2001). "Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases". In: *ACM Computing Surveys (CSUR)* 33.3, pp. 322–373.

Boldyreva, A. (2003). "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme". In: *Proceedings of the International Workshop on Practice and Theory in Public Key Cryptography - PKC'03*. Vol. 2567, pp. 31–46.

Boldyreva, A., N. Chenette, Y. Lee, and A. O'neill (2009). "Order-preserving symmetric encryption". In: *Proceedings of the 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques - EUROCRYPT'09*. Springer, pp. 224–241.

Boldyreva, A., N. Chenette, and A. O. Neill (2011). "Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions". In: *Proceedings of the 31st International Cryptology Conference - CRYPTO'11*.

Boneh, D. and M. Franklin (2001). "Identity-based encryption from the Weil pairing". In: *Proceedings of the 21st International Cryptology Conference - CRYPTO'01*. Vol. 32. 3. Springer, pp. 213–229.

Boneh, D., G. Di Crescenzo, R. Ostrovsky, and G. Persiano (2004). "Public key encryption with keyword search". In: *Proceedings of the 23th Annual International Conference on the Theory and Applications of Cryptographic Techniques - EUROCRYPT'04*. Springer, pp. 506–522.

Boneh, D., C. Gentry, and B. Waters (2005a). "Collusion resistant broadcast encryption with short ciphertexts and private keys". In: *25st International Cryptology Conference - CRYPT'05*. Springer, pp. 258–275.

Boneh, D., E. Goh, and K Nissim (2005b). "Evaluating 2-DNF formulas on ciphertexts". In: *Proceedings of the Second Theory of Cryptography Conference - TCC'05* 3378, pp. 325–341.

Boneh, D., A. Sahai, and B. Waters (2011). "Functional encryption: Definitions and challenges". In: *Proceedings of the 8th Theory of Cryptography Conference - TCC'11*, pp. 253–273.

Bösch, C., P. Hartel, W. Jonker, and A. Peter (2015). "A Survey of Provably Secure Searchable Encryption". In: *ACM Computing Surveys (CSUR)* 47.2, 18:1–18:51.

Boufounos, P. and S. Rane (2011). "Secure binary embeddings for privacy preserving nearest neighbors". In: *IEEE International Workshop on Information Forensics and Security*. IEEE, pp. 1–6.

Brakerski, Z. and V. Vaikuntanathan (2014). "Efficient fully homomorphic encryption from (standard) LWE". In: *SIAM Journal on Copmputing* 43.2, pp. 831–871.

Brakerski, Z., C. Gentry, and V. Vaikuntanathan (2012). "(Leveled) Fully homomorphic encryption without bootstrapping". In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference - ITCS'12*, pp. 309–325.

Brandenburger, M., C. Cachin, and N. Knezevic (2015). "Don't Trust the Cloud, Verify: Integrity and Consistency for Cloud Object Stores". In: *Proceedings of the 8th ACM International Systems and Storage Conference - SYSTOR'15*. ACM.

Brin, S. and L. Page (1998). "The Anatomy of a Large-Scale Hypertextual Web Search Engine". In: *Proceedings of the Seventh International World-Wide Web Conference - WWW'98*.

Brunette, G., R. Mogull, and Others (2009). "Security guidance for critical areas of focus in cloud computing v2.1". In: *Cloud Security Alliance*, pp. 1–76.

Canetti, R. (2001). "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science - FOCS'01*, pp. 136–145.

Canetti, R., O. Goldreich, and S. Halevi (2004). "The Random Oracle Methodology, Revisited". In: *Journal of the ACM* 51.4, p. 38.

Cao, N., C. Wang, M. Li, K. Ren, and W. Lou (2014). "Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data". In: *IEEE Transactions on Parallel and Distribibuted Systems* 25.1, pp. 222–233.

Cash, D., J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M Rosu, and M. Steiner (2014). "Dynamic searchable encryption in very-large databases: Data structures and implementation". In: *Proceedings of the The 21th Annual Network and Distributed System Security Symposium -NDSS'14*. Vol. 14.

Cash, D., P. Grubbs, J. Perry, and T. Ristenpart (2015). "Leakage-Abuse Attacks Against Searchable Encryption". In: *Proceedings of the 22nd ACM Conference on Computer and Communications Security - CCS'15*. ACM, pp. 668–679.

Castro, M., B. Liskov, and Others (1999). "Practical Byzantine fault tolerance". In: *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation - OSDI'99*. Vol. 99, pp. 173–186.

Catteddu, D. and G. Hogben (2009). *Cloud Computing - Benefits, risks and recommendations for information security*. Tech. rep. European Network and Information Security Agency (ENISA).

Chase, M. and S. Kamara (2010). "Structured encryption and controlled disclosure". In: *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security - ASIACRYPT'10*. Springer, pp. 577–594.

Chen, A. (2010). *GCreep: Google Engineer Stalked Teens, Spied on Chats*. Gawker. URL: http://gawker.com/5637234.

Chow, R., P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina (2009). "Controlling data in the cloud: outsourcing computation without outsourcing control". In: *Proceedings of the ACM Cloud Computing Security Workshop - CCSW'09*.

Cisco (2016). *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015 – 2020*. Tech. rep. URL: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white{_}paper{_}c11-520862.html.

ComScore (2016). *The 2016 U.S. Mobile App Report*. Tech. rep.

Cook, T. (2016). *A Message to Our Customers*. Apple. URL: https://www.apple.com/customer-letter/.

Coron, J.-S., A. Mandal, D. Naccache, and M. Tibouchi (2011). "Fully Homomorphic Encryption over the Integers with Shorter Public Keys". In: *Proceedings of the 31st International Cryptology Conference - CRYPTO'11*. Vol. 6841, pp. 487–504.

Cowling, J., D. Myers, B. Liskov, R. Rodrigues, and Liuba Shrira (2006). "HQ replication: A hybrid quorum protocol for Byzantine fault tolerance". In: *Proceedings of the 7th Symposium on Operating Systems Design and Implementation - OSDI'06*, pp. 177–190.

Curtmola, R., J. Garay, S. Kamara, and R. Ostrovsky (2006). "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions". In:

Proceedings of the 13th ACM Conference on Computer and Communications Security - CCS'06, pp. 79–88.

Danezis, G. and S. Gürses (2010). "A critical review of 10 years of Privacy Technology". In: *Proceedings of Surveillance Cultures: a Global Surveillance Society*, pp. 1–16.

Datta, R., D. Joshi, J. Li, and J. Z. Wang (2008). "Image retrieval". In: *ACM Computing Surveys (CSUR)* 40.2, pp. 1–60.

Dautrich, J., E. Stefanov, and E. Shi (2014). "Burst ORAM: Minimizing ORAM Response Times for Bursty Access Patterns". In: *Proceedings of the 23rd USENIX Security Symposium - Security'14*, pp. 749–764.

Desmedt, Y. (1987). "Society and group oriented cryptography: a new concept". In: *Proceedings of the 7th International Cryptology Conference - CRYPTO'87*, pp. 120–127.

Devadas, S., M. van Dijk, C. W. Fletcher, L. Ren, E. Shi, and D. Wichs (2016). "Onion ORAM: A constant bandwidth blowup oblivious RAM". In: *Proceedings of the 13th International Theory of Cryptography Conference - TCC'16*. Springer, pp. 145–174.

Dong, C., G. Russello, and N. Dulay (2011). "Shared and searchable encrypted data for untrusted servers". In: *Journal of Computer Security* 19.3, pp. 367–397.

Dropbox (2016). *Dropbox Cloud Storage Service*. URL: https://www.dropbox.com/.

ElGamal, T. (1984). "A public key cryptosystem and a signature scheme based on discrete logarithms". In: *Proceedings of the 4h International Cryptology Conference - CRYPTO'84*. Springer.

Esteves-Verissimo, P. and J. Decouchant (2016). *The big data deluge in biomedicine: addressing the privacy vs. sharing dilemma*. Tech. rep. University of Luxembourg - SnT, pp. 1–2.

Feldman, A. J., W. P. Zeller, M. J. Freedman, and E. W. Felten (2010). "SPORC: Group Collaboration using Untrusted Cloud Resources." In: *Proceedings of the 9th Symposium on Operating Systems Design and Implementation - OSDI'10*. Vol. 10, pp. 337–350.

Ferreira, B. and H. Domingos (2012a). "Gestão e Pesquisa de Dados Privados em Nuvens de Armazenamento". In: *Atas do 4º Simpósio Nacional de Informática - INFORUM'12*. Monte da Caparica, Portugal: Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa.

— (2012b). "Management and search of private data on storage clouds". In: *Proceedings of the First International Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management - SDMCMM '12*. New York, New York, USA: ACM Press, pp. 1–6.

— (2013a). "CloudCryptoSearch: a prototype for secure searching of private data in cloud encrypted domains". In: *Proceedings of the the 10th International Conference in the RIAO series - OAIR'13*. C.I.D., pp. 209–210.

— (2013b). "Searching private data in a cloud encrypted domain". In: *Proceedings of the the 10th International Conference in the RIAO series - OAIR'13*, pp. 165–172.

Ferreira, B., J. Rodrigues, J. Leitão, and H. Domingos (2014). "Towards an Image Encryption Scheme with Content-Based Image Retrieval Properties". In: *Proceedings of the 9th International Workshop on Data Privacy Management - DPM'14*. Springer.

Ferreira, B., J. Leitão, and H. Domingos (2015a). "Cifra Multimodal Indexável para Aplicações Móveis baseadas na Nuvem". In: *Atas do 7º Simpósio Nacional de Informática - INFORUM'15*.

— (2015b). "Indexable Encryption: Searching Cloud-Stored Multimodal Data on Mobile Devices". In: *PhD Forum of the 34th International Symposium on Reliable Distributed Systems*.

— (2015c). "Multimodal Indexable Encryption for Mobile Cloud-based Applications". In: *Poster Session of the ACM European Conference on Computer Systems - EuroSys'15*.

Ferreira, B., J. Rodrigues, J. Leitão, and H. Domingos (2015d). "Privacy-Preserving Content-Based Image Retrieval in the Cloud". In: *Proceedings of the 34th International Symposium on Reliable Distributed Systems - SRDS'15*. IEEE.

Ferreira, B., J. Leitão, and H. Domingos (2016a). *Multimodal Indexable Encryption for Mobile Cloud-based Applications*. Tech. rep. NOVA LINCS.

Ferreira, B., J. Rodrigues, J. Leitão, and H. Domingos (2016b). *Practical Privacy-Preserving Content-Based Retrieval in Cloud Image Repositories*. Tech. rep. NOVA LINCS.

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

Fung, B. (2015). *In 5 years, 80 percent of the whole Internet will be online video*. The Washington Post. URL: https://www.washingtonpost.com/news/the-switch/wp/2015/05/27/in-5-years-80-percent-of-the-whole-internet-will-be-online-video/.

Gentry, C. (2009). "A fully homomorphic encryption scheme". PhD thesis. Stanford University.

Gentry, C. and S. Halevi (2011). "Implementing Gentry's Fully-Homomorphic Encryption Scheme". In: *Proceedings of the 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques - EUROCRYPT'11* 6632, pp. 129–148.

Gentry, C., S. Halevi, and N. P. Smart (2012). "Homomorphic evaluation of the AES circuit". In: *Proceedings of the 32nd International Cryptology Conference - CRYPTO'12*. Springer, pp. 850–867.

Global Web Index (2013). *Instagram tops the list of social network growth*. GWI Q4 2013 Social Report. URL: http://blog.globalwebindex.net/instagram-tops-list-of-growth.

Goh, E., H Shacham, N Modadugu, and D Boneh (2003). "SiRiUS: Securing remote untrusted storage". In: *Proceedings of the The 10th Annual Network and Distributed System Security Symposium - NDSS '03*, pp. 131–145.

Goh, E. J. (2003). "Secure Indexes". In: *IACR Cryptology ePrint Archive*, pp. 1–19.

Goldreich, O (1987). "Towards a theory of software protection and simulation by oblivious RAMs". In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing - STOC'87*, pp. 182–194.

Goldreich, O. and R. Ostrovsky (1996). "Software protection and simulation on oblivious RAMs". In: *Journal of the ACM* 43.3, pp. 431–473.

Goldwasser, S. and Y. T. Kalai (2015). "Cryptographic Assumptions: A Position Paper". In: *Theory of Cryptography Conference*. Springer, pp. 505–522.

Goldwasser, S., Y. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich (2013). "Reusable garbled circuits and succinct functional encryption". In: *Proceedings of the 45th Annual ACM Symposium on Theory of Computing - STOC'13*, pp. 555–564.

Goldwasser, S. and S. Micali (1982). "Probabilistic encryption & how to play mental poker keeping secret all partial information". In: *Proceedings of the 14th Annual ACM Symposium on Theory of Computing - STOC '82*, pp. 365–377.

Google (2016a). *Android Developer Center*. URL: http://developer.android.com.

— (2016b). *GMail - Google Email Service*. URL: https://mail.google.com.

— (2016c). *Google Cloud Storage Service*. URL: https://cloud.google.com/storage/.

— (2016d). *Google Photos*. URL: https://www.google.com/photos.

— (2016e). *Power Profiles for Android*. URL: https://source.android.com/devices/tech/power/index.html.

Goyal, V., O. Pandey, A. Sahai, and B. Waters (2006). "Attribute-based Encryption for Fine-grained Access Control of Encrypted Data". In: *Proceedings of the 13th ACM Conference on Computer and Communications Security - CCS'06*, pp. 89–98.

Greenwald, G. and E. MacAskill (2013). *NSA Prism program taps in to user data of Apple, Google and others*. The Guardian. URL: http://tinyurl.com/oea3g8t.

Hacigümüş, H., B. Iyer, C. Li, and S. Mehrotra (2002). "Executing SQL over encrypted data in the database-service-provider model". In: *Proceedings of the 2002 ACM SIGMOD international Conference on Management of Data - SIGMOD'02*, pp. 216–227.

Hahn, F. and F. Kerschbaum (2014). "Searchable Encryption with Secure and Efficient Updates". In: *Proceedings of the 21st ACM Conference on Computer and Communications Security - CCS'14*. ACM, pp. 310–320.

Halderman, J. A. and S. D. Schoen (2009). "Lest we remember: cold-boot attacks on encryption keys". In: *Communications of the ACM*. Vol. 52. 5.

Hartigan, J. A. (1975). *Clustering algorithms*. Wiley, 364 p.

Heinly, J., E. Dunn, and J.-M. Frahm (2012). "Comparative evaluation of binary features". In: *Proceedings of the 12th European Conference on Computer Vision - ECCV '12*. Springer, pp. 759–773.

Heinz, S. and J. Zobel (2003). "Efficient single-pass index construction for text databases". In: *Journal of the American Society for Information Science and Technology* 54.8, pp. 713–729.

Hjaltason, G. R. and H. Samet (2003). "Index-driven similarity search in metric spaces". In: *ACM Transactions on Database Systems (TODS)* 28.4, pp. 517–580.

Howard, J. H. (1988). *An overview of the andrew file system*. Carnegie Mellon University, Information Technology Center.

Hsu, C.-Y., C.-S. Lu, and S.-c. Pei (2012). "Image Feature Extraction in Encrypted Domain With Privacy-Preserving SIFT". In: *IEEE Transactions on Image Processing* 21.11, pp. 4593–4607.

Huiskes, M. J. and M. S. Lew (2008). "The MIR Flickr Retrieval Evaluation". In: *Proceedings of the 1st ACM iIternational Conference on Multimedia Information Retrieval - MIR'08*. New York, NY, USA: ACM.

Ilker Nadi Bozkurt, K. Kaya, A. A. Selçuk, and A. M. Güloglu (2008). "Threshold Cryptography Based on Blakely Secret Sharing". In: *Information Sciences*.

International Organization for Standardization (2015). *ISO/IEC 11889-1:2015 - Trusted platform module library*. Tech. rep., p. 257.

Ion, I., N. Sachdeva, P. Kumaraguru, and S. Čapkun (2011). "Home is safer than the cloud! Privacy Concerns for Consumer Cloud Storage". In: *Proceedings of the Seventh Symposium on Usable Privacy and Security - SOUPS '11*, p. 1.

Islam, M. S., M. Kuzu, and M. Kantarcioglu (2012). "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation". In: *Proceedings of the The 19th Annual Network and Distributed System Security Symposium - NDSS'12*.

Itseez (2016). *OpenCV: Open Source Computer Vision*. URL: http://opencv.org.

Jegou, H., M. Douze, and C. Schmid (2008). "Hamming embedding and weak geometric consistency for large scale image search". In: *Proceedings of the 10th European Conference on Computer Vision - ECCV'08*. Springer, pp. 304–317.

Jeon, J, V Lavrenko, and R Manmatha (2003). "Automatic image annotation and retrieval using cross-media relevance models". In: *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval - SIGIR'03*, pp. 119–126.

Jeong, S., C. S. Won, and R. M. Gray (2004). "Image retrieval using color histograms generated by Gauss mixture vector quantization". In: *Computer Vision and Image Understanding* 94.1-3, pp. 44–66.

Jones, K. S., S. Walker, and S. E. Robertson (2000). "A probabilistic model of information retrieval: development and comparative experiments: Part 2". In: *Information Processing & Management* 36.6, pp. 809–840.

Juels, A. and M. Wattenberg (1999). "A Fuzzy Commitment Scheme". In: *Proceedings of the 6th ACM Conference on Computer and Communications Security - CCS'99*. ACM, pp. 28–36.

Kallahalla, M., E. Riedel, R. Swaminathan, Q. Wang, and K. Fu (2003). "Plutus: Scalable Secure File Sharing on Untrusted Storage". In: *Proceedings of the 2nd Usenix Conference on File and Storage Tecnhologies - Fast'03*. Vol. 3, pp. 29–42.

Kamara, S. and K. Lauter (2010). "Cryptographic Cloud Storage". In: *Proceedings of the Workshop on Real-Life Cryptographic Protocols and Standardization*, pp. 136–149.

Kamara, S. and C. Papamanthou (2013). "Parallel and dynamic searchable symmetric encryption". In: *Proceedings of the 7th International Conference on Financial Cryptography and Data Security - FC'13*, pp. 1–15.

Kamara, S., C. Papamanthou, and T. Roeder (2012). "Dynamic searchable symmetric encryption". In: *Proceedings of the 19th ACM Conference on Computer and Communications Security - CCS'12*. ACM, pp. 965–976.

Katz, J. and Y. Lindell (2007). *Introduction to Modern Cryptography*. CRC PRESS.

Kaya, K. and A. A. Selçuk (2007). "Threshold cryptography based on Asmuth-Bloom secret sharing". In: *Information Sciences* 177.19, pp. 4148–4160.

Kerschbaum, F. (2007). "Distance-preserving pseudonymization for timestamps and spatial data". In: *Proceedings of the Workshop on Workshop on Privacy in the Electronic Society - WPES'07*. ACM.

Kerschbaum, F. (2015). "Frequency-Hiding Order-Preserving Encryption". In: *Proceedings of the 22nd ACM Conference on Computer and Communications Security - CCS'15*, pp. 656–667.

Kerschbaum, F. and A. Schröpfer (2014). "Optimal Average-Complexity Ideal-Security Order-Preserving Encryption". In: *Proceedings of the 21st ACM Conference on Computer and Communications Security - CCS'14*, pp. 275–286.

Khalaf, S. (2014). *Health and Fitness Apps Finally Take Off, Fueled by Fitness Fanatics*. Flurry Insights. URL: http://tinyurl.com/q4wyl7j.

Kim, B. H. and D. Lie (2015). "Caelus: Verifying the Consistency of Cloud Services with Battery-Powered Devices". In: *Proceedings of the 36th IEEE Symposium on Security and Privacy - S&P'15*. IEEE.

Klensin, J. C. (2008). *RFC 5321 - Simple Mail Transfer Protocol*. Tech. rep. Internet Engineering Task Force.

Klimt, B. and Y. Yang (2004). "Introducing the Enron Corpus". In: *CEAS*.

Kotla, R., L. Alvisi, M. Dahlin, A. Clement, and E. Wong (2009). "Zyzzyva: Speculative Byzantine Fault Tolerance". In: *ACM Transactions on Computer Systems (TOCS)* 27.4, 7:1–39.

Kuzu, M, M. S. Islam, and M Kantarcioglu (2012). "Efficient Similarity Search over Encrypted Data". In: *Proceedings of the 28th IEEE International Conference on Data Engineering - ICDE'12*, pp. 1156–1167.

Lamport, L., R. Shostak, and M. Pease (1982). "The Byzantine Generals Problem". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4.3, pp. 382–401.

Lazebnik, S., C. Schmid, and J. Ponce (2006). "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories". In: *Proceedings of the 19th IEEE Conference on Computer Vision and Pattern Recognition - CVPR'06*. Vol. 2. IEEE, pp. 2169–2178.

Lewis, D. (2014). *iCloud Data Breach: Hacking And Celebrity Photos*. Forbes. URL: https://tinyurl.com/nohznmr.

Li, J., M. N. Krohn, D. Mazières, and D. Shasha (2004). "Secure Untrusted Data Repository (SUNDR)." In: *Proceedings of the 6th Symposium on Operating Systems Design and Implementation - OSDI'04*. Vol. 4, p. 9.

Liu, Q., C. C. Tan, J. Wu, and G. Wang (2012). "Efficient information retrieval for ranked queries in cost-effective cloud environments". In: *Proceedings of the 31st IEEE International Conference on Computer Communications - INFOCOM'12*. IEEE, pp. 2581–2585.

Lowe, D. G. (2004). "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision* 60.2, pp. 91–110.

Lu, W., A. Swaminathan, A. L. Varna, and M. Wu (2009). "Enabling Search over Encrypted Multimedia Databases". In: *IS&T/SPIE Electronic Imaging* 7254, pp. 725418–725418–11.

Luna Cloud (2016). *Luna Cloud Storage Service*. URL: http://www.lunacloud.com/.

Lv, Y. and C. Zhai (2011). "When documents are very long, BM25 fails!" In: *Proceedings of the 34th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval - SIGIR'11*. I. New York, New York, USA: ACM Press, pp. 1103–1104.

Lyubashevsky, V., C. Peikert, and O. Regev (2013). "On ideal lattices and learning with errors over rings". In: *Journal of the ACM* 60.6, p. 43.

Mahajan, P., S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish (2011). "Depot: Cloud Storage with Minimal Trust". In: *ACM Transactions on Computer Systems* 29.4, pp. 1–38.

Manning, C. D., P. Raghavan, and H. Schütze (2009). *An Introduction to Information Retrieval*. Vol. 1. Cambridge University Press.

Mayberry, T., E.-O. Blass, and A. H. Chan (2014). "Efficient private file retrieval by combining ORAM and PIR". In: *Proceedings of the The 21th Annual Network and Distributed System Security Symposium -NDSS'14*.

MediaBuzz (2010). *Email Integrity: An Emerging Business Issue*. URL: http://www.mediabuzz.com.sg/best-practices-march-april-10/email-integrity-an-emerging-business-issue.

Meeker, M. (2015). "Internet Trends 2015". In: *Code Conference*.

— (2016). "Internet Trends 2016". In: *Code Conference*.

Mell, P. and T. Grance (2011). *The NIST definition of cloud computing*. Tech. rep.

Menezes, A. J., P. C. Van Oorschot, and S. A. Vanstone (1996). *Handbook of applied cryptography*. CRC press.

Microsoft (2016a). *Azure Cloud Storage Service*. URL: https://azure.microsoft.com/.

— (2016b). *HealthVault*. URL: https://www.healthvault.com/.

— (2016c). *Hotmail - Microsoft Email Service*. URL: https://www.hotmail.com/.

Mitra, S., R. K. Panta, M.-R. Ra, and S. Bagchi (2016). "Partial-Parallel-Repair (PPR): A Distributed Technique for Repairing Erasure Coded Storage". In: *Proceedings of the ACM European Conference on Computer Systems - EuroSys'16*.

Mourão, A., F. Martins, and J. Magalhães (2013). "NovaSearch at TREC 2013 Federated Web Search Track : Experiments with rank fusion". In: *Proceedings of the 22nd Text REtrieval Conference (TREC'13)*. Task 1, pp. 1–8.

— (2014). "Multimodal medical information retrieval with unsupervised rank fusion". In: *Computerized Medical Imaging and Graphics*.

Müller, H., W. Müller, D. M. Squire, S. Marchand-Maillet, and T. Pun (2001). "Performance evaluation in content-based image retrieval: overview and proposals". In: *Pattern Recognition Letters* 22.5, pp. 593–601.

Myers, J. G. and M. T. Rose (1996). *RFC 1939 - Post Office Protocol*. Tech. rep. Internet Engineering Task Force.

National Vulnerability Database (2016). *CVE Statistics*. URL: http://web.nvd.nist.gov/view/vuln/statistics.

Naveed, M. (2015). *The Fallacy of Composition of Oblivious RAM and Searchable Encryption*. Tech. rep. Cryptology ePrint Archive, Report 2015/668.

Naveed, M., M. Prabhakaran, and C. A. Gunter (2014). "Dynamic Searchable Encryption via Blind Storage". In: *Proceedings of the 35th IEEE Symposium on Security and Privacy - S&P'14*.

Neuman, B. C. and T. Ts'o (1994). "Kerberos: An authentication service for computer networks". In: *IEEE Communications Magazine* 32.9, pp. 33–38.

Nirvanix (2013). *Nirvanix Cloud Storage Service*. URL: http://nirvanix.com.

Nistér, D., H. Stewénius, D. Nister, and H. Stewenius (2006). "Scalable recognition with a vocabulary tree". In: *Proceedings of the 19th IEEE Conference on Computer Vision and Pattern Recognition - CVPR'06*. IEEE, pp. 2161–2168.

Nourian, A. and M. Maheswaran (2013). "Privacy aware image template matching in clouds using ambient data". In: *The Journal of Supercomputing* 66.2, pp. 1049–1070.

OpenSSL Software Foundation (2016). *OpenSSL: Cryptography and SSL/TLS Toolkit*. URL: https://www.openssl.org.

Ostrovsky, R. (1990). "Efficient Computation on Oblivious RAMs". In: *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing - STOC'90*. ACM.

Paillier, P. (1999). "Public-key cryptosystems based on composite degree residuosity classes". In: *Proceedings of the 18th Annual International Conference on the Theory and Applications of Cryptographic Techniques - EUROCRYPT'99*, pp. 223–238.

Paillier, P. and D. Pointcheval (1999). "Efficient public-key cryptosystems provably secure against active adversaries". In: *Proceedings of the 5th International Conference on the Theory and Application of Cryptology and Information Security - ASIACRYPT'99*, pp. 1–13.

Popa, R. A., E. Stark, J. Helfer, and S. Valdez (2014). "Building web applications on top of encrypted data using Mylar". In: *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI'14)*.

Popa, R. A., C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan (2011). "CryptDB: Protecting Confidentiality with Encrypted Query Processing". In: *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11)*. ACM, pp. 85–100.

— (2012). "CryptDB : Processing Queries on an Encrypted Database". In: *Communications of the ACM (CACM)* 55.9, pp. 103–111.

Popa, R. A., F. H. Li, and N. Zeldovich (2013). "An Ideal-Security Protocol for Order-Preserving Encoding". In: *Proceedings of the 34th IEEE Symposium on Security and Privacy - S&P'13*. IEEE.

Privacy Rights Clearinghouse (2009). *A chronology of data breaches*.

Rabin, M. O. (1989). "Efficient dispersal of information for security, load balancing, and fault tolerance". In: *Journal of the ACM* 36.2, pp. 335–348.

Rackspace (2016). *Rackspace Cloud Storage Service*. URL: https://www.rackspace.com.

Rane, S. and P. T. Boufounos (2013). "Privacy-Preserving Nearest Neighbor Methods". In: *IEEE Signal Processing Magazine* 30.2, pp. 18–28.

Rivest, R. L., A Shamir, and L Adleman (1978a). "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM (CACM)* 21.2, pp. 120–126.

Rivest, R. L., L. Adleman, and M. L. Dertouzos (1978b). "On data banks and privacy homomorphisms". In: *Foundations of Secure Computation* 4.11, pp. 169–180.

Rodrigues, J., B. Ferreira, and H. Domingos (2013a). "A Secure Email Repository Service using Public Untrusted Storage Clouds". In: *Atas do 5º Simpósio Nacional de Informática - INFORUM'13*. Évora, Portugal.

— (2013b). "TMS: A Trusted Mail Repository Service using Public Storage Clouds". In: *Proceedings of the 8th International Workshop on Middleware for Next Generation Internet Computing - MW4NG'13*. ACM.

Rodrigues, J., B. Ferreira, J. Leitão, and H. Domingos (2014). "DocNuvem: Edição Colaborativa de Documentos na Nuvem com Garantias de Privacidade". In: *Atas do 6º Simpósio Nacional de Informática - INFORUM'14*.

Rodrigues, R. and B. Liskov (2005). "High availability in DHTs: Erasure coding vs. replication". In: *Peer-to-Peer Systems IV*. Vol. 3640. Section 2, pp. 226–239.

Rushe, D. (2013). *Google: don't expect privacy when sending to Gmail*. The Guardian. URL: http://tinyurl.com/kjga34x.

Santos, N., R. Rodrigues, K. P. Gummadi, and S. Saroiu (2012). "Policy-sealed data: A new abstraction for building trusted cloud services". In: *Proceedings of the 21st USENIX conference on Security Symposium - Security'12*, pp. 175–188.

Schneider, M. and T. Schneider (2014). "Notes on Non-Interactive Secure Comparison in " Image Feature Extraction in the Encrypted Domain with Privacy-Preserving SIFT "". In: *Proceedings of the 2nd ACM Workshop on Information Hiding and Multimedia Security*.

Schneier, B. (1996). *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons.

Services, A. W. (2011). *Client-Side Data Encryption with the AWS SDK for Java and Amazon S3*. URL: http://aws.amazon.com/articles/2850096021478074.

Shamir, A. (1979). "How To Share a Secret". In: *Communications of the ACM (CACM)* 22.11, pp. 612–613.

Shaw, J. A. and E. A. Fox (1995). "Combination of multiple searches". In: *NIST SPECIAL PUBLICATION SP*, p. 105.

Shen, E., E. Shi, and B. Waters (2009). "Predicate privacy in encryption systems". In: *Proceedings of the 6th Theory of Cryptography Conference - TCC'09*. Springer, pp. 457–473.

Shepler, S., M. Eisler, D. Robinson, B. Callaghan, R. Thurlow, D. Noveck, C. Beame, and N. Appliance (2003). *RFC 3530: NFS version 4 protocol*. Tech. rep. Internet Engineering Task Force.

Shoup, V. (2000). "Practical threshold signatures". In: *Proceedings of the 19th Annual International Conference on the Theory and Applications of Cryptographic Techniques - EUROCRYPT'00*, pp. 207–220.

Shraer, A., C. Cachin, A. Cidon, I. Keidar, Y. Michalevsky, and D. Shaket (2010). "Venus: Verification for untrusted cloud storage". In: *Proceedings of the ACM Cloud Computing Security Workshop - CCSW'10*. ACM, pp. 19–29.

Smart, N. P. and F. Vercauteren (2010). "Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes". In: *Proceedings of the 13th International Conference on Practice and Theory in Public-Key Cryptography - PKC'10*. Vol. 6056, pp. 420–443.

Song, D. X., D. Wagner, and A. Perrig (2000). "Practical techniques for searches on encrypted data". In: *Proceedings of the 21st IEEE Symposium on Security and Privacy - S&P'00*. IEEE, pp. 44–55.

Sparck Jones, K. (1972). "A statistical interpretation of term specificity and its application in retrieval". In: *Journal of Documentation* 28.1, pp. 11–21.

Stefanov, E. and E. Shi (2013a). "Multi-Cloud Oblivious Storage". In: *Proceedings of the 20th ACM Conference on Computer and Communications Security - CCS'13*, pp. 247–258.

Stefanov, E. and E. Shi (2013b). "ObliviStore: High performance oblivious cloud storage". In: *Proceedings of the 34th IEEE Symposium on Security and Privacy - S&P'13*, pp. 253–267.

Stefanov, E., 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 (2013). "Path oram: An extremely simple oblivious ram protocol". In: *Proceedings of the 20th ACM Conference on Computer and Communications Security - CCS'13*. ACM, pp. 299–310.

Stefanov, E., C. Papamanthou, and E. Shi (2014). "Practical Dynamic Searchable Encryption with Small Leakage". In: *Proceedings of the The 21th Annual Network and Distributed System Security Symposium -NDSS'14*.

Swain, M. J. and D. H. Ballard (1991). "Color Indexing". In: *International Journal of Computer Vision* 7.1, pp. 11–32.

Troncoso-Pastoriza, J. R. and F. Perez-Gonzalez (2013). "Secure signal processing in the cloud: enabling technologies for privacy-preserving multimedia cloud processing". In: *IEEE Signal Processing Magazine* 30.2.

Verissimo, P., A. Bessani, and M. Pasin (2012). "The TClouds Architecture: Open and Resilient Cloud-of-clouds Computing". In: *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W'12)*.

Verissimo, P. E. and A. Bessani (2013). "E-biobanking : What Have You Done to My Cell Samples?" In: *IEEE Security & Privacy* 11.6, pp. 62–65.

Wang, C., N. Cao, K. Ren, and W. Lou (2012). "Enabling Secure and Efficient Ranked Keyword Search over Outsourced Cloud Data". In: *IEEE Transactions on Parallel and Distributed Systems* 23.8, pp. 1467–1479.

Wang, J. Z., J. Li, and G. Wiederhold (2001). "SIMPLIcity: Semantics-sensitive Integrated Matching for Picture LIbraries". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23.9, pp. 947–963.

Wang, X. S., K. Nayak, C. Liu, T.-H. H. Chan, E. Shi, E. Stefanov, and Y. Huang (2014). "Oblivious Data Structures". In: *Proceedings of the 21st ACM Conference on Computer and Communications Security - CCS'14*, pp. 215–226.

Weinzaepfel, P., H. Jégou, and P. Pérez (2011). "Reconstructing an image from its local descriptors". In: *Proceedings of the 24th IEEE Conference on Computer Vision and Pattern Recognition - CVPR'11*. ACM, pp. 337–344.

Weng, L., L. Amsaleg, A. Morton, and S. Marchand-maillet (2015). "A Privacy-Preserving Framework for Large-Scale Content-Based Information Retrieval". In: *IEEE Transactions on Information Forensics and Security* 10.1, pp. 152–167.

Wikimedia Foundation (2016). *Wikipedia Database dump*. URL: https://en.wikipedia.org/wiki/Wikipedia:Database{_}download.

Wright, C. P., J. Dave, and E. Zadok (2003). "Cryptographic File Systems Performance: What You Don't Know Can Hurt You". In: *Proceedings of the Second IEEE International Security in Storage Workshop - SISW'03*.

Xia, Z., Y. Zhu, X. Sun, Z. Qin, and K. Ren (2015). "Towards Privacy-preserving Content-based Image Retrieval in Cloud Computing". In: *IEEE Transactions on Cloud Computing* PP.99, pp. 1–11.

Yuan, X., X. Wang, C. Wang, A. Squicciarini, and K. Ren (2014). "Enabling Privacy-preserving Image-centric Social Discovery". In: *Proceedings of the 34th IEEE International Conference on Distributed Computing Systems - ICDCS'14*. IEEE.

Zhang, Y., J. Katz, and C. Papamanthou (2016). "All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption". In: *Proceedings of the 25th USENIX Security Symposium - Security'16*. USENIX Association.

Zheng, P. and J. Huang (2013). "An efficient image homomorphic encryption scheme with small ciphertext expansion". In: *Proceedings of the 21st ACM international Conference on Multimedia - MM '13*.

Zobel, J. and A. Moffat (2006). "Inverted files for text search engines". In: *ACM Computing Surveys (CSUR)* 38.2, p. 6.

Appendix A

A Multimodal SSE Scheme

In this appendix chapter we detail how we implemented and extended a recent SSE scheme from the literature (Cash et al. 2014) to support multimodal querying. This scheme, which we call MSSE, was used in Chapter 5.7 as a baseline comparison for the experimental evaluation of MIE.

An Exact-Match Text Searching Scheme. From the recent literature on SSE schemes (Cash et al. 2014; Hahn and Kerschbaum 2014; Kamara and Papamanthou 2013; Kamara et al. 2012; Naveed et al. 2014; Popa et al. 2014; Stefanov et al. 2014), whose authors have been focusing on single keyword exact-match search on text documents, we found the approach by Cash et al. (Cash et al. 2014) 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 (Katz and Lindell 2007)) and a second key derived from the keyword. To search with a query keyword (in the Random Oracle Model (Canetti et al. 2004)) 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.

Algorithm A.1 MSSE scheme, Create Repository Operation

procedure User(U).createRepository(ID_R, sp_R)
  $rk1_R \leftarrow PRG(sp_R)$
  $rk2_R \leftarrow PRG(sp_R)$
  Cloud.CreateRepository(ID_R)
  RepUsers.ShareKey($\{rk1_R, rk2_R\}$)
  return $rk_R = \{rk1_R, rk2_R\}$

procedure Cloud.CreateRepository(ID_R)
  $Rep[ID_R] \leftarrow$ InitializeRepository()
  $Fvs[ID_R] \leftarrow$ InitializeFeatureVectorsList()

Algorithm A.2 MSSE scheme, Remove Operation

procedure User(U).Remove(ID_R, ID_p)
  Cloud.Remove(ID_R, ID_p)

procedure Cloud.Remove(ID_R, ID_p)
  if $Rep[ID_R][ID_p] != \{\}$ **then**
    $Rep[ID_R][ID_p] \leftarrow \{\}$; $Fvs[ID_R][ID_p] \leftarrow \{\}$
    if IsTrained(ID_R) **then**
      for all $\{Idx[ID_R][ID_{m_i}]\}_{i=0}^{n}$ **do**
        for all $\{Idx[ID_R][ID_{m_i}][l_j]\}_{j=0}^{|Idx[ID_R][ID_{m_i}]|}$ **do**
          if $Idx[ID_R][ID_{m_i}][l_j].ID == ID_p$ **then**
            $Idx[ID_R][ID_{m_i}][l_j] \leftarrow \{\}$

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 (Manning et al. 2009). 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) (Nistér et al. 2006). Similar approaches can be used for indexing audio and video features.

Algorithm A.3 MSSE scheme, Update Operation

procedure User(U).Update(ID_R, ID_p, p, dk_p, $\{rk1_R, rk2_R\}$, $\{ID_{m_i}\}_{i=0}^{n}$)
  $e \leftarrow Enc(dk_p, p)$
  for all $\{ID_{m_i}\}_{i=0}^{n}$ **do**
    $fvs_{m_i}^{p} \leftarrow$ ExtractFeatureVectors(p, ID_{m_i})
    $efvs_{m_i}^{p} \leftarrow ENC(rk1_R, fvs_{m_i}^{p})$
  if !IsTrained(ID_R) **then**
    Cloud.UntrainedUpdate(ID_R, ID_p, e, $\{efvs_{m_i}^{p}\}_{i=0}^{n}$)
  else
    $\{ectrs_{m_i}\}_{i=0}^{n} \leftarrow$ Cloud.GetCtrs($ID_R, \{ID_{m_i}\}_{i=0}^{n}$)
    for all $\{ID_{m_i}\}_{i=0}^{n}$ **do**
      $ctrs_{m_i} \leftarrow DEC(rk1_R, ectrs_{m_i})$
      $hist_{m_i}^{p} \leftarrow$ ClusterizeAndSort($fvs_{m_i}^{p}$)
      $L_{m_i} \leftarrow$ InitializeList($|hist_{m_i}^{p}|$)
      for all $\{fv_j, freq_{fv_j}\}_{j=0}^{|hist_{m_i}^{p}|}$ **do**
        if $ctrs_{m_i}[fv_j] == \{\}$ **then**
          $ctrs_{m_i}[fv_j] \leftarrow 0$
        $k1 \leftarrow PRF(rk2_R, fv_j \| 1)$
        $k2 \leftarrow PRF(rk2_R, fv_j \| 2)$
        $l \leftarrow PRF(k1, ctrs_{m_i}[fv_j])$
        $ctrs_{m_i}[fv_j]++$
        $d \leftarrow ID_p \| ENC(k2, freq_{fv_j})$
        L_{m_i}.Add($\{l, d\}$)
      $ectrs_{m_i} \leftarrow ENC(rk1_R, ctrs_{m_i})$
    Cloud.TrainedUpdate(ID_R, ID_p, e, $\{ID_{m_i}, L_{m_i}, efvs_{m_i}^{p}, ectrs_{m_i}\}_{i=0}^{n}$)

procedure Cloud.GetCtrs($ID_R, \{ID_{m_i}\}_{i=0}^{n}$)
  for all $\{ID_{m_i}\}_{i=0}^{n}$ **do**
    $ectrs_{m_i} \leftarrow Ctrs[ID_R][ID_{m_i}]$
    LockCounterAccess($Ctrs[[ID_R][ID_{m_i}]$)
  return $\{ectrs_{m_i}\}_{i=0}^{n}$

procedure Cloud.UntrainedUpdate(ID_R, ID_p, e, $\{efvs_{m_i}^{p}\}_{i=0}^{n}$)
  $Rep[ID_R][ID_p] \leftarrow e$
  $Fvs[ID_R][ID_p] \leftarrow \{efvs_{m_i}^{p}\}_{i=0}^{n}$

procedure Cloud.TrainedUpdate(ID_R, ID_p, e, $\{ID_{m_i}, L_{m_i}, efvs_{m_i}^{p}, ectrs_{m_i}\}_{i=0}^{n}$)
  for all $\{ID_{m_i}, ectrs_{m_i}\}_{i=0}^{n}$ **do**
    $Ctrs[ID_{m_i}] \leftarrow ectrs_{m_i}$
    UnLockCounterAccess($Ctrs[ID_{m_i}]$)
  Cloud.Remove(ID_R, ID_p)
  $Rep[ID_R][ID_p] \leftarrow e$; $Fvs[ID_R][ID_p] \leftarrow \{efvs_{m_i}^{p}\}_{i=0}^{n}$
  for all $\{ID_{m_i}, L_{m_i}\}_{i=0}^{n}$ **do**
    for all $\{l, d\} \in L_{m_i}$ **do**
      $Idx[ID_R][ID_{m_i}][l] \leftarrow d$

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 (Mourão et al. 2013).

Updates and Removals. One of the main limitations of the approach by Cash et al. (Cash et al. 2014) 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 (Cash et al. 2014) the revocation list can fluctuate in size, if removed keywords are later re-added to their documents however, in our case the updated

Algorithm A.4 MSSE scheme, Train Operation

procedure $\text{User}(U).\text{Train}(ID_R, \{rk1_R, rk2_R\}, ID_{m_i}, ip_{m_i}\}_{i=0}^{n})$
  $efvs \leftarrow \text{Cloud.GetFeatures}(ID_R)$
  $fvs \leftarrow DEC(rk1_R, efvs)$
  for all $\{ID_{m_i}, ip_{m_i}\}_{i=0}^{n}$ **do**
    $D[ID_{m_i}] \leftarrow \text{InitializeIndex}(ip_{m_i})$
    if $\text{DenseMediaType}(ID_{m_i})$ **then**
      $CB_R^{m_i} \leftarrow \text{TrainIndex}(D[ID_{m_i}], ip_{m_i}, fvs_{m_i})$
      $\text{RepUsers.ShareCodebook}(CB_R)$
    $\text{IndexData}(D[ID_{m_i}], fvs_{m_i})$
  $\text{Cloud.StoreIndex}(ID_R, \{ID_{m_i}, D[ID_{m_i}]\}_{i=0}^{n})$

procedure $\text{Cloud.GetFeatures}(ID_R)$
  return $Fvs[ID_R]$

procedure $\text{Cloud.StoreIndex}(ID_R, \{ID_{m_i}, D[ID_{m_i}]\}_{i=0}^{n})$
  for all $\{D[ID_{m_i}]\}_{i=0}^{n}$ **do**
    $Idx[ID_R][ID_{m_i}] \leftarrow D[ID_{m_i}]$

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 (Cash et al. 2015).

Multiple Clients and Client Storage. The methodology proposed by Cash et al. (Cash et al. 2014) 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

Algorithm A.5 MSSE scheme, Search Operation

procedure User(U).Search(ID_R, q, $\{rk1_R, rk2_R\}$, $\{ID_{m_i}\}_{i=0}^{n}$, k)
  for all $\{ID_{m_i}\}_{i=0}^{n}$ **do**
    $fvs_{m_i}^{q} \leftarrow$ ExtractFeatureVectors(q, ID_{m_i})
  if !IsTrained(ID_R) **then**
    $\{efvs, Rep\} \leftarrow$ Cloud.GetFeaturesAndObjects(ID_R)
    $fvs \leftarrow DEC(rk1_R, efvs)$
    for all $\{ID_{m_i}\}_{i=0}^{n}$ **do**
      $Res_{m_i} \leftarrow$ LinearRankedSearch($fvs_{m_i}^{q}$, fvs_{m_i})
    $Res \leftarrow$ FusionRank($\{ID_{m_i}, Res_{m_i}\}_{i=0}^{n}$, k)
    return $\{ID_{p_i}, Rep[ID_{p_i}], Res[ID_{p_i}]\}_{i=0}^{k}$
  else
    $\{ectrs_{m_i}\}_{i=0}^{n} \leftarrow$ Cloud.GetCounters(ID_R, $\{ID_{m_i}\}_{i=0}^{n}$)
    for all $\{ID_{m_i}\}_{i=0}^{n}$ **do**
      $ctrs_{m_i} \leftarrow DEC(rk1_R, ectrs_{m_i})$
      $hist_{m_i}^{q} \leftarrow$ ClusterizeAndSort($CB_R^{m_i}$, $fvs_{m_i}^{q}$)
      $L_{m_i} \leftarrow$ InitializeList($|hist_{m_i}^{q}|$)
      for all $\{fv, freq_{fv}\} \in hist_{m_i}^{q}$ **do**
        $ll \leftarrow$ InitializeList($ctrs_{m_i}[fv] + 1$)
        $k1 \leftarrow PRF(rk2_R, fv||1)$
        $k2 \leftarrow PRF(rk2_R, fv||2)$
        for $ctr \leftarrow 0 \ldots ctrs_{m_i}[fv]$ **do**
          $l \leftarrow PRF(k1, ctr)$
          ll.Add(l)
        L_{m_i}.Add($\{ll, k2, freq_{fv}\}$)
    $\{ID_{p_i}, p_i, score_{p_i}^{q}\}_{i=0}^{k} \leftarrow$ Cloud.Search(ID_R, $\{L_{m_i}\}_{i=0}^{n}$, k)
    return $\{ID_{p_i}, p_i, score_{p_i}^{q}\}_{i=0}^{k}$

procedure Cloud.GetFeaturesAndObjects(ID_R)
  return $\{Fvs[ID_R], Rep[ID_R]\}$

procedure Cloud.Search(ID_R, $\{ID_{m_i}, L_{m_i}\}_{i=0}^{n}$, k)
  for all $\{ID_{m_i}\}_{i=0}^{n}$ **do**
    $Res_{m_i} \leftarrow$ InitializeList(k)
    for all $\{ll, k2, freq_q\} \in L_{m_i}$ **do**
      $tfs \leftarrow$ InitializeList($|ll|$)
      for all $l \in ll$ **do**
        if $Idx[ID_R][ID_{m_i}][l]! = \{\}$ **then**
          $\{ID_p, efreq\} \leftarrow Idx[ID_R][ID_{m_i}][l]$
          $freq \leftarrow DEC(k2, efreq)$
          tfs.Add($\{ID_p, freq\}$)
      for all $\{ID_p, freq\} \in tfs$ **do**
        $idf \leftarrow log(|Rep[ID_R]|/|tfs|)$
        $tfidf \leftarrow freq_q \times freq \times idf$
        if $Res_{m_i}[ID_p] == \{\}$ **then**
          $Res_{m_i}[ID_p] \leftarrow tfidf$
        else
          $Res_{m_i}[ID_p] \leftarrow Res[ID_p] + tfidf$
    $Res_{m_i} \leftarrow$ Sort(Res_{m_i})
  $Res \leftarrow$ FusionRank($\{ID_{m_i}, Res_{m_i}\}_{i=0}^{n}$, k)
  return $\{ID_{p_i}, Rep[ID_R][ID_{p_i}], Res[ID_{p_i}]\}_{i=0}^{k}$

information leakage and practical performance require client-storage (Cash et al. 2014; Hahn and Kerschbaum 2014). 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).

Algorithms A.1 through A.5 present a formalization of scheme MSSE. In the Algorithms, 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 a Pseudo-Random Number Generator.

A.1 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 Section 5.7 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 (Paillier 1999), 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 (Baldimtsi and Ohrimenko 2015) 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

Algorithm A.6 Scheme Hom-MSSE.

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** $fv_j \in \text{Padding}(fvs_{m_i}^{p})$ **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

4: **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: $Res_{m_i} \leftarrow \text{Sort}(Res_{m_i})$
29: $Res \leftarrow \text{FusionRank}(\{ID_{m_i}, Res_{m_i}\}_{i=0}^{n}, k)$
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** $\{\{ID_{p_j}$, $Rep[ID_R][ID_{p_j}]$, $Res_{m_i}[ID_{p_j}]\}_{j=0}^{|Rep[ID_R]|}\}_{i=0}^{n}$

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 (Cash et al. 2015), padding by 1.6x of the request size would be enough to stop keyword-retrieval attacks).

Algorithm A.6 formalizes scheme Hom-MSSE, which is presented as an iteration over scheme MSSE from Algorithms A.1 through A.5. Ellipsis represent skipped lines from the previous Algorithms, and each line after an ellipsis represents a re-written line with the same line number. Lines marked with *Removed Line* are lines from the previous Algorithms that should be removed in Hom-MSSE.