THEORY AND APPLICATION OF ENCRYPTED SEQUENTIAL DATA PROCESSING: SEARCH AND COMPUTATION

by

Hoi Ting Poon

M.A.Sc. University of Ottawa, Ottawa (ON), Canada, 2008

B.A.Sc. University of Ottawa, Ottawa (ON), Canada, 2005

A dissertation

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

in the program of

Computer Science

Toronto, Ontario, Canada, 2018

Author's Declaration

I hereby declare that I am the sole author of this dissertation. This is a true copy of the dissertation, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this dissertation to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this dissertation by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my dissertation may be made electronically available to the public for the purpose of scholarly research only.

Abstract

Theory and Application of Encrypted Sequential Data Processing: Search and Computation

Hoi Ting Poon

Doctor of Philosophy, Computer Science

Ryerson University, 2018

Cloud Computing has seen a dramatic rise in adoption in the past decade amid security and privacy concerns. One area of consensus is that encryption is necessary, as anonymization techniques have been shown to be unreliable. However, the processing of encrypted data has proven to be difficult. Briefly, the goal is to maintain security over remotely stored and accessed data while achieving reasonable storage cost and performance. Search is the most basic and central functionality of a privacy-protected cloud storage system actively being investigated. Recent works have looked at enabling more specialized search functions. In this thesis, we explore the problem of searching and processing of sequential data. We propose three solutions targeting textual data, with emphasis respectively on security, storage cost and performance. Our first solution achieves a high level of security with reduced communication, storage and computational cost by exploiting properties of natural languages. Our second solution achieves a minimal storage cost by taking advantage of the space efficiency of Bloom filters. Both proposals were also first to enable non-keyword search in phrases. Using a subsequence-based solution, our final phrase search scheme is currently the fastest phrase search protocol in literature. We also show how sequential data search schemes can be extended to include auditing with minimal additional cost. The solution is capable of achieving proof of retrievability with unbounded number of audits. A sample application which enables

searching and computing over target values of encrypted XML files is also demonstrated. In terms of media, we describe an encrypted cloud media storage solution that simultaneously protects user privacy and enables copyright verification, and is the first to achieve security against dishonest participants. We also describe a framework where practical scalable privacy-protected copyright detection can be performed. Finally, an application of sequence querying over generic data in the form of an Anti-Virus over encrypted cloud storage is demonstrated. A private scanning solution and a public Anti-Virus as a service solution are described, noting that the technique can be conceptualized as a generic pattern matching solution on encrypted data. We also include some directions on future work and unexplored applications.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

There has been significant interest in encrypted data processing in the research community over the past decade. Arguably, it all started with VMWare: customizable virtual machines with hard drive, RAM, CPU, etc, all adjustable as needed, easily and rapidly deployable at a much lower cost than physical machines. As a result, small companies can quickly set up their computing resources while established companies can reduce the financial burden of setting up and maintaining physical data centers. Its scalability and flexibility also allows clients to easily adjust the resources according to their computing needs and quickly adapt to unexpected increases in traffic, such as a popular app on launch day or defense against denial of service attacks. This is what fueled the commercialization of cloud computing technology. Despite the numerous advantages of out-sourcing data storage and computations, cloud computing and storage systems also raise security and privacy concerns. Many organizations have become alarmed by increasingly frequent data breaches and many, such as health services, require handling private and classified information. The security and privacy concerns that come with outsourcing of computing resources, combined with a timely breakthrough in homomorphic encryption, is what led many researchers to consider encrypted data processing solutions for cloud services.

1.1 Motivation

Despite the proliferation of cloud technologies today, many commercial cloud service providers do not offer data encryption. Those that do encrypt the data using a private key that they control, allowing them to read any data they desire and offering no privacy to their clients. The promise of reliable, scalable and cost-effective service is therefore

hampered by the lack of data security and privacy. The latter can be particularly important in industries required to handle classified and confidential information such as health care and financial services. In response, regulators establish standards requiring anonymization of private data, through removal of personally identifiable information such as names and identification numbers from client data. These techniques have been in use by many companies such as Facebook and Netflix. Yet, many studies have shown that they do not provide reliable protection. For instance, a study by MIT [1] showed that knowing a person's location four times in a year is enough to uniquely identify 95% of users in a set of 1.5 million cellphone usage records. In genomics, short subsequences of chromosomes were found to be enough to identify individuals with high probability [30]. The anonymized Netflix Prize dataset was famously deanonymized using publicly available information [49]. In all cases, the conclusion seems to be that reliable anonymization could well be infeasible since information about individuals is so widely available and easily accessible largely due to the Internet.

As an alternative to anonymization, encryption has well defined security properties that have endured under the scrutiny of academics and security professionals. Rather than maintaining seemingly non-identifying information in plain, all data is encrypted with mechanisms in place to perform the required functionalities. Much of the difficulty in securing distributed computation and data storage is due to the fact that strong encryption tends to require significant computations, which in turn reduces the throughput of the system. Therefore, the balance between cost, performance and security is central to research in encrypted data processing. An insurance company that needs to secure client data may be willing to sacrifice access speed while a stock exchange relying on fast order processing would value response time above all. Another challenge rests in enabling the functionalities required for remote access of encrypted data. However, many functions available in storage systems and databases, such as search, deduplication and auditing, do not translate intuitively to encrypted data. A trivial solution would have the data owner download and decrypt the entire data set before performing the required function on the plaintext. However, this would be impractically expensive on large data stores with petabytes of data. As a result, researchers have been actively investigating solutions for processing encrypted data on cloud storage. Particularly, search has been identified as one of the most important features needed in encrypted storage systems.

1.1.1 Privacy-aware keyword search

Considered to be the core function of a privacy protected cloud storage system, conjunctive keyword search over encrypted data has been the focus of many research works

in the past decade. More recently, researchers have begun exploring more specialized functions such as fuzzy search [70, 77], ranking [18, 24, 33] and auditing [8, 65, 72]. Along the same line, our work aimed to extend basic keyword search functionality to include phrases.

Since applications have different requirements, solutions are needed to satisfy different levels of security, cost and performance. For instance, medical and financial services may have sensitive data that require a high level of security. Despite many advantages in outsourcing data management, current unencrypted cloud storage services generally cannot meet the industry's security needs. Therefore, a solution which enables secure and private storage of sensitive documents while enabling search is of great value.

The emergence of the Internet of Things also introduced resource limited devices that require lightweight solutions. For example, a network of devices in the field may need to access a cloud database. However, due to geographical distance and power limitations, a small edge device with limited amount of storage is instead placed close to the end nodes to facilitate queries. Since the edge device is in the field, its storage needs to be secured and the resource for supporting queries must have a low storage requirement.

Time sensitive applications such as those related to Big Data often require fast processing to match the rate at which data are being generated. One of the emerging trends in security is to perform large scale monitoring and intrusion/anomaly detection. Security companies offering such services may require scanning security logs on their clients' machines for network traffic, https requests, file transfers, etc. These information could potentially be sensitive. They are also time critical. A fast detection of a newly released malware can significantly reduce its spread and the damage it causes. A mechanism for querying encrypted security logs with fast response time could present a practical solution.

1.1.2 Cloud auditing: data retrievability

Another area of interest is that of cloud auditing. In particular, storing data in servers that the data owner does not control presents a dilemma where there's no obvious way to ensure that the data is faithfully being stored. Furthermore, sensitive data needs to be encrypted to ensure security and privacy. If the data is not frequently accessed, how can one ensure that it remains uncorrupted and available without downloading and decrypting the data set? Researchers describe the problem as proof of possession or retrievability of untrusted cloud stored data. A practical solution to the problem is especially beneficial to archives and backups that are rarely accessed but can be extremely important when required. Imagine the scenario where a company's financial dealings and

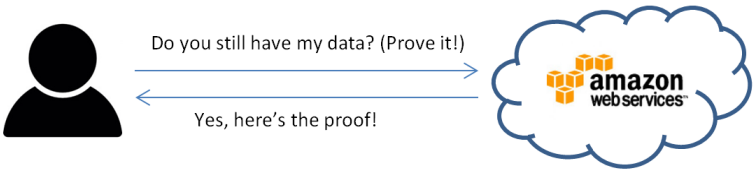

Figure 1.1: Cloud auditing: proof of data retrievability

statements are stored in the cloud and a legal dispute arises some years later when proofs of contractual obligations are required. Such documents are naturally confidential and their availability and authenticity are equally important.

1.1.3 Privacy-preserving computations

While search is considered a basic functionality, the ability to compute over encrypted data provides many interesting possibilities. Although generic computation over encrypted data has so far remained out of reach, application specific solutions have shown promise.

In this thesis, we consider three problems where computing over encrypted sequences is required.

XML Documents

XML is a data format that structures documents for human and machine readability. It is widely used across the Internet, e.g. HTML, JSON. The textual documents consist of attributes, tags and contents. Often, there are fields where contents are numeric, such as prices, time and weight, where meaningful computations can be performed. When documents are in plaintext, identifying contents of specific tags and computing over them is straight forward. However, computing over encrypted XML documents is much less obvious. Locations and ordering of tags are not necessarily consistent and may even be missing across different documents of the same format. Size of contents can also vary. A solution that addresses these challenges could provide a valuable tool to process encrypted online forms and compute aggregated results while protecting privacy of individual users.

Digital rights management

Since Napster introduced its pioneering peer-to-peer file sharing service, copyright advocates had been in a constant struggle to enforce copyright laws over the Internet. While

peer-to-peer file sharing remains popular today, cloud storage services have become the tool of choice for many illegal distributors. Unlike peer-to-peer, there is no need to maintain a server to seed a file. Instead, the distributor simply uploads the file to the cloud and provides the link to the public. There is no cost to the distributor and no maintenance required.

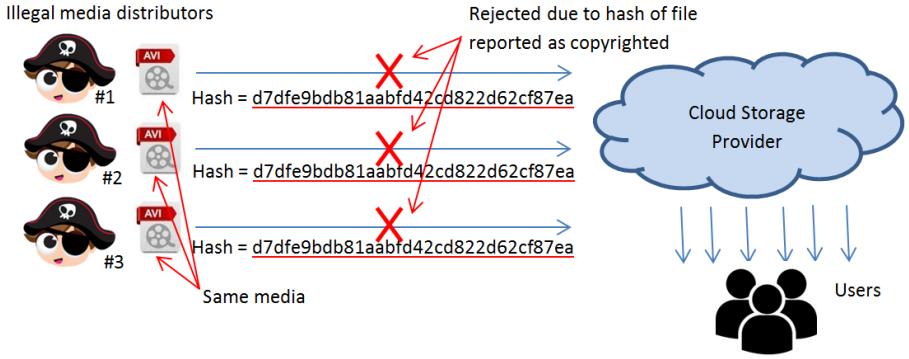

Figure 1.2: Hash based copyright detection for cloud storage services

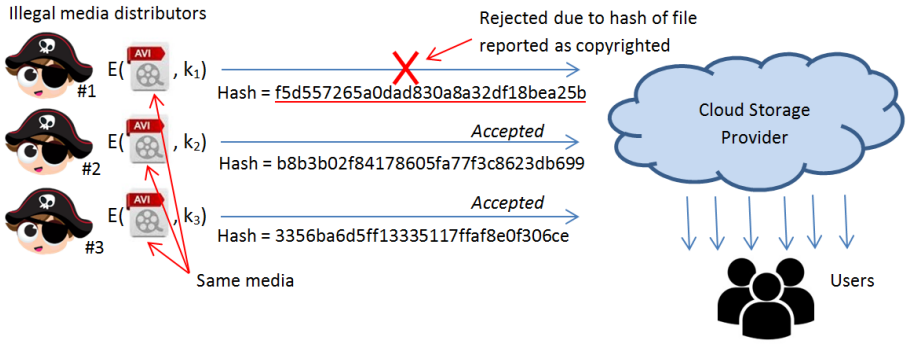

Figure 1.3: Hash based copyright detection for encrypted cloud storage services (e.g. Mega)

Companies such as Amazon EC2, Microsoft Azure and Mega are among cloud service providers who have dedicated significant resources to comply with copyright laws in order to continue operations. However, efforts to prevent copyrighted media from being placed on cloud servers have had limited success. Furthermore, the need for user security and privacy has led to cloud storage services offering encryption where the data owner controls the private key. The latter renders traditional hash-based automatic detection impossible. Figure 1.2 illustrates how a media previously reported by a copy

claimant as copyrighted can be automatically identified using the file's hash signature should other user uploads the same media. Figure 1.3 shows the equivalent scenario in an encrypted cloud service where the encrypted media have different hashes due to encryption with different keys, disabling traditional hash based detection. Without an automatic detection system, cloud services can rely only on manual reports to remove copyrighted media from their servers, a slow and ineffective process that exposes them to legal threats. Without encryption, they cannot alleviate security and privacy concerns. There is therefore a need for a solution that can effectively address both user privacy and digital rights management on cloud services.

Malware detection

Anti-virus companies have come under scrutiny in recent years amid accusation of cyber warfare between nations. As recent as in September 2017, US ordered removal of Kaspersky's products from government computers citing national security concerns on fear that Kaspersky is used by the Russian government to further national interest [48]. These concerns are in no small part due to the significant control required by an anti-virus software to perform its functions and its unrestricted access to any data stored on a host machine. This level of access allows an anti-virus company to extract data from any target machine with its software installed should it desires to. Anti-virus software operates similarly on cloud servers. They run locally and are given administrator-like access on machines. On a storage system, an anti-virus software's main function is to identify malware or, more specifically, the signatures and code sequences representing malware. Knowledge of the data itself is not technically required, unless it matches those representing malware. How would one then allow the anti-virus to detect malware without giving it access to the data in plain? This question may be answered using cryptographic techniques in a solution that provides both user privacy and control over the amount of information that an anti-virus has access to.

1.2 Contributions

The contribution of this thesis revolves around techniques for searching and computing over encrypted sequences and their applications to secure and privacy-protected cloud services.

To establish the basic techniques, three search schemes for textual sequences are described, with respective emphasis on:

1. Security: This construction, denoted our $\text{scheme}_{\text{sec}}$, is the first practical phrase search scheme with provable security in the literature. It is also the first to enable queries for non-indexed words with basic ranking capability. More generally, the scheme demonstrates techniques to improve storage and communication cost in an index based search scheme when the data under search, be it textual or otherwise, exhibits a highly non-uniform distribution. It is also demonstrated that data symmetrically encrypted in counter mode can be searched directly without indexing.

2. Storage: This construction, denoted our $\text{scheme}_{\text{sto}}$, is currently the phrase search solution with the lowest storage cost in the literature. Non-indexed words are also searchable with basic ranking capability. More generally, the scheme demonstrates techniques to trade computational cost and processing speed for lower storage by combining data and location, be it textual or otherwise, in a single searchable data structure.

3. Speed: This construction, denoted our $\text{scheme}_{\text{spd}}$, is currently the fastest phrase search scheme in the literature. The solution introduces a new way of searching data sequences, shifting away from traditional location based search. The technique allows the search to be completed in a single round of communication using only efficient hash computations.

Our solutions can also adapt to the fog computing architecture, where edge devices are placed closer to users to provide localized service as opposed to a pure cloud architecture where only centralized servers are used. This reduces latency to clients and reduces bandwidth to backbone cloud data center, improving quality of service. Adapting our phrase search protocols to the fog computing architecture consists of placing and replicating the indexes or Bloom filters at edge devices while storing the encrypted data sets at the cloud storage.

Noting the connection with cloud data auditing and proof of retrievability schemes, we also show that resources required for a location based search scheme can be reused for proof of retrievability with minimal additional cost. To this end, we describe a private auditing scheme and a public auditing scheme where a third party auditor is used. The solutions maintain data and search privacy while enabling conjunctive keyword search, phrase search and proof of retrievability with unbounded number of audits.

Using XML as example, we show how phrase search can be used in a document processing solution that enables SQL-like queries, such as SUM and PRODUCT, while still being relatively efficient. The solution is applicable to partially and fully encrypted XML

documents. Our solution is also the first to consider fully encrypted XML documents, where full protection of content and structure of the XML is available.

In terms of media, we describe a solution which enables both data privacy and copyright verification on encrypted media uploaded to cloud servers, resolving the apparent incompatibility of the two features. By providing a technique to perform automatic detection, this gives a way forward from the industry's status quo where manual reports are required on encrypted cloud storage such as Mega. It is the first solution to consider the digital rights management issue for cloud storage in practise, where different parties are considered malicious due to each having incentives to deviate from the protocol for individual benefits. We also describe a framework where efficient copyright detection can be performed by combining with an encrypted media search scheme.

Our final contribution involves an application of sequential encrypted binary data processing for cloud based Anti-virus software. Instead of allowing an anti-virus access to plaintext, all data are encrypted. The approach provides data owner control over the anti-virus's access to the data set as opposed to traditional anti-virus with unrestricted access. More specifically, our solutions allow an anti-virus to scan for malware but learn nothing more on the data. The cloud service provider assists in performing the scan, also without knowledge of the data. The private scenario allows a single user to efficiently scan his own encrypted data set, without revealing data to the cloud. We also showed an interesting solution in the form of an anti-virus as a service where an anti-virus server scans user's encrypted data stored on the cloud. The solution has the property that the anti-virus server does not require knowledge of the data in plain and the data owner does not require knowledge of the malware database to perform malware scanning. The cloud service provider does not acquire knowledge of the malware database or the data being scanned. Furthermore, only a simple encrypted scanner is installed on the cloud server, which does not require frequent software updates. This protects against reverse-engineering of anti-virus software, prevents malware writers from gaining access to malware databases, provides data control to users while maintaining data privacy and security on the cloud.

1.3 Thesis organization

In this thesis, we explore the problem of searching and processing of sequential data. In the form of text, this represents phrases. In media, this could be pixels or spectral information. We begin our formal discussion in chapter 2 by proposing three solutions targeting textual data, with an emphasis respectively on security, storage cost and per-

formance. Business and organizations have varying needs in security, privacy and costs. Agencies handling classified documents may opt for a highly secure solution in spite of slower response time and higher storage requirement. A network of small devices with limited storage may require the smallest storage requirement possible. An intrusion detector scanning encrypted logs may require fast response time in order to process the logs as fast as they are being generated. Each of these scenarios would require a different solution among the ones proposed. To help understand our solutions, section 2.2 provides background on the tools we will use throughout this thesis. Cloud auditing is another area of interest. It involves mechanisms for a user to verify the integrity and availability of one's outsourced data when one no longer has access to it. In section 2.7, we show that, under certain condition, a solution for sequential data search can also be used as a cloud auditing solution with minimal additional cost. In other words, an encrypted sequential data search scheme would not only allow a user to search but also ensure that the cloud had not tempered with or lost any user data.

There are also scenarios where one might want to compute over certain portions of a document. For example, a finance department may want to add up the costs of items related to marketing in encrypted invoices. In section 2.8, we describe how one can search and compute over structured text such as an encrypted XML file to perform SQL-like queries.

Media, due to the large bandwidth and storage requirement, is commonly placed on cloud storage services. While there are efforts towards enabling search for encrypted media, our focus is on finding a solution to the status quo of digital rights management in cloud services, where the need for user privacy runs counter to the need for respecting copyright law. Namely, to protect user privacy, cloud services offer client-side data encryption. Without access to user data in plain, they can no longer identify copyrighted material. In chapter 3, we examine and describe an encrypted cloud media storage solution that protects user privacy while enabling copyright verification by computing over encrypted pixel or coefficient sequences. Our solution considers the malicious setting, which is particularly relevant in practise where there are incentives for users to upload copyrighted media onto the cloud.

Finally, we show an application of sequential search over generic data in chapter 4 in the form of an Anti-Virus over encrypted cloud storage. Recent world events and frequent network breaches, some of which claimed to be state sponsored, have brought attention to Anti-Virus companies. Anti-Virus software requires administrative rights over all aspects of computers to perform its functionalities. However, questions arose over time on whether these companies can be trusted to have such complete control

over our data. Furthermore, traditional Anti-Virus software stores the virus database and performs malware scanning locally, which allows malicious parties to adopt and test their malware accordingly. To address these issues, we proposed private and public remote malware scanning solutions for encrypted cloud storage that not only enable user privacy but also protect the content of the virus database from malware writers. To conclude, some directions on future work and unexplored applications will be presented.

The following is a list of our publications and the section(s) associated with it:

- Section 2.3, 3.1 and 3.2: H. Poon and A. Miri. Privacy-aware search and computation over encrypted data stores. In S. Srinivasan, editor, *Guide to Big Data Applications*, chapter 11, pages 273–293. Springers International, 2018
- Section 2.4: H. Poon and A. Miri. An efficient conjunctive keyword and phrase search scheme for encrypted cloud storage systems. In *IEEE International Conference on Cloud Computing*, pages 508–515, 2015
- Section 2.5: H. Poon and A. Miri. A low storage phrase search scheme based on bloom filters for encrypted cloud services. In *IEEE International Conference on Cyber Security and Cloud Computing*, pages 253–259, 2015
- Section 2.6: H. Poon and A. Miri. Fast phrase search for encrypted cloud storage. *IEEE Transactions on Cloud Computing*, DOI: 10.1109/TCC.2017.2709316, to appear
- Section 2.7: H. Poon and A. Miri. A combined solution for conjunctive keyword search, phrase search and auditing for encrypted cloud storage. In *IEEE Conference on Advanced and Trusted Computing*, pages 938–941, 2016
- Section 2.8: H. Poon and A. Miri. Computation and search over encrypted XML documents. In *IEEE International Congress on Big Data*, pages 631–634, 2015
- Section 4.1: H. Poon and A. Miri. Scanning for viruses on encrypted cloud storage. In *IEEE Conference on Cloud and Big Data Computing*, pages 954–959, 2016
- Section 3.3.3: S. Ghaffaripour, F. Younis, H. Poon, and A. Miri. An analysis of the security of compressed sensing using an artificial neural network. In *Privacy, Security and Trust*, pages 1–3, 2017

As the motivations, contributions and organization of this thesis have been stated, next chapter will begin the formal discussion with sequential search and computations with textual data.

Chapter 2

Text

Many sensitive and confidential information are stored in texts. Documents containing medical records, financial spreadsheets, business transactions, credit card records and customer information are among the most cited that require privacy protections. Search, being one of the central needs of text processing systems, is of particular importance.

Investigation into privacy-protected keyword search began in the early 2000s, when Boneh *et al.* [13] proposed one of the earliest works on keyword searching. Their scheme uses public key encryption to allow keywords to be searchable without revealing data content. Waters *et al.* [74] investigated the problem for searching over encrypted audit logs. Many of the early works focused on single keyword searches. Recently, researchers have proposed solutions on conjunctive keyword search, which involves multiple keywords [20, 41]. Other interesting problems, such as the ranking of search results [18, 24, 33] and searching with keywords that might contain errors [70, 77] termed fuzzy keyword search, have also been considered.

The ability to search for phrases was also only recently considered [69, 79]. Phrases are sequential textual data and the most common form encountered in practise. The various encryption schemes and techniques presented in this chapter form the basis of our work on search and computation over encrypted sequences. While English is assumed in our discussions, the techniques can be extended to other natural languages.

In this chapter, we will describe three approaches to implementing phrase search over encrypted documents, with respective focus on security, storage cost and performance. We then describe a search and audit scheme that reuses the resources for enabling phrase search, effectively granting audit capability for free. Then, an application in encrypted XML processing will be demonstrated in section 2.8, which, in addition to search, also allows computing over field values.

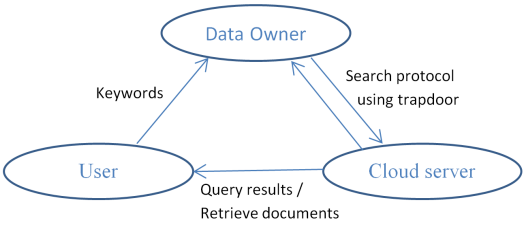

Figure 2.1: Communication model for keyword search over encrypted data

2.1 Model for keyword search over encrypted data

The communication model for a keyword search protocol generally involves up to three parties: The data owner, the cloud server and the user. In a private cloud, the user is simply the data owner. For most of our discussions, we will be considering the public cloud scenario involving three different parties. A typical protocol is illustrated in Figure 2.1. The user begins by sending a search request containing the queried keywords to the data owner. To prevent the cloud from learning the keywords, the data owner computes and sends a trapdoor to the cloud to initiate a protocol to search for the requested keywords in the corpus. Finally, the cloud responds to the user with the search results and, if required, indexes to the requested documents.

Our framework differs from some of the earlier works [13, 74], where keywords generally consist of meta-data rather than content of the files to be retrieved and where a trusted key escrow authority is used due to the use of identity-based encryption. When compared to recent works, our setup is equivalent to that of [75,79], where an organization wishes to outsource computing resources to a cloud storage provider and enable search for its employees, and similar to [24,71], where the aim is to return properly ranked files. Most other recent works related to search over encrypted data have considered similar models such as [69], where the client acts as both data owner and user.

The cloud server usually wields significant computational power compared to the data owner and users. Therefore, it is desirable that the computational and storage cost be asymmetrically placed on the cloud.

Note that, depending on the application, the encrypted documents may or may not require retrieval once the query is resolved. Should retrieval be required, further privacy issues may arise. These issues are considered in oblivious storage [28] and private information retrieval schemes [17]. Our discussions will mainly restrict to the protocol leading to the query resolution. Direct retrieval is assumed where appropriate to better

compare against existing solutions for phrase search.

2.1.1 Security

In terms of security, we assume a semi-honest cloud server, which is interested in learning about stored data but will follow our keyword search protocol as described and will not modify or misrepresent any data in order to gain an advantage. Two of the main security issues regarding keyword searches are the privacy of the document sets and the privacy of the queried keywords. Briefly, a secure keyword search protocol should prevent the cloud server from obtaining non-negligible amount of information on the stored documents or the keywords in the query requests. The latter is a very difficult issue to address if we consider an adversary with some prior knowledge about the documents, such as the language and the content (ex: business, technical specifications), or the pattern of the search requests. Consider a scenario where the cloud operator with statistical knowledge of user data were able to identify the documents where the keywords 'John', 'Smith' and 'Cancer' reside. It would have learned with some level of confidence that John Smith has cancer. Furthermore, should the user retrieves the queried documents at some point, the association of subsets of documents to search patterns is inevitable. Although private information retrieval protocols can be used to lessen its effect, they are costly and can still leak information over time. As with all existing works, we will mostly restrict our security discussions of search privacy on the scheme prior to document retrieval. Note that, in our target application, users are employees of the data owner's organization and are authorized to search for any documents in the data set. Should an application requires that users be restricted from accessing certain files, an access control system such as [61] would be required to verify the matched results and returned only those which the user has the required credential to access. In summary, our protocols are designed to be secure against semi-honest Cloud services with statistical knowledge, such as distribution, of the stored data, protecting user content from the Cloud service provider. Our first protocol is also secure against semi-honest Cloud services with statistical knowledge of incoming queries, while our other protocol can be modified to be secure against the stronger adversary with a communication cost.

2.2 Background

Over the past decade, a few tools have been central to the development of encrypted search schemes. In this section, we present some of the tools that we will use throughout our discussions. Other related techniques can also be found in our survey [57]. Research

work related to our proposed solutions will be presented in the relevant chapters.

2.2.1 Symmetric-key Encryption

Symmetric-key cryptographic algorithms are widely used for securing data. They consists of encryption and decryption algorithms using the same secret key:

$$C = E_K(P) \tag{2.1}$$

$$P = D_K(C) \tag{2.2}$$

where P is the plaintext, C is the ciphertext, $E_K()$ and $D_K()$ are the encryption and decryption using the secret key K. The most popular symmetric key encryption in practise is the block cipher AES. A block cipher is a symmetric-key algorithm that uses a fixed block size. That is, P and C have fixed lengths, e.g. 128 bits. A secure symmetric encryption has the following properties:

- Without the secret key and prior knowledge about the plaintext, it is infeasible to learn the plaintext that generated a ciphertext other than through brute force
- Any two plaintexts, even with minor differences, yield uncorrelated ciphetexts
- A ciphetext should always decrypt to the correct plaintext under the secret key

Modes of Operation

To encrypt messages of different length using a block cipher, they must first be partitioned into cipher blocks. Since the same message will encrypt to the same ciphertext under the same secret key, encrypting each block independently is generally insecure, as blocks representing identical plaintext can be easily identified. Modes of encryption address this issue by allowing a block cipher to securely encrypt long messages. Two of the most commonly used modes of encryption are Cipher Block Chaining (CBC) and Counter (CTR) mode.

CBC mode, illustrated in figure 2.2, creates a chain by making each block's encryption dependent on the previous block, through an XOR of the plaintext with the previous ciphertext block. Effectively, the encryption of a plaintext block is dependent on all plaintexts processed up to that block. To ensure that the first block remains unique, an initialization vector (IV) is used for the first block. Note that the initialization vector should not be reused under the same secret key.

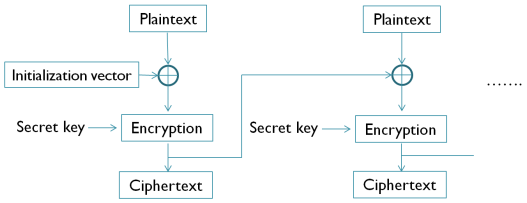

Figure 2.2: Cipher Block Chaining (CBC) mode encryption

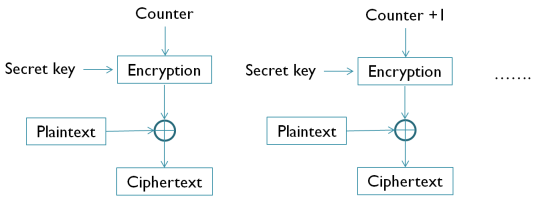

Figure 2.3: Counter (CTR) mode encryption

Counter mode, illustrated in figure 2.3, instead uses a counter to ensure uniqueness of ciphertext, by making it dependent on the position of the block in the message. The ciphertext is simply the XOR of the encryption of the counter with the plaintext. One significant advantage of counter mode over CBC mode is that it is parallelizable.

2.2.2 Cryptographic Hashing Functions

A cryptographic hash function, $H()$, is a hash function with properties that are suitable for use in cryptography. In addition to mapping a message of variable length to a hash value of fixed size as in non-cryptographic hash functions, it must also have the following properties:

- Infeasible to determine the message that generated a hash value other than through brute force
- Infeasible to identify two messages that generate the same hash value
- Any two messages, even with minor differences, yield uncorrelated hash values

Cryptographic hash functions share many similar properties to an encryption algorithm, except the output is uninvertible, fixed size, smaller than the message and collision is possible. Common cryptographic hash functions include MD5, SHA-256 and BLAKE.

2.2.3 Keyed Hashing Functions

Message authentication codes (MAC), also called keyed hash functions, are cryptographic hash functions often used as a mean to ensure integrity and authenticity/origin of files. With keyed hash functions, $H_K()$, a secret key is used to generate the hash to ensure only those possessing the secret key can generate and verify the hash value. HMAC is a keyed hash function specified as follows:

$$HMAC_K(M) = H(K_1|H(K_2|M)) \tag{2.3}$$

where $K = (K_1|K_2)$ is the secret key, M is the message to be hashed and $H()$ is a cryptographic hash function.

2.2.4 Encrypted Indexes

Indexing has been one of the most efficient approaches to search over data. The technique can also be extended to encrypted data.

An index works by first parsing a data set for keywords and then generating a table that maps the keywords to the data. Consider a document set with three books with the following keywords:

Book A	'Horror','Fiction'
Book B	'World War','Biography'
Book C	'World War', 'Pandemic', 'Fiction'

Parsing the document set would result in the following index :

'Horror'	A
'Fiction'	A,C
'World War'	B,C
'Biography'	B
'Pandemic'	C

Extending the approach to encrypted data consists simply of hashing and encrypting the keys and entries in a manner that is consistent with the index structure [27]. The data set itself is symmetrically encrypted using a separate secret key.

E_K('Horror')	$E_K(A)$
E_K('Fiction')	$E_K(A, C)$
E_K('World War')	$E_K(B, C)$
E_K('Biography')	$E_K(B)$
E_K('Pandemic')	$E_K(C)$

Suppose a user wishes to upload a document collection, $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$. It is first parsed for a list of keywords, $\{kw_1, kw_2, \ldots, kw_{n'}\}$, which may include document content or meta-data such as date and department. An index is generated mapping keywords to documents such that $I(kw_j) = (d_1, d_2, \ldots, d_n)$, where $d_i = 1$ if kw_j is a keyword for the i^{th}document and $d_i = 0$ otherwise. The index is then encrypted and uploaded to the cloud server:

$$I(H_K(kw_j)) = E_K(d_1, d_2, \ldots, d_n), \tag{2.4}$$

where $H_K()$ is a keyed hash function and $E_K()$ is a symmetric encryption algorithm such as AES. Briefly, cryptographic hash functions are mapping $H_K(x) : \mathcal{A} \to \mathcal{B}$, where $x \in \mathcal{A}$, $|\mathcal{A}| \geq |\mathcal{B}|$ and where it is computationally infeasible to determine any information about x given $H(x)$. The use of $H_K()$ for keywords is to ensure that only the data owner can perform the queries and that the resulting hash value is fixed size even for keywords with different length. Furthermore, the scheme does not require the extraction of keyword from the hash values so a cryptographic hash function would represent a lower cost option compared to encryption.

For the discussed example, the encrypted index would be

E_K('Horror')	$E_K(100)$
E_K('Fiction')	$E_K(101)$
E_K('World War')	$E_K(011)$
E_K('Biography')	$E_K(010)$
E_K('Pandemic')	$E_K(001)$

To perform a search for a set of keywords $kw' = \{kw_1, kw_2, \ldots, kw_q\}$, the data owner computes their hashes, $H_K(kw') = \{H_K(kw_1), H_K(kw_2), \ldots, H_K(kw_q)\}$, using the secret key and sends them to the cloud server. The cloud server looks up entries in the index tables corresponding to $H_K(kw')$ and return the encrypted index entries to the data owner. The data owner then decrypts and finds the intersection of index entries and identifies the matching documents:

$$D_K(I(H_K(kw_1))) \,\&\, D_K(I(H_K(kw_2))) \cdots \,\&\, D_K(I(H_K(kw_q))), \tag{2.5}$$

where & is a bitwise AND operation. Suppose a query was made for all biographies from World war veterans, a search for 'World War' and 'Biography' would require E_K('World War') and E_K('Biography') to be sent to the cloud server. $E_K(011)$ and $E_K(010)$ would respectively be returned to the data owner, who identifies B as the matching results from $011\&010 = 010$

2.2.5 Bloom Filters

While indexes provide a reliable and familiar approach to searching encrypted data, the need for decryption and encryption during search can be computationally expensive for certain applications. As an alternative, Bloom filters offer similar level of performance without the need for decryption, but, unlike indexing, results can contain false positives. While generally undesirable, false positives can provide some level of privacy protection [27].

Bloom filters are space-efficient probabilistic data structure used to test whether an element is a member of a set. A Bloom filter contains m bits and μ non-cryptographic hash functions, $\mathcal{H}_i(x)$, are used to map elements to the m-bits in the filter. All bits in the filter are initially set to zeros. To add an element, a, to the filter, we compute $\mathcal{H}_i(a)$ for $i = 1$ to μ, and set the corresponding positions in the filter to 1. For example, for $\mu = 2$ and $m = 5$, to add 'Happy' to the filter, we compute $\mathcal{H}_1$('Happy') $= 1$ and $\mathcal{H}_2$('Happy') $= 4$. Setting the position 1 and 4, the Bloom filter becomes $1, 0, 0, 1, 0$. To test for membership of an element, b, in a sample Bloom filter, we compute $\mathcal{H}_i(b)$ for $i = 1$ to μ, the element is determined to be a member if all corresponding positions of the sample Bloom filter is set to 1. For example, 'Happy' would be a member of the Bloom filter, $1, 1, 0, 1, 1$.

While Bloom filters have no false-negatives, it can falsely identify an element as member of a set. Given μ hash functions, n items inserted and m bits used in the filter, the probability of false positives is approximately:

$$p = (1 - e^{-\mu\frac{n}{m}})^{\mu}. \tag{2.6}$$

Applying Bloom filters for search consists of viewing the keywords associated with a document as a set and individual keywords as its members. Using the same example as in previous section, Book A would need to add 'Horror' and 'Fiction' to its filter. Suppose $\mu = 2$, $m = 5$, $\mathcal{H}_1$('Horror') $= 1$, $\mathcal{H}_2$('Horror') $= 4$, $\mathcal{H}_1$('Fiction') $= 2$ and

$\mathcal{H}_2$('Fiction') = 4, Book A's keyword filter would be $1, 1, 0, 1, 0$. Proceeding similarly for the remaining documents yield the following Bloom filters analogous to the index table in previous section:

Book A	11010
Book B	01101
Book C	11011

To search for 'World War' and 'Biography', we would construct a query filter where $\mathcal{H}_1$('World War'), $\mathcal{H}_2$('World War'), $\mathcal{H}_1$('Biography') and $\mathcal{H}_2$('Biography') are set and send it to the server. Suppose, the query filter is 01101, the server identifies all filters with the 2nd, 3rd and 5th bits set and returns Book B as the result.

Using Bloom filters for encrypted data proceeds in the same manner except members of filters consist of keyed hashes of keywords. That is, to add 'Happy' to a filter, we first compute its keyed cryptographic hash, H_K('Happy'). Then, we hash the result and set the filter bits as before using $\mathcal{H}_1(H_K(\text{'Happy'}))$ and $\mathcal{H}_2(H_K(\text{'Happy'}))$. To perform a search, we construct a query using the cryptographic hash of keywords under search as members. Since the cloud server does not have access to k, it cannot perform searches without data owner's authorization.

Note that if the file names also require privacy, a small lookup table matching numerical identifiers to file names can be stored privately by the data owner. The matching numerical identifiers can then be used in place of file names on the cloud server. A sample file name to identifier table is as follows:

Book A	83546
Book B	15378
Book C	43879

When compared to the encrypted indexes approach, the use of Bloom filters will generally lead to a much smaller storage requirement at the cost of having false positives.

2.2.6 Homomorphic Encryption

Homomorphic encryption allows computations to be carried out on ciphertexts, where the results would decrypt to the corresponding computation on plaintexts. For example, an additively homomorphic scheme would have the following property:

$$Add(E(A), E(B)) = E(A + B) \tag{2.7}$$

where $Add()$ is a function of the ciphertexts and depends on the encryption algorithm. It may be as simple as a XOR or a multiplication. This feature allows for third parties to perform computations without exposing confidential information.

Paillier cryptosystem

Paillier cryptosystem [51] is one of the most popular probabilistic homomorphic encryption algorithms in the literature. The scheme involves three algorithms:

1. Key Generation: Generate two large primes, p and q, of equal length. Set $n = pq$, $g = n + 1$, $\lambda = \phi(n)$ and $\mu = \phi(n)^{-1} \bmod n$, where $\phi(n) = (p-1)(q-1)$

2. Encryption: For a message, $m \in \mathbb{Z}_n$, compute ciphertext, $c = g^m r^n \bmod n^2$, where $r \in \mathbb{Z}_n^*$

3. Decryption: For a ciphertext, $c \in \mathbb{Z}_{n^2}$, compute the message, $m = L(c^\lambda \bmod n^2)\mu \bmod n$, where $L = \left\lfloor \frac{\mu-1}{n} \right\rfloor$

The public key, (n, g), is used for encryption and made public and the private key, (λ, μ), is used for decryption and kept private by the owner. The scheme is additively homomorphic:

$$E(m_1, r_1)E(m_2, r_2) \bmod n^2 = E(m_1 + m_2, r_1 r_2) \bmod n^2. \tag{2.8}$$

Although there's no known method to compute the multiplication of two ciphertexts, multiplication with plaintexts can be performed by

$$E(m_1, r_1)^{m_2} \bmod n^2 = E(m_1 m_2, r_1^{m_2}) \bmod n^2. \tag{2.9}$$

The latter is used in protocols for more complex functions such as secure scalar products [26].

Until recently, most homomorphic encryption algorithms are either additive or multiplicative, but not both. Gentry [25] described the first fully homomorphic encryption algorithm which supports both addition and multiplication over ciphertext, opening the door to many applications and a dramatic increase in interest on computing over

encrypted data. Fully homomorphic encryption algorithms can now run in relatively reasonable time. However, its computational cost remains several orders of magnitude higher than all popular encryption algorithms [32]. Therefore, it is generally an impractical option.

2.3 Related Work

Many work can be found in the literature towards enabling privacy preserving search on encrypted data for single keyword search, conjunctive keyword search and phrase search. We describe some of the most relevant results in this section.

2.3.1 Single and conjunctive keyword search

Boneh *et al.*'s work [13] on an encrypted keyword search scheme based on public key encryption was among the most cited in the area. At a time when email was the primary method of communication online, the author considered a scenario where a user wishes to have an email server verify messages associated with certain keywords without revealing the content of the emails. The usefulness of the system, shown in Figure 2.4, is demonstrated in a scenario where certain emails sent by various people may be urgent and required immediate attention from the recipient. Hence, rather than waiting for an email retrieval request, the recipient may be immediately alerted to the urgent matter, all while maintaining the secrecy of the email contents. The proposed solution uses identity-based encryption (IBE) and a variant using bilinear mapping.

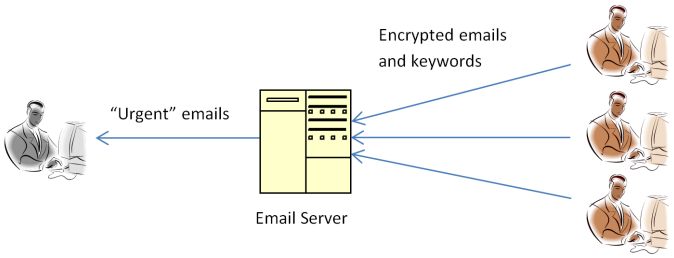

Figure 2.4: Email filtering system

Another interesting application was proposed by [74] regarding searching through encrypted audit logs, where only relevant logs are retrieved. The scenario involves an auditor which acts as a key escrow authorizing investigators to search audit records. The scheme uses an extension of Boneh's scheme using identity-based encryption. Song *et al.*

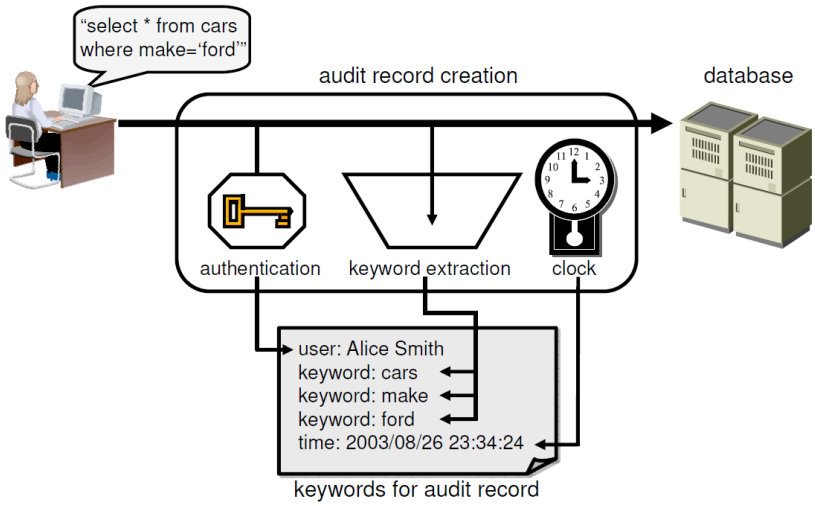

Figure 2.5: Audit records, with keywords and meta-data such as user and time [74]

[66] also considered the scenario introduced by Boneh *et al.* and proposed a probabilistic search solution based on stream cipher.

Many recent works have focused on conjunctive keyword search, that is performing queries using multiple keywords. Ding *et al.* [20] extended Boneh *et al.*'s scheme using bilinear mapping to perform multiple keyword search and described a solution that did not include expensive pairing operations in the encryption and trapdoor generation phase. Kerschbaum *et al.* [41] considered the search of unstructured text, where positions of keywords are unknown. The use of encrypted index for keyword search was examined in [27] and a scheme secure against chosen keyword attack was proposed. The ranking of search results was looked at by Wang *et al.* in [71]. The authors described a solution based on the commonly used TF-IDF (Term Frequency $\times$ Inverse Document Frequency) rule, which computes a value representing the importance of a keyword to a document in the corpus, and the use of order preserving symmetric encryption. Liu *et al.* [44] considered the search for potentially erroneous keywords termed fuzzy keyword search. The index-based solution makes use of fuzzy dictionaries containing various misspelling of keywords including wildcards.

2.3.2 Phrase search

Solutions for searching for phrases over encrypted data were only recently proposed by researchers. The main difference between conjunctive keyword search and phrase search is that the queried keywords must appear contiguously in the specified order in addition

ID	Word Vectors				
3	diseas:**3**	heart:**6**	induc:**8**		
5	heart:**1**	diseas:**2**	led:**5**	caus:**6**	death:**8**
10	his:**1,6** becom:**8**	diseas:**2** heart:**9**	idea:**3**	led:**4**	employe:**7**
13	his:**1** size:**6**	small:**2** day:**8**	heart:**3**	grew:**4**	three:**5**

Figure 2.6: Keyword location index [79]

ID	Word Vectors				
3	f7b:0—**3**	487:0—**6**	477:0—**8**		
5	487:0—**1**	f7b:0—**2**	55d:0—**5**	d37:0—**6**	ff3:0—**8**
10	110:0—**1,6** 498:0—**8**	f7b:0—**2** 487:0—**9**	aef:0—**3**	55d:0—**4**	7e9:0—**7**
13	110:0—**1** 667:0—**6**	99f:0—**2** eef:0—**8**	487:0—**3**	2f3:0—**4**	498:1—**5**

Figure 2.7: Encrypted keyword location index [79]

to all being present in the document.

Zittrower *et al.* [79] were the first to investigate the problem. His solution uses a keyword-to-document index and a keyword location index. The keyword-to-document index provides the mapping of keywords to the documents which contain them while the location index contains the position of the keywords within each document. The researchers identified potential statistical attacks on the indexes: Since certain words are more common than others in every natural language, the distribution of keywords in the indexes could reveal information on the documents. To defend against statistical attack, truncation of encrypted keywords was used to generate false positives in query results to hide the true search terms. To identify false positives, indicators are included in the index entries and also stored client-side. Figure 2.6 shows a sample keyword location index and figure2.7 shows the corresponding truncated partially encrypted index. The locations are also encrypted in the location index stored in the cloud. Note that 498 appears twice but actually represents different words. Indicators, i.e. 0 and 1 for the example, is attached to differentiate between them. A table stored client-side maps indicators to keywords. Figure 2.8 shows the keyword-to-document index. With a large enough corpus, many keywords can be associated with the same truncated encrypted form, providing a measure of privacy and defense against statistical analysis. The scheme requires a fairly high communication and computational cost due to the large amount of false positives used to provide security. Much of the computation is also performed client-side.

Enc. Word	ID
487	3,5,10,13
498	10,13
...	...
f7b	3,5,10
ff3	5

a: Encrypted

Word	ID
heart	3,5,10,13
become/three	10,13
...	...
disease	3,5,10
death	5

b: Unencrypted

Figure 2.8: Keyword-to-document index [79]

Tang *et al.* [69] focused on the security of phrase search in a solution with provable security using normalization, which consists of creating lookup tables for encrypted keywords and locations of seemingly uniform distribution. Their technique also uses two index tables: a keyword-to-document index and a keyword chain table. Central to their solution is the keyword chain table used to verify existence of pairs of keywords. Each location is represented as a random value, r_i. In addition to storing the location of the keyword in the index, it also stores the hash, $h_S(r_i)$, of the preceding location. Figure 2.9 illustrates this process. Thus, to verify a phrase, one needs to access the row entries corresponding to the keyword and document using $\psi_Z(word_d||id(\mathcal{D}_i))$ and compute the chain of hashes starting with the first word. In order to achieve provable security against statistical attacks, the keyword chain table is normalized against all documents in the corpus. Random data is used to fill in the table so that the same number of elements is listed under every entry. This results in a uniform distribution of entries in the table. However, the solution has a high storage cost as the index tables require significant storage, which hinders its practicality.

2.4 An Efficient and Secure Phrase Search Scheme for Encrypted Cloud Storage

Our first solution [53] aims to achieve high security with a much lower storage cost than [69]. The scheme is capable of basic ranking and was the first published work with the ability to search for non-indexed keywords. It's suitable for applications where security is of high importance and where client-side computation and higher storage are acceptable trade-offs. For example, a private company's financial statements or patient health care records at a clinic are confidential documents that would require a high level of security and privacy. We begin by providing the basic conjunctive search algorithm in section 2.4.1 and the basic phrase search algorithm in section 2.4.2. An in-depth discussion of the

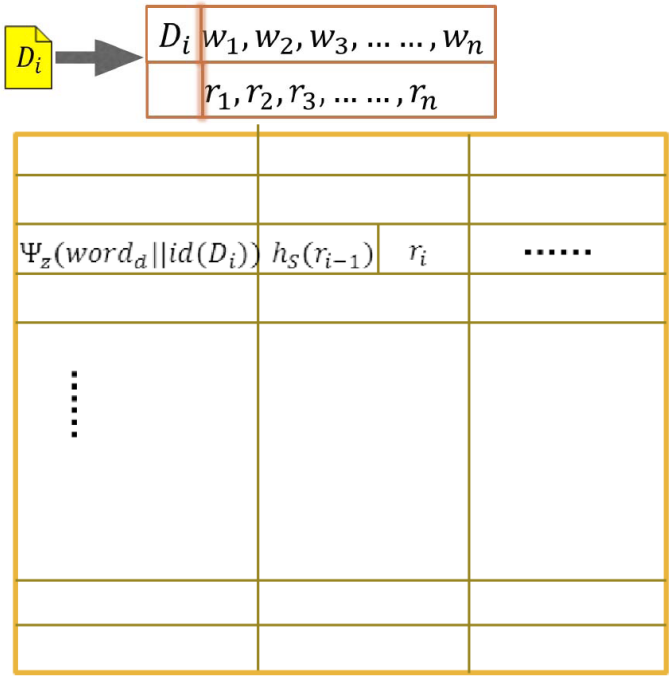

Figure 2.9: Keyword chain table [69]

modifications to balance the different security, storage and communication requirements can be found in section 2.4.3.

Note this work has also appeared in:

- H. Poon and A. Miri. An efficient conjunctive keyword and phrase search scheme for encrypted cloud storage systems. In *IEEE International Conference on Cloud Computing*, pages 508–515, 2015

2.4.1 Basic conjunctive keyword search protocol

Our starting point is a simple index based keyword search scheme, similar to the one described in section 2.2.4.

For a document collection, $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$, we parse each document, $\mathcal{D}_i$, for a list of keywords, $\{kw_1, kw_2, \ldots, kw_{n'}\}$. An index, I, is then generated mapping keywords to documents such that $I(kw_j) = (d_1, d_2, \ldots, d_n)$, where $d_i = 1$ if kw_j is linked to the document. The documents are then encrypted and uploaded to the server. The index is also encrypted prior to being placed on the cloud server:

$$I(E_K(kw_j)) = \{E_K(d_1, d_2, \ldots, d_n)\}. \tag{2.10}$$

Alternatively, a local dictionary mapping keywords to unique word id's, $I(Word_{ID}(kw_j))$ instead of $I(E_K(kw_j))$ can be used, but must be stored and maintained by the data owner.

To perform a search, the user sends a set of keywords $kw' = \{kw_1, kw_2, \ldots, kw_q\}$ to the data owner. The data owner computes $E_K(kw')$ and sends it to the cloud server. The cloud server returns the encrypted index entries to data owner, who then finds the documents matching the requested keywords from the intersection of index entries:

$$D_K(I(E_K(kw_1))) \,\&\, D_K(I(E_K(kw_2))) \cdots \,\&\, D_K(I(E_K(kw_q))), \tag{2.11}$$

where & denotes a bitwise *and* operation. If required, the data owner may now request that the matched documents be sent to the user from the cloud server.

2.4.2 Phrase search based on symmetric encryption

For querying phrases, the proposed solution by Zittrower [79] uses truncation of encrypted keywords to generate false positives in query results to hide the true search terms. As a result, part of the index must be stored client-side and the search is also performed by the client rather than the cloud. The use of false positives to provide security can be unreliable since the number of false positives associated to individual search terms is random. That is, by luck, one encrypted keyword may match several others after truncation while a different keyword may have no matches. Tang [69] addresses the privacy concerns by proposing a solution with provable security using normalization. The technique uses an index table that allows for verifications of chains of keywords instead of having the word locations stored. However, the index table requires significant storage, which hinders its practicality.

Searching for a phrase is searching for keywords that appear contiguously in the specified order. In another word, we must have some knowledge on the locations of the keywords. In [79], all keyword locations are stored alongside encrypted keywords in an index. In [69], the relative location of the keywords are stored in a table. Note that a standard single keyword index is also used in both cases. We first observed that the use of a single keyword index alongside the keyword location is enough for determining whether the phrase is present. We then note that we do not need to learn the location of more than a single keyword within the phrase.

To add phrase search capability to the basic scheme, a keyword location index is used. To generate the keyword location index for a document, $\mathcal{D}_i$, it is first parsed for keywords. Each keyword, $kw_{i'}$, would have its locations $j_{i''}$ encrypted and stored in the

index table, resulting in:

$$\{H(\mathcal{D}_i|kw_{i'}), E_K(j_1, j_2, \ldots, j_n)\}, \tag{2.12}$$

where $H()$ is a cryptographically secure hash function and j_x are the locations of $kw_{i'}$ within $\mathcal{D}_i$. Given a user phrase search request $kw' = (kw_1, kw_2, \ldots, kw_q)$, the data owner proceeds as in section 2.4.1 to determine documents containing all keywords. It then selects a random keyword, kw_{rand}, in the phrase and queries its location by sending $H(\mathcal{D}_i|kw_{rand})$ to the cloud. Given its locations in $\mathcal{D}_i$, the owner can identify potential starting locations of the phrase and return hashes of the phrase as if it starts at those locations:

$$\{H(E_{K_{\mathcal{D}_i}, j_s}(kw_1, kw_2, \ldots, kw_q)), i, j_s\} \tag{2.13}$$

where i is the index of the matched document and j_s is an identified candidate starting location of the phrase, for each match. $E_{K_{\mathcal{D}_i, j_s}}()$ represents the symmetric encryption of the phrase if it were at location j_s of document $\mathcal{D}_i$. Note that the secret key, $K_{\mathcal{D}_i}$, used for encrypting the stored document $\mathcal{D}_i$ is different from the secret key K used for encrypting the indexes. The cloud then computes $H(E(w_{j_s}, w_{j_s+1}, \ldots, w_{j_s+q}))$, where $E(w_j)$ is the actual j^{th} encrypted stored word in document i. Matched phrases are found where the following equality holds:

$$H(E(w_{j_s}, \ldots, w_{j_s+q})) = H(E_{K_{\mathcal{D}_i, j_s}}(kw_1, \ldots, kw_q)). \tag{2.14}$$

That is, if the queried phrase indeed appears at location j_s, then the ciphertext of the queried phrase would appear at location j_s of the document, $\mathcal{D}_i$, and their hashes must match.

Modes of operation

Unlike previous works by [79] and [69], we do not rely purely on indexes to determine matches. Instead, we process the encrypted document themselves on the fly. The advantage is that we require less information to be stored in the index and a lower storage requirement by the cloud server. The ability to process encrypted data, however, poses some challenges. With asymmetric encryption, the computational cost can be prohibitive. With deterministic symmetric encryption, the data must be encrypted using a mode of operation that prevents the same plaintexts from mapping to the same ciphertexts.

Cipher-block chaining (CBC), a commonly used mode of operation, would require an extra step where the cloud must return the ciphertext, $E(w_{j_s-1})$, directly ahead of the expected starting position of the phrase to the owner in order for the owner to compute $H(E(kw_{j_s}), kw_{j_s+1}, \ldots, kw_{j_s+q}))$. A simple initialization vector can be $H(\mathcal{D}_i)$, where $\mathcal{D}_i$ is the document id, should a phrase starting at the first position be queried.

Alternatively, we propose the use of counter mode (CTR), which does not require this extra step and also renders the encryption parallelizable, an advantage today where multi-processor computers are ubiquitous. The initialization vector can be stored alongside the document in plain or simply use $H(\mathcal{D}_i)$ to provide further saving.

Non-keyword search

An interesting property of the proposed scheme is that it has the ability to search encrypted documents for words that are not part of the keyword space. To the best of our knowledge, this is currently the only proposed phrase search algorithm with this ability. In many scenarios, it is impractical to index every word in the document sets. Common words such as 'it' or 'and' are often omitted. Some may choose to go further and index only distinctive words relevant to the document.

As long as the queried phrase contains at least one keyword, the owner can determine the expected starting position of the phrase and proceed with the query. Although possible, querying a phrase containing no keywords would require expensive brute-force matching through the document set.

Ranking

The scheme provides the basic ability to rank the returned results during the final step of the phrase matching process: The number of matched phrases per document can be tracked and used to rank the returned results. Though not discussed here, further ranking capability can be added by incorporating TF-IDF as in [71,76] during the initial conjunctive keyword matching phase using order preserving encryption [6]. This can even be used to provide the best odds of relevant results should an application wishes to limit the number of results for the query by proceeding to the phrase search on only the highest ranked documents in the initial step.

Proximity ranking, which ranks results based on the distance between keywords, was suggested by Zittrower [18, 79] and can also be used here since the location of the keywords can be queried.

2.4.3 Security

At rest, the cloud server contains the encrypted documents, $E_{K_{\mathcal{D}_i}}(\mathcal{D}_i)$, the keyword index, $I(E_K(kw_j)) = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_y\}$ and the keyword location indexes, $\{H(\mathcal{D}_i|kw_i), j_1, j_2 \ldots j_n\}$. The security of the encrypted documents at rest is equivalent to that of the symmetric encryption $E_{K_{\mathcal{D}_i}}()$ and cryptographic hash function, $H()$. However, the keyword and location indexes are particularly susceptible to statistical attacks, since certain words are more common than others in every natural language. The location of the keywords may also reveal information to an adversary with partial knowledge [79]. For example, if knowledge of statistical distribution of documents allowed the cloud to identify a document containing 'FBI', 'North Korea' and 'assassination', it could potentially represent a compromise in national security.

To alleviate the problem, Tang [69] opted to encrypt the single word index and have the keyword chain index normalized by filling in random data so that every entry contains the same number of elements as the largest entry prior to normalization. Encrypting the single word index is sufficient to provide provable security for hiding single word statistics. However, hiding the statistical properties of the keyword chain index used to provide phrase search capability came at a high cost in storage. It is known that most natural languages roughly follows Zipf's law, which states the word frequency is inversely proportional to its rank in the frequency table [59]. By filling in entries up to the maximum occurrence of a word in the entire document set, the resulting index comprises almost entirely of random data. Figure 2.10 shows the distribution of a particular research paper with approximately 6000 words, which follows roughly the distribution of documents of English language. We'll refer to this example for the remaining of this section to explain the design of our scheme. If all words in the sample document are indexed, the most common word is 'the', with 445 instances, $\approx 7\%$ of all words. It was found that the ten most common words comprises 25% of all words in the document. In fact, almost 1200 of the 1300 distinct words occurred less than 10 times in the document. Using Tang's scheme, the keyword chain table for this document would contain 446*1300 entries where over 436*1200 contains random data. Thus, less than 10% is actually used for searching. If the most common English words are excluded, we would have 1066 distinct words indexed and the 20 most common words comprising 25% of all words indexed. The most common word occuring 84 times and over 1000 words occuring less than 10 times, resulting in less than 26% of the data useful for search.

During query, one must also account for the statistical property of search terms, that is users may be more likely to perform a search for a subset of keywords. Then,

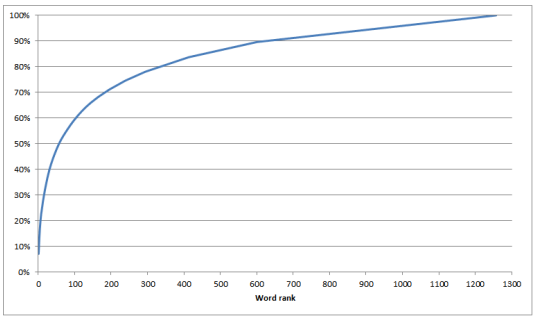

Figure 2.10: Cumulative distinct word distribution of a sample document

it may be possible for the cloud server to establish a known plaintext scenario, where kw_j and $E_K(kw_j)$ are discovered. Since the indexes, d_j, are encrypted, the link between document and keywords are not available from the keyword to document index only. Hence, knowing the encrypted keywords in a conjunctive keyword query would not allow an attacker, i.e. the Cloud, to identify the documents which the keywords reside in. However, if an attacker can identify the keywords in a phrase search, he would be able to identify which keyword location table is accessed in our original setup where each table is stored alongside the document and hence be able to link certain keywords to the documents. A simple modification would mitigate this attack: Simply store all the keyword location table together as a corpus-wide keyword location table and have location search use this table for any document. Note that the hash values for $H(D_i|kw_i)$ would have to larger in size to reduce risk of collision. This prevents the Cloud from learning which document the keywords belong to in the same way that the conjunctive keyword table does. Since the document ID is already part of the information in the table, the simple change is sufficient.

Reducing storage cost in providing security of the location index

The algorithm provided in section 2.4.2, although efficient, is vulnerable to statistical analysis. Namely, the number of entries for each word in a document is not hidden. A modification can be made to provide similar privacy protection as in Tang's scheme [69]. The goal of normalization is to hide the fact that certain keywords appear more frequently than others. Since the number of entries for each word cannot be reduced without losing potential matches, the simplest approach would be to insert random data until every word is associated with the same number of entries, as suggested by Tang. Furthermore, to hide the statistics of the most frequent word of each document in the document set,

the normalization can be applied over the entire set rather than per document. This results in every query returning the same number of entries independent of the keyword or document, but at a high storage cost.

As noted in section 2.4.3, keyword frequencies are distributed such that most common words dominate the entries. Although not as pronounced when the most common words in English are excluded, the trend remains. In the example provided, the two most common words were 'decoding' and 'algorithm', the topic of the research paper, appearing 84 and 64 times respectively. By the 21st most common word, the count has reduced to 18. Therefore, the high cost in Tang's scheme is due to having to adjust the majority of the keywords to accommodate these very few common keywords. Perhaps it would be preferable to do the reverse. Instead of increasing all entries to match the most common keyword, we could 'reduce' the number of entries taken up by the most common keywords.

Rather than normalizing according to the most common keyword in the document set, we could normalize to the keyword at, say, the $p = 95\%$ percentile when ordered from the least to the most common. For 95% of the keywords, it suffices to increase the number of entries to match the target keyword. For the most common $1-p = 5\%$, each keyword's entries must be split into multiple equivalent keywords, each comprising of the target number of entries. In the example, $p = 95\%$ corresponds to a keyword which appeared 10 times in the text. Keywords that appear between 11 and 20 times will be split into 2 equivalent keywords, etc. This results in $1*31+2*7+3*4+4*4+5*1+6*2+8*1 = 98$ extra keywords, each with 10 entries, for a total of $1164 * 10 = 11640$ entries in addition to the 1164 indexed keywords. If normalization to maximum keyword had been used, the resulting table would contain $1066 * 84 = 89544$ entries instead, a 87% reduction in number of entries in the proposed approach. In fact, the reduction in storage cost is greater still since an entry in a keyword chain index requires two parts containing the hashed previous word's location and the current word's location while our scheme requires only the latter.

Note that a split keyword requires more storage for its different variations and for padding the number of entries to a multiple of the target value. It is also necessary to store the number of times a keyword's been split. As p decreases, the amount of storage we save increases, but at a slower rate, potentially reaching a point where the momentum reverses and storage cost instead increases. Depending on the keyword frequency distributions, the optimal value for p can be any value between 0 and 1. At the limit case of normalizing according to the least common keyword, reasonably assumed to have only a single entry, the scheme reduces to indexing every word in the document, resulting in

3224 entries with 3224 keywords in the example. Assuming that keywords require same storage as a location entry, the optimal value for p was heuristically found to be 70%, where 2010 keywords were used with each containing only 2 entries. A table listing the number of times 149 keywords is split must also be stored by the client. Note that the client side storage cost is comparatively low due to the low number of common keywords in typical documents. Section 2.4.4 contains a more in-depth discussion and experiment to find the optimal value for p and an optimal split value.

Another advantage of this approach is the relative ease in adding and removing documents from the set. A new document can be parsed as usual and each keyword index is split and padded to fit the norm of the set. Conversely in [69], a problem could arise where a new document's most common term appears more frequently than the most common term in the document set, and could not be added without regenerating the entire index. Removing documents is also straight forward, with each document corresponding to a bit in the single keyword index and each document having its own keyword location index. The task becomes more difficult using Tang's approach since the keyword chain index is applied to the entire document set. When adding new documents, our scheme does require updating certain entries in the keyword to document index and since they are encrypted, the entry corresponding to the keywords in the added documents would have to be updated in entirety. A practical approach is to schedule an index update during low traffic and maintain a local index of newly added files until the scheduled update. Note that our scheme does leak information on the statistics on the number of distinct keywords for the documents. However, since its value is typically a function on the document size [22], which is available to the cloud server, we do not believe it to be a significant source of security leak. Therefore, we did not normalize the number of distinct keywords between different document indexes.

Reducing communication cost in providing security of the location index

There's also a communication cost to querying more keywords. However, since the cloud server does not need to return a significant number of unused entries, the communication cost is also reduced. For the sample research paper, a query for 'polynomial code' requires returning only 37 entries using the proposed scheme whereas 168 entries would have to be returned if maximal normalization were used. On this note, one may notice that the number of entries queried are susceptible to statistical analysis. If a query for a single keyword returns a very small number of entries, one can reasonably assume that the queried keyword(s) is not a common word. However, if a query for multiple keywords returns a larger number of entries, it does not imply a common keyword was queried since

multiple keywords can be assigned to a common term or many less common terms. There is no obvious way for the cloud server to decide whether two encrypted keywords are equivalent. Similar to previous case, maximal normalization would provide guaranteed protection. Assume the most common keyword in the document set is associated with x equivalent keywords, then a query for n keywords should proceed as follows:

a) Select random $nx - n$ keywords

b) Perform the query for the combined nx keywords

In this way, every keyword queried returns the same number of entries. Based on similar reasoning as for storage, we argue that we would achieve only marginally less protection for common keywords if we instead set a minimum of $q \leq x$ keywords queried instead of nx. A query for $q = x$ keywords can then be associated with the most common keyword or a combination of many less common keywords. From the example, less than 2% of the keywords appear more than 18 times while the most common term appears 84 times. Choosing $q = 18$ would then only reveal that a query of 18 keywords does not include 2% of the keywords in the document. Therefore, choosing $q < x$ could provide significant savings at the cost of minimal security loss.

2.4.4 Analysis

To provide a better understanding of the scheme on large document sets, we retrieved 1500 documents made available by Project Gutenberg [2] and compared our results against other phrase search schemes. The Natural Language Toolkit [12] was used to determine the statistical properties of the corpus and the performance for the various schemes.

Effect of common passages in a corpus

The retrieved documents often include headers and footers outlining the copyright, contact and source information. The inclusion of such texts that are repeated throughout the document set can skew the statistical property of a corpus. Other examples of such documents include forms, contracts and technical documentations. The possible inclusion of repeated texts highlights the need for hiding the word frequency distribution of documents. To better understand the skewing effect, we attempted to remove the header and footer from the documents. However, due to inconsistent placement and wording, we were only able to remove most, but not all of them.

While examining the frequency distribution of words in each document, we found that the most frequent word in each document was particularly affected by the skewing effect. The following are the most common words in the list when headers and footers are included: said, would, *ebook*, upon, thou, king, state, *etext*, love and like. When they are removed, the list becomes: said, would, upon, thou, king, love, state, like, little and could. Note the terms, *ebook* and *etext*, are frequently used words in the header and footer rather than the content. Further examinations also revealed that, while the corpus averaged 4000 distinct words per document, the documents whose most common words were ebook or etext contain an average of only 1600 distinct words. Therefore, the skewing effect affects mainly shorter documents where the header and footer represent a significant portion of the words.

Should the basic scheme in section 2.4.1 be used without any protection against statistical analysis, an examination of the index tables could reveal significant portion of the content of the shorter documents by matching the frequency distribution of words in the header and footer against the encrypted keywords, $E_K(kw_j)$, in the index table. We illustrate such an attack in the following scenario: A company is storing application forms on a cloud service provider. An employee of the cloud service provider legitimately uses the company's service to submit a form. In addition to having knowledge of the form's content, the employee also has access to the corresponding encrypted file along with the encrypted index tables. Any texts that are not entered by a user must be in every encrypted form. If the amount of texts entered by the user is much less than the texts already on the form, the employee could have reasonable chance of success at determining the keywords, kw_j, corresponding to, $E_K(kw_j)$, by comparing the number of entries for the keywords with the frequency distribution of words over the form. Any discrepancies must then be user-entered texts.

Effect of indexing common words in a corpus

Common words such as "the", "on" and "at", also called *stop words*, are often filtered out before indexing since they are functional terms of little relevance to the document's content. Indexing common words can also be expensive in both storage and computation. However, filtering them out could limit the ability to search for phrases that include such terms, as is the case in [69, 79]. Recall that our scheme can search for non-indexed keywords. To evaluate the effect of indexing common words, We processed the corpus with and without stop words.

When common words are included, the most frequent word in 96% of the documents becomes "the". In terms of security, if protection against statistical analysis is not

included, this can easily be exploited as described in previous section. In terms of storage, we observed that the highest number of instance of "the" in a document was 129070. When stop words are removed, the number of instance of the most frequent word becomes only 10757, which is 92% less frequent than "the". This illustrates that not indexing common words can lead to significant reduction in storage cost. To correctly search for phrases that include stop words, the approaches by [69, 79] would require indexing common words, which translates to significantly more storage than our scheme.

Finding an optimal p value relative to storage cost

We examined the optimization of the scheme in terms of storage. Since symmetric encryption is used, storage cost of the encrypted documents is optimal. The storage cost of our keyword-to-document index is similar to existing schemes. Since there's only one such index for a corpus, its cost is relatively small compared to the location index which is assigned to each document. Therefore, we focus our evaluation on the location index. To this end, we first define the storage cost to be the total number of entries the table maintains. The storage cost of the scheme is dependent on the amount of times a keyword is split. We initially defined a p value which determines the percentage of the keywords in a document that are left unsplit. Figure 2.11 shows the histogram of optimal p value for the sample document set. For 90% of the documents, the optimal p value was observed to be between 60% and 70% percentile. The histogram for the corresponding number of entries assigned to each keyword is shown in Figure 2.12. The majority of the optimal split values are between 2 and 4. Although the p value was found to be more consistent throughout the corpus, it might be more practical to have a fixed split value to reduce the preprocessing cost. It may also be more secure. Since the optimal split value is dependent on the keyword distribution of the document, Indexes with different split values could leak information on the document content.

2.4.5 Comparison with other schemes

We compare our phrase search scheme to Zittrower's [79] and Tang's [69]. Since symmetric encryption is used, the encrypted documents require roughly the same amount of storage as when in unencrypted form. This is true for all three schemes in consideration.

Using Zittrower's approach, the user must store a dictionary mapping every distinct word in each document to an index value used to distinguish between different keywords since different keywords can map to the same entries in the index table. Suppose there are x distinct keywords in the corpus and suppose optimal representation for the index

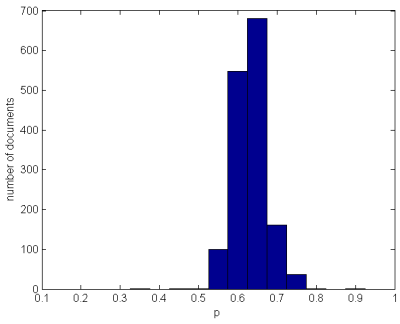

Figure 2.11: Optimal p value across 1500 sample documents

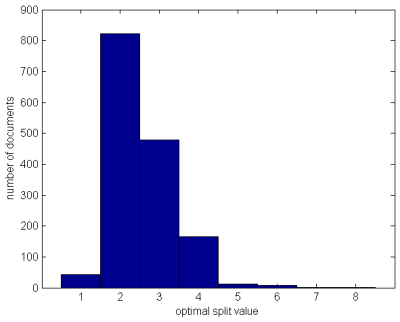

Figure 2.12: Optimal split value across 1500 sample documents

value was used, this represents $x(log_2(x) + b)$ bits of storage, where b is the average number of bits per keyword. On the server, two index tables are stored, one mapping truncated encrypted keywords to documents and another to their locations within the documents. Each encrypted keyword was truncated to 12 bits. Suppose that a location value requires y bits, that there's an average of x' distinct keywords and that each keyword appears q times on average per document. Then, the location table requires $x'(12+qy)$ bits and the keyword-to-document table requires $x(12+p\,log_2(n))$, where p is the average number of documents associated with a keyword in the corpus and n is the total number of documents in the corpus. The keyword-to-document table here requires the least amount of storage among the three schemes since normalization was not used. Instead, the scheme relies on false positives to defend against statistical analysis. The author stated an average of 300 collisions among the encrypted keywords when 12 bits are retained. This means a query for a single keyword would on average also return results belonging to 300 other keywords that must then be processed by the client. Although it is possible to reduce the communication cost by increasing the number of bits retained, this leads to lower collision rate and susceptibility to statistical analysis. Although it does not lead to a higher storage cost, the technique requires a higher communication and computational cost, especially on client side. Since the amount of false positives generated is random, the technique also has the risk of generating very low number of false positives for certain keywords, leading to a variability in security throughout the keyword set.

In Tang's scheme, the user similarly stores a dictionary mapping keywords to index values, requiring $x(log_2(x) + b)$ bits of storage. The server also stores two tables: The keyword-to-document index, requiring $x(log_2(x)+n)$ bits of storage, and a location index table requiring $x'(h + d(u + y))$ bits of storage, where h is the number of bits to store a hashed keyword, u is the number of bits to store a hashed location value, y is the number of bits to store a location value and d is the number of instance of the most frequent keyword in the corpus. Although the storage requirement for the client and the keyword-to-document index is similar to Zittrower's scheme, the location index table is several orders of magnitude greater, due to the normalization to the value of d. The goal of normalization was to achieve security against statistical attacks, but at a high storage cost. Unlike in Zittrower's scheme, the technique allows the cloud to perform the majority of the computations. In terms of communication, the scheme includes returning irrelevant results since normalization requires insertion of random data into the index tables, although much less than in Zittrower's scheme.

In our scheme, denoted our $\text{scheme}_{\text{sec}}$, the user must also keep track of the keywords

that are split in addition to the dictionary mapping keywords to index values. Our experiments showed that splitting keyword entries into pairs were ideal for reducing the size of the location index tables on server. Splitting into pairs, on average, increases the number of keywords by 150% in the location table, where approximately 27% of the keywords were to be split. Suppose that there is an average of x' distinct keywords per document and that k is the average number of instance of most frequent word per document, then the split keyword table would require $0.27x'(log_2(x') + log_2(k/2))$ bits of storage. Note that this client side table can be reduced in storage by splitting into triplets or quartets instead, which would respectively reduce the number of keywords to be split to 18% and 14%, although it would also have the effect of increasing the server side storage cost. It is also possible to encrypt the split table and store on the cloud similar to the index tables, eliminating the need for client side storage aside from the keyword dictionary. However, it must be retrieved in entirety and decrypted during search. Retrieving only split values of the relevant keywords would leak information on the number of common words among the search terms since only common terms are split. On the cloud server, our scheme requires a keyword-to-document index table which uses $x(log_2(x)+n)$ bits of storage and a location index table that requires on average $2.5x'(h+2y)$ bits, where h is the number of bits used to represent the hashed keyword. Although our scheme requires a higher client-side storage than both Tang's and Zittrower's schemes, the size of the split table is significantly smaller than the normalization cost in Tang's scheme and far less irrelevant data are used in normalizing the tables as explained in section 2.4.3.

Table 2.2 summarize the results of the schemes on the sample Gutenberg document set with the experimental values for the various parameters outlined in Table 2.1. For the hash values of keywords and locations, we assumed that each would require 16 bits in all cases. Since there are approximately 1 million words in the English language today, we would need no more than 20 bits to guarantee no collision. We believe 16 bits is reasonable since the number of words used in most scenarios is far less than the limit. As shown, the proposed approach can achieve significant savings in cloud storage at a modest increase in user storage.

2.5 A Low Storage Phase Search Scheme based on Bloom Filters for Encrypted Cloud Services

In this section, we present a phrase search scheme that emphasizes on achieving the lowest storage requirement [54]. Instead of encrypted indexes, the solution makes use of two sets of space efficient bloom filters. The scheme is capable of basic ranking and can

Table 2.1: Properties of the sample document set

Average number of documents associated with a keyword, p	885.6
Total number of documents, n	1530
Total distinct keywords, x	285396
Average number of distinct keywords per document, x'	3959.6
Average number of times each keyword appears per document, q	5.6
Number of instance of the most frequent keyword, d	10757
Average number of instance of most frequent word per document, k	369.1

Table 2.2: Comparison of phrase search schemes for a sample of 1500 documents

	Zittrower [79]		Tang [69]		Our scheme$_{\text{sec}}$	
	user	cloud	user	cloud	user	cloud
Storage	1.98MB	392.5MB	1.98MB	242.8GB	5.78MB	139.3MB

be adapted to search for non-indexed keywords and defend against Inclusion-Relation attacks [15]. It's suitable for applications where a low remote storage is required in exchange for higher computational cost or slower response time. For example, a network of small devices with limited storage or a startup looking to minimize financial cost. The scheme's conjunctive keyword and phrase search protocols are presented in section 2.5.1, along with explanations on various features and design choices.

Note this work has also appeared in:

- H. Poon and A. Miri. A low storage phrase search scheme based on bloom filters for encrypted cloud services. In *IEEE International Conference on Cyber Security and Cloud Computing*, pages 253–259, 2015

2.5.1 Phrase search scheme based on Bloom filters

A Bloom filter is a space efficient data structure used to test whether an element is a member of a set. In a search scheme, it can be used to test whether a keyword is associated with a document. The data structure can also be adopted for our phrase search scheme. A traditional phrase search scheme, as in [79] [69], uses a keyword-to-document index and a location/chain index to map keywords to documents and match phrases. We describe the use of Bloom filters to support the functionality of these indexes to further reduce the storage cost.

2.5.2 Conjunctive keyword search protocol

To provide conjunctive keyword search capability, each document, $\mathcal{D}_i$, is parsed for a list of keywords kw_j. A Bloom filter of size m is initialized to contain all zeros. Each keyword is hashed using a private key to produce $H_{k_c}(kw_j)$ before passing into k Bloom filter hash functions and the result is used to set k bits in the Bloom filter. This addition of keywords as members results in a Bloom filter for each document: $B_{\mathcal{D}_i} = (b_1, b_2, ...b_m)$ where $b_i \in \{0, 1\}$. The document collection, $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$, is then encrypted and uploaded along with the Bloom filters to the cloud server. Note that each Bloom filter can be viewed as the equivalent of an entry in an index table.

To perform a conjunctive keyword search, the user sends a set of keywords $kw' = \{kw_1, kw_2...kw_q\}$ to the data owner. The data owner computes a keyed hash, $H_{k_c}(kw')$, of the keywords, randomizes their order, and sends them to the cloud server. Upon receipt, the cloud server computes a query Bloom filter, $T = (t_1, t_2, ...t_m)$, for $H_{k_c}(kw')$ and verifies the keywords by comparing against the Bloom filters for each document. A match is found if $T = T\&B_{\mathcal{D}_i}$, where & denotes a bitwise *and* operation. Finally, the cloud server sends the matched documents to the user.

2.5.3 Phrase search protocol

To provide phrase search capability, each keyed hashed keyword is concatenated with its location, $H_{k_l}(kw_j)|j$, and passed into k hash functions and the result is used to set k bits in the keyword location Bloom filter, $Bl_{\mathcal{D}_i}$, stored alongside the conjunctive keyword filter on the cloud. To perform a phrase search, the user performs a conjunctive keyword search for all the keywords in the phrase as previously described, except the user also sends the hash of the phrase under a different private key,

$$H_{k_l}(kw') = (H_{k_l}(kw_1), H_{k_l}(kw_2)...H_{k_l}(kw_q)), \tag{2.15}$$

to the cloud. For each document containing all the keywords, the location of the first keyword in the phrase is queried. To do so, the cloud server verifies whether $H_{k_l}(kw_1)|s$ is a member of $Bl_{\mathcal{D}_i}$, for $s = 1$ to r', where r' is the size of the candidate document. Then, for each $H_{k_l}(kw_1)|s'$ found to be a member of $Bl_{\mathcal{D}_i}$, the cloud verifies whether $H_{k_l}(kw_2)|(s' + 1)$ is a member of the set to determine whether the next keyword is in sequence. The cloud then proceeds similarly to determine whether the keywords in the phrase follows in proper order by verifying the membership of $H_{k_l}(kw_{2+i})|(s' + i + 1)$ for each $H_{k_l}(kw_{1+i})|(s' + i)$ found, narrowing the search at each step. The documents

where phrases were found are returned, that is the set of $\mathcal{D}_i$ where

$$\{H_{k_l}(kw_{1+j})|s' + j\} \in Bl_{D_i} \text{ and } j = 1 \text{ to } q. \tag{2.16}$$

Note that the phrase search requires only 2 exchanges with the cloud server. The data owner relays the query Bloom filter T and the encrypted phrase $H_{k_l}(kw')$ to the cloud. The cloud performs the conjunctive keyword search and location search locally and returns the results.

Despite its advantage in reducing storage, Bloom filter does have some drawbacks in security and computational cost. Its use leads to a probabilistic generation of false positives to hide the search terms. Using a larger filter leads to lower false positive rate, but also lowers protection of search terms and higher storage cost. Using a small filter leads to greater protection, low storage cost but larger false positives and computational/communication cost from searching for non-relevant terms. Aside from the trade-off in security, the scheme can require a higher computational cost than an indexing approach, due to the iterative hash computations during phrase search to verify the keyword-location membership. The choice of hash function is therefore an important factor in the performance of the scheme.

Incremental hash functions

While popular cryptographic hash functions such as SHA2 and MD5 are often cited, they are less suitable for Bloom filter constructions. Since keywords are encrypted prior to addition into the filter to hide the search terms from the cloud, security properties such as preimage and collision resistance are not needed for our application. Instead, speed and low collision rate are valued.

From section 2.5.3, it can be observed that $H_{k_l}(kw_1)$ does not change as we iterate through different values of s when computing $\mathcal{H}_i(H_{k_l}(kw_1)|s)$. Therefore, a hash function that can efficiently compute $\mathcal{H}_i(x_1|x_2|x_3')$ given $\mathcal{H}_i(x_1|x_2|x_3)$ can greatly improve performance. These are generally called incremental hash functions introduced by Bellare [9]. A series of efforts into their development culminated in a randomize-then-combine construction [10], in which standard cryptographic hash functions such as SHA2 and MD5 can be made incremental. Since the hash functions in the Bloom filter are not security-critical, we simplify the construction to increase performance. The resulting design is shown in Figure 2.13, where $h()$ is a standard hash function and b_i are input blocks. The key features being a simple XOR combine function and a parallel design for fast computation. Briefly, the goal is to separate a message into blocks, $b = (b_1, b_2, \ldots, b_n)$,

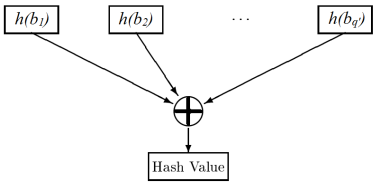

Figure 2.13: Incremental hash function for Bloom filter

and hash the blocks separately instead of hashing b as a whole. In this way, incremental changes that affect blocks require only recomputation of the affected block rather than recomputation the entire message.

While the randomize-then-combine construction by Bellare [10] holds a slight advantage in performance compared to the more popular Merkle-Damgård construction. The latter can also be used to perform incremental hashing provided that the location is attached at the end since the Merkle-Damgård process is iterative rather than parallel.

For the hash function, $h()$, MurmurHash3a is currently among the fastest non-cryptographic hash functions available while providing randomness and collision properties similar to its cryptographic counterparts [3].

Non-keyword search

Note that our scheme does not restrict the encryption algorithm used for the document set. However, its choice could enable certain features such as the ability to perform searches on non-keywords, i.e. words that are not added to the Bloom filters. Common words in a language, such as "the", "a", "is", also called *stop words*, are often omitted since they are functional terms of little relevance to the document's content. While this improves performance, it also limits the ability to search for phrases that include such terms, as is the case in [79] [69]. Consider the scenario where one wishes to search for "Flights from Canada to US" as opposed to "Flights to Canada from US". Despite having different meanings, the query results would be identical without the ability to consider stop words in the phrase. A more extreme example would be the popular quote "To be, or not to be", a phrase containing only stop-words. Note that, while possible, searching for a phrase with only unindexed stop-words would require brute-force matching. Interestingly, a slight extension of the quote "To be, or not to be, that is the question" would be searchable using our technique due to inclusion of the keyword "question", but would be problematic for existing solutions such as in [79] and [69].

To enable non-keyword search, we adapt a technique presented in section 2.4. Briefly,

we encrypt the document set using symmetric encryption, such as AES, running in counter mode with the initialization vector set to $H(\mathcal{D}_i)$. A list of the 500 most common words in order of frequency in the language of the document set excluding stop words is maintained by the data owner. To perform a phrase search, a conjunctive keyword search is performed with all the keywords in the phrase as in section 2.5.2. Then, the least common word in the phrase is identified using the frequency list and its location is queried. Then, given its locations, we can compute the candidate starting location of phrases by subtracting the offset of the least common word from the first word in the phrase. We then verify whether the first word is at the candidate starting location, s, by verifying whether $H_{k_l}(kw_1)|s$ is a member of $Bl_{\mathcal{D}_i}$ for each candidate document. Given the matching locations, the owner returns

$$\{H(E_{K_{D_i},j_s}(kw_1, kw_2, \ldots, kw_q)), i, j_s\} \tag{2.17}$$

where i is the index of the matched document, j_s is the identified starting location of the phrase and $H()$ is a non-cryptographic hash function, for each match. $E_{K_{\mathcal{D}_i,j_s}}()$ represents the symmetric encryption of the phrase at location j_s of document $\mathcal{D}_i$. The cloud then computes $H(E_{K_{\mathcal{D}_i,js}}(w_{j_s}, w_{j_s+1}, \ldots, w_{j_s+q}))$, where $E_{K_{\mathcal{D}_i,j_s}}(w_j)$ is the j^{th} stored word in document i. Matched phrases are found where the following equality holds:

$$H(E_{K_{\mathcal{D}_i,js}}(w_{j_s}, \ldots, w_{j_s+q})) = H(E_{K_{\mathcal{D}_i,js}}(kw_1, \ldots, kw_q)). \tag{2.18}$$

Ranking and adding/removing documents

Our scheme can also rank the query results by tracking the number of matched phrases per document. Proximity ranking, suggested by Zittrower [79], can also be used since the location of the keywords can be queried.

Another advantage of this approach is the ease in adding and removing documents since each document retains its own pair of Bloom filters. In the algorithm proposed by Tang [69], a problem could arise where a new document's most common term appears more frequently than the most common term in the document set. As a result, it could not be added without regenerating the index.

2.5.4 Modified phrase search scheme against IR attacks

In [15], Cai described the notion of inclusion-relation (IR) attacks, which states that two query sets, a and b, where a is the subset of b, would imply b includes the set of keywords from a. If a cloud server has some knowledge of the statistical properties of the incoming

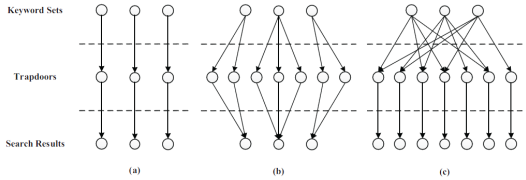

Figure 2.14: Relationship between keyword set, trapdoor and search result [15]

search terms, it can potentially discover some of the keywords.

An interesting property of the proposed technique is that it can be adapted to defend against such attacks by ensuring that a set of keywords can lead to many possible queries (trapdoors) due to the inclusion of false positives. Also, since different sets of keywords can lead to the same positions in the Bloom filter being set to 1, different keyword sets can also lead to the same query. This provides some level of privacy. Figure 2.14 shows the different mapping of keywords to trapdoors and search results between various conjunctive keyword search schemes. Our proposed technique is of type C, where the same phrase can lead to different encrypted queries, which was suggested to provide the best defense against inclusion-relation attacks [15].

An inclusion-relation attack can be carried out against the basic scheme if an attacker gains access to a significant amount of known queries and their associated search results. The class of searching algorithms of type C noted in Figure 2.14 was proposed to defend against such attacks.

To adapt our basic scheme in section 2.5.1 to defend against IR attacks, we increase the false positive rate by randomly removing z terms from the beginning and the end of the phrase being queried. Due to the uniqueness of long phrases, more terms can be removed to generate false positives. For our experimental data set, $z = \lfloor q/3 \rfloor$ was found to be effective for $q \leq 6$ and $z = q - 3$ for $q > 6$. For example, a query phrase, $kw' = (kw_1, kw_2, kw_3)$, would be queried randomly as $kw' = (kw_1, kw_2)$ or $kw' = (kw_2, kw_3)$. This results in false matches that contain only sub-phrases, severing the inclusion relation between search terms and query results for queries with common terms.

2.5.5 Security

The cloud server contains the encrypted documents, $E_{K_{\mathcal{D}_i}}(\mathcal{D}_i)$, the conjunctive keyword Bloom filter, $B_{\mathcal{D}_i}$, and the location Bloom filter, $Bl_{\mathcal{D}_i}$. The security and privacy of the documents are ensured by the symmetric encryption algorithm. The filters do not reveal meaningful information since all members are meshed together in the structure. The

words added to the conjunctive keyword Bloom filter are encrypted to prevent the cloud from learning the keywords that are contained in the documents. The location Bloom filter operates similarly, but the keywords are encrypted with a different key to sever the link between the two filters. If the same encryption key had been used, the cloud would be able to perform a location query even when it's not requested.

During query, users may be more likely to perform a search for a subset of keywords. Then, it may be possible for the cloud server to establish a known plaintext scenario, where kw_j and $E_K(kw_j)$ are discovered. Since the filters are stored per document, if the attacker can identify which filter was accessed during a search, he would have identify the documents that contain the keywords. While the low-storage construction is not designed defend against Cloud with statistical knowledge of incoming queries, it can be modified to defend against such attack, by recognizing that the distribution of incoming queries seen by the Cloud is controlled by the data owner. In a similar manner to the defense against IR attacks, the data owner can shape the distribution of keyword queries to be uniform by injecting fake queries. That is, the data owner maintain a table of keyword queries with a count value and whenever, a frequently searched keyword is queried, it injects extra keywords or phrases with probability equal to the inverse of their query probability such that infrequent keyword or phrase is more frequently injected.

2.5.6 Performance Analysis

As outlined in section 2.5.3, our scheme, denoted our $\text{scheme}_{\text{sto}}$, requires two Bloom filters per document, one for mapping keywords to document and one for determining keyword location. The two sets of filters require $N(b_k + b_l)$ bits of storage on the cloud server, where b_k is the size of a conjunctive keyword Bloom filter, b_l is the size of a keyword location Bloom filter and N is the number of documents in the corpus. The data owner needs only to store cryptography keys. The communication cost also compares favorably with existing schemes. The proposed protocol requires only two messages to be sent, one containing the keyed hash of the keywords for each filter, and the other containing the results of the query. Altogether, the scheme requires $2qh$ bits to be sent to the cloud server and $ulog_2(N)$ bits sent to the user, where h is the number of bits per hashed keyword and u is the number of matched documents. In terms of computation, the scheme requires $2q$ keyed hash computations on the client side to generate the trapdoor query sent to the server. Upon receiving the query, the server performs qk Bloom filter hash computations to produce the query filters, followed by a bitwise AND operation for each conjunctive keyword filter, $B_{\mathcal{D}_i}$, in the document set. Then, for each matched document, rk Bloom filter hash computation is needed to locate

the first word in the phrase, followed by $r_i k$ hash computation for each additional word in the phrase, where r is the number of keywords in the candidate document and r_i is the number of matches for the i^{th} word in the phrase. In most practical scenarios, $r \gg r_i$, resulting in approximately $u_i r k$ hash computations required during phrase search, where u_i is the number of matched documents during conjunctive keyword search. The response time of the scheme is dependent on the execution time/computational power of the server and the client, and also the transmission and propagation time of the messages. While the computational load will likely be the more important factor, our scheme also has the advantage of a short transmission time due to a small query filter and requiring only a single round-trip delay time to complete. Therefore, the distance between the server and client has a lower impact on the performance of the system.

Zittrower's truncated ciphertext approach requires a client-side dictionary mapping distinct keywords to index values to differentiate between entries associated with different keywords under the same key in the index table [79]. Assuming optimal representations, this table requires $x(log_2(x) + b)$ bits of storage, where x is the number of distinct keywords in the corpus and b is the average number of bits per keyword. On the server, two index tables are stored: one mapping truncated encrypted keywords to documents and another to their locations within the documents. The two tables require $x(12 + p' log_2(N))$ and $x'(12 + gy)$ bits respectively, where p is the average number of documents associated with a keyword in the corpus and y is the number of bits needed to store a location value. The scheme relies on false positives to provide security, with an average of 300 collisions among the encrypted keywords. Hence, a query would also return results belonging to 300 other unrelated keywords, leading to a significant amount of wasted bandwidth and processing. In terms of communication, this results in up to $300 u_i log_2(N)$ bits wasted during conjunctive keyword search, where u_i is the number of candidate documents matched in the search, and $300g(y + log_2(x)) + salt)$ bits wasted during phrase search for every keyword in the query. Together with the true results, the scheme would respond to a client's $12q$ bits query with up to $301 u_i(log_2(N) + 301q(g(y + log_2(x)) + salt))$ bits of data. In terms of computation, the client must first decrypt all the returned entries and then use the dictionary to identify the relevant results while discarding collision. Further processing follows to identify keywords that are in order based on the decrypted locations. The computational load is naturally higher compared to the non-cryptographic hashing used in the proposed scheme and must be carried out by the client. Although the scheme requires only a single round-trip delay, the transmission time is relatively long due to the high number of false positives. The response time also suffers due to the significant amount of decryption operations. Since the client likely possesses much lower

computational power than the server, it would also require a longer execution time to complete the decryptions.

Tang's scheme [69] similarly stores a keyword dictionary on the client while a keyword-to-document index and keyword chain tables are stored on the cloud server, requiring $x(log_2(x) + b)$ bits of client-side storage and $x(log_2(x) + N) + Nx'(h + d(h + y))$ bits of server-side storage, where b is the average number of bits per keyword, h is the number of bits to store a hashed keyword or location value, y is the number of bits to store a location value and d is the number of instance of the most frequent keyword in the corpus. Although the storage requirement for the client and the keyword-to-document index is similar to Zittrower's scheme, the keyword chain table, analogous to the location index, is several orders of magnitude greater, due to the normalization to the value of d. In terms of communication, the client sends $qlog_2(x)$ bits to the server during the conjunctive keyword search and receives qN bits as results. For phrase matching, the client sends $u_i(qlog_2(x)+(q-1)h)$ bits to the server, where u_i is the number of candidate documents, and receives $ulog_2(N)$ bits in matching document ID's. Although its communication cost is lower than Zittrower's scheme, communicating index entries and keys still costs more than the transmission of a single Bloom filter and the encrypted key terms required by the proposed technique. One of the main advantages of Tang's approach is in allowing the cloud to perform the majority of the computations. During conjunctive keyword search, the client performs a dictionary lookup and computes the keyed-hash of the keywords and sends them to the server. The server similarly performs a table lookup for the queried entries. During phrase search, the client hashes the keywords under a different private key in addition to $q-1$ chain keys for the chain digests. Upon receiving the hashed phrase and chain keys, the server finds the corresponding entries in the index via binary search, and performs up to $d(q-1)$ keyed hashes. Due to the size of the keyword chain table, a significant amount of hash computation is required to verify a phrase, especially when a candidate document contains all the keywords but not in consecutive order, where the maximum number of hashes would then be required to reject it as a match. The response time of the scheme is then largely dependent on the server's ability to compute the keyed hashes. When compared to the proposed scheme, it's important to note that the hashing algorithm used in Bloom filters is non-cryptographic and incremental, which is significantly faster than the cryptographic hash functions used in Tang's scheme. It should also be noted that the scheme requires two round-trip delays compared to the one required in our scheme, with a similar transmission time.

2.5.7 Experimental Results

We evaluate the proposed algorithm on a corpus consisting of 1500 documents made available by Project Gutenberg [2] and compared our results against existing phrase search schemes. The documents were preprocessed to exclude headers and footers, which include copyright, contact and source information to reduce skewing in the statistics of the data set. Stop words are also omitted. To determine the statistical properties of the corpus and the performance for the schemes, the Natural Language Toolkit [12] was used.

As Bloom filter forms the basis of our approach, its design parameters are an important factor on the scheme's performance. We rewrite the false positive rate equation 2.6 as follows:

$$\frac{m}{n} = \frac{-k}{ln\left(1 - e^{\frac{ln(p)}{k}}\right)}, \tag{2.19}$$

where k hash functions are used to insert n items into a m bits Bloom filter with a false positive rate of p. Figure 2.15 shows the number of bits needed per entry to achieve false positive rates between 1% and 10% depending on the number of hash functions used. To reduce storage cost, a small filter is desirable. A low false positive rate would be beneficial in terms of communication and computational cost. Most importantly, using a small number of hash functions greatly improves the execution time since the computational cost is proportional to the number of hash function used. Note that the the optimal k for minimizing false positive rate is rarely used in practice since there is often very little improvement in false positive rate as we increase the number of hash functions past a certain threshold. While the limit case of using a single hash function, $k = 1$, would minimize the computational cost, it also more than doubles the storage cost compared to using more hash functions. As shown in the figure, the number of bits per entry and the false positive rate is fairly stable for $k \geq 2$. Although we could improve the false positive rate by using more hash functions, increasing the number of hash function used from $k = 2$ to $k = 3$ would increase our computational cost by 50% while reducing storage cost by no more than 25%. Having considered the various trade-offs, the operating point was heuristically chosen for a false positive rate of $p = 5\%$ with $k = 2$ and $m/n = 7.9$. To determine the number of bits needed for the Bloom filters, B_{D_i} and Bl_{D_i}, we need the number of items that each filter must store. That is the total number of keywords per document and the number of distinct keywords per document. Their values among various parameters of the sample Gutenberg document set are listed in table 2.3. At 7.9 bits per entry, a conjunctive keyword Bloom filter would require, on

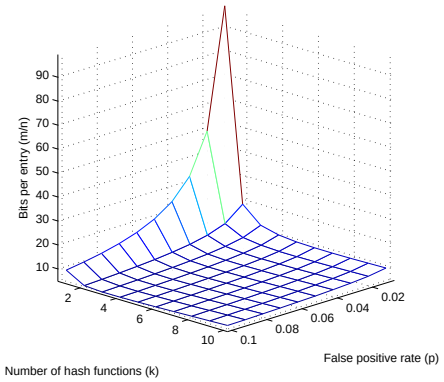

Figure 2.15: Bits per entry (m/n) as a function of the number of hash function (k) and false positive rate (p)

average, $b_k = mx'/n = 3.82$kB and a location filter would require $b_l = mr/n = 29.286$kB.

Table 2.5 summarize the results of the schemes on the sample Gutenberg document set with the experimental values for the various parameters outlined in table 2.3. The hash values of keywords and locations are assumed to require 16 bits. Since communication and computational costs are query-dependent, related parameters are kept in the formulas and approximations were made to retain dominant terms for clearer comparisons. In practical scenarios, $u_i > u > q$ and $Dec(x) > H_k(x) > H_{bf}(x) > LUT(x)$, where $Dec(x)$, $H_k(x)$, $H_{bf}(x)$ and $LUT(x)$ represent the cost of a decryption of x bits, a keyed hash computation of x bits, a Bloom filter hash computation of x bits and a table look up of x elements. Note that the communication and computational cost values for Zittrower's scheme are worst-case estimates. As shown in the table, our scheme requires almost 8 times lower storage cost than Zittrower's scheme. Aside from the low storage cost, it also achieves a lower communication cost and would vastly outperform both schemes when there are many candidate documents. As for computational cost, our scheme requires more work to be performed by the cloud server, but the load is far lighter on the client, which is desirable in a cloud based solution.

Table 2.3: Properties of the sample document set

Average number of documents associated with a keyword, p'	885.6
Total number of documents, N	1530
Total distinct keywords, x	285396
Average number of keywords per document, r	30364.7
Average number of distinct keywords per document, x'	3959.6
Average number of times each keyword appears per document, g	5.6
Number of instance of the most frequent keyword, d	10757
Average number of instance of most frequent word per document, k'	369.1

Table 2.4: Comparison of phrase search schemes

		Storage	Communication (download)	Computation
Zittrower [79]	data owner	$x(log_2(x)+b)$	$301u_i(log_2(N)+301q(g(y+log_2(x))+salt))$	$301u_iDec(301q(g(y+log_2(x))+salt))+LUT(x)$
	cloud	$x(p'log_2(N)+12)+Nx'(12+gy)$	$12q$	$LUT(x-x/2^{12})+u_iLUT(x'-x'/2^{12})$
Tang [69]	data owner	$x(log_2(x)+b)$	$qN+ulog_2(N)$	$LUT(x)+H_k(qb)+u_iH_k(qb)+u_iH_k(b(q-1))$
	cloud	$x(log_2(x)+N)+Nx'(h+d(h+y))$	$qlog_2(x)+u_i(qlog_2(x)+(q-1)h)$	$LUT(x)+LUT(Nx')+d(q-1)H_k(h)$
Our scheme$_{\text{sto}}$	data owner	0	$ulog_2(N)$	$2H_k(qb)$
	cloud	$N(b_k+b_l)$	$2qh$	$kH_{bf}(qb)+u_irkH_{bf}(b+log_2(r))$

Table 2.5: Comparison of phrase search schemes for a sample of 1500 documents

	Zittrower [79]		Tang [69]		Our scheme$_{\text{sto}}$	
	data owner	cloud	data owner	cloud	data owner	cloud
Storage	1.98MB	392.5MB	1.98MB	242.8GB	0MB	49.47MB
Communication	$16740.5u_iq$ kb	$12q$ bits	$10.6u+1530q$ bits	$(34q-16)u_i$ bits	$10.6u$ bits	$32q$ bits
Computation	$17*10^6u_iDec(q)$	$u_iLUT(3959)$	$u_iH_k(40(2q-1))$	$10^4(q-1)H_k(16)$	$2H_k(5q)$	$6*10^4u_iH_{bf}(55)$

2.6 Fast Phrase Search for Encrypted Cloud Storage

In this section, we present a phrase search scheme [58], which achieves a much faster response time than existing solutions, including those discussed in section 2.4 and 2.5. Instead of dedicating resources for matching keywords to documents and keyword locations, the scheme considers the presence of subsequences as indication of the presence of the sequence they derived from. The scheme is scalable, where documents can easily be removed and added to the corpus. We also describe modifications to the scheme to lower storage cost at a small cost in response time and to defend against cloud providers with statistical knowledge on stored data. The solutions is suitable for applications where a fast response time is valued, such as businesses that provide fast data access to clients and in secure big data applications where data must be processed as they are rapidly generated. For example, a stock exchange that processes private client orders and an intrusion detection software that scans encrypted security logs uploaded to the cloud. Although phrase searches are processed independently using our technique, they are typically a specialized function in a keyword search scheme, where the primary function is to provide conjunctive keyword searches. Therefore, we describe both the basic conjunctive keyword search algorithm and the basic phrase search algorithm in section 2.6.2 along with design techniques in section 2.6.2. Complete Performance analysis of all our phrase search schemes and experimental results are included in section 2.6.6 and 2.6.7.

Note this work has also appeared in:

- H. Poon and A. Miri. Fast phrase search for encrypted cloud storage. *IEEE Transactions on Cloud Computing*, DOI: 10.1109/TCC.2017.2709316, to appear

2.6.1 Conjunctive keyword search protocol

In a keyword search scheme, Bloom filters can be used to test whether a keyword is associated with a document. Many existing phrase search schemes [69, 79] use a keyword-to-document index and a location/chain index to map keywords to documents and match phrases. We describe an alternative approach using Bloom filters to support this functionality with an emphasis on response time. Our scheme can be summarized as the use of multiple n-gram Bloom filters, $B^n_{\mathcal{D}_i}$, to provide conjunctive keyword search and phrase search.

To provide conjunctive keyword search capability, each document, $\mathcal{D}_i$, is parsed for a list of keywords kw_j. A Bloom filter of size m is initialized to zeros. Each keyword is hashed using a secret key to produce $H_{k_c}(kw_j)$ and passed into k Bloom filter hash

functions to set k bits in the Bloom filter. This results in a 1-gram Bloom filter for each document: $B^1_{\mathcal{D}_i} = (b_1, b_2, ...b_m)$ where $b_i \in \{0, 1\}$. The document collection, $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}$, is encrypted and uploaded along with the Bloom filters to the cloud server. The Bloom filters are then organized into a matrix with the first row containing the filter $B^1_{\mathcal{D}_1}$ for the first document and the last row containing $B^1_{\mathcal{D}_N}$. Its transpose is stored as a Bloom filter index I_{BF} where each row corresponds to a bit in the Bloom filters. Note that the i^{th} row in I_{BF} contains information on which document's filter has its i^{th} bit set. This arrangement allows us to quickly identify the documents for a specific query by working only with bits that are set.

To perform a conjunctive keyword search, the user sends a set of keywords $kw' = \{kw_1, kw_2...kw_q\}$ to the data owner. The data owner performs the Bloom filter hash computation to determine the set of bit locations, $Q = \{q_1, q_2, ...q_x\}$, that would be set in the query filter and sends them to the server. The server then computes $T = I_{BF,q_1} \& I_{BF,q_2} ... \& I_{BF,q_x}$, where I_{BF,q_i} is the q_i^{th} row in I_{BF}. The index of bits that are set in T are identified as the matched documents. Note that the size of the set Q is much smaller than m since the query filter contains only a few keywords while a conjunctive keyword Bloom filter contains all the keywords in a document. Therefore, this approach can identify the matched documents much faster, performing fewer operations than individual filter verification.

Note that an entry in the Bloom filter index has as many bits as the number of documents. A query generally involves only a few words and very few bits set. These lead to only a few rows being extracted for matching. Furthermore, when performing the bit-wise AND testing, computer processors would generally test 32 or 64 bits at a time. Should a test results in all zeroes for any subset of bits in a row, the corresponding documents are no longer candidates and the subset of bits no longer require testing in subsequent rows.

2.6.2 Phrase search protocol

To provide phrase search capability, each document is parsed for lists of keyword pairs and triples. For example, 'Happy Day, Happy Night' would yield the pairs, 'Happy Day', 'Day Happy' and 'Happy Night', and the triples, 'Happy Day Happy' and 'Day Happy Night'. A keyed hash for each keyword pair is computed, $H_{k_p}(kw_j|kw_{j+1})$, and passed into k hash functions and the result is used to set k bits in the Bloom filter, $B^2_{D_i}$. Keyword triples are similarly hashed to generate the Bloom filter, $B^3_{D_i}$. The resulting Bloom filters for pairs and triples are organized into matrices with the first rows containing the filters $B^x_{D_1}$ for the first document. The matrices are then transposed to produce the pairs and

triples Bloom filter indexes, I_{BF^2} and I_{BF^3}, which are stored alongside the encrypted documents on the cloud.

To perform a phrase search, the user begins by sending the phrase, $kw' = (kw_1, kw_2, \ldots kw_q)$, to the data owner. The data owner performs the Bloom filter hash computation of the pair, $H_{k_p}(kw_1|kw_2)$, to determine the set bits in the query filter if the phrase contains two keywords. If the phrase contains more than two keywords, the hashes of triples within the phrase, $H_{k_p}(kw_j|kw_{j+1}|kw_{j+2})$ where $j = 1$ to $q-2$, are evaluated instead. The set bit locations are sent to the server, who then computes $T = I_{BF^2,q_1} \& I_{BF^2,q_2} ... \& I_{BF^2,q_x}$, where I_{BF^2,q_i} is the q_i^{th} row in I_{BF^2} if the phrase contains two keywords, and similarly using I_{BF^3} for longer phrases. The set bits in T identify the matched documents. That is, for each set bit index, i, in T, the following is true:

$$\{H_{k_p}(kw_1|kw_2)\} \in B^2_{\mathcal{D}_i} \tag{2.20}$$

for pairs and

$$\{H_{k_p}(kw_j|kw_{j+1}|kw_{j+2})\} \in B^3_{\mathcal{D}_i}, \text{ where } j = 1 \text{ to } q-2, \tag{2.21}$$

for triples.

Our phrase search scheme requires only 2 exchanges: a) The initial message to the cloud server containing the set bit locations of the query Bloom filter T for pairs or triples and b) The query results from the phrase search performed locally by the cloud. Performing the phrase search requires $k(q-2)$ hash computations for phrases of length $q > 2$ and a simple bit-wise AND operations. The protocol is computationally efficient. Its performance is dependent on the length of the phrase and largely independent of the size of the document set. Due to the space efficiency of Bloom filters, our scheme also requires less storage than index based schemes. Since filters are assigned per document, adding or removing documents consists simply of adding or removing the associated filters, providing a scalable solution.

While a document containing a phrase will always be correctly identified as such, our scheme can falsely identify documents as containing a phrase when it doesn't. The source of the false positive is not only the natural property of Bloom filter, but also in how a phrase match is determined. If a user queries n-grams for $n = 2$ or $n = 3$, our scheme has no false positives other than ones arising from the use of Bloom filters. For $n > 3$, however, it is possible that keyword triples within a phrase appear in different parts of a document without the complete phrase being present. Using the previous

example of 'Happy Day Happy Night', a false positive would occur if a document does not contain the phrase but instead contains 'Happy Day Happy Day' and 'Snowy Day Happy Night'. The validity of the scheme is based on low occurrence of such scenarios in practical settings.

Designing for target precision in large corpuses

In information retrieval, precision and recall are often used to measure the performance of a system in its ability to retrieve relevant data.

Recall is defined as the fraction of documents relevant to a query that is retrieved:

$$Recall = \frac{TP}{TP + FN} \tag{2.22}$$

Precision is defined as the fraction of retrieved documents that are relevant to the query:

$$Precision = \frac{TP}{TP + FP} \tag{2.23}$$

Since our scheme has no false negatives, it achieves 100% recall rate. However, precision tends to decrease when querying longer phrases due to a higher number of false positives (FP) relative to true positives (TP). While not ideal, it is unlikely that this would negatively affect the performance of querying corpuses of typical size, due to the uniqueness of long phrases, as will be demonstrated in section 2.6.7.

When working with particularly large corpuses, the number of Bloom filters can be used to tune the precision rate. One may notice that the number of false positives can be reduced by extending the scheme to include a quadruple Bloom filter, $B^4_{\mathcal{D}_i}$, and beyond, at the cost of extra storage. On the other hand, one can also reduce storage by using fewer filters. While it's possible to use only the pairs filter, it's generally preferable to use at least one for pairs and one for triples when processing English documents since 2-grams and 3-grams are fairly common in the language. Our experimental results over a corpus of 1500 documents showed a significant number of false positive when using 2-grams to infer presence of 3-grams while almost no false positive were seen when using 3-grams to infer presence of 4-grams. This motivates our design choice of using two filters: a 2-gram and a 3-gram filter.

Suppose an application have a target precision, $Prec_n$, for n-grams and a corpus with N documents, we can estimate the probability that a file be matched by computing $p_T = 1 - (1 - u_n)^{x'-n+1}$, where x' is the average number of keywords per file and u_n is the probability of two random n-grams being identical. Then, the number of true

positive is approximately $p_T N$ and the target number of false positive is

$$FP_T = \frac{p_T N - Prec_n p_T N}{Prec_n}. \tag{2.24}$$

Assume the probability that a file being retrieved as false positive is p_f, then the probability that no more than FP_T files were found as false positive is

$$p = \sum_{i=1}^{FP_T} \binom{N}{i} p_f^i (1 - p_f)^i. \tag{2.25}$$

Suppose we wish that the probability $p > 90\%$, the above equation can then be solved for a target p_f.

Given the probability, $p_{f(n,m)}$, that an n-gram is a false positive when verified using a m-gram Bloom filter and the number of n-grams in a file, fs, the probability that the file is retrieved as a false positive is $1 - (1 - p_{f(n,m)})^{fs}$. If we desire a false positive probability of no greater than p_f, then we would verify if $1 - (1 - p_{f(n,m)})^{fs} < p_f$. If the inequality holds, the target is achieved otherwise a $m + 1$-gram Bloom filter is added. The process repeats until the inequality holds.

Note that when choosing a target precision, $Prec_n$, it is helpful to consider the expected number of files returned per n-gram query. For example, if a query is expected to return less than 5 results, a precision of 80% would yield only 1 false positive per query on average. Depending on the application, it may be then preferable to target shorter sequences such as $n - 1$ where the higher number of false positive could be problematic.

2.6.3 Modified phrase search scheme against IR attacks

The basic scheme can be adapted to provide additional defense against inclusion-relation attacks where an attacker has access to a significant amount of query Bloom filters and search results associated with known queries. To do so, we modify our algorithm to a type C searching algorithm in the same manner as in section 2.5.4.

In an IR-secure scheme, keywords are randomly removed from the beginning and the end of the query phrase. More terms can be removed from longer phrases to generate false positives. For example, a query phrase, $kw' = (kw_1, kw_2, kw_3)$, would be queried randomly as $kw' = (kw_1, kw_2)$ or $kw' = (kw_2, kw_3)$. This results in an increase in false matches that only contain sub-phrases, severing the inclusion relation between search terms and query results.

2.6.4 Security

At rest, the cloud server contains the encrypted documents, $E_{K_{\mathcal{D}_i}}(\mathcal{D}_i)$, the conjunctive keyword Bloom filter, $B_{\mathcal{D}_i}$, and the n-gram Bloom filters, $B^n_{\mathcal{D}_i}$. The security and privacy of the documents are ensured by the symmetric encryption algorithm. The words added to the conjunctive keyword Bloom filter and the n-grams added to the n-gram Bloom filters are hashed with a secret key to prevent the cloud from learning the keywords contained in the documents.

The situation is more complex during query. In order to achieve high efficiency, the basic scheme uses the same secret key for the Bloom filters of different documents. As a result, it is possible for the cloud to knowingly verify the existence of an encrypted keyword or n-gram in every document in the corpus. Given enough queries, the cloud could build a statistical distribution of encrypted words. If the cloud has any prior knowledge on the statistics of the corpus, such as that the language is English or that it contains legal documents, it may be able to learn partial information on the data. An intuitive defence against this statistical attack would use different secret keys for different documents. However, this would incur significant overhead since filters would have to be computed and verified separately for every document. Instead, we propose a hybrid approach as described in the following section.

2.6.5 A hybrid approach against statistical attacks

In a typical keyword search scheme, the majority of queries consist of conjunctive keyword searches. Being a specialized search option, phrase searches occur far less frequently. Therefore, the availability of statistical information for individual keywords would be far greater than that for n-grams. To defend against statistical attacks, the more secure, albeit more expensive, approach of encrypted indexing is used for conjunctive keyword matching, where the statistics of individual keywords are better protected. The approach, described in section 2.4, provides security for individual keywords at the cost of having to perform client-side encryption/decryption and to re-encrypt the index when adding files. The use of n-gram Bloom filters for phrase search is retained. In addition to the low availability of statistical information due to the infrequent occurrence of phrase searches, the number of distinct n-grams is also far greater than the number of distinct keywords [14], resulting in a distribution that shows individual probability of occurrence several orders lower than that of keywords [40]. This means it is significantly more difficult to mount a statistical attack against n-grams because far more data is required to recognize the rare occurrences of n-grams while, at the same time, far fewer data is

Table 2.6: Average number of distinct n-grams for a sample of 150 documents

Number of words	$n = 1$	$n = 2$	$n = 3$
37626	4833	32023	36884

available. Table 2.6 illustrates this property on our experimental data set. Regarding updates to the corpus, it should be noted that index update can be delayed to avoid constantly decrypting and re-encrypting the index. That is, the data owner can maintain a small local index which includes recently added and removed files until the next scheduled index update.

The encrypted index approach to conjunctive keyword search proceeds as follows. A document collection, $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$, is parsed for a list of keywords, kw_j. An keyword-to-document index, I, is generated mapping keywords to documents such that $I(kw_j) = (d_1, d_2, \ldots, d_n)$, where $d_i = 1$ if kw_j is linked to the document and $d_i = 0$ otherwise. The resulting index is encrypted and uploaded to the cloud server:

$$I(H_K(kw_j)) = E_K(d_1, d_2, \ldots, d_n). \tag{2.26}$$

To perform a search, the user sends a set of keywords $kw' = \{kw_1, kw_2, \ldots, kw_q\}$ to the data owner. The data owner computes their hashes, $H_K(kw')$, using a secret key and sends them to the cloud server. The encrypted index entries are returned to the data owner, who computes the intersection of index entries and identifies the matching documents:

$$D_K(I(H_K(kw_1))) \,\&\, D_K(I(H_K(kw_2))) \cdots \,\&\, D_K(I(H_K(kw_q))), \tag{2.27}$$

where & is a bitwise AND operation. Note that the phrase search protocol, which runs independently, in the hybrid construction is identical to that described in section 2.6.2. Therefore, the response time, communication cost and computational cost associated with phrase search are also identical.

2.6.6 Performance Analysis

As outlined in section 2.6.2, our scheme, denoted our $\text{scheme}_{\text{spd}}$, requires two Bloom filters per document for the purpose of phrase search, one for storing pairs and another for triples. Note that the conjunctive keyword Bloom filters are not required for the purpose of phrase search. The two sets of filters require $N(b_2 + b_3)$ bits of storage on

the cloud server, where b_2 is the size of a pairs Bloom filter, b_3 is the size of a triples Bloom filter and N is the number of documents in the corpus. The data owner needs only to store cryptographic keys. Since all existing schemes include conjunctive keyword search capability, we have included the 1-gram filter, b_1, in our comparison tables 2.8 and 2.9. The communication cost of our scheme is similar to [54], descibed in section 2.5. The proposed protocol requires only two messages to be sent, one containing the set bit locations of the query Bloom filter for pairs or triples and the other containing the matched results. Assuming the queried phrase contain $q > 2$ keywords and that k is small, the scheme requires $(q-2)klog_2(b_3)$ bits to be sent to the cloud server and $ulog_2(N)$ bits to be sent to the user, where u is the number of matched documents. In terms of computation, the scheme requires hashing of triples in the phrase using a secret key, with a total of $3(q-2)b$ bits, where b is the average number of bits per keyword, and $k(q-2)$ standard hash computations on the client side to determine the set bit locations of the query filter sent to the server. Upon receiving the query, the server performs a bitwise AND operation between the Bloom filter index entries, each consisting of N bits, for the $k(q-2)$ set bit location. The matches are then immediately available from the result. The response time of the scheme is dependent on the execution time of the server and the client, and also the transmission time and propagation delay of the messages. The most expensive computation in the protocol is the $3(q-2)b$ keyed hashing performed by the client while non-cryptographic hashing and bit-wise AND are both very efficient operations. In addition, the scheme also enjoys a short transmission time due to a compact description of the query filter and requires the minimal propagation delay of a single round-trip communication.

Zittrower's proposal [79] uses an encrypted keyword truncation table that maps distinct keywords with the same truncated ciphertext to different index values. The table allows the data owner to differentiate between entries associated with different keywords but have the same truncated ciphertext in the index table stored on the cloud. Assuming optimal representations, this table requires $x(log_2(x)+b)$ bits, where x is the number of distinct keywords in the corpus and b is the average number of bits per keyword. On the cloud server, two index tables, one mapping truncated encrypted keywords to documents and another to their locations within the documents, are stored. The two tables require $x(12+p'\ log_2(N))$ and $x'(12+gy)$ bits respectively, where p is the average number of documents associated with a keyword in the corpus and y is the number of bits needed to store a location value. On average, The scheme results in 300 collisions among the encrypted keywords to hide the true search terms. In other words, a query response contains, on average, results belonging to 300 other unrelated keywords, leading to a

significant amount of wasted bandwidth and processing. In terms of communication, this implies that up to $300u_i\ log_2(N)$ bits are spent transferring unrelated data during conjunctive keyword search, where u_i is the number of candidate documents matched in the search, and $300g(y + log_2(x)) + salt)$ bits are wasted during phrase search for every keyword in the query. When combined with the desired query results, a $12q$ bits query would yield a response with up to $301u_i(log_2(N) + 301q(g(y + log_2(x)) + salt))$ bits of data from the cloud server. Regarding computational cost, the data owner must decrypt all returned entries and, using the encrypted keyword truncation table, identify the results belonging to the searched keywords while discarding collisions. Then, based on the decrypted locations, the data owner identifies documents where the keywords appear in order. Due to the need for client-side encryption and the large amount of false positives processed during search, the scheme has a higher computational requirement than our proposed scheme, where the most expensive operation consists of hash computations. Although the protocol results in only a single round-trip delay, it has a long transmission time due to the high number of false positives. The response time also degrades due to a high processing time because of a significant amount of decryption operations required at the client which likely possesses much lower computational power than the cloud server.

Tang's scheme [69] also uses a table mapping keywords to index values kept locally by the data owner. On the cloud server, a keyword-to-document index and a keyword chain table are stored. While the keyword-to-document index is similar to other existing solutions, the keyword chain table is a structure that allows the verification of keyword chains, combining the use of cryptographic hash and randomly generated location indicators. In order to achieve security across all parameters, it was proposed that the keyword chain table be applied to the entire data set and normalized according to the highest occurring keyword in the document set. In all, $x(log_2(x) + b)$ bits of storage is required by the data owner and $x(log_2(x) + N) + Nx'(h + d(h + y))$ bits of storage is required by the cloud, where b is the average number of bits per keyword, h is the number of bits used to store a hashed keyword or location value, y is the number of bits to store a location value and d is the number of instance of the most frequent keyword in the corpus. While the client-side storage requirement is low, the keyword chain table, analogous to the location index, is several orders of magnitude greater, due to the normalization to the value of d. In terms of communication, the client sends $qlog_2(x)$ bits to the server during the conjunctive keyword search phase and receives qN bits as results. For matching phrases using the keyword chain table, the data owner sends $u_i(qlog_2(x) + (q - 1)h)$ bits to the cloud server, where u_i is the number of candidate

documents, and receives $ulog_2(N)$ bits in matching document ID's. While the communication cost of the scheme is an improvement over Zittrower's scheme, communicating index entries and keys is still more costly than the transmission of a single Bloom filter required in our proposed technique. One of the main advantages of Tang's approach is the asymmetric distribution of computational load, where the cloud server, typically possessing significant computational power, performs the majority of the computations. During the conjunctive keyword search phase, the client performs a table lookup and computes the keyed-hash of the queried keywords and sends them to the cloud server. The server then performs a table lookup for the queried entries and return the results to the data owner. Given the candidate documents, the client hashes the keywords under a different secret key in addition to $q-1$ chain keys for the chain digests. Upon receiving the hashed phrase and chain keys, the server finds the corresponding entries in the index using binary search, and performs up to $d(q-1)$ keyed hashes. Due to the size of the keyword chain table, a significant amount of hash computation is required to verify a phrase, especially when a candidate document contains all the keywords but not in consecutive order, where the maximum number of hashes would then be required to reject it as a match. The processing time of the scheme is largely dependent on the server's ability to compute the keyed hashes. The two phase protocol also requires two round-trip delays. Compared to our proposed technique, the scheme has a higher propagation delay and a higher processing time since the hashing algorithm used in Bloom filters is non-cryptographic, which is significantly faster than the cryptographic hash functions used in Tang's scheme.

In section 2.4, we proposed a solution to address the high communication cost in Zittrower's scheme and the high storage cost noted in Tang's scheme while maintaining a high level of security. We'll designate this scheme as our $\text{scheme}_{\text{sec}}$. The central idea was to exploit properties of natural languages to better design the indexes. By considering the almost exponential distribution of words in most languages, it was shown that splitting keywords location entries into pairs dramatically reduces the storage cost of the system. The scheme requires data owners to maintain a dictionary mapping keywords to index values and a list showing the number of times that keywords were split in each file. It was found that the data owner would require $x(log_2(x)+b)$ bits for the conjunctive keyword index and $0.27x'N(log_2(x')+log_2(k'/2))$ bits of storage for the split tables. The cloud server would store the encrypted keyword-to-document index using $x(log_2(x)+N)$ bits and the location index tables using $2.5x'N(h+2y)$ bits, where h is the number of bits used to represent a hashed keyword and y is the number of bits to store a location value. During the first phase of a phrase query, $qlog_2(x)$ bits would be

sent to the cloud to identify candidate document and qN bits would be received by the data owner. On average, a query for a single keyword's locations requires 2.5 encrypted keywords to be sent due to splitting. A location query for a random keyword in phrase would then require sending $2.5h$ bits to the server. Given u_i candidates identified in the conjunctive keyword search, $2.5hu_i$ bits would need to be sent. In response, the cloud returns the encrypted location entries requiring $2.5u_i * 2y = 5u_iy$ bits. Hash signatures of the phrase at the identified starting locations are then sent to cloud for matching, requiring $u_ig(h + log_2(N) + y)$ bits, where g is the number of times a keyword appears on average per document. In terms of computation, the cloud server must look up the index entries on each step and perform hash computations of qb bits at u_ig locations. The data owner must encrypt q keywords for conjunctive keyword search and hash one random keyword for location query. Then, u_ig encryption and hash computation of q keywords are required to generate the hash signatures for matching. The scheme presents a significant improvement in terms of practicality over Zittrower and Tang's scheme by offering a much lower storage cost at the cloud and not having to rely on a large number of false positives to maintain security. Phrase search is performed in two rounds of communication, as in Tang's scheme, with the first step identifying candidate documents that contain all keywords.

In section 2.5, we noted that further reduction in storage can be achieved and proposed a scheme based on Bloom filters that focused on minimizing storage cost. We'll designate this solution as our $\text{scheme}_{\text{sto}}$. For conjunctive keyword search, all distinct keywords in a document is placed in a conjunctive keyword filter to enable keyword-to-document search. For phrase search, keywords are concatenated with their locations and placed inside a keyword location filter to enable location queries. The scheme requires two Bloom filters per document, one for mapping keywords to document and one for determining keyword location. The filters are stored on the cloud server, requiring $N(b_k + b_l)$ bits of storage, where b_k is the size of a conjunctive keyword Bloom filter, b_l is the size of a keyword location Bloom filter and N is the number of documents in the corpus. The data owner retains only cryptographic keys. In terms of communication cost, the protocol requires that the keyed hash of the keywords for each filter be sent to the cloud server, and the results of the query returned to the data owner. Altogether, the scheme requires $2qh$ bits to be sent to the cloud server and $ulog_2(N)$ bits to be sent to the user, where h is the number of bits per hashed keyword and u is the number of matched documents. Regarding computational cost, the data owner must perform $2q$ keyed hash computations to generate the trapdoor query. Upon receiving the query, the server performs qk Bloom filter hash computations to produce the query

filters. A bitwise AND operation for each conjunctive keyword filter in the document set identifies the candidate documents. Then, for each candidate document, rk Bloom filter hash computation is needed to find the locations of the first word of the phrase, followed by $r_i k$ hash computation for each additional word to determine matches, where r is the number of keywords in the candidate document and r_i is the number of matches for the i^{th} word in the phrase. Generally, $r \gg r_i$. Therefore, approximately $u_i rk$ hash computations are required during phrase search, where u_i is the number of matched documents during conjunctive keyword search. The space-efficiency of Bloom filters allowed the scheme to achieve the lowest storage cost among existing solutions but it required a brute-force approach to identify keyword locations. Although incremental hash functions can improve the verification speed, the computational cost remains high and increases proportionally to the size of the documents. While the scheme requires a single round of communication, the response time of the scheme suffers due to the high processing time required to identify keyword locations.

Table 2.8 shows a comparison of all our schemes to existing techniques. For clarity, some terms that do not have a significant impact on the associated cost were omitted. Note that the variants, our $\text{scheme}_{\text{spd}}$ (speed) and our $\text{scheme}_{\text{spd}}$ (storage) are defined in section 2.6.7 and our $\text{scheme}_{\text{spd}}$ (hybrid/*) corresponds to the hybrid approach described in section 2.6.5. While our $\text{scheme}_{\text{spd}}$ (speed) exhibits a high storage cost in order to achieve the fastest processing time by taking full advantage of the Bloom filter index, variants with different values of t, such as our $\text{scheme}_{\text{spd}}$ (storage), can achieve fast processing time with a storage cost similar to our $\text{scheme}_{\text{sec}}$ and 3 times lower than Zittrower's scheme. Regarding the hybrid approach, the technique differs from the scheme in section 2.6.2 only in its conjunctive keyword search functionality. Its phrase search functionality remains the same. Therefore, its response time, communication cost and computational cost associated with phrase search are identical to our base scheme. The sole difference is the storage cost of the system where resource dedicated to conjunctive keyword search, namely the index, requires higher storage than a 1-gram Bloom filter.

2.6.7 Experimental Results

To compare our results against existing phrase search schemes, we evaluate our algorithm on a corpus consisting of 1500 documents made available by Project Gutenberg [2]. The documents were preprocessed to exclude headers and footers, which include copyright, contact and source information to reduce skewing in the statistics of the data set. Stop words are also omitted. To determine the statistical properties of the corpus and the performance for the various schemes, the Natural Language Toolkit [12] was used.

The design of Bloom filters is important to our scheme's performance. In particular, the use of a Bloom filter index requires the filters to be of the same length. Recall that the equation for false positive rate is as follows:

$$p = (1 - e^{-k\frac{n}{m}})^k \tag{2.28}$$

where k hash functions are used to insert n items into a m bits Bloom filter with a false positive rate of p. Figure 2.16 shows false positive rates between 1% and 10% relative to the number of hash functions and the number of bits needed per entry. A small filter size is preferable in terms of storage. A low false positive rate would reduce communication and computational cost. In particular, using a small number of hash functions greatly improves the execution time since the computational cost is proportional to the number of hash function used. In practice, the number of hash functions, k, needed to minimize false positive rate is rarely used since there is very little improvement in false positive rate as we increase the number of hash functions past a certain threshold. Using a single hash function, $k = 1$, would reduce the computational cost, but also more than doubles the storage cost to achieve the same false positive rate. The high variance in false positive rate when $k = 1$ can also be problematic for corpus with high variance in document sizes. As shown in the Figure 2.17, the number of bits per entry and the false positive rate is fairly stable for $k \geq 2$ and $m/n \geq 10$.

Selecting the Filter Size

Optimizing for response time requires normalizing the filter size to take advantage of the Bloom filter index. A simple design approach is to ensure that the parameters meet the requirements in the worst case scenario. If an application requires a certain false positive rate, then the filter size can be chosen such that the document with the largest number of distinct keywords or n-grams in the corpus falls within the required rate. In most practical scenarios, it would be the largest document in the corpus. All other smaller documents would exhibit lower than required false positive rate. However, this approach has a high cost in storage. In corpuses where there's a large variance in document sizes, much of the storage is wasted. For example, if we consider the entire Gutenberg corpus of 15620 English documents, the largest document contains 2.8 million words. However, only 88 documents contain more than 140 thousand words and half of the documents in corpus contain less than 20 thousand words. In such scenarios, a trade-off between response time and storage cost can be made by using t sets of filter sizes where only one of the document set would conform to the largest filter size used, effectively

generating t Bloom filter indexes. In the previous example, we can have the largest 88 documents conform to the largest filter size, the following 7722 documents conform to the document containing 140 thousand words and the remaining 7800 documents conform to the document containing 20 thousand words. This simple change would lead to 30 times lower storage requirement than the fastest solution. This approach requires a slight modification of the protocol in section 2.6.2, where the output of the k hash functions would be sent to the server instead of the set bit locations. To determine the set bit locations, the server simply computes the hash values modulo the filter sizes. The server then proceeds to compute T as usual for each set of filters to determine matches. Note that the limit case of setting $t = N$, where every document uses exactly the filter size needed to achieve the desired false positive rate, would require the least amount of storage, but would also require the server to verify each filter separately. Nonetheless, it would still achieve much improved response time when compared to existing approaches. We will refer to the approach with $t = 1$ as "Our $\text{scheme}_{\text{spd}}$ (speed)" and $t = N$ as "Our $\text{scheme}_{\text{spd}}$ (storage)" in tables 2.8 and 2.9 to highlight that the former is designed for the highest speed and the latter is designed for the lowest storage cost possible with a trade-off in speed. It should be noted that the best value for t depends on the application and especially on the distribution of file sizes. If all documents have the same number of distinct keywords or n-grams, there is no advantage in storage for choosing $t > 1$ as all filter sizes would be the same.

Having considered various trade-offs, the operating point was heuristically chosen for a false positive rate of $p = 5\%$ with $k = 2$ and $m/n = 7.9$ for our experimental corpus. A more stringent false positive rate of 1% can be achieved by increasing the number of bits per entry to $m/n = 19$ while keeping the number of hash functions at $k = 2$, maintaining a high response time at the cost of more storage. Although we could improve the false positive rate by using more hash functions, increasing the number of hash function from $k = 2$ to $k = 3$ would increase our computational cost by 50%, leading to lower performance.

The storage requirement is dependent on the number of items that the filters must store. Using the simple approach of normalizing according to the largest document, we would require the number of bits needed to store the number of distinct keywords in the largest document for each conjunctive keyword filter. Similarly, the number of distinct pairs and triples determine the size of each pairs or triples filter. The values of various parameters of the sample Gutenberg document set are listed in Table 2.7. To provide consistent comparison with existing schemes, we consider 1530 documents from the Gutenberg corpus. At 7.9 bits per entry and optimized for speed, a conjunctive

keyword Bloom filter would require $b_1 = x''m/n = 41$kB, a pairs filter would require $b_2 = r_2m/n = 2.17$mB and a triples filter would require $b_3 = r_3m/n = 2.66$mB. Should we wish to minimize storage, we would require $b_1 = x''m/n = 3.82$kB, a pairs filter would require $b_2 = r_2m/n = 23.43$kB and a triples filter would require $b_3 = r_3m/n = 28.7$kB.

Usually, the number of distinct words/pairs/triplets is proportional to the document size. However, in the event of outliers, this could allow an attacker to easily recognize the non-conforming documents. To defend against this, one can design a minimum filter size to file size ratio so that a file with an unusually small number of distinct words would have a filter size larger than what is needed to protect the file's identity. Equivalently, a minimum file size can also be set so that a file requiring an unusually large filter size would be padded to have a correspondingly large file size associated with it.

Long phrases

Long phrase queries are often used to locate known items rather than to locate resources for a general topic. In many cases, the user's goal is to retrieve a single document. Longer phrases also have a very low probability of occurrence and yield fewer matches. Therefore, even with a precision rate of 50%, we would rarely see more than a single false positive for a search query of longer phrases. In our experiment, we never encountered more than a single false positive in queries with phrases containing more than 4 keywords. The small amount of false positives can also be easily identified and removed client-side. As a result, the effect of low precision rate in longer phrases should not have a noticeable detrimental effect in practice.

Table 2.9 summarizes the results of the schemes on the sample Gutenberg document set with the experimental values for the various parameters outlined in Table 2.7. For the hash values of keywords and locations, we assumed that each would require 16 bits in all cases. English words have an average length of 5 letters. Hence, b is set to 40. Since communication and computational costs are query-dependent, related parameters are kept in the formulas and dominant terms were retained for clearer comparisons. In practical scenarios, $u_i > u > q$ and $Enc(x) \approx Dec(x) > H_k(x) > H_{bf}(x) \approx H(x) > LUT(x) \approx Mod(x) > And(x)$, where $Enc(x)$, $Dec(x)$, $H_k(x)$, $H_{bf}(x)$, $H(x)$, $LUT(x)$, $Mod(x)$ and $And(x)$ represent respectively the cost of an encryption of x bits, a decryption of x bits, a keyed hash computation of x bits, a Bloom filter hash computation of x bits, a standard hash computation of x bits, a table look up of x elements, the modulus computation of x numerical values and bit-wise AND operation of x bits . Note that the communication and computational cost values for Zittrower's scheme are worst-case estimates. As shown in the table, our schemes$_{\text{spd}}$ require far lower computational cost

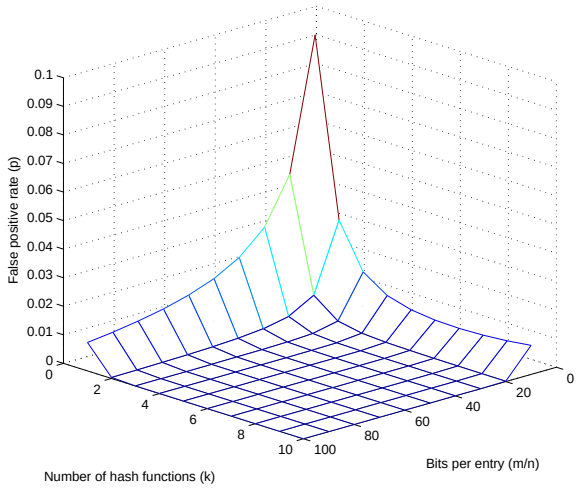

Figure 2.16: False positive rate (p) as a function of the number of hash function (k) and bits per entry (m/n)

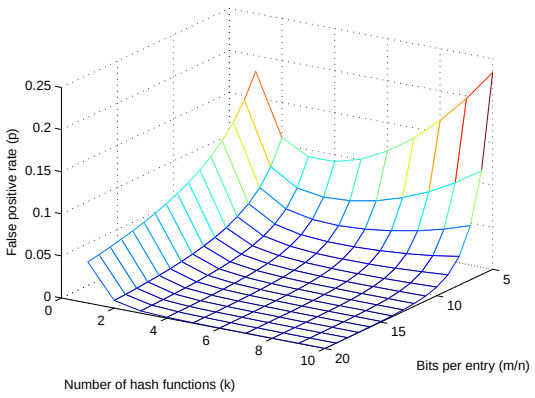

Figure 2.17: False positive rate (p) as a function of the number of hash function (k) and bits per entry (m/n) (close up)

Table 2.7: Properties of the sample document set

Average number of documents associated with a keyword, p'	885.6
Total number of documents, N	1530
Total distinct keywords, x	285396
Average number of keywords per document, r	30364.7
Maximum number of keywords per document, r'	2815415
Average number of distinct keywords per document, x'	3959.6
Maximum number of distinct keywords per document, x''	42489
Average number of distinct pairs per document, $r2$	25177.6
Maximum number of distinct pairs per document, $r2'$	2252332
Average number of distinct triples per document, $r3$	30842.5
Maximum number of distinct triples per document, $r3'$	2759106
Average number of times each keyword appears per document, g	5.6
Number of instance of the most frequent keyword, d	10757
Average number of instance of most frequent word per document, k'	369.1

on both the data owner and the cloud server. The advantage is particularly notable on the cloud where only basic operations are needed. Our $\text{scheme}_{\text{spd}}$ (storage) also achieves almost 5 times lower storage cost than Zittrower's scheme. In terms of communication, our $\text{schemes}_{\text{spd}}$ require only a single round trip much like Tang's solution, but requires fewer bits to be sent by either party. It is also interesting to note that the bulk of the communication/computation cost is not dependent on the number of matches for the keywords.

Table 2.8: Comparison of all phrase search schemes

		Storage	Communication (download)	Computation
Zittrower [79]	data owner	$x(log_2(x)+b)$	$301u_i(log_2(N)+301q(g(y+log_2(x))+salt))$	$301u_iDec(301q(g(y+log_2(x))+salt))+LUT(x)$
	cloud	$x(p'log_2(N)+12)+N\,x'(12+gy)$	$12q$	$LUT(x-x/2^{12})+u_iLUT(x'-x'/2^{12})$
Tang [69]	data owner	$x(log_2(x)+b)$	$qN+ulog_2(N)$	$LUT(x)+H_k(qb)+u_iH_k(qb)+u_iH_k(b(q-1))$
	cloud	$x(log_2(x)+N)+N\,x'(h+d(h+y))$	$qlog_2(x)+u_i(qlog_2(x)+(q-1)h)$	$LUT(x)+LUT(Nx')+d(q-1)H_k(h)$
Our scheme$_{sec}$	data owner	$x(log_2(x)+b)+0.27x'N(log_2(x')+log_2(k'/2))$	$qN+5u_iy$	$qDec(p'log_2(N))+u_i(2.5(H_k(log_2(N)+b)+Dec(gy))+g(Enc(qb)+H(qb+log_2(N)+y)))$
	cloud	$x(log_2(x)+N)+2.5x'N(h+2y)$	$qlog_2(x)+u_i(2.5h+g(h+log_2(N)+y))$	$LUT(x)+u_iLUT(2.5x')+u_igH(qb)$
Our scheme$_{sto}$	data owner	0	$ulog_2(N)$	$2H_k(qb)$
	cloud	$N(b_k+b_l)$	$2qh$	$kH_{bf}(qb)+u_irkH_{bf}(b+log_2(r))$
Our scheme$_{spd}$ (speed)	data owner	0	$ulog_2(N)$	$H_k(3(q-2)b)+k(q-2)H_{bf}(16)$
	cloud	$N(b_1+b_2+b_3)$	$(q-2)klog_2(b_3)$	$And(k(q-2)N)$
Our scheme$_{spd}$ (storage)	data owner	0	$ulog_2(N)$	$H_k(3(q-2)b)+k(q-2)H_{bf}(16)$
	cloud	$N(b_1+b_2+b_3)$	$(q-2)klog_2(b_3)$	$Mod(k(q-2)N)+And(Nb_3)$
Our scheme$_{spd}$ (hybrid/speed)	data owner	$x(log_2(x)+b)$	$ulog_2(N)$	$H_k(3(q-2)b)+k(q-2)H_{bf}(16)$
	cloud	$x(log_2(x)+N)+N(b_2+b_3)$	$(q-2)klog_2(b_3)$	$And(k(q-2)N)$
Our scheme$_{spd}$ (hybrid/storage)	data owner	$x(log_2(x)+b)$	$ulog_2(N)$	$H_k(3(q-2)b)+k(q-2)H_{bf}(16)$
	cloud	$x(log_2(x)+N)+N(b_2+b_3)$	$(q-2)klog_2(b_3)$	$Mod(k(q-2)N)+And(Nb_3)$

Table 2.9: Comparison of all phrase search schemes for a sample of 1500 documents

	Zittrower [79]		Tang [69]		Our scheme$_{sec}$ [53]		Our scheme$_{sto}$ [54]	
	data owner	cloud	data owner	cloud	data owner	cloud	data owner	cloud
Storage	1.98MB	392.5MB	1.98MB	242.8GB	5.78MB	139.3MB	0MB	49.47MB
Communication	$16740.5u_iq$ kb	$12q$ bits	$10.6u+1530q$ bits	$(34q-16)u_i$ bits	$1530q+80ui$ bits	$18q+ui(34q-16)$ bits	$10.6u$ bits	$32q$ bits
Computation	$17*10^6u_iDec(q)$	$u_iLUT(3959)$	$u_iH_k(40(2q-1))$	$10^4(q-1)H_k(16)$	$5.6u_iEnc(40q)+Dec(9369q)$	$5.6u_iH(40q)$	$2H_k(40q)$	$6*10^4u_iH_{bf}(55)$
	Our scheme$_{spd}$ (speed)		Our scheme$_{spd}$ (storage)		Our scheme$_{spd}$ (hybrid/speed)		Our scheme$_{spd}$ (hybrid/storage)	
	data owner	cloud	data owner	cloud	data owner	cloud	data owner	cloud
Storage	0MB	7.28GB	0MB	85.6MB	1.98MB	7.27GB	1.98MB	130.55MB
Communication	$10.6u$ bits	$44(q-2)$ bits	$10.6u$ bits	$30(q-2)$ bits	$10.6u$ bits	$44(q-2)$ bits	$10.6u$ bits	$30(q-2)$ bits
Computation	$H_k(120(q-2))$	$And(3060(q-2)))$	$H_k(120(q-2))$	$Mod(3060(q-2))$	$H_k(120(q-2))$	$And(3060(q-2)))$	$H_k(120(q-2))$	$Mod(3060(q-2))$

2.7 A Combined Solution for Search and Auditing for Encrypted Cloud Storage

In this section, we propose a solution for storing, auditing and searching through encrypted document sets on cloud storage [55]. Instead of including resources that enable the functionalities separately, our setup uses the same pool of resources, leading to a smaller overhead than simply using two separate solutions. Our solution maintains many desirable features such as privacy of documents and search terms, proof of retrievability with theoretically unbounded number of audits and public verifiability. It is applicable to any sequential data search schemes with location query capability and provides protection against data tempering and ensure data integrity.

Note this work has also appeared in:

- H. Poon and A. Miri. A combined solution for conjunctive keyword search, phrase search and auditing for encrypted cloud storage. In *IEEE Conference on Advanced and Trusted Computing*, pages 938–941, 2016

2.7.1 Auditing cloud services

It's generally agreed that auditing is needed for cloud storage services, in particular where they are used as data archives and backups, where they may not be accessed for long periods of time. Since the data is not in the owner's possession, verifying that the data is indeed available is not a simple task. Where privacy and security is concerned, the cloud service provider should also not learn the content of the encrypted data as a result of the auditing process. Ateniese *et al.* were among the first researchers to consider the problem of auditing encrypted cloud storage. They defined the notion of provable data possession (PDP) and proposed various solutions over the past decade [7, 8]. Their solution generally involves the addition of homomorphic tags, used for data verification. One of the common shortfalls of existing schemes such as in [8] is that there's often a limit on the number of audits that can be performed before the data have to be reprocessed. Shacham *et al.* [64] argued that the previous definition is also insufficient, in that it allows the verification of a subset of blocks without guaranteeing the availability of the data as a whole, and proposed the use of a stronger proof of retrievability (POR). Researchers have, over the years, devised various POR schemes, generally requiring the use of an erasure code to achieve the retrievability criterion without verifying all stored data. The resulting data expansion from the use of erasure code is the cost of achieving this stronger guarantee [65]. Wang *et al.* [72] investigated the possibility of a public

auditing service, where a third party may perform the auditing task for the data owner. The challenge, however, is that the auditing service provider should not be considered any more trusted than the cloud service provider. As such, the auditing service provider must perform auditing without knowledge of the data content. To this end, Wang *et al.* [72] proposed a scheme using public key encryption that could meet these requirements. Another recent work noted the vulnerability of verifying data ownership by knowledge of its signature that was identified in many cloud service at the time [31] and proposed two-way auditing where the user must also prove to the cloud its knowledge and, thus, ownership of the data.

2.7.2 Communication Model

For keyword search, our communication model involves up to three parties: The data owner, the cloud server and the user. Our discussions will assume the public cloud scenario involving all three parties. The algorithms can easily be adapted to the private cloud scenario where the user is simply the data owner. A standard keyword search protocol is shown in Figure 2.18. The user begins by sending a search request containing the queried keywords to the data owner. To prevent the cloud from learning the keywords, the data owner computes and sends a trapdoor to the cloud to initiate a protocol to search for the requested keywords in the corpus. Finally, the cloud responds to the user with the indexes to the requested documents.

For auditing, the private model on which our solution is based on is shown in Figure 2.19. The data owner begins by selecting and querying a set of keywords or rows from the indexes. Upon retrieving the results, the data owner randomly selects a number of bits to audit. The cloud must respond by providing the hash of the bit stream queried. Our solution can also be used in a public auditing model, shown in Figure 2.20, where a third party verifies data integrity in data owner's stead. In the public model, a set of pre-computed challenge and signature response must be sent to the auditor, who then independently engages with the cloud server in an auditing protocol.

In each scenario, the cloud operator is assumed to be semi-honest, following our protocol without deviation, but is interested in learning about stored user data. The auditor is also assumed to be semi-honest in the public auditing model.

2.7.3 Keyword and Phrase search schemes

Our combined audit and search solution works with any keyword and phrase search scheme that has the ability to query a keyword's location within a document and uses a

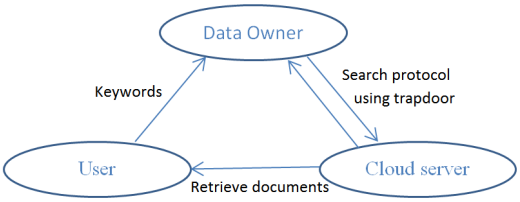

Figure 2.18: Communication model for keyword search over encrypted data

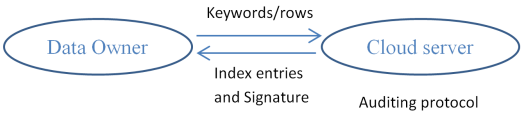

Figure 2.19: Communication model for private auditing over encrypted data

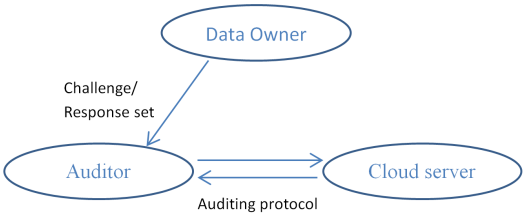

Figure 2.20: Communication model for public auditing over encrypted data

symmetric encryption algorithm in counter mode. Examples include our $\text{scheme}_{\text{sec}}$ [53], our $\text{scheme}_{\text{sto}}$ [54] and Zittrower's solution [79]. Without loss of generality, we'll describe a basic phrase search scheme here.

Given a document set, $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$, each document, $\mathcal{D}_i$, is parsed for a list of keywords, kw_j. An index, I, is generated mapping keywords to documents such that $I(kw_j) = (d_{1,j}, d_{2,j}, \ldots, d_{n,j})$, where $d_i = 1$ if kw_j is found in the document. The index is then encrypted:

$$I(H_K(kw_j)) = E_K(d_{1,j}, d_{2,j}, \ldots, d_{n,j}). \tag{2.29}$$

where $H_K()$ is a cryptographically secure hash function. For each document, $\mathcal{D}_i$, a keyword location index, I_l, is also generated containing entries:

$$\{H_K(\mathcal{D}_i|kw_i), E_K(j_1, j_2, \ldots, j_{n'})\} \tag{2.30}$$

where j_x are the byte locations of kw_i within $\mathcal{D}_i$. The documents are then encrypted using any symmetric encryption, such as AES, under counter mode and uploaded along with the encrypted indexes to the cloud server. The initialization vector is set to the hash of the document index, $H(\mathcal{D}_i)$

To perform a standard keyword search, the user sends the queried keywords $kw' = \{kw_1, kw_2, \ldots, kw_q\}$ to the data owner. The data owner computes $H_K(kw')$ and sends to the cloud server. The cloud server returns the encrypted index entries to the data owner, who then finds the documents matching the keywords from the intersection of index entries:

$$D_K(I(H_K(kw_1))) \,\&\, D_K(I(H_K(kw_2))) \cdots \,\&\, D_K(I(H_K(kw_q))), \tag{2.31}$$

where & denotes a bitwise *and* operation.

To perform a phrase search, the user begins with a standard keyword search to identify candidate documents. Then, the locations of the keywords in the candidate documents are queried by sending $H_K(\mathcal{D}_i|kw_i)$ to the cloud. Upon decrypting the location information, the location of the first keyword in the phrase, kw_1, is extracted. For each location, $Loc(kw_1) = \{j_1, j_2 \ldots, j_m\}$, the data owner verifies if the following keyword is kw_2. For every positive match, the process repeats for the following keyword until the last keyword in the phrase is verified or until no candidates remain.

2.7.4 Auditing protocol

To enable auditing, we add a row-based query mechanism to the basic phrase scheme described in previous section so that each row of the index, I_l, corresponding to a keyword, also includes $E_K(row_{no}|kw)$, where row_{no} is the row number of the keyword, kw, and, optionally, the row number can also be added to the keyword locations as follows: $E_K(row_{no}|j_1|j_2\ldots)$. The latter provides earlier fault detection should a malicious cloud returns a different set of entries than asked. The data owner must also maintain a record of the number of rows available for each file's index. An audit query proceeds as follows: The auditor performs a keyword location or phrase search for a set of keywords, $kw_a = \{kw_{a1}, kw_{a2}, kw_{a3}, \ldots kw_{an}\}$, or randomly chooses a set of documents and performs a row-based query by sending a set of row numbers $row_a = \{r_{a1}, r_{a2}, r_{a3} \ldots, r_{an}\}$. The former has the advantage that the first step is simply a conjunctive keyword search and can be gathered based on user queries over time, without the added complexity of a row-based query mechanism. The latter has the advantage of not requiring knowledge over what keywords may be present in the document set and increasing the randomness of audits. In both cases, the server would return the encrypted locations for the set of keywords or the rows requested. Upon receiving an entry, $\{E_K(row_{no}|kw), E_K(j_1, j_2, \ldots, j_{n'})\}$, the first term is decrypted to verify that the row number or the keyword matches the one being queried. If there are discrepancies, the audit fails. Otherwise, the data owner is now aware of the locations of the keywords and can verify them by specifying a bit sequence belonging to a subset of the locations queried and requesting the cloud server to compute a hash signature of the bit sequence.

While it is possible to audit any bit sequence using this approach, it is useful to have a simple description of the locations to avoid having to specify the locations of every bit under audit. To this end, we chose to fix the number of bits to audit per keyword location. The following audit message is sent to the cloud server, describing the bits under audit: $\{run\_size, \{\mathcal{D}_{ID_1}, byte_{loc_1}, byte_{loc_2}, \ldots byte_{loc_{n''}}\}, \ldots \{\mathcal{D}_{ID_n}, byte_{loc_1}, byte_{loc_2}, \ldots byte_{loc_{n''}}\}\}$. Upon receipt, the server locates the document, $\mathcal{D}_{ID_1}$, and retrieves run_size bytes at each byte location, $byte_{loc_i}$ for $i = 1$ to n'', and place them in a buffer in order. Document $\mathcal{D}_{ID_2}$ is then processed in the same manner and the requested bytes added to the buffer until all bytes under audit are placed in the buffer. Aside from the bits under audit, the data owner also sends a salt value, $Salt$, which the cloud must also placed in the buffer. Finally, a hash signature of the bit sequence in the buffer is computed and returned. By decrypting $E_K(row_{no}|kw)$, the data owner knows the keywords at all the specified locations and can compute their corresponding

ciphertexts. Because the auditor can recompute the ciphertexts, he can reproduce the buffer that the cloud server should obtain and verify that the signatures match. The inclusion of the salt serves to defend against replay attack should the same set of bit sequence gets audited at a later time.

To make use of a public auditor, the data owner must pre-compute sets of bit sequence, salt and signature and provide them to the auditor. An audit would be performed by sending the message describing the run_size and bit locations, then verifying the signature the cloud server responds with.

Modes of operation

The choice of symmetric encryption was natural due to its efficiency and well-studied security. For the purpose of our auditing protocol, it is also important that the data owner can systematically reproduce the ciphertext at any byte location.

Cipher-block chaining (CBC), a commonly used mode of operation for symmetric encryption, would require the preceding ciphertext block in order to compute the ciphertext at any location. This could significantly increase communication cost depending on the number of disjoint bit sequences under audit.

The choice of counter mode (CTR) is more suitable for our application as each ciphertext can be computed independent of other blocks, averting the communication cost of performing encryption in CBC mode. The initialization vector can be stored in plain with the document or use $H(\mathcal{D}_i)$ to further reduce storage cost.

Furthermore, it is also possible to perform non-indexed keyword search under CTR mode. Detailed discussion can be found in section 2.4.

Parameters for row and keyword selections

On average, English word length is approximately 5 characters, which translates to 40 bits. For a sample document set obtained from Project Gutenberg [2], keywords appear on average 15 times per document. Therefore, 600 bits on average would be available per row returned. Auditing 1024 kbits would require, on average, a minimum of 1707 rows. Suppose we request 3500 rows, we can then select $run\_size = 20$ bits at each keyword location. Then, for each keyword location, the auditor randomly chooses a starting bit location with an offset between 1 and $keyword\_length - 20$. The bits under audit are the 20 bits following the specified start location.

2.7.5 Analysis

In terms of storage, our scheme requires only that the encryption for each distinct keyword and the row number, $E_K(row_{no}|kw)$, be added in the keyword location indexes. The communication and computational complexity of keyword and phrase search is identical to the underlying phrase search schemes.

When compared to audit schemes, our solution uses encrypted indexes in place of tags and erasure codes. The storage cost of proof of retrievability schemes heavily depend on the recoverability of the erasure codes used and grows proportionally to the document set size. Our storage cost also increases as the document set size increases. However, as noted in section 2.4, the probability distribution of natural languages is far from uniform. The number of distinct keywords, and so the size of the indexes used in our scheme, grows much slower than the document sizes do. The communication cost is similar to existing audit schemes in that the bits under audit must be specified in some way. The main computational cost during setup is that of generating the indexes for the entire document set. During audit, the data owner must decrypt the retrieved keywords and encrypt the audit bit sequence while the cloud hashes the requested bits.

Note that the number of audits allowed in our scheme is theoretically upper bounded only by the information available in the document set and the size of the salt chosen. Any set of bits in the document can be audited, provided that the keyword at the location is indexed. As such, indexing all words in the document set would allow our scheme to achieve proof of retrievability, provided the index itself is retrievable. This can be achieved by storing the index locally or using existing proof of retrievability schemes on the indexes. Note that the indexes are much smaller than the document set. Therefore, the data expansion and computational cost resulting from the use of erasure code would also be much lower.

In terms of security, the stored documents are symmetrically encrypted and the private key remains with the data owner at all times. The indexes are also symmetrically encrypted for privacy. In the public auditing model, the auditor is given a set of byte locations and the cryptographic hash of the bit sequence it should obtain. Since it is computationally infeasible to compute the pre-image of a cryptographic hash, the auditor would not be able to manipulate bit sequences to obtain different valid signatures. Bit locations also do not reveal information on the stored data. Attempts to store and replay audit queries by a malicious cloud operator would only be successful if exactly the same bit sequence and salt were chosen. Attempts to suggest the requested row do not exist would be caught since the data owner is aware of the number of rows available. Should

the cloud operator send a set of locations for a different keyword than the one requested, it would also result in an audit failure since a different keyword would result in a different bit sequence or a mismatched row_{no} should it be included with the locations.

2.8 Computation and Search over Encrypted XML Documents

To the best of our knowledge, the problem of obtaining aggregated data from encrypted XML documents has not been addressed though there have been efforts towards querying of such documents. In [38], the authors described a technique for encrypting and querying XML documents in a tree structure. While the scheme is reasonably efficient, it is limited to *select* type queries and the use of trees exposes the structure of the XML documents. Sung [67] proposed using Elliptic curve cryptosystem to encrypt XML documents and Juan [39] investigated techniques to reduce the overhead in decrypting XML data. Both of these solutions considered only partially encrypted XML documents and are also limited to *select* type queries.

In this section, we present a scheme which combines search and computations over encrypted documents to extract aggregated data from XML documents [52]. Unlike existing schemes, our approach can be used for partial or fully encrypted XML documents and allows computations to be performed server-side, expanding to queries that include functions such as *sum* and *avg*. This solution is particularly useful for processing user uploaded forms to the cloud. We begin by presenting the communication model of the proposed scheme in section 2.8.1 and provide a simple description of the XML format in section 2.8.2. Then, we will introduce the basic protocols required for keyword search in section 2.8.3. Finally, in Section 2.8.4, the full protocol for search and computations is presented.

Note this work has also appeared in:

- H. Poon and A. Miri. Computation and search over encrypted XML documents. In *IEEE International Congress on Big Data*, pages 631–634, 2015

2.8.1 Model for keyword search and computation over encrypted data

The communication model for our protocol involves up to three parties: The data owner, the cloud server and the user. In a private cloud, the user is simply the data owner. Figure 2.21 illustrates a standard protocol where the user initiates the request by sending the keywords, kw_i, and the function, $F(x)$, to the data owner. The data owner then

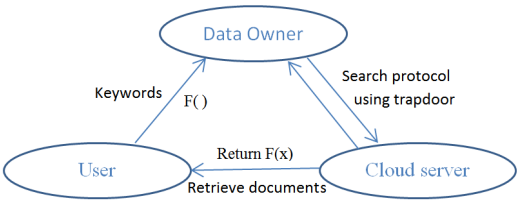

Figure 2.21: Communication model for search and computation over encrypted data

generates a trapdoor and sends it to the cloud. A protocol to search over the requested keywords and function arguments follows. Finally, the cloud responds to the user with the indexes to the requested documents and the result of $F(x)$.

2.8.2 XML format

XML is a markup language that specifies how documents encoded in a format that is simple and easy to use [4]. Since its inception, it has seen wide-spread use across the Internet and many variants, such as JSON, have since been specified.

A typical XML file includes tags, attributes and contents. A tag is a markup that begins with a '<' and ends with '>'. Attributes are sometimes included in a tag to better describe the content. The following is an example of a XML file describing music in an album, where various tags such as title and artist are used and a number attribute describes ordering of the songs.

```
<ALBUM>
  <SONG number="1">
    <TITLE>Blowing in the Wind</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <PRICE>5.99</PRICE>
    <YEAR>1963</YEAR>
  </SONG>
  <SONG number="2">
    <TITLE>Lost in France</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <PRICE>6.99</PRICE>
    <YEAR>1976</YEAR>
  </SONG>
...
</ALBUM>
```

It is important to note that the location of a tag within a document is not fixed. The

ordering of the tags may differ without affecting the content of the document. Certain elements may be present in some, but not all documents. The size of the content may also be inconsistent. Therefore, a searching algorithm is required for processing XML files.

2.8.3 Search over encrypted XML documents

We first require the basic conjunctive keyword search scheme from section 2.4.1, which allows a user to identify documents matching a set of keywords. However, it would not be able to access specific elements within the XML documents. For example, in order to answer the question, "What is the average price of a song by Bob Dylan?", one would first require identification of albums that contain Bob Dylan in the artist tag and retrieve the corresponding price. Note that to determine the price, one must first determine the documents listing the artist's songs, and that the artist's name contains two consecutive keywords. Matching of consecutive terms is also termed phrase search. We present a light weight version of the protocol discussed in section 2.4 below.

To provide phrase search capability, a keyword location index is used. To generate the keyword location index, we compute

$$I_L(H(\mathcal{D}_i|kw_j)) = E_K(l_1, l_2, \ldots, l_n), \tag{2.32}$$

where $H()$ is a cryptographically secure hash function and l_x are the locations of kw_j within $\mathcal{D}_i$. Given a user phrase search request $kw' = (kw_1, kw_2, \ldots, kw_q)$, the data owner proceeds as in section 2.4.1 to identify n documents containing all keywords. It then queries the location of the keywords by sending $H(\mathcal{D}_i|kw_j)$ for $i = 1$ to n and $j = 1$ to q to the cloud. Given the locations, the data owner can verify that a phrase exists if, for any $l_x \in I_L(H(\mathcal{D}_i|kw_1))$, $(l_x + j) \in I_L(H(\mathcal{D}_i|kw_{j+1}))$, where $j = 1$ to q.

It should be noted that, even when querying single keywords, it is possible that a tag or attribute contains the queried keyword. It is rarely the intent to return results where keywords belong to both markup and content, therefore leading to false matches. When applying the basic phrase search protocol to XML documents, it is important then to also index certain symbols as keywords, in particular $<$, $>$ and /, to allow differentiation between tags and contents, and facilitate identification of sibling, child and parent elements from the XML tree.

2.8.4 Computations over encrypted XML documents

The previous phrase search scheme would be sufficient to perform most common searches. However, despite the computational and storage efficiency of symmetric encryption, such as AES, computations cannot be performed on symmetrically encrypted data without decryption, leading to a potentially expensive process of retrieving thousands of values, if, for example, one were to compute the average price of all albums in a catalog.

Homomorphic encryption

As discussed in section 2.2.6, homomorphic encryption allows computations to be carried out on ciphertexts, where the results would decrypt to the corresponding computation on plaintexts. An additively homomorphic scheme, such as Paillier cryptosystem would have the following property:

$$Add(E(A), E(B)) = E(A + B) \tag{2.33}$$

By extension, multiplication by constant is also possible in additively homomorphic schemes. This feature allows computations to be performed without exposing confidential information.

Combining Symmetric and Homomorphic Encryption

To enable computations over encrypted data, homomorphic encryption is a natural choice. However, it is significantly more expensive than symmetric encryption in both storage and computations. To benefit from its feature while minimizing its cost, we propose adapting the encryption scheme based on the content of the document. That is, only content which may be used for computations, assumed here to be numeric data, are homomorphically encrypted, while remaining content are symmetrically encrypted. Using the example in section 2.8.2, we may have $E_{AES}(</\text{ARTIST}>)$, $E_{AES}(<\text{PRICE}>)$, $E_{Paillier}(5.99)$, $E_{AES}(</\text{PRICE}>)$. Without loss of generalization, we will describe our scheme using Paillier's cryptosystem.

To enable computations over encrypted XML documents,

1. Document Encryption: Each document, $\mathcal{D}_i$, is parsed and encrypted such that numeric data are encrypted using $E_{Paillier}(m)$, as described in section 2.2.6 and the remaining content is symmetrically encrypted using AES, $E_{AES}(m)$.

2. Document Indexing: Each document, $\mathcal{D}_i$, is parsed and indexed as described in section 2.4.1 and 2.8.3.

3. Keyword Search: To compute $F(x)$ where $tag_m = kw_1, kw_2 \ldots kw_q$ and $x \in tag_x$, we must first resolve the clause by searching for keyword, kw_i using the keyword-to-document index and the keyword location index if phrases are queried. The location of tag_x is then queried for the matched documents. Markup symbol locations are also queried to resolve potential conflicts between keywords and tags.

4. Function computation: Once the location of the function arguments are determined, the cloud server computes $F(x)$ using $E_{Paillier}(x)$ where $x \in tag_x$. For example, if the average price is desired, then $tag_x =$ '*price*' and $F(x) = \sum E_{Paillier}(x)$. Upon receiving $F(x)$, it is decrypted to obtain the sum and the average is obtained by dividing the number of matched elements.

Note that the function, $F(x)$, is not limited to summations. With minor modifications, other more complex functions can also be used such as scalar product [26] and comparison [19]. Our solution requires the exchange of two rounds of queries and the encrypted result of the function. In terms of computations, the queries consist of binary searches, hash computations and equality comparisons, each of which can be done efficiently. The efficiency of the computation of $F(x)$ depends on the function and the encryption algorithm used. In terms of summation, Paillier's cryptosystem consists of a simple integer addition modulo n^2. Since most XML documents consists of few numeric value relative to text, the computational cost of the document encryption is not believed to be significantly worse than pure symmetric encryption.

It is interesting to note that it is infeasible to determine whether a ciphertext resulted from AES encryption or Paillier by examining the ciphertext alone. Therefore, a cloud server would yield no additional information from the encrypted documents. If further security is desired, it is possible to hide the location of the desired numeric values by requesting the computation of $F(x)$ over other symmetrically encrypted data, at the expense of computational and communication cost. Other security measures can also be included in the design of the keyword-to-document index and keyword location index to defend against statistical analysis at the expense of efficiency [69, 79].

In this chapter, we presented three sequential text search schemes, each targeted at different applications requiring respectively high security, low storage and fast response

time. Detailed analysis comparing against existing works in the literature was provided, along with discussions on security and design issues. We also showed that a sequential data search scheme can be extended to solve the cloud auditing problem with little additional cost. Finally, we illustrated how a phrase search scheme can be a component of a more complex query system where computations such as sum and products can be performed over encrypted XML documents. In the following chapter, we will present our work on encrypted sequential processing in media.

Chapter 3

Media

The growing importance of media cannot be understated. It is estimated that, by 2019, 80% of the world's bandwidth would be consumed by media. At the end of 2015, the number sits at 70%, with Netflix and Youtube combining for over 50% of data sent over the Internet. Naturally, the need to process videos, audios or images in a secure and privacy-aware manner will be of growing interest in coming years. While studies in encrypted media processing is still at its infancy, there exists some interesting and working solutions. For our discussions, we will consider the media to be an image, as techniques used for images generally form the basis of techniques used for audio and video.

In this chapter, we will begin with a brief discussion on existing solutions on privacy-aware media search. Then, we will present our solution on privacy-aware copyright detection on cloud. Our solutions targets cloud storage services such as DropBox, Amazon Webservices and Mega.nz, where cloud service provider may want to offer client side data encryption to protect user privacy while also respect copyright laws to avoid litigation. Our discussion will begin with background on watermarking techniques for digital rights management. Related work in the area follows, in particular compressive sensing for media encryption, along with discussions on strengths and weaknesses. Then, we will present our protocols and setup of a framework for copyright detection in cloud services.

3.1 Keyword based media search

The simplest approach to adapt existing text-based searchable encryption to media is through a meta-data only media search. Consider a set of images, $I = \{I_1, I_2, \ldots, I_n\}$, we first extract a list of keywords for each image. This extraction process can be manual,

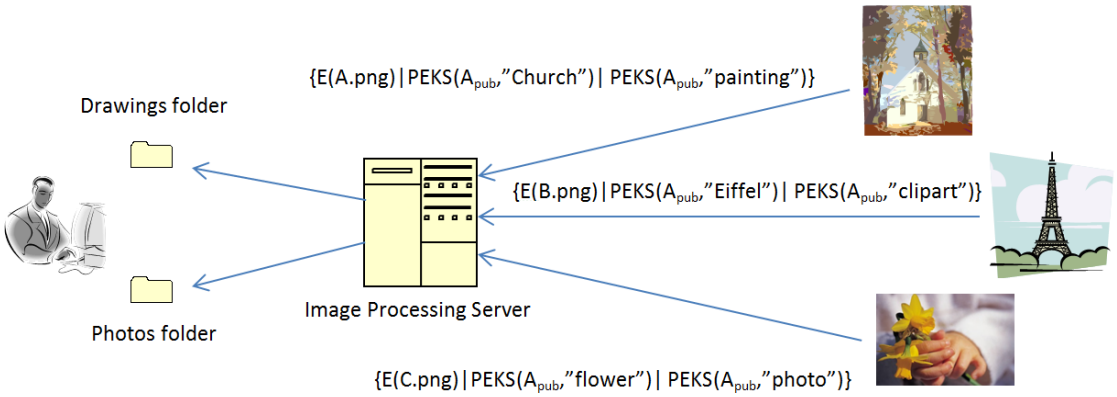

Figure 3.1: Keywords-based image filtering system based on IBE

i.e. a person assigning keywords such as 'Man', 'bird', 'table', 'HighRes', or it may be done through artificial intelligence (AI) and image recognition systems.

Once all images have been assigned keywords, solutions for conjunctive keyword search apply as is by considering the image set as the document set. This is achievable because the search mechanism for the text-based solutions are all based on extracted keywords and the documents are encrypted separately from the search mechanism, be it an index, Bloom filters or IBE encrypted keywords. The reason for the separation is because searching data content on-the-fly is computationally expensive, even when unencrypted, since each file must be scanned as a whole if no pre-processing was performed. Due to the security guarantees of standard symmetric and asymmetric encryption algorithm, processing of encrypted data is often impossible or very computationally intensive.

Figure 3.1 shows a media filtering system based on the Email filtering system in Figure 2.4. The system depicts an image bank that receives images from various sources. Occasionally, some images may contain sensitive or personally identifiable information that requires blurring before being placed in the image bank. The data owner would like to maintain privacy and prevent the host server from learning the image contents. However, due to the quantity of images received, the data owner would like an automatic sorting system that would facilitate post-processing upon decryption. The depicted system achieves this by requiring each user encrypts their images using a symmetric encryption algorithm and encrypts a series of keywords using IBE to describe the image.

3.2 Content based media search

Searching for media based on content is a far more challenging task. Generally, images are first processed to produce feature vectors, analogous to keywords in documents, using algorithms such as SIFT [45]. Feature vectors are values that are deemed representative of the image according to a feature extraction algorithm, which may consider features such as edges, lines, contrast, brightness, position of eyes and noses, etc. The Euclidean distance between feature vectors provides a mean to measure the similarity between different images. However, doing so in encrypted domain is generally impossible with standard encryption algorithms. We'll describe a solution based on homomorphic encryption.

Recall that content-based image search typically relies on Euclidean distances between feature vectors as a measure of similarity. That is to compute

$$\left\|F_Q - F_{D_j}\right\| = \sqrt{\sum_{i=1}^{N}(F_{Q,i} - F_{D_j,i})^2} \tag{3.1}$$

where $F_Q = \{F_{Q,1}, F_{Q,2}, \ldots F_{Q,N}\}$ is the feature vector of the querying image and $F_{D_j} = \{F_{D_j,1}, F_{D_j,2}, \ldots F_{D_j,N}\}$ is that of an image under test. Since the square root operator applies to all values, it is easy to see that using the squared Euclidean distance is equally valid as a distance measure. Naturally, computing the summation is possible using fully homomorphic encryption although at a high computational cost. To do so, the data owner encrypts the features $F_{D_j,i}$ for each image D_j and uploads to server. To perform a query for an image, Q, the data owner encrypts all features, $F_{Q,i}$, of the query image and uploads to the server. The server computes $dist_j = \sum_{i=1}^{N}(F_{Q,i} - F_{D_j,i})^2$ for all images in the database and returns $dist_j$'s to data owner. The smallest distance values are identified as matches by decrypting the results.

Additive homomorphic encryption, such as Paillier cryptosystem [51], can also be used to query an image in plaintext against an encrypted image database, by exploiting the following property:

$$\sum_{i=1}^{N}(F_{Q,i} - F_{D_j,i})^2 = \sum_{i=1}^{N} F_{Q,i}^2 - 2\sum_{i=1}^{N} F_{Q,i}F_{D_j,i} + \sum_{i=1}^{N} F_{D_j,i}^2 \tag{3.2}$$

Since $F_{Q,i}$ is in plain, the first term is available. With $E(F_{D_j,i})$ as ciphertext, the second term can be computed using equation 2.9. The third term can be uploaded with the features during setup. The server can then compute the encrypted distance without

interacting with data owner. The encrypted distance must then be sent back to the data owner for decryption. It should be noted, however, that the ability to perform similarity search using images in plain could allow the server to infer that the matched encrypted image retrieved is close to the query image presented.

3.3 Privacy-aware digital rights management for cloud storage

Digital rights management (DRM) is a relatively new concept born as a result of the ease of data sharing on Internet, in particular music, images and videos. The Digital Millennium Copyright Act (DMCA) is the most well-known law in the world that criminalize technology enabling copyright infringement. Since the birth of Napster, the entertainment industries had been in a struggle to enforce copyright laws on the Internet, where traditional laws limited by countries and borders do not hold. The ongoing war had led to high profile shutdowns of popular file sharing sites. One of the most notable cases involves Megaupload, which is a company that RIAA claims to facilitate piracy of copyrighted media, despite allowing owners to report copyrighted contents. The current push towards cloud storage had been widely praised for its economic advantage. However, companies offering cloud storage, such as Amazon EC2 and Microsoft Azure, operate much like these file sharing sites and are arguably equivalent. They allow users to place potentially copyrighted material onto the company's servers and share with other users. If a cloud storage provider provides full access to authorities for scanning for copyrighted content, it would compromise the privacy of its users. If it refuses, the provider would be exposed to legal threats and would risk having its operations shutdown.

3.3.1 Current state of DRM on cloud services

To provide a sense of the needs of cloud services and copyright claimants, we present the case of Megaupload, which had previously been the largest file sharing site in the world and was targeted by Recording Industry Association of America (RIAA) as the largest piracy site behind bittorrent. The file sharing service was shut down in 2012 on claims that it was dedicated to copyright infringement. When Megaupload was shut down by RIAA following court orders, it had few options. After a period of considerations, the service was relaunched as an encrypted cloud service citing user privacy while following copyright laws by accepting DMCA complaints and processing file takedowns. It operated much like any other cloud storage company, except all files are stored encrypted

while users maintain the secret keys. In short, Mega, using encryption as a means to enable user privacy, can continue to operate while protecting itself against litigations claiming that it's willfully operating a business that profits from piracy. Since data stored on its servers are encrypted and it does not have the secret key, it cannot know what the data represents and so cannot be legally responsible for its content unless given the decryption key. As of today, Mega continues to operate while likely aware that much of its customer base remains those seeking to distribute and download copyrighted media. The current state is such that DRM over cloud services is still largely performed by monitoring of known file sharing sites and human reporting of DMCA complaint forms.

There are a few caveats to Mega's claims. The cloud service provider is still required to be trusted for user privacy to be protected since the encryption and decryption scripts are loaded from the cloud service provider. It would be trivial for Mega to compromise user privacy should it desire to. Users also do not have the ability to search, rendering it difficult to manage a large collection of files. Copyright claimants must provide both the file link and the decryption keys in DMCA complaints. Should the decryption key be kept secret within a well-knit community, copyrighted claimants would have no recourse.

Other non-encrypted file sharing services exist, but with a high turn-over, presumably due to legal challenges. Our goal is to provide the tools that would allow both copyright claimants and cloud services to achieve their respective goal of copyright detection and user privacy. The private keys remain private to each party while copyright infringement can be detected without revealing the file content.

3.3.2 Digital watermarking

As a result of the ease of data sharing over the Internet, Digital watermarking was developed as a technique to protect media from illegal distribution. It is widely used in the film and music recording industries. For example, an unreleased movie may be distributed to critics and reviewers for advanced screening to help gauge the movie's performance at the risk of being leaked to the Internet. Should it happen, digital watermarks can be used to track the source of the leak and served as proof in a court of law.

Digital watermarking is the process of hiding a watermark signal in a carrier signal. The watermark signal or message is typically hidden in such a way that the carrier signal is not disrupted from its operation. In practise, the carrier signal, also called the cover work, is generally a media such as an image, audio or video while the watermark signal is a white noise like sequence. Once inserted into a host media, the watermarked version is perceptually identical to the original.

The embedding of an invisible watermark allows the verification of the owner of

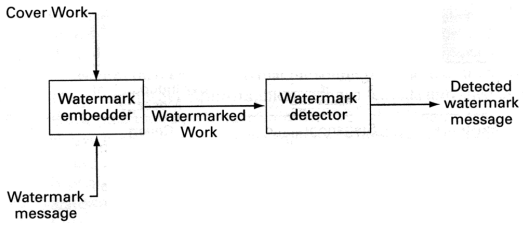

Figure 3.2: Basic watermarking model

media content. In the case of a copyright dispute, the owner can be verified using the watermark. Since the marked media is perceptually identical to the original, it can be distributed commercially without issue. Figure 3.2 shows the basic model of embedding and detecting watermarks.

There are generally two criteria for watermark algorithms: Robustness and imperceptibility. Robustness refers to the ability of the watermark to survive transformations, e.g. image processing. Imperceptibility refers to the quality of the watermarked media relative to the original. It is generally desirable but difficult to achieve both since a stronger watermark signal is more difficult to remove while also causes more degradation to the cover work.

There are many watermark embedding techniques, such as fragile watermarks [42,46] and spread spectrum content dependent watermarking [29,35]. For our solution, we will consider a simple watermarking scheme: A watermark, $W_m = (W_1, W_2, \ldots, W_N)$, is to be embedded into a cover work (image), $C_0 = (C_1, C_2, \ldots, C_N)$ where C_i are pixel values. C_i would be coefficients if spectral representation is used. The watermarked image is computed as follows [36]:

$$C_w = C_0 + \alpha W_m \tag{3.3}$$

where $(W_1, W_2, \ldots, W_N)$ is each randomly selected as 1 or -1 with equal probability and α is the watermark strength, a parameter used to balance between robustness and imperceptibility. C_0 is shifted so the sequence of pixel values is zero-mean. Note that C_0 and W_m are transformed from 2-D to a one dimensional sequence.

Watermark detection allows verification of a watermark in an image under test. Given the watermark, detectors can be separated into two categories: blind and non-blind. A blind detector detects the watermark without knowledge of the cover work. A non-blind detector detects the watermark given the cover work. Naturally, the latter is far easier to achieve and provides a higher accuracy.

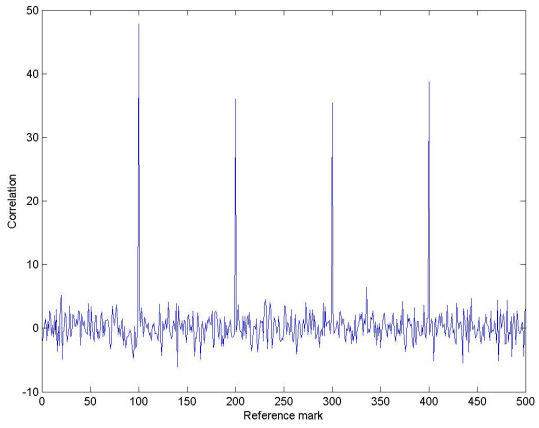

Figure 3.3: Correlation values in a random reference mark test

Detection is commonly done using a slight variation of the Pearson's sample correlation coefficient. A non-blind detector for the simple watermarking scheme above would be

$$\rho_{nb} = \frac{1}{N}\sum_{i=1}^{N}(C - C_0)W_m \tag{3.4}$$

where C is the image under test and N is the number of elements in C. A blind detector would be

$$\rho = \frac{1}{N}\sum_{i=1}^{N}CW_m. \tag{3.5}$$

If $C = C_w$,

$$\rho_{nb} = \frac{1}{N}\sum_{i=1}^{N}(C - C_0)W_m = \frac{1}{N}\sum_{i=1}^{N}\alpha W_m^2 = \alpha \tag{3.6}$$

since $||W_m|| = 1$. For blind detection,

$$\rho = \frac{1}{N}\sum\sum_{i=1}^{N}CW_m = \frac{1}{N}\sum\sum_{i=1}^{N}C_0W_m + \frac{1}{N}\sum\sum_{i=1}^{N}\alpha W_m^2 = \frac{1}{N}\sum\sum_{i=1}^{N}C_0W_m + \alpha \approx \alpha \tag{3.7}$$

because C_0 is zero mean then the expected value of the first term is zero.

Since the blind detector does not include the cover work in its calculation, the corre-

Figure 3.4: Original image (Spatial domain)

lation value is noisier. Figure 3.3 shows the correlation, ρ, of watermarked images with the embedded watermark and randomly generated watermarks. The correlation values with embedded watermarks are clearly distinguished. A threshold value is then used to determine whether a positive detection occurred. For our discussion, we will consider a blind detector as the media is encrypted. Even if the encrypted cover work is made available, there can be enough uncertainty regarding the identity of the cover work that a blind detector would provide a more suitable solution in our scenario.

While it is straightforward to embed the watermark into image pixels, embedding in the spatial domain may not be as robust against common image processing techniques. Embedding in frequency domain is often suggested as an alternative. Figure 3.4 and Figure 3.5 show respectively a sample original image and its wavelet transformed version.

Frequency transform, such as Wavelet or DCT, leads to an alternate representation of the media in terms of frequency levels. In Figure 3.5, the smallest sub image, in top left, represents the coefficients for the lowest horizontal and vertical frequency components. It shows the rough layout of the image. The largest portion, in bottom right, represents the highest frequency components and outlines the detailed values in the image. Embedding in low frequencies would grant the highest robustness and the lowest quality. Conversely, embedding in high frequencies would yield the highest quality image but the watermark could be easily removed.

Note that the simple embedding algorithm is applicable to both spatial and frequency

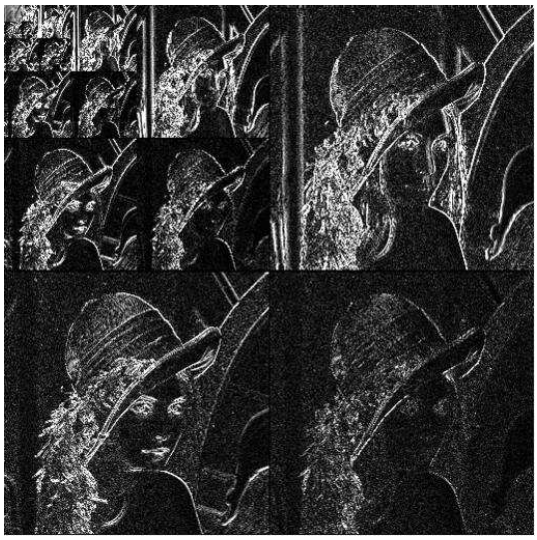

Figure 3.5: Wavelet transformed image (Frequency domain)

domain as the cover work sequence may represent pixel values as well as frequency coefficients. For our solution, it is only important that the embedding technique be known. Both spatial and frequency domain embedding can be used.

3.3.3 Related work

Long before cloud computing's mass adoption, researchers noted an ambiguity in the target application of digital watermarking. Namely, in order to prove that a media is copyrighted, the claimant must reveal the watermark in order to prove his ownership. However, in doing so, he would have also provided the information necessary to remove the watermark from the media. Early proposals to the problem include asymmetric watermarking [37, 68], where a public and a private key were used allowing public watermark detection, but was determined to be vulnerable to sensitivity attacks [5]. Later on, Craver [62] proposed Zero-knowledge watermarking which relies on zero-knowledge proofs as a means to prove the existence of the watermark without revealing what the watermark is. Research into Zero-knowledge watermarking, however, does not concern with the privacy of or require hiding the watermarked image, and generally uses a mixture of image processing technique alongside commitment schemes to achieve its goal [5, 62].

More recently, there have been a few proposals [34, 73] regarding privacy-preserving image outsourcing and authentication. Most of which have revolved around using com-

pressed sensing as an encryption algorithm to achieve privacy over the outsourced images.

Compressed sensing as encryption and its drawbacks

Compressed Sensing (CS) [21] is a signal processing technique for reconstructing a signal known to be sparse through fewer samples than required by the Shannon-Nyquist sampling theorem. This means signals that can only be sampled infrequently and previously thought to be impossible to reconstruct may now be determined. It also means that sparse signals can be compressed to fewer samples than stated by the Nyquist criterion. The breakthrough discovery led to significant interest in the area. Among them, few researchers [43, 50, 60, 78] also suggested that it is usable as a compression-encryption algorithm.

The recoverability of a sparse signal depends on two conditions: a) The signal must be sparse in some domain, that is most of its elements are zero. b) The transform matrix to the domain satisfies Restricted Isometric Property (RIP) [21]. The latter is known to be satisfied by a vast majority of signals in practise. Most media are sparse. Since human visions and hearing can only perceive a small range of frequencies and are most sensitive to low frequencies, compression algorithms often convert to the frequency domain and remove all elements with low magnitude, resulting in a sparse signal in the frequency domain.

The CS compression algorithm is a simple matrix multiplication:

$$Y = AX \tag{3.8}$$

where $X \in \mathbb{R}_{n\times 1}$ is a sparse signal, $A \in \mathbb{R}_{m\times n}$ is a compressed sensing (CS) transform matrix and $Y \in \mathbb{R}_{m\times 1}$ is the compressed signal. Note that $m \ll n$. The CS matrix is often chosen to be zero mean normally distributed. When given Y and A, the matrix multiplication represents an underdetermined system and can have many solutions. Compressed Sensing states that, if there is a unique sparse solution to the underdetermined system, then it is recoverable. We omit the reconstruction algorithms as they require significant background and are not relevant to our discussion.

The argument for adoption of CS as an encryption algorithm is simply that the CS matrix is necessary for signal reconstruction. Without it, reconstruction fails and the result signals appear random. Furthermore, the underdetermined nature of the system seems to lend well to the security of the system. As an encryption scheme, X is the plaintext, A is the secret key and Y is the ciphertext. It is not required that $m \ll n$, only $m \leq n$. For the purpose of image processing, all media are frequency transformed

and compressed by removing low magnitude elements (making them zero) to a sparse representation, used as X. Each element in the CS matrix, i.e. the secret key, is generated as zero mean normal distributed random number. The ciphertext Y is computed as a matrix multiplication of X and A.

The basic construction is almost identical to Hill cipher. Works on analyzing its security had mostly arrived at similar properties, e.g. A high number of possible keys, computationally infeasible to brute force, and vulnerable to known-plaintext attack. Namely, for a $m \times n$ matrix, we would need at most n known X to Y pairs to completely determine A. It was also noted by [16] that CS matrix satisfying RIP is energy preserving. We noted that the analysis [50,60] generally omitted the effect of the restrictions on plaintexts and CS matrix, that is the sparsity of plaintext and the normal distribution of CS matrix. Both of which limits the plaintext and key space. In particular, a normal distributed matrix could have many small elements. In image processing, these small elements might be considered zero with little disruption on the quality of the reconstruction. Through a series of experiments using artificial neural networks, we [63] found non-uniform distribution of ciphertext to plaintext and private key, suggesting other vulnerabilities may be present if compressed sensing is used as an encryption scheme.

Although we were unable to devise specific attacks aside from ones known in the literature, there are other security and practical issues regarding its use. Wang [73] was among the first to devise a fairly complete solution on multimedia storage outsourcing framework. The main observation in the result is that watermarking can be performed with relative success in a CS transformed media. Since CS transformed images and watermarks cannot be used to reconstruct the original image and watermark without the CS matrices, they do not leak user privacy and are considered secure. However, the practical weakness of CS as encryption becomes apparent in the experiments.

To evaluate performance, a 512×512 original image is used along with a watermark of the same size. By definition, a CS matrix of size $512^2 \times 512^2$ would be required. That is the matrix would have over 68 billion elements. Instead, the researchers opted to consider the image as 64^2 8×8 blocks. Each block uses a separate 64×64 CS matrix. While this strategy reduces the total number of elements in the matrices to $64^4 = 16777216$, it also leads to more serious information leakage due to the energy preserving property of RIP [16]. The CS transformed image in [73] showed a fairly clear coarse representation of the original image as a result of this block separation to reduce the CS matrix size. In its complexity evaluation, it was also noted that a 1000×1 image, that is equivalent to a 20×50 image, would require sending 256MB of data to represent the 1000×1000 CS matrix.

To summarize, to achieve the original intended level of security, the key size needs to be N^2, where N is the plaintext size. For media, N is often relatively large compared to block size of typical encryption algorithms. With the key size growing exponentially relative to the plaintext size, CS as an encryption system becomes quickly impractical. Reducing the media into smaller blocks to address the issue instead leads to information leakage that is perceptible by the naked eye, removing user privacy and security. Even assuming that CS is secure as encryption, the system is completely impractical even for very small images. Attempts at improving the practicality all but remove any security on the plaintext. For these reasons, Compressed sensing do not appear suitable as a technique for privacy-protected digital rights management. We instead propose the use of homomorphic encryption for our constructions.

Furthermore, all existing solutions use the semi-honest model for all participants, which states that the participants would not inject false or malicious data. When considering our copyright detection for cloud services, this is a fairly weak assumption as each participant would have some incentive to do so. Particularly, the user holding the media would, in many cases, want to pass the detection to have his media deemed legitimate whether it is the case or not.

3.3.4 Model for copyright detection over encrypted data

The standard communication model for a copyright verification protocol involves up to three parties: the copyright claimant, the cloud server and the user (file uploader). Digital watermarking is used for copyright detection. The user wishes to upload a media to the cloud server. He first encodes the media as specified (e.g. JPEG, wavelet, etc) and encrypts the file using homomorphic encryption. The file is then uploaded to the cloud server which initiates a three party protocol with copyright claimant to verify the legitimacy of the media. The copyright claimant in turn provides the encrypted watermarks to verify against the user's file. The protocol would result in a correlation value that indicates whether the claimant's watermark was found to be present in the user's file. A sample protocol is illustrated in Figure 3.6.

Should an application requires more flexibility, where the user may encode the media in different formats, such as wavelet instead of discrete cosine transform (DCT), multiple equivalent versions of the detection algorithm can be implemented. A possible scenario to circumvent detection would be to encrypt the file prior to upload. Unfortunately, this scenario is unsolvable without the private key and barring any weakness in the encryption scheme the file uploader uses. A secure encryption scheme, such as AES, does not produce ciphertexts where plaintext can be revealed. In general, any encoding on the media must

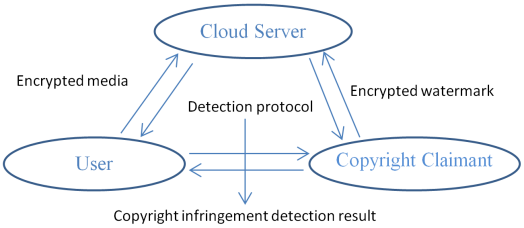

Figure 3.6: Communication model for copyright detection over encrypted data

be known, i.e. the encrypted media must be a known representation of the actual media. Furthermore, pre-encryption would incur a performance cost, in particular, to the clients that the file uploader wishes to distribute the files to and may present a technical barrier to some and limit the audience. This makes it an undesirable option for illegal file uploaders. Furthermore, we also require no collusion between the parties because the computations required to arrive at a copyright infringement decision is such that any two colluding parties could yield meaningful information on the remaining member's inputs. In fact, the central function being performed is between the copyright claimant and the user, where any collusion between them would naturally reveal all the information in the system. The cloud service provider's role is to act as a mediator and protects the privacy of the two sides by producing a final result that does not reveal information on either side's input.

An ideal solution would consider all three parties to be malicious since each party may have interest in gaining information on the other parties:

- The file uploader could be a user wishing to share copyrighted media using the cloud service provider's resources or obtain information on the copyright holder's watermark in order to circumvent detection.
- The cloud service provider could be interested in the user's files to better position its business in relation to its clients. It may even be interested with colluding with either party for financial incentives in compromising the third party's private information.
- The copyright claimant may have interest in the user's or the cloud service provider's private information, such as a government agency wishing to uncover a person of interest's data.

In our solution, we assume the semi-honest model for the cloud, where the service

provider would not deviate from the protocol, but would attempt to obtain extra information on stored media or the watermark. The data holder and the watermark holder are assumed to be potentially malicious and would attempt to alter the data to its favor. Note that we will begin our discussion with a base algorithm without the assumption of malicious participants before leading into our full protocol.

Our solution differs from earlier works by making use of encryption algorithms with established security. It is also the first to achieve security and privacy against dishonest participants in a setting representative of the expected behavior in cloud storage services. We are also the first to describe the components required in a practical framework for digital rights management involving both search and copyright detection.

3.3.5 A construction with semi-honest participants

Before we describe our solution against malicious participants, we begin with a construction under the semi-honest model to illustrate the basic setup. There are three parties involved:

- The data holder or user holds the media, X, and wishes to place the media on the cloud without revealing the media to the cloud service provider
- The cloud service provider operates the cloud server and provides a data outsourcing service that accepts encrypted data, alleviating its clients' privacy concerns. However, it also needs to protect itself against the legal threat posed by users potentially uploading copyrighted content.
- The watermark holder or copyright claimant holds and embeds the watermark, W, into the copyrighted media prior to releasing it to public.

In a semi-honest model, the data holder would honestly use the media, X, that it uploads to cloud server and the watermark holder would likewise use, W, and neither would deviate at any step of the protocol. Similarly, the detection result would also be honestly reported. As such, the cloud service provider can be assured that if the detection algorithm works as intended, any copyright infringement would be identified. He may therefore trust the two parties to perform the bulk of the detection prior to reporting the results. The key is to prevent any leak of information on the watermark and media to each party during the process. Figure 3.7 shows the communication model with semi-honest participants.

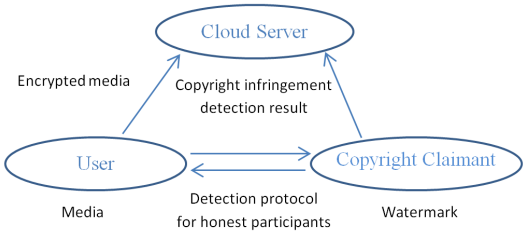

Figure 3.7: Model for construction with semi-honest participants

As in any watermarking application, the copyrighted media must be watermarked prior to release. A sample watermark embedding process for images in the frequency domain is as follows:

1. Perform DCT/Wavelet transform to obtain a set of coefficients for an image of size q.

2. The set of transform coefficients will be real numbers. In order to limit error and apply encryption, the values are multiplied by a large factor and truncated into integers.

3. The watermark vector, W, of size k, is chosen from the standard normal distribution and also multiplied by the large factor and truncated. The watermarked image, X', is obtained by computing:

$$X' = X + \alpha W \tag{3.9}$$

where α is the watermark strength and X represents the k coefficients selected for embedding, typically values representing the middle frequencies to balance robustness and imperceptibility. To simplify our discussion and without loss of generality, we will assume X and W are the same size, i.e. $q = k$.

The copyrighted image, having been released, may not be distributed without permission. The copyright claimant, wishing to prevent unauthorized mass distribution, sets up a server to work with the cloud service provider to prevent copyrighted material from being accepted. When a user wants to upload a media to the cloud server, it must also engage in a secure two party computation protocol with the copyright claimant's server to verify the legitimacy of the media being uploaded. At the end of the protocol, the copyright claimant will notify the cloud server of any infringement and the encrypted media uploaded by the user would be accepted or rejected accordingly.

Secure watermark detection with semi-honest participants

To determine whether data holder's media, X_{DH}, is copyrighted, watermarked detection is performed. We first note that the criteria for a positive watermark detection,

$$\rho = \frac{1}{N}\sum_{i=1}^{N} X_{DH}W > \delta_{th} \tag{3.10}$$

can be rewritten as

$$\sum_{i=1}^{N} X_{DH}W > N\delta_{th}, \tag{3.11}$$

where the left hand side of the equation is a scalar product and the right is the threshold value, δ_{th}, scaled by the number of elements, N. From equation 3.11, the watermark detection then consists of two steps:

- Compute the scalar product of the media under test and the watermark
- Compare against threshold value

To compute the scalar product, we use a secure protocol by Goethals [26, 73] and Paillier cryptosystem as the underlying homomorphic encryption scheme 2.2.6. The protocol involves one round of exchange between the data holder and the watermark owner, and one round between each party and the cloud service provider to compute the results and render the decision. Figure 3.8 shows the secure scalar product protocol.

From pixels and coefficients to field elements

Transformation into frequency domain generally results in coefficients that are real numbers, $\mathbb{R}$. Since Paillier and most other encryption algorithms require integer as plaintexts, the coefficients are scaled up and fractional values removed before encryption is applied. For higher accuracy representation, a higher scaling factor is used, but also requires more storage and computational cost. Watermarking in the spatial domain, however, does not require scaling since pixel values are typically already integers. Note that the threshold value for detection must also be scaled up by the same factor.

Another issue that needs addressing is that shifted pixel values and coefficients may be negative or positive while plaintext elements are strictly between 0 and C_{max}. To accommodate negative values, we separate the field into a positive and a negative region [23]. That is, for $x = 0$ to $\frac{C_{max}}{2} - 1$, it is mapped to $[0, \frac{C_{max}}{2} - 1]$, representing positive

$SecScal(X_{DH}, W, N\delta_{th})$:

1. The user runs the key generation algorithm for Paillier cryptosystem to produce a public and private key pair, (pk, sk).
2. The user then encrypts his media, $X_{DH} = \{x_1, x_2 \dots x_N\}$, using pk and sends the encrypted media, $E_{pk}(X_{DH})$, to the watermark holder.
3. Given the encrypted media, the watermark holder computes the scalar product in the encrypted domain: $z_1 = \sum_{i=1}^{N} E_{pk}(x_i)^{w_i}$, where $W = \{w_1, w_2 \dots w_N\}$ is the watermark.
4. The watermark holder also encrypts a random plaintext, s_B, to obtain $z_2 = E_{pk}(s_B)$. Then, he sends $z = z_1 z_2$ to the user and $\{s_B, N\delta_{th}\}$ to the cloud service provider, where $N\delta_{th}$ is the threshold value.
5. The user decrypts ρ to obtain $s_A = D_{sk}() = (\sum_{i=1}^{N} X_{DH} W) + s_B$ and sends it to the cloud service provider.
6. The cloud service provider computes $\rho = s_A - s_B = \sum_{i=1}^{N} X_{DH} W$. The encrypted media is rejected if $\rho > N\delta_{th}$ and accepted otherwise.

Figure 3.8: Secure scalar product for copyright detection with semi-honest participants

values. For $x = -\frac{C_{max}}{2}$ to -1, it is mapped to $[\frac{C_{max}}{2}, C_{max} - 1]$, representing negative values. Due to properties of modular arithmetic, the representations will remain consistent through ciphertext additions and multiplications so long as the values never exceed the assigned range at any point in the computation. Therefore, it is important that $N = pq$ be sufficiently large to accommodate the maximum image size and accuracy required.

3.3.6 A construction secure against dishonest participants

The construction for semi-honest model becomes problematic if we consider that each participant may want to cheat in some manner. In particular, a dishonest user would want to convince the cloud of its media's legitimacy regardless of the nature of the media. This is certainly the case in our target application of file sharing sites where illegal media distribution is rampant. One simple approach to subvert the semi-honest construction is to send a much smaller value instead of the true value of s_A. This would lead to a smaller correlation computed by the cloud and less likely to be flagged as copyright infringement.

To address this issue, we propose a construction secure against dishonest participants. Our solution is based on a secure multiparty computation scheme by Bendlin [11]. The scheme works with any homomorphic encryption algorithm as long as the computations involved do not increase size of the input over a limit. It is efficient during function evaluation as it does not require any expensive cryptographic operations. The latter are performed instead in a setup phase between the participants.

The original description allows for any n parties to perform multiplications and additions of their inputs. They exchange sets of tokens along with cryptographically secure MAC during the setup phase. Proof of knowledge protocols are used to ensure that the tokens are well formed and within a system limit, where any message and randomness used, at any point, must be less than a specified limit: $m < M$ and $r < R$. In addition, a smaller limit, $\tau < r$ and $\rho < R$, is used for specifying limits on inputs and randomness, as these will grow as computations are performed. For $C = E_{pk}(x, r)$, we call C a (τ, ρ)-ciphertext if there exists (x, r) such that $|x| \leq \tau$ and $|r| \leq \rho$. Note that these limits are set by the system, i.e. the participants, and is dependent on the required level of security. These ensure that the computations are correct. During function evaluation, the tokens are used to specify inputs, perform additions, multiplications, output values, ensure correctness and that no parties deviate from the protocol.

Zero-knowledge proofs

There are two proofs of knowledge protocols required to ensure correctness of computations from all participants:

- Proof of plaintext knowledge: This protocol provides proof to a verifier that a prover knows the plaintext that formed the ciphertext and that the ciphertext are within limit required by the system
- Proof of correct multiplication: This protocol provides proof to a verifier that a prover had correctly performed multiplication of a ciphertext encrypted under prover's public key with a plaintext plus some randomness provided by the verifier

The proof of plaintext knowledge protocol is illustrated in Figure 3.9. Broadly, the process involves the prover producing a separate sets of ciphertexts and the verifier generating a random set of coefficients. The prover then proves that he knows the plaintexts by successfully computing a linear combination of the ciphertexts using the verifier's coefficients and sending them along with the exact randomness and the plaintext of the separate sets of ciphertexts. The verifier verifies that the linear combination and randomness provided gives the correct ciphertexts.

The proof of multiplication protocol is illustrated in Figure 3.10. The basis of proof is similar to $PoPK()$. The process also involves the prover producing a separate set of ciphertexts and the verifier generating a random sets of coefficients. The main difference is that the verification involves not only the linear combination of the ciphertexts, but also must satisfy the multiplication and addition of the random ciphertext. An alternate protocol specifically for Paillier can be found in [11].

$PoPK(\tau, \rho)$:

1. Given a set of ciphertexts, $\{C_k = E_P(x_k, r_k)\}_u^{k=1}$, where $E_P(x_k, r_k)$ is a Paillier encryption of plaintext, x_k, and randomness, r_k, with prover's public key. We define vectors $c = (C_1, \ldots, C_u)$ and $x = (x_1, \ldots x_u)$ and the matrix $R = (r_1, \ldots r_u)$, where r_k are rows.
2. Prover constructs m $(2^{u-1+log(u)}\tau, 2^{u-1+log(u)}\rho)$-ciphertexts: $\{A_i = E_P(y_i, s_i)\}_{i=1}^m$ and send them to the verifier. We define vectors $a = (A_1, \ldots, A_u)$ and $y = (y_1, \ldots y_u)$ and the matrix $S = (s_1, \ldots s_u)$, where s_k are rows.
3. Verifier selects a random vector $e = (e_1, \ldots e_u) \in \{0, 1\}^u$, and sends it to the prover.
4. Prover computes and sends $z = y + M_e x$ and $T = S + M_e R$ to the verifier, where $M_e =$.
5. Verifier accepts the proof if $a + M_e c = \{E_P(z_1, t_1), \ldots E_p(z_m, t_m)\}$, $|z_i| \leq 2^{u-1+log(u)}\tau$ and $|t_i| \leq 2^{u-1+log(u)}\rho$.

Figure 3.9: Proof of plaintext knowledge

$PoCM(\tau, \rho)$:

1. Given sets of ciphertexts $\{(A_k, B_k, C_k)\}_{k=1}^u$, where $A_k = E_P(a_k, h_k)$, $C_k = a_k B_k + E_V(r_k, t_k)$ and $E_V()$ is a Paillier encryption using the verifier's public key.
2. Prover constructs u random $(2^{3u-1+log(u)}\tau, 2^{3u-1+log(u)}\rho)$-ciphertexts: $\{D_k = E_P(d_k, s_k)\}_{k=1}^u$
3. Prover also constructs u ciphertexts: $F_k = d_k B_k + E_V(f_k, y_k)$, where $E_V(f_k, y_k)$ are random $(2^{4u-1+log(u)}\tau^2, 2^{4u-1+log(u)}\tau\rho)$-ciphertexts.
4. Prover sends D_k and F_k to the verifier.
5. Verifier selects a random e_k, and sends it to the prover.
6. Prover computes $z_k = d_k + e_k a_k$, $v_k = s_k + e_k h_k$, $x_k = f_k + e_k r_k$ and $w_k = y_k + e_k t_k$.
7. Prover sends $\{(z_k, v_k)\}_{k=1}^u$ and $\{(x_k, w_k)\}_{k=1}^u$ to the verifier.
8. Verifier accepts the proof if $D_k + e_k A_k = E_P(z_k, v_k)$, $F_k + e_k C_k = z_k B_k + E_v(x_k, w_k)$, $|z_k| \leq 2^{3u-1+log(u)}\tau$, $|v_k| \leq 2^{3u-1+log(u)}\rho$, $|x_k| \leq 2^{4u-1+log(u)}\tau^2$ and $|w_k| \leq 2^{4u-1+log(u)}\tau\rho$
9. repeat for every k.

Figure 3.10: Proof of correct multiplication

Setup phase

The first step in our privacy-aware solution is to set up the tools we would need to perform secure computations. This includes the ability to generate the public and private keys as well as single and triple shares. For better clarity on the description of the proto-

Share: ([x])

1. Each participant, P_i, randomly chooses x_i and sends its encryption $E_{P_i}(x_i)$, which are (τ, ρ)-ciphertexts, to all other participants.
2. Each pair of participants, P_i and P_j, use $PoPK$ on $E_{P_i}(x_i)$ to prove that they are correct and well formed.
3. Each participant P_i holds x and the randomness, r, used to generate $E_{P_i}(x_i)$ as private values.

Figure 3.11: Subprotocol to create additive shares of a random value

cols, we will omit the number of variables, u, which was included previously to highlight that many instances of the function can be performed in parallel, such as the creation of many single shares or proving correct multiplications on a collection of triples. Single shares are representations of a value shared among all participants. We denote $[a]$ as the value a shared among all participants along with cryptographically secure MAC. This enables secure sharing of participant's inputs. Triple shares are representations of values a, b, c with a multiplicative relationship, i.e. $c = ab$. They are similarly accompanied with MAC to ensure correctness. Zero-knowledge proofs are used to ensure the computations are performed honestly. We denote $([a], [b], [c])$ a triple share. This enables secure multiplication. Figure 3.15 shows the functions required during setup.

Generation of private additive shares is the most basic operation required, shown in Figure 3.11. The shares are kept private until a result needs to be revealed. Message authentication codes (MAC) provide the mechanism to detect dishonest participants and ensure correct computations. Figure 3.14 shows the steps to acquire message authentication codes on additive shares. Note that α_j^i is kept consistent for all variables shared a pair of participants while $\beta_{a_j}^i$ varies. This design allows us to update the MAC consistently throughout operations in the compute phase. Figure 3.12 show the subprocess required to obtain additive shares of a product of two values between two participants. This pairwise operation is used as a subprocess to compute multiplicative sharings between all parties. Figure 3.13 lists the steps required to perform a multiplication of two values shared between all participants, based on the conversion of a product of sums to a sum of products and using $2mult$ to perform the latter.

Note that the setup phase can be performed without knowing the participants' inputs or the functions they need to compute. For example, the setup phase may be performed when a user sign up for the cloud storage service. Then, when a user uploads a file, the service proceeds to the compute phase.

2mult(x,y): ($z = xy$)

1. Participant, P_i, computes and sends $c = xE_{P_j}(y) + E_{P_j}(r)$ to P_j, where $E_{P_j}(r)$ is a random $(2^{3u+log(u)}\tau^2, 2^{3u+log(u)}\tau\rho)$-ciphertexts.
2. Participant, P_i, use $PoCM$ to prove that c is constructed correctly.
3. Participant, P_j, decrypts c to obtain his share z_j and P_i sets his share $z_i = -r$.

Figure 3.12: Subprotocol to compute a two-party sharing of the product of their encrypted inputs

nmult($[a]$,$[b]$): ($[c] = [a][b]$)

1. Each participant, P_i computes $c'_i = a_i b_i$.
2. Each pair of participants, P_i and P_j, use $2mult()$ with input $E_{P_i}(a_i)$ and $E_{P_j}(b_j)$ and adds the outputs to their c'_i and c'_j respectively: $c_i = c'_i + z_i$ and $c_j = c'_j + z_j$.
3. Each participant, P_i, uses $Share$ except, instead of selecting random x_i, he uses c_i. This results in a shared product $[c]$, with each participant holding c_i.

Figure 3.13: Subprotocol to compute a sharings of a product of two other shared values

InitMAC:

1. For each pair of participants, P_i and P_j, P_i chooses a random α^i_j and send a (τ, ρ)-ciphertext, $E_{P_i}(\alpha^i_j)$ to P_j.
2. P_i then runs $PoPK$ with $E_{P_i}(\alpha^i_j)$ as input and with P_j as verifier.

AddMAC:

1. To add MAC to a shared representation of a, i.e. $\{a_1, a_2 \dots a_u\}$, each pair of participants, P_i and P_j use $2mult()$ with input $E_{P_i}(\alpha^i_j)$ from P_i and $E_{P_j}(a_j)$ from P_j. This results in P_i holding $z_i = -\beta^i_{a_j}$ and α^i_j. P_j holds $z_j = \alpha^i_j a_j + \beta^i_{a_j}$, where $\alpha^i_j a_j = z_i + z_j$.
2. This results in P_i holding a MAC key, $K^i_{a_j} = (\alpha^i_j, \beta^i_{a_j})$, which can be used to verify P_j's MAC, $MAC(a_j) = z_j$.

Figure 3.14: Subprotocol supporting message authentication codes (MAC)

Compute phase

Once the setup is complete, each participant holds single shares of random values and shares of triples representing products, all with verifiable MAC's. Each participant now has the ability to verify all other participants' shares to ensure they are correct. The objective in the compute phase is to enable the participants to perform additions and multiplications of inputs that the participants select using the collection of singles and

InitKeyMac:

1. Each participant generates a (pk, sk) pair and publishes his public key, pk.
2. Run $InitMAC()$ to create authentication keys.

Singles: $[a]$

1. Run the $Share()$ protocol to create shares of a random a.
2. Run $AddMAC()$ to obtain committed $[a]$.

Triples:$([a], [b], [c])$

1. Run $Share()$ 4 times to obtain $[a], [b], [f], [g]$.
2. Run $nmult([a], [b])$ and $nmult([f], [g])$ to obtain $[c]$ and $[h]$, where $c = ab$ and $h = fg$.
3. Run $AddMAC()$ on $[a], [b], [c], [f], [g], [h]$.
4. The parties verify that $[a], [b], [c]$ are correct representation of the product of a and b producing c, using $[f], [g], [h]$:
   (a) Parties select a random e.
   (b) They then compute $\epsilon = e[a] - [f]$ and $\delta = [b] - [g]$, and open ϵ and δ to all parties.
   (c) Having ϵ and δ, they now compute and open $e[c]-[h]-\delta[f]-\epsilon[g]-[f]-\delta\epsilon$ to all parties.
   (d) If the result is zero, $([a], [b], [c])$ is added to the collection of triples.

Figure 3.15: Setup functions

triples while updating the MAC to ensure consistency.

We first describe some basic operations or subprotocols on shares, shown in Figure 3.16. Opening of a shared representation, a, to a participant, P_i, is to reveal the plaintext of a, to P_i. Opening to all participants is to reveal the value to all participants while having every participant verify the correctness of all shares. Addition of two shared representation is simply the additions of their shares and MAC's. Multiplication of a value by a public constant similarly involves multiplying the shares and MAC's by the constant. To add a constant to a shared representation involves one participant adding the constant to his share and all other participants updating their MAC keys to maintain consistency.

With these, we now have all the tools to completely describe the secure multi-party computation protocol against dishonest participants. The setup phase involves the key generation and creating enough committed singles and triples for the functions that the participants would need for the computations. In the compute phase, a participant can create shares of his inputs by computing the offset of a shared random value generated during setup and denoting it as a variable, say x. Addition of two variables is

Open:

1. To open a shared representation to a participant, P_i, each participant, P_j, sends his share a_j and $MAC(a_j)$ to P_i.
2. P_i, who holds the MAC key, $(\alpha^i_j, \beta^i_{a_j})$, verifies that $MAC(a_j) = \alpha^i_j a_j + \beta^i_{a_j}$.
3. If the equality holds for all j, he computes and obtain the value for $a = \sum a_i$.
4. To open a shared representation to all participants, repeat for every participant, P_i.

Add: $([c] = [a] + [b])$

1. To add shared representations, a and b, each participant, P_i, computes c_i=a_i+b_i and updates the MAC keys and MAC as follows: $K^i_{a_j} = K^i_{a_j} + K^i_{b_j} = (\alpha^i_j, \beta^i_{a_j} + \beta^i_{b_j})$ and $MAC(c_j) = MAC(a_j) + MAC(b_j)$

AddConstant: $([b] = [a] + \delta)$

1. To add a constant δ to a shared representation, $[a]$, a participant, say P_1, adds δ to his share a_1.
2. All participants, P_j, update the MAC keys by replacing $\beta^i_{a_j}$ with $\beta^i_{a_j} - \delta\alpha^i_j$.

MultConstant: $([b] = \delta[a])$

1. All participants multiply their shares, MAC keys and MAC's by δ: $a_j \Rightarrow \delta a_j$, $\beta^i_{a_j} \Rightarrow \delta\beta^i_{a_j}$ and $MAC(a_j) \Rightarrow \delta MAC(a_j)$

Figure 3.16: Computing over shared $[a]$ representations, including MAC updates

straightforward.

Multiplication of two shared representations, $[x]$ and $[y]$, is performed by considering the offsets of the representation against an established triple. First, we write $x = a + \epsilon$ and $y = b + \delta$. Then, we have:

$$xy = (a + \epsilon)(b + \delta) = ab + \epsilon b + \delta a + \epsilon\delta. \tag{3.12}$$

Given a triple $([a], [b], [c])$, we can compute $\epsilon = [x] - [a]$ and $\delta = [y] - [a]$. The first term is given by $[c]$. Since x and a remains privately shared, ϵ and, similarly δ, can be opened to all parties. The second and third term can then be computed by multiplying a public constant. The fourth term is an addition by a constant. Figure 3.17 summarizes all the operations to perform secure multi-party computation.

Secure copyright detection protocol against dishonest participants

With the secure AMPC protocols, we can now describe our solution for secure watermark detection. There are three parties involved in our solution. The data holder holds the media under test. The copyright claimant holds the watermark. The cloud service provider is providing a data storage and distribution service, and is prohibited by law

Init: (Setup phase)

1. The participants run $InitKeyMAC()$.
2. Then, run $Singles()$ and $Triples()$ enough times to allow for the number of additions and multiplications required in the application. Note that more can be generated as needed, so long as the bound are respected in computations

Input:

1. To create shares of a participant's input, a random single, $[a]$, is opened to the participant.
2. Having learned a, he computes and sends $\delta = x_i - a$ to all participants
3. All parties perform $[x_i] = [a] + \delta$.

Add:

1. Compute $[z] = [x] + [y]$

Mult: ($[z] = [x][y]$)

1. Parties select a random triple $([a], [b], [c])$
2. Open $\epsilon = [x] - [a]$ and $\delta = [y] - [b]$ to all parties
3. Compute $[z] = [c] + \epsilon[b] + \delta[a] + \epsilon\delta$
4. remove $([a], [b], [c])$ from the list of triples

Output:

1. To output $[z]$ to a participant, open $[z]$ to that participant.

Figure 3.17: Arithmetic multi-party computation (AMPC) protocols

from hosting copyrighted content.

After setup, the first step is to generate the inputs representing the image and the watermark. The media and watermark are converted, if necessary, to the required representation, such as DCT, wavelet or spatial. If the inputs are not integers, scaling is performed. Then, shared representations of the media and watermark are obtained using $Input()$. The core computation in the detection algorithm is the scalar product of the image and watermark, consisting of multiplication and addition of shared values. Once a shared correlation value is obtained, the threshold value is subtracted from it so that a positive value indicates that it is higher than the threshold. Two random values are then generated and used to hide the actual correlation value while maintaining its sign. Further discussion on these two random values is found in section 3.3.6. Finally, the sanitized correlation value is revealed to all participants and a positive value indicates a positive detection. If the media is found to be legitimate, encrypted shares of data-holder's inputs are added up and stored. Proofs of knowledge and correct multiplication are used on each sharing and computation to ensure honesty of participants. Message authentication code is also used by every pair of participants to ensure that the final

Using the functions from AMPC protocol:

1. Data holder, Watermark holder and Cloud service provider run $Init()$. This represents the setup phase.
2. Data holder uses $Input(X)$ to generate a shared representation of his image, $X = \{x_1, \dots x_N\}$.
3. Watermark holder uses $Input(W)$ and $Input(-N\delta_{th}+1)$ to generate a shared representation of his watermark, W, and the correlation threshold, $N\delta_{th} - 1$.
4. All parties compute the scalar product of data holder and watermark holder's inputs using $[xw_{prod,i}] = Mult([x_i], [w_i])$ for all i and $Add([xw_{prod,1}], \dots [xw_{prod,N}])$ to compute the correlation value, $corr = \sum_{i=1}^{N} xw_{prod,i}$.
5. Cloud service provider chooses two random positive values: $mixA$ and $mixB$, where $mixA \geq MixA_{min} \geq 2$ and $mixA > mixB \geq 0$. Then, use $Input(mixA)$. and $Input(-mixB)$.
6. All parties compute $[corr_s] = Add([corr], [-N\delta th + 1])$ and then $[result] = Add(Mult([corr_s], [mixA]), [-mixB])$.
7. $[result]$ is opened to all parties. A positive value indicate copyright infringement has been detected, i.e. $corr > N\delta_{th}$, and a negative value indicate otherwise.
8. If the media is deemed legitimate, the cloud service provider adds up the shares of the media, $\sum E_{pk}(x_i)$, and stores as user's file.

Figure 3.18: Protocol for copyright detection with semi-honest cloud

output is correct. That is, the correlation value must be verified by all three parties to be accepted. Figure 3.18 shows our base protocol, which is secure against dishonest data/watermark holders and semi-honest cloud.

Hiding the correlation value

The correlation value is hidden to prevent data holder or watermark holder from deducing each other's inputs. If the correlation value, ρ, is known, either party can construct an equation with s unknown, where s is the size of the media or watermark. Suppose the data holder wishes to deduce the watermark, he knows:

$$N\rho = x_1w_1 + x_2w_2 \dots x_sw_s \tag{3.13}$$

where N is the size of the media, x_i is a pixel or coefficient of the media and w_i is the corresponding element of the watermark. Since he knows X, he can compute ρ, given s equations of the same form with different X's. That is, uploading s media and obtaining their correlation against the same watermark would allow the data holder to solve for the watermark. Therefore, $mixA$ and $mixB$ are used to randomize the correlation without

changing its sign. Figure 3.18 shows a construction where the cloud generates $mixA$ and $mixB$ as required. Since there are conditions on $mixA$ and $mixB$ but no mechanism is available to ensure that the cloud service providers follow, we denote this protocol as secure against dishonest data holder and watermark holder with a semi-honest cloud. $MixA_{min} \geq 2$ should be chosen to be large enough to provide adequate security and hide the true correlation value. Note that a trusted fourth party can also be used in place of the cloud to generate these values.

Note that the secure computation protocol cannot detect participants lying about their inputs. That is, a user providing false media, a copyright claimant providing a false watermark or the cloud service provider generating a $mixA$ or $mixB$ that do not follow the protocol are scenarios that the protocol do not consider. However, should the participant provide false inputs, they must continue in every step until the end. In the user's case, providing a false media would lead to the false media being accepted for storage, which does not benefit the user. In the copyright claimant's case, providing a false watermark yields no information on the user's media, provided no collusion occurs between the participants, and would fail to detect copyright infringement. In the cloud service provider's case, falsifying $mixA$ or $mixB$ could lead to a false detection result, but yield no information on the watermark or media.

However, there are scenarios where the cloud service provider may benefit in falsifying $mixA$, $mixB$ and consequently the correlation value. A cloud storage provider whose primary clients are illegal media distributors may want to purposely select a small $mixA$ and large $mixB$, where $mixB \gg mixA$ contrary to the protocol, to lower the correlation result to ensure most media go through, legitimate or otherwise. We propose an alternative construction that ensures $mixA$ and $mixB$ satisfy the required conditions using only random shares generated in setup. We ensure values are positive through squaring and achieves the condition of $A > B$ through the relationship between the sum of squares and the difference of squares.

We require

$$mixA > mixB \geq 0 \;\; \& \;\; mixA > MixA_{min}. \tag{3.14}$$

To satisfy the first condition, we use the following relation:

$$mixA' = (a^2 + b^2)^2 > mixB = (a^2 - b^2)^2 \geq 0 \tag{3.15}$$

where a and b are arbitrary integers. The condition is true for any values of a and b, which means they can be taken from any shared representations generated during setup.

Using the functions from AMPC protocol:

1. Data holder, Watermark holder and Cloud service provider run $Init()$. This represents the setup phase.
2. Data holder uses $Input(X)$ to generate a shared representation of his image, $X = \{x_1, \dots x_N\}$.
3. Watermark holder uses $Input(W)$ and $Input(-N\delta_{th}+1)$ to generate a shared representation of his watermark, W, and the correlation threshold, $N\delta_{th} - 1$.
4. All parties compute the scalar product of data holder and watermark holder's inputs using $[xw_{prod,i}] = Mult([x_i], [w_i])$ for all i and $Add([xw_{prod,1}], \dots [xw_{prod,N}])$ to compute the correlation value, $corr = \sum_{i=1}^{N} xw_{prod,i}$.
5. All parties select two random values $mixA$ and $mixB$, where $mixA > mixB \geq 0$ and $mixA > MixA_{min}$, by performing the following:
   (a) Select two random single shared values from the collection of singles. We denote them $[randA]$ and $[randB]$.
   (b) Compute $[randA^2] = Mult([randA], [randA])$ and $[randB^2] = Mult([randB], [randB])$ to obtain sharings of their squares.
   (c) Compute $[sumA] = Add([randA^2], [randB^2])$ and $[diffB] = Add([randA^2], [-randB^2])$, where $[-randB^2] = -1 * [randB^2]$.
   (d) Compute $[mixA] = Mult([sumA], [sumA]) + MixA_{min}$, where $MixA_{min}$ is a public constant, and $[mixB] = Mult([diffB], [diffB])$
6. All parties compute $[corr_s] = Add([corr], [-N\delta th + 1])$ and then $[result] = Add(Mult([corr_s], [mixA]), [-mixB])$.
7. $[result]$ is opened to all parties. A positive value indicate copyright infringement has been detected, i.e. $corr > N\delta_{th}$, and a negative value indicate otherwise.
8. If the media is deemed legitimate, the cloud service provider adds up the shares of the media, $\sum E_{pk}(x_i)$, and stores as user's file.

Figure 3.19: Protocol for copyright detection against all dishonest participants

The second condition is equivalent to $mixA - MixA_{min} > 0$. Since $mixA > 1$ from the first condition, adding $MixA_{min}$ to any positive value would yield a valid solution for $mixA$. Combined with the first condition, we have:

$$mixA = (a^2 + b^2)^2 + MixA_{min} > MixA_{min} \tag{3.16}$$

Figure 3.19 illustrates our privacy-aware copyright detection protocol against dishonest participants.

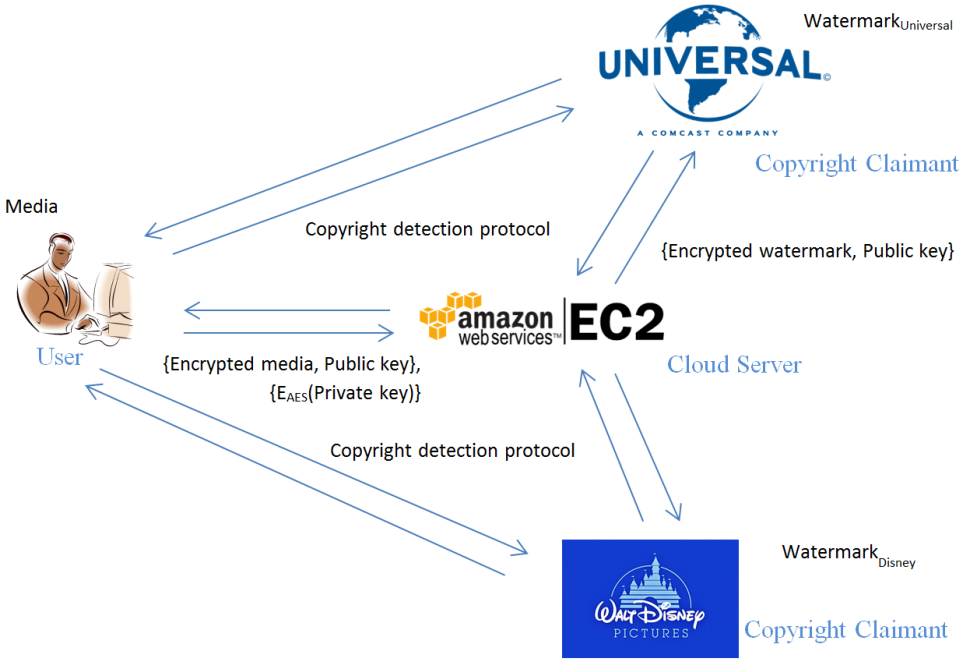

Figure 3.20: Copyright detection with multiple copyright claimants

3.3.7 Example with multiple copyright claimants

In practise, many different organizations generate media and may want to limit distribution (copyright detection), trace the source or even track the popularity of its media. The same techniques discussed in this chapter can be used in each case, except media is not rejected when watermarking is used for the purpose of data mining as in the last case.

Figure 3.20 shows a setup where many organizations work with the cloud service provider to ensure legitimacy of the data it stores. When a user uploads a media, he also uploads his public key, which is distributed to all copyright claimant servers. Each copyright claimant engages in the protocol illustrated in Figure 3.19 with the user and the cloud service provider. The user may use the same public key in every instance. At the end, if all copyright claimants are satisfied that the media is not copyrighted, the cloud service provider sums up the encrypted shares and stores the symmetrically encrypted private key, $E_{AES}(sk)$, that the user sends upon acceptance.

3.4 A framework for secure and privacy-aware copyright detection in cloud storage system

Our protocol can detect on copyright infringement on individual media. However, in practise, users upload many media and copyright claimants would have many copyrighted works and potentially many watermarks. On its own, our protocol would require performing detection on every pair of media and watermarks on every user-uploaded file, which is fairly inefficient.

We propose a copyright verification framework, shown in Figure 3.21, which addresses the efficiency issue by separating into two steps:

1. Candidate identification
2. Copyright detection

We first perform a secure media (image/video) search against a database of copyrighted media. When a copyright claimant releases a watermarked media, it also creates an encrypted feature sets associated with the watermark. When a user uploads a media, he first engages in a secure search of his media's feature sets over the collection of copyrighted media feature sets. If any matches were found, the copyright verification protocol would follow on only the candidate media identified. Since feature sets are much smaller than the media itself, the solution would be more efficient than a brute-force verification of all media.

The content-based searching algorithms described in section 3.2 are applicable in our framework. As discussed, feature sets are compared using Euclidean distance. In particular, equation 3.2 can be computed using the secure AMPC protocol from Figure 3.17. The second term is a scalar product of the feature sets. Any copyrighted media whose Euclidean distance with user's media is below a certain threshold is retained as candidate to proceed to copyright detection presented in section 3.3.6.

Since feature sets are used to identify the copyrighted works that the user uploads, a non-blind detector with its higher accuracy, may even be possible if the feature sets are able to accurately identify the copyrighted works. However, due to the searching algorithms' inability to ensure that the encrypted feature sets provided by user belong to the encrypted media, our framework works only in the semi-honest model for all participants.

Depending on the application's security requirement, feature sets may be considered to contain negligible information on the original media and allowed to be publicly released. In which case, a search over plaintext would be far more efficient and accurate.

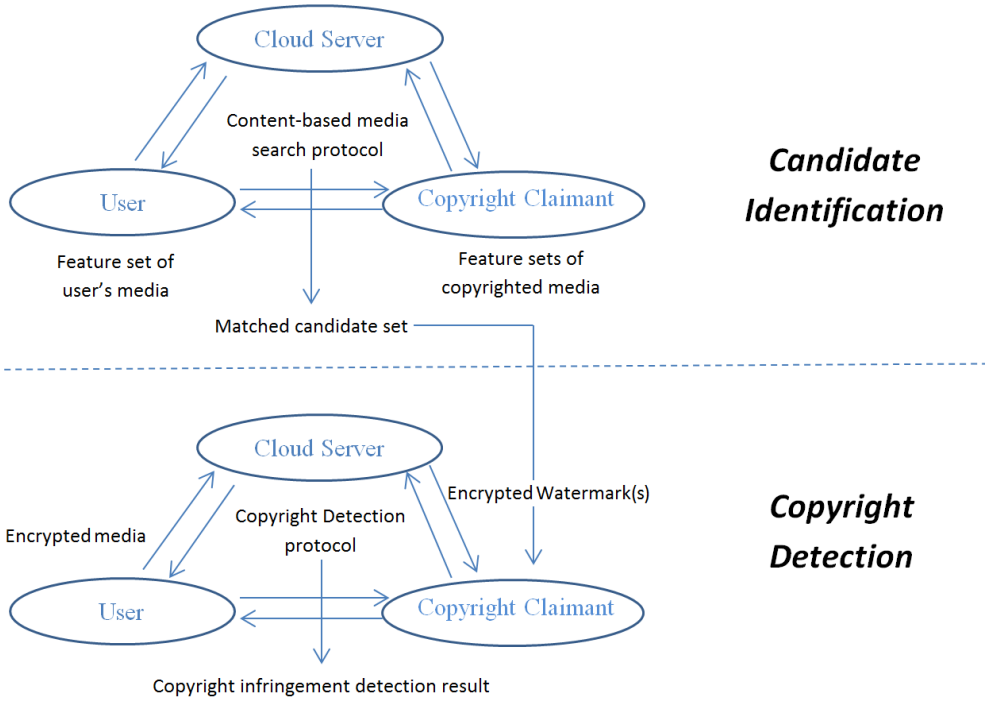

Figure 3.21: Framework incorporating search and copyright detection

The feature sets of copyrighted media can be stored on cloud server and candidate search needs not involve the copyright claimant. A non-blind detector would greatly improve performance.

In this chapter, we presented a digital rights management solution for encrypted cloud storage where the privacy of users and copyright claimants are both protected. Copyright infringement is detected using digital watermarking over encrypted pixel or coefficient sequences. A construction was provided for the practical setting where the user, cloud and copyright claimant may be malicious. Our solution provides an automated DRM solution, targeted at cloud storage providers such as Dropbox and Mega, where the need to respect copyright laws presents a constant legal threat and a resource-consuming burden. In the next chapter, we will present an application of search over encrypted binary data sequences.

Chapter 4

Data

While text and media represent the most common form of outsourced data where privacy and security are of importance, there are scenarios where privacy-protected search and computing over generic data is of interest.

In this chapter, we present a novel solution to malware scanning where an Anti-virus service may be performed with different levels of privacy protection [56]. At the highest level, virus can be scanned without knowledge of the data being tested except in the event of a positive match. It is also another example where computation and search are combined as in section 2.8, except over binary data. It should be noted that our technique can be generalized as an approach to match any patterns in encrypted data, and is not restricted to the application of malware detection.

Note this work has also appeared in:

- H. Poon and A. Miri. Scanning for viruses on encrypted cloud storage. In *IEEE Conference on Cloud and Big Data Computing*, pages 954–959, 2016

4.1 Scanning for Viruses on Encrypted Cloud Storage

We look at a problem that has yet to be examined by the community: that of malware detection on encrypted data. The issue is particularly relevant where organizations use cloud storage to archive data or to back up their systems. It is not unusual that companies would archive a significant amount of data in case the information becomes needed again in the future and such data may not be accessed for long periods of time. While many malwares are identified soon after released, some remain dormant long before they are discovered. Should the archive contain malwares, retrieval could activate them. Similarly, restoring a system to a backup containing malwares would also be problematic.

In these scenarios, a methodology to detect malwares in the encrypted archive would be invaluable.

Furthermore, anti-virus companies often share information on newly discovered malwares to aid in protecting users against the latest threats. The shared information was always considered trusted. Recently, it has emerged that a reputed anti-virus company may have intentionally injected false positives in an attempt to harm the reputation of its competitors, resulting in a spike of legitimate files being identified as malware in the early 2000s [47]. Such malicious behavior from a partner was not considered in the past. One of the main reasons that a malicious party was able to launch such an attack was due to the direct access to the competitors' anti-virus programs and their malware database. This allows them to reverse engineer the software to determine and to mislead their identification algorithms. Note that malware writers have also always been able to do the same to verify that their latest malwares can avoid detection or to be alerted when their released malware has been identified. All are good reasons why an anti-virus tool can be invaluable when offered as a cloud service, where the detection algorithms and virus database can be kept private and server-side updates are seamless and invisible to public.

In this chapter, we present two schemes for performing virus scanning over encrypted data: a private cloud solution, where the data owner possesses the anti-virus tools and database, and an anti-virus as a service solution, where the data owner requests the scanning service from the anti-virus company, which controls the anti-virus tools and database. We also describe how the service can be performed over unencrypted data.

4.1.1 Background

Modern malware identification techniques include signature-based detection, heuristics-based detection, behavioral-based detection (including Sandbox detection) and data-mining techniques (AI). While each technique has its strengths and flaws, combining them has been an effective strategy in practice. A malware's standard signature often refers to the file's hash signature, e.g. The MD5 hash of an executable file. While its detection capability is limited on its own due to the proliferation of polymorphic viruses, it is still one of the most reliable tools available and is highly effective when combined with other identification techniques. The combination of the various properties, behavioral and structural, of a malware is often referred to as a generic signature, which is one of the most commonly used identification techniques in anti-virus software today.

Our solutions for encrypted cloud storage will be restricted to detecting malware based on structural rather than behavioral properties. More specifically, in addition

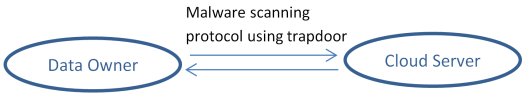

Figure 4.1: Communication model for two-party malware scanning over encrypted data

to being able to identify perfect matches of malware through standard signatures, the solutions allow for detection of sequences of potentially malicious computer instructions, which may appear at different positions in a file, as in generic signatures. Malware that evolves and has several versions in the wild typically exhibits such properties. By extension, our schemes also allow the use of wildcards.

Our proposed solution in a private scanning model relies on the use of encrypted indexes, originating from the field of keyword search over encrypted data.

While efficient, the private model can be limiting as the data owner must manage the anti-virus tools and perform the scan using his own resources. Alternatively, we propose an anti-virus as a service model where the malware scanning tools and database are maintained by the anti-virus company and the data owner requests to have the scan performed. Our solution in this model is based on the use of homomorphic encryption. In particular, a demonstration of our scheme based on Paillier's cryptosystem will be presented.

4.1.2 Private scanning of malware over encrypted data

In many cases, an organization is the sole user accessing and writing data onto the cloud server. We describe here an efficient solution to provide malware scanning capability in this scenario, where security concerns are restricted to that of the cloud operator.

Communication Model

Our communication model involves two parties, as shown in Figure 4.1, where the data owner encrypts and uploads the data to the cloud server and subsequently requests virus scanning by following a communication protocol. The data owner is assumed to have a set of virus definition consisting of standard signatures such as MD5 or SHA-256 and generic signatures, restricted to structural properties, describing snippets of malicious codes.

In terms of security, we assume the cloud operator to be semi-honest, following our protocol without deviation, but is interested in learning information on the stored data. The main requirement of our scheme is that the content of the files stored on the cloud

server remains private and no information is leaked as a result of the malware scanning setup and protocol.

Malware scanning protocol

Our proposed solution is based on encrypted indexes, using techniques similar to [53] and [27], and allows for standard and generic signature (with wildcards) based detections. Briefly, two indexes are used: A block-to-file index, I, and a block location index, I_L. The block-to-file index enables a coarse scan to quickly identify potentially infected files in the data set while the block location index allows for detailed scan on individual files for virus identification. Our scheme uses symmetric encryption to protect the content of the files and indexes. During setup, N files, forming the data set to be uploaded, are parsed and indexed as n-bit blocks, resulting in a block-to-file index and N block location indexes. The indexes and files are then encrypted and uploaded onto the server. To perform a virus scan request, the data owner first queries the block-to-file index, I, for files containing instruction blocks corresponding to virus definitions followed by a query on I_L to determine code sequence matches. The queries sent to the cloud service provider contain encrypted blocks while the returned entries are also encrypted. To speed up standard signature detection, a hash signature, such as SHA-256, is attached at the beginning of every file. Our scheme achieves the required functionalities without additional complexity in terms of storage, communication and computational cost when compared to search schemes for encrypted cloud storage [79].

A detailed description of the algorithms is as follows. A file collection, $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}$, is parsed for n'-bit blocks, x_j. A block-to-file index, I, is generated mapping blocks to files such that $I(x_j) = (d_1, d_2, \ldots, d_N)$, where $d_i = 1$ if x_j is found in the file and $d_i = 0$ otherwise. The resulting index is encrypted and uploaded to the cloud server:

$$I(H_K(x_j)) = E_K(d_1, d_2, \ldots, d_N). \tag{4.1}$$

In addition, a block location index, $I_{L(i)}$, is generated for each file:

$$I_{L(i)}(H_K(\mathcal{D}_i|x_i)) = E_{K(x_i)}(j_1, j_2, \ldots, j_{N_{x_i}}), \tag{4.2}$$

where $H_K()$ is a cryptographically secure hash function and j_x are the locations of x_i in the file with identifier, $\mathcal{D}_i$.

To perform a standard signature scan for a malware with a n_h-bit hash signature, $Sig(y) = (y_1, y_2 \ldots, y_{n_h/n'})$, the user begins by sending the keyed hash of the signature, $H_K(Sig(y))$, to the cloud. The corresponding encrypted index entries in I are returned

to the data owner. A set of candidate files, $\mathcal{D}_C$, are then found by decrypting the entries and identifying their intersection:

$$D_K(I(H_K(y_1))) \,\&\, D_K(I(H_K(y_2))) \cdots \,\&\, D_K(I(H_K(y_{n_h/n'}))), \tag{4.3}$$

where & denotes a bitwise AND operation. The data owner then sends $H_K(\mathcal{D}_i|Sig(y))$, where $\mathcal{D}_i \in \mathcal{D}_C$ to query the block location of the signatures. Upon receiving the encrypted location entries, $I_{L(i)}(H_K(\mathcal{D}_i|y_i)) = E_{K(y_i)}(j_1, j_2, \ldots, j_n)$ from the cloud server, the data owner decrypts and identifies files, $\mathcal{D}_i$, as matches if $1, 2 \ldots, n_h/n'$ are respectively in $D_K(E_{K(y_1)}), D_K(E_{K(y_2)}) \ldots, D_K(E_{K(y_{n_h/n'})})$. Note that multiple malwares can be verified simultaneously by sending multiple keyed hash signatures at the same time.

To perform a generic signature scan for a malware with the following characteristic code snippets $y_a = (y_1, y_2 \ldots, y_{q'})$ and $y_b = (y_1, y_2 \ldots, y_{q''})$, the user computes and sends $H_K(y_a \cup y_b)$ to the cloud. The cloud server returns the corresponding encrypted index entries to data owner, who then finds the set of candidate files from their intersection in the same way as when performing a standard signature scan. To determine the existence of the code snippets, the block locations are then queried by sending $H_K(\mathcal{D}_i|y_i)$ where $y_i \in \{y_a \cup y_b\}$ to the cloud server. Upon decrypting the location information, the location of the first block, y_1, in y_a is extracted. For each location, $Loc(y_1) = \{l_1, l_2 \ldots, l_m\}$, the data owner verifies if the following block is y_2. For each candidate found, $\{l_1 + 1, l_2 + 1 \ldots, l_{m'} + 1\} \in Loc(y_2)$, the data owner continues with the following block. The process iterates until the last block in the snippet is verified or until no candidates remain. The algorithm then proceeds in the same manner with y_b. Files where the locations of the code snippets appear as ordered are identified as matches. Note that a query with wildcard is equivalent to a query for multiple code snippets. For example, querying (y_a, wildcard$_{5 \text{ blocks}}$,y_b) is equivalent to querying y_a and y_b, with the additional step of ensuring the location of the first block of y_b is 5 blocks after the last block of y_a. Since there is no limit to the length of sequences or wildcard, this is equivalent to generic pattern matching.

An alternative approach for verifying code snippets that may require less communication would proceed as before for identifying candidate files. Instead of querying the locations of all blocks within candidates, a random block in the code snippets is queried, $H(\mathcal{D}_i|y_i)$. Given the locations, the owner returns

$$\{H(E_{K_{\mathcal{D}_i}, j_s}(y_1, y_2, \ldots, y_{q'})), i, j_s\}, \tag{4.4}$$

where i is the index of the matched file and j_s is the expected starting location of the

code snippet, for each match. $E_{K_{\mathcal{D}_i,j_s}}()$ represents the symmetric encryption of the code snippet at location j_s of file $\mathcal{D}_i$. The cloud then computes $H(E(x_{j_s}, x_{j_s+1}, \ldots, x_{j_s+q}))$, where $E(x_j)$ is the j^{th} stored block in file i. Matches are found where the following equality holds:

$$H(E(x_{j_s}, \ldots, x_{j_s+q'})) = H(E_{K_{\mathcal{D}_i,j_s}}(y_1, \ldots, y_{q'})). \tag{4.5}$$

The communication cost of the alternative approach depends on the frequency of the random block chosen. Instead of randomly choosing the block, one approach to ensure lower communication cost is to select the block with the lowest frequency in the file set. To do so, a block frequency list would be stored locally by the data owner. To allow for fast computation of encrypted block sequences, the data should be encrypted using a block cipher in counter mode.

The scheme is fairly efficient, requiring mainly decryption of index entries and lookup of sequences by the data owner. Any symmetric encryption algorithm can be used and security is easily observed since the index entries are encrypted as a whole. The setup and protocol are flexible and can also be used for keyword and phrase searches with proper choice of parameters. A hierarchical setup of indexes could also lead to better efficiencies.

4.1.3 Anti-virus as a service for encrypted cloud storage

While simple, the previous scenario requires the data owner to manage the malware scanning process, including the anti-virus software and database. Furthermore, an anti-virus software designed to run entirely client-side exposes the virus database and detection algorithms to the public, including malicious users which may use the information to aid in malware development or to disrupt normal operation of the anti-virus software.

Therefore, we propose the implementation of anti-virus as a cloud service, where only essential scanning is performed client-side and malware detection based on the scanning results is performed by the anti-virus company's server. Aside from simplifying the frequent updates currently required by anti-virus software, denying malicious parties access to a significant portion of the scanning algorithm would also hinder efforts to produce malware that evades detection and protect against reverse-engineering of the detection algorithms. In addition, our solution maintains user privacy by requiring the data stored on the cloud be encrypted under data owner's private key.

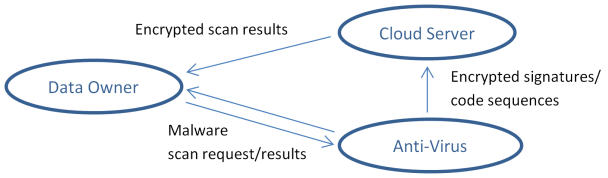

Figure 4.2: Communication model for three-party malware scanning service on encrypted data

Communication Model

We consider a three-party model, as shown in Figure 4.2, where the data owner encrypts and uploads the data to the cloud server and the anti-virus company offers a malware scanning service on the encrypted data. Virus scanning is performed by following a communication protocol. The anti-virus company controls the malware database and the detection algorithm. The data owner generally initiates by sending a scan request to the anti-virus server. Then, a set of encrypted signatures/code sequences are sent to the cloud for tests. The encrypted results are sent back to the data owner, who sends the decrypted scan results back to the anti-virus server for analysis.

In terms of security, we assume both the cloud operator and anti-virus service provider to be semi-honest, following our protocol without deviation, but are interested in learning information on the stored data. The main requirement of our scheme is that the content of the files stored on the cloud server remains private and no information is leaked as a result of the malware scanning setup and protocol except where malware matches are found.

Another highly desirable property is that the virus definitions and malware identification methodologies, which include the weighing of the various matches and combinations of results that would lead to positive identifications, are not leaked as a result of the scanning protocol.

Malware scanning protocol

Our proposed solution is based on homomorphic encryption, discussed in section 2.2.6. Without loss of generalization, we will describe the scheme using Paillier's cryptosystem. Briefly, the scheme works as follows. The data owner encrypts the data set using Paillier's cryptosystem and sends the result to the cloud. The public key is also uploaded with the data set. A hash signature, such as SHA-256, is attached to each file to enable standard signature verification. To perform a malware scan, the anti-virus company encrypts the

standard and generic signatures using the data owner's public key and sends them to the cloud for testing. The cloud computes the difference between the encrypted data and test sequences. Since the data and the hash signature/code snippets are both encrypted, the cloud gains no information on either and sends the results back to the data owner. Using the homomorphic property that $E(X)+E(-Y)=E(0)$ if $X=Y$, the data owner decrypts and sends the individual test results to the anti-virus server. Finally, the anti-virus server sends the malware scan results to data owner or cloud, depending on end user agreement.

A detailed description of the algorithms is as follows. We first generate the publickey, (n, g), and the private key, (λ, μ), for a Paillier cryptosystem which accepts n'-bit plaintexts. Given a file collection, $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}$, a hash signature, $Sig(\mathcal{D}_i)$, such as SHA-256, is computed and attached to each file. The file collection is then encrypted as n'-bit blocks, x_j. Recall that the ciphertext, c, for a given plaintext, m, is given by $c = g^m r^n \bmod n^2$, where $r \in \mathbb{Z}_n^*$.

To perform a standard malware signature scan, the anti-virus server encrypts the negation of a n_h-bit hash signature, $Sig(y) = \{y_1, y_2 \ldots, y_{n_h/n'}\}$ using the user's public key. This results in the encrypted signature:

$$C_{Sig(y)} = \{g^{-y_1} r_1^n, g^{-y_2} r_2^n \ldots g^{-y_{n'_h}} r_{n'_h}^n\}, \tag{4.6}$$

where $n'_h = n_h/n'$ is the number of blocks needed to represent the hash signature. The anti-virus server then sends the encrypted hash signature. For each file, the cloud storage provider computes

$$Q_{Sig(\mathcal{D}_i)} = \{Q_1, Q_2 \ldots Q_{n'_h}\} = C_{Sig(\mathcal{D}_i)} + C_{Sig(y)} \tag{4.7}$$

and homomorphically multiplies each block by a random value before passing to the data owner:

$$Q'_{Sig(\mathcal{D}_i)} = \{Q_1^{r_1}, Q_2^{r_2} \ldots Q_{n'_h}^{r_h}\} \tag{4.8}$$

where $r_i \in \mathbb{Z}_n$ are randomly generated. The data owner decrypts each result and a match is detected if

$$D(Q'_{Sig(\mathcal{D}_i)}) = \{0, 0 \ldots 0\}. \tag{4.9}$$

The scan results are sent back to the anti-virus server as a bit sequence

$$Q_{Std}(Sig(y)) = \{Q_{\mathcal{D}_1}, Q_{\mathcal{D}_2} \dots Q_{\mathcal{D}_n}\} \tag{4.10}$$

where $Q_{\mathcal{D}_i}$ represents a single bit and is set to 1 if a match is detected for $\mathcal{D}_i$ and is set to 0 otherwise. Based on the scan results, the anti-virus server determines if a malware is detected.

The previous step can be performed more efficiently at the cost of a small chance of false positive by aggregating the encrypted blocks. That is, compute

$$Q_A = \sum Q_{Sig(\mathcal{D}_i)} = \sum_{i=1}^{n'_h} Q_i \tag{4.11}$$

and returning $Q'_A = Q_A^{r_A}$ to the data owner, where a match is identified if

$$D(Q'_A) = 0. \tag{4.12}$$

The description on generic signature will proceed using this aggregated approach for clarity. Note that multiple malware can be verified simultaneously by sending multiple keyed hash signatures at the same time.

A generic signature verification with code snippets proceeds in the same manner using a sliding windows approach. For the following malware block sequence, $y_a = \{y_1, y_2 \dots, y_{q'}\}$, the anti-virus server encrypts the sequence using the data owner's public key to obtain

$$C_{y_a} = \{g^{-y_1} r_1^n, g^{-y_2} r_2^n \dots, g^{-y_{q'}} r_{q'}^n\} \tag{4.13}$$

and sends it to cloud server. For an encrypted file $C_{\mathcal{D}_i} = \{C_{x_1}, C_{x_2} \dots C_{x_{n_{\mathcal{D}_i}}}\}$, the cloud computes

$$Q(j_s) = \{Q_1, Q_2 \dots Q_{q'}\} = C_{x(j_s)} + C_{y_a}, \tag{4.14}$$

where $C_{x(j_s)} = \{C_{x_{j_s}}, C_{x_{j_s+1}} \dots C_{x_{j_s+q'-1}}\}$, for the starting block, $j_s = 1$ to $n_{\mathcal{D}_i} - q' + 1$. The results are then aggregated and

$$Q_A(j_s)' = (\sum Q(j_s))^{r_A} \tag{4.15}$$

is sent back to the data owner, who decrypts the results to determine matches where

$$D(Q_A(j_s)') = 0. \tag{4.16}$$

The individual results are sent to the anti-virus server as a bit sequence:

$$Q_{Gen}(\mathcal{D}_i, y_a) = \{Q_{\mathcal{D}_i}(1), Q_{\mathcal{D}_i}(2) \dots Q_{\mathcal{D}_i}(n_{\mathcal{D}_i} - q' + 1)\} \tag{4.17}$$

where $Q_{\mathcal{D}_i}(j_s)$ represents a single bit and is set to 1 if a match is detected for $\mathcal{D}_i$ at position j_s and is set to 0 otherwise. Based on the scan results, the anti-virus server determines if a malware is detected. Scanning for snippets with wildcards proceeds similarly. To verify $\{y_a, \text{wildcard}_{5 \text{ blocks}}, y_b\}$, 5 blocks are skipped at the end of y_a when computing equation 4.14. That is,

$$Q(j_s) = \{Q_1, Q_2 \dots Q_{q'+q''}\} = \{C_{x(j_s,a)}, C_{x(j_s,b)}\} + \{C_{y_a}, C_{y_b}\} \tag{4.18}$$

where

$$\begin{aligned} C_{x(j_s,a)} &= \{C_{x_{j_s}}, C_{x_{j_s+1}} \dots C_{x_{j_s+q'-1}}\} \\ C_{x(j_s,b)} &= \{C_{x_{j_s+q'+4}}, C_{x_{j_s+q'+5}} \dots C_{x_{j_s+q'+q''-2}}\} \end{aligned} \tag{4.19}$$

It is interesting to note that the cloud can only access the encrypted data, the encrypted code sequences and hash signatures under test. Without the user's private key, the cloud cannot learn their content. Similarly, the anti-virus server is never granted access to the encrypted data and only receives the test results in the form of a bit sequence representing a match versus non-match. An encrypted sequence test result, Q'_A, is first randomized such that no information is divulged except in the event that $Q'_A = 0$. Thus, other than the event of a match, little is revealed on the content of the data under test.

4.1.4 Anti-virus as a service for unencrypted cloud storage

Despite the promise of better user privacy, much of today's cloud storage providers do not provide encryption services where the private key is controlled by the data owner. In fact, most providers continue to work with unencrypted data due to efficiency and various functionalities that are available only data remains unencrypted. Nonetheless, anti-virus as a service would still be valuable in an unencrypted cloud for its ease in keeping the malware scanning tools up to date and its ability to hinder the viability testing of malwares and to prevent reverse engineering of detection algorithms. The scenario would also be interesting in applications where privacy is not of concern. Furthermore, detection can be performed based on behavioral in addition to structural characteristics of malwares, leveraging techniques that currently do not work on encrypted data.

Communication Model

The communication model for an anti-virus as a service is the same three-party model as in the encrypted case, shown in Figure 4.2. The data owner uploads the data to the cloud server and the anti-virus company offers a malware scanning service on the data. Virus scanning is performed by following a communication protocol. The anti-virus company controls the malware database and the detection algorithm. A client scanning software runs on the cloud server. Unlike the encrypted case, there are no privacy or security requirements on unencrypted data. The objective is to hide as much as possible the malware detection algorithm.

Malware scanning protocol

To illustrate the technique, consider the following set of rules:

- *sudo* followed by self-decryption and execution, (Behavioral), Weight = 2
- SHA-256 hash signature is $\{x, y \text{ or } z\}$, (Structural), Weight = 1
- Code snippet $\{cs_1, \text{wildcard}_{10\text{bytes}}, cs_2\}$, (Structural), Weight = 1

A score of 2 results in a positive malware match. In standalone anti-virus software, the entire sequence of tests is performed locally and visible to anyone monitoring the software. As a cloud service, the client performing the scan may relay the information that a *sudo* had been called, that a file with hash x' is found or a code snippet is detected, but only the anti-virus server could decide whether a positive malware match had occurred, based on the scan results. While our simple example may contain only a few rules, a practical anti-virus may perform hundreds of tests. A positive scan result may not necessarily be a factor that led to the positive match of the malware detected, and vice-versa. This separation of the anti-virus scanning process effectively turns the anti-virus server into a black box to a malicious user. The ease and invisibility of software updates can also alter the behavior of the anti-virus server without an outsider becoming aware. This dramatically complicates any reverse engineering efforts and attempts at testing malware viability without being detected.

It should be noted, however, that behavioral detection must be performed by the client scanner due to the time-sensitive nature of executing codes, although behavior deemed to be high-risk may be interrupted to await further instructions from the anti-virus server, which may include structural verification of the executing file/code/ram, or

permission for the executing code to continue to run under restricted conditions while monitoring further suspicious behaviors.

In this chapter, we explored the problem of malware detection on cloud services, and proposed three anti-virus solutions for cloud services. Our solution based on encrypted search provides an intuitive approach to perform malware detection on an encrypted cloud storage whose access is limited to the data owner. The scheme allows for any symmetric encryption algorithm to be used, with performance comparable to the leading keyword and phrase search algorithms.

Aside from user privacy, we also examined the disadvantages of implementing anti-virus as a software that performs scanning locally. In particular, the current approach exposes the virus database and detection algorithm, potentially aiding malware writers to evade detection and malicious agents to reverse engineer the detection algorithms. Therefore, we propose an anti-virus as a service solution, which, in addition to mitigating the aforementioned risks, also eases the frequent updates required for the critical service. Our solution is based on homomorphic encryption and demonstrated using Paillier's cryptosystem. Detection is performed in the encrypted domain, ensuring privacy.

While encryption leads to greater security and privacy, many valuable functionalities are currently not possible in the encrypted domain and unencrypted cloud services will continue to operate in the foreseeable future. Nonetheless, the merit of implementing anti-virus as a cloud service extends to unencrypted cloud services. In addition, operating over unencrypted data allows for behavioral detection that plays a significant role in anti-virus software today.

Chapter 5

Conclusion and Future Work

In this thesis, we explored techniques for searching and processing encrypted data, in particular, that of sequences.

Upon analyzing existing works in the area of encrypted text and phrase search, we identified several key challenges that needed to be addressed. It culminated into three main proposals for searching phrases in encrypted documents.

Our first solution from section 2.4, denoted our $\text{scheme}_{\text{sec}}$, describes a technique for splitting keywords, inspired from the properties of natural languages, to reduce wasted storage, computational and communication cost found in existing solutions. It maintains almost the same level of security as the current leading solution while drastically improve upon storage and communication cost. For our experimental corpus, we found a reduction of 65 % in cloud storage cost compared to the solution with the lowest storage [79] in the literature. When compared to the provably secure solution by [69], the storage cost is reduced by 95 % while sacrificing only small distinguisability between the number of distinct keywords per document in a corpus. The latter, being largely correlated with document size, corresponds to information that is available to the cloud operator anyways. Because all existing solutions relied on some form of false/random data, their storage, communication and computational cost all scaled with the security and corpus size. By reducing/removing the irrelevant data, our solution also significantly reduces communication and computational cost in addition to storage. By carefully selecting the encryption algorithm, it was also the first solution proposed in the literature that allows non-indexed keywords to be queried and includes basic ranking capability. Note that the degree of improvement depends on the property of the corpus and is summarized in Table 2.8.

Our $\text{scheme}_{\text{sto}}$ from section 2.5 exploits the space efficiency of Bloom filters to achieve

the lowest storage cost reported among all phrase search schemes. The scheme stores location information and keywords into same Bloom filter structures. We also discussed how incremental hash functions can be used to improve the performance cost traded for the storage reduction. The solution showed a further reduction of 65% in storage over our previous solution of section 2.4, an overall 87% reduction in cloud storage over the leading solution in literature. Due to the small representation of Bloom filters, the scheme also reduces communication cost. In place of encryption and decryption operations, a series of non-cryptographic hash functions are required instead. The scheme also supports non-keyword search and basic ranking.

Noting that all existing solutions had a relatively low response time, our final $\text{scheme}_{\text{spd}}$, described in section 2.6, proposes a new paradigm of viewing sequences as a collection of subsequences (n-grams) that may be queried independent of their locations. Relying on the unique properties of long textual sequences, the technique allows for a dramatic speed-up of existing phrase search techniques, reducing the protocol from the usual two to three rounds into a single round of communication while requiring only two Bloom filter verifications and cryptographic hash functions, as opposed to expensive cryptographic operations of most existing schemes. When compared to our $\text{scheme}_{\text{sto}}$, it requires far fewer hash function evaluations and achieves a much faster processing time. While the fastest implementation can require a high storage cost, the scheme can be adjusted to balance storage and response time as required by the application. In the lowest storage implementation, the scheme still enjoys much faster response time than existing works while requiring about 40 % less storage than our $\text{scheme}_{\text{sec}}$ and 70 % more than our $\text{scheme}_{\text{sto}}$. To summarize our findings, we also included an in-depth performance and security analysis of all schemes and supported our proposals with experimental results on a sample corpus. Tables 2.7, 2.8 and 2.9 give an overall comparison of all existing phrase search solutions in terms of storage, communication and computational cost.

Furthermore, we noted that a keyword location based phrase search scheme can effectively provide both search and audit capabilities to cloud services while satisfying the strong requirement of proof of retrievability with unbounded number of audits. Section 2.7 describes our solution which is applicable to our $\text{scheme}_{\text{sec}}$ and our $\text{scheme}_{\text{sto}}$. The scheme requires the same resources as a standard phrase search solution, requiring only a new protocol to perform auditing and without the use of tags and erasure codes traditionally used in auditing solutions. Aside from search, we also demonstrated an application in XML processing in section 2.8. The proposal uses symmetric encryption for phrase search and homomorphic encryption for values. This enables SQL-like queries in XML at a relatively low overhead computational cost to encrypt the documents as

the number of values in a XML is generally far lower than text.

Processing encrypted media presents an interesting challenge. A sequence of pixels or music notes contains far greater amount of information than texts. While an exact match is considered appropriate in text, media may be perceptually identical while quite different in description. Our work focused on the area of digital rights management in cloud storage services, where the need for user privacy and copyright law enforcement seem to contradict. To this end, we described the first privacy-aware watermark detection scheme for cloud storage service in section 3.3.6 that is secure against dishonest participants. The solution uses image processing, watermarking, homomorphic encryption, message authentication codes and zero-knowledge proofs to allow data holder, copyright claimants and cloud service providers to securely convert media into ciphertexts where watermark detection can be performed while ensuring that all parties are following the protocol as described. We also presented a complete practical framework in section 3.4 incorporating both content-based media search and copyright detection.

Finally, we present a novel application in section 4.1 based on search of generic bit sequences in the form of malware detection. Noting that existing anti-virus solutions often require super-user like control of the host and daily updates, Anti-virus companies could potentially gain control of or monitor any computers and data store on which it is installed. We consider alternative designs where user maintain greater control over the scanning process and, in particular, the privacy of his data. We described three solutions, based on private scanning, anti-virus as a privacy-aware service and as a public service. We showed that standard hash-based detection and code snippet scanning can be performed over encrypted data. Our solution for private scanning makes use of our phrase search protocols extended to binary data to achieve an efficient one to one anti-virus scanning. Then, we note that, with all the data being stored in cloud, an anti-virus can be performed as a service. We then described a three way protocol to performed virus scanning over encrypted data using homomorphic encryption. Our solutions protect user privacy such that file content is only revealed upon a match and only where the match occurs within the file.

5.1 Future work

During our research, we encountered many interesting problems we had hoped to, but were unable to explore or complete. We briefly discuss a few here which we believe are most promising.

5.1.1 Privacy-protected queries for DNA/Chromosome Sequence

Sequential data processing may find use in genomics, where individuals' DNA sequencing may post a privacy risk. Typical operations by medical professionals include locating genes at certain starting and ending positions, and specific start and end sequences such as ATG and TAA. Since certain chromosome sequences and genetic mutations are linked to diseases and medical conditions, data confidentiality is of high importance.

A notable difference between genetics and the data types considered in this thesis is the plaintext space and statistical distribution of the data. Unlike text and media, the number of possible chromosomes is much lower and has far lower diversity, which may require higher security to manage the heightened risk of statistical and brute force attacks.

5.1.2 Privacy-protected feature extraction and search scheme against dishonest participants

In chapter 3, we described a solution to perform the central functionality of copyright detection in a privacy-aware setting. While the algorithms work as intended, they do not scale well in practice. As the number of copyright claimants increase, the performance deteriorates since each uploaded media would have to compare against every claimant's watermark and perform the algorithm as many times as there are claimants.

The performance issue can be addressed by incorporating search as a first step to narrow down the set of watermarks that the uploaded media required testing against, as described in section 3.4. To do so, the copyright claimants upload sets of feature vectors for their works along with the watermarks. When a media is uploaded, a search is performed to identify similar copyrighted media. Finally, the detection algorithm is performed against the watermarks of copyrighted works identified to be most similar to the media under test. Different levels of computational cost and detection accuracy can be achieved by varying the threshold of similarity.

There are significant challenges to achieving this practical framework. While our detection algorithms can defend against dishonest participants, we had required that the uploader and the copyright claimants be honest during search in section 3.4. However, this is a fairly weak assumption in practise, where illegal media distributers are incentivized to cheat in order to convince the cloud providers to distribute their content.

Namely, security and privacy issues must be addressed in the search algorithm as in the detection algorithm. Furthermore, the ciphertext of the media under test must be identical or provably equivalent in both search and copyright detection. In a copyright

detection scenario, the media uploader may want to deceive the cloud by providing a feature vector belonging to a legitimate media other than the one he wishes to upload. We are not aware of any work on search over encrypted data in settings where a searcher would have incentive to deceive the search engine, or vice-versa.

A possible extension of our construction from chapter 3 would have Paillier cryptosystem as the basis for both search and detection. A feature extraction algorithm consisting only of operations permitted in homomorphic encryptions would be used on the encrypted media to extract the encrypted feature sets. The feature sets are compared against those of copyrighted works extracted in the same manner to identify similar works where the squared Euclidean distance is below a certain threshold. This can be achieved using the multiparty computation scheme described in section 3.3.5 and 3.3.6. Since the feature sets are extracted from the encrypted media the user uploads and the similarity measure is computed using an algorithm secure against dishonest parties, the cloud can be assured that the similarity testing results can be trusted. Furthermore, since both search and detection algorithm are based on Paillier, the user would not be able to switch out the encrypted media for the detection step.

The key challenge in the above solution lies in devising a media feature extractor in the encrypted domain. Unfortunately, many of the most popular feature extraction algorithms, such as SURF, SIFT and HOG, contain non-linear operators, which are operations not directly available in homomorphic encryption.

5.1.3 Aggregation of matching results in privacy-protected Anti-Virus as a service

The major drawback of the privacy-aware cloud based Anti-Virus described in section 4.1.3 is the high communication cost. Particularly, if a thorough indiscriminate scan on all files is requested, the communication cost is equivalent to the size of the database. Note that the scheme still achieves data privacy of stored data and anti-virus database, and that the trivial solution of sending the database is not a valid alternative.

Generally, the goal of the anti-virus code snippet pattern matching process is to determine the existence of snippets regardless of their positions within the file. Our analysis found that related techniques were unable to satisfy the privacy requirements of maintaining the privacy of the keywords from the data owner and privacy of documents from the cloud operator, despite improved communication cost. Using IBE such as Boneh's keyword search in private email database and Water's delegation of auditors for log investigation schemes, the data owner must give authorization to any third party wishing to search over the encrypted data by providing the plaintext of the requested

keywords. Using a public key scheme enables the anti-virus server to send the data without notifying data owner. The only primitive for sequential data search among our solutions which omit location information is the n-tuple Bloom-filter based query. However, the data owner must also provide permission to a third party in order to perform the query and, in doing so, reveals the plaintext to data owner. While it is possible to insert public key encrypted keywords into the filter, it would allow anyone including the cloud operator to perform queries without the data owner's permission. Note that, if wildcards are used, relative positioning of code snippets may be required.

The presented scheme in section 4.1.3 provides information on not only whether matches occurred, but also the location of the matches. If both sets of information are required and cryptographic strength protecting the matches is to be maintained, then we believe that the communication cost can only be improved if an approach is devised to map a ciphertext representing the difference of plaintext and code snippet to a ciphertext representing a match (1 or 0) and have the size of the ciphertext reduced to that needed for the binary plaintext space. That is, the minimum number of bits required to represent the information of whether a match occurred (Y or N) and the corresponding locations within a file cannot be less than the number of bits required to represent the ciphertext of a (Y or N) times the number of locations in the file. Furthermore, should the location of the matches be not required, then, further improvement may be possible since the entire file can now be processed to produce a single (Y or N) response.

Should it be possible to map the plaintext-snippet difference to an encrypted 1 or 0 value, the matching results can be meaningfully aggregated to determine if a match has occurred over any part of the file by summing them up at all the tested locations in a file. Assume that a match at location i results in $E_i(1)$ and a non-match results in $E_i(0)$, then the matching result for a file would equal:

$$E(x) = \sum_{i=1}^{n} (E_i(x_i)) \tag{5.1}$$

If a match is found in any location within the file, $x \neq 0$. If matches are rare occurring events, further savings can be achieved by increasing the aggregation level to multiple files or folders, and then narrowing down to identify matched files if needed. That is, a single matched result of $E(0)$ can be used to signal non-matched status for many files. If an algorithm can map matches to $E_i(0)$ instead, it is easy to convert to the above scenario by providing a ciphertext for 1, $E_s(1)$, to the cloud operator and convert the result by computing $E'_i(x_i) = E_s(1) - E_i(x_i)$.

The difficulty of the idea lies in devising a scheme that would map an encrypted

difference representing a match to a ciphertext of 1 and 0. A possibility we considered is Euler's theorem, which states that

$$a^{\varphi(n)} = 1 \ (mod \ n), \tag{5.2}$$

where $\varphi(n)$ denotes Euler's totient function. That is, if a is non-zero and coprime to n, the above operation results in 1. If a is zero, the above operation results in 0. However, $\varphi(n)$ is equivalent to the private key in Paillier cryptosystem, as is the case in most public key cryptosystems and cannot be revealed to the cloud operator. One would require a solution that would enable the cloud to compute the exponentiation without revealing the exponent, such as performing a homomorphic exponentiation, which we believe is an open problem.

5.1.4 Privacy-protected Big Data: Homomorphic neural network

The ability to compute over encrypted data presented by homomorphic encryption could lead to cryptographically secure outsourcing of computation. In addition to allowing a remote server to store and search over encrypted data without revealing their content, it may even enable data to be manipulated in meaningful ways without learning its content, providing privacy and security in services such as cloud based word processing software (e.g. Office 365) or Internet email services.

Another sample application would be privacy-protected recommender system. Suppose a recommender system has identified that users belonging to certain clusters are interested in certain products, a customer may send his feature sets, which may include previously view products and set preferences, in encrypted form and have the recommendation be computed by the server and sent back in encrypted form without learning what products he had viewed or his preferences. A possible methodology would be to perform privacy-protected collaborative filtering. The required computations, such as the Pearson correlation, to identify similarity between feature sets are comparable to those used in our privacy-aware copyright detection solution in section 3.3.

A more general and broadly applicable solution would be a homomorphic neural network. Artificial Neural Networks mimic brain functions through a network of equations. They have shown to be very effective at solving many practical problems, finding use in computer vision, speech recognition, health monitoring, network traffic balancing, etc.

Construction of neural networks typically involves layers of connected neurons. Each neuron generates a single output based on an activation function, $f_a()$, of a set of inputs. Each connection to the next layer is associated with a weight that scales this output.

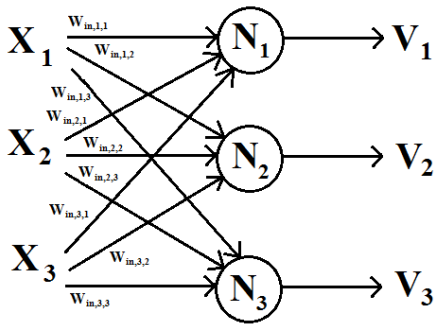

Figure 5.1: Input layer of a neural network with three inputs

The output from previous layer is similarly scaled. Figure 5.1 illustrates a layer of a sample network. The output of the layer is: $V = \{V_1, V_2, V_3\}$. The individual outputs are defined by $V_i = f_a(W_{in,i,1}X_1 + W_{in,i,2}X_2 + W_{in,i,3}X_3)$, where $W_{in,i,j}$ are the weights and X_i are inputs:

$$\begin{aligned} V_1 &= f_a(W_{in,1,1}X_1 + W_{in,1,2}X_2 + W_{in,1,3}X_3) \\ V_2 &= f_a(W_{in,2,1}X_1 + W_{in,2,2}X_2 + W_{in,2,3}X_3) \\ V_3 &= f_a(W_{in,3,1}X_1 + W_{in,3,2}X_2 + W_{in,3,3}X_3) \end{aligned} \tag{5.3}$$

The scale and add operations are naturally available in homomorphic encryption algorithms. However, the activation function poses a significant challenge. Typical activation functions used in neural networks are non-linear, e.g. $tanh()$ and $logsig()$. The common use of threshold functions, another non-linear operation, also poses a problem. As is often the case, another limiting factor towards achieving secure and private big data is the performance of homomorphic encryptions, even in the simpler collaborative filtering solution mentioned earlier. Nonetheless, a positive result towards the realization of a homomorphic neural network would have wide implications.

Bibliography

[1] How hard is it to 'de-anonymize' cellphone data? `http://news.mit.edu/2013/how-hard-it-de-anonymize-cellphone-data`. Accessed: 10-Sept-2016.

[2] Project Gutenberg. https://www.gutenberg.org/wiki/Main_Page. Accessed: 1-Sep-2014.

[3] Smhasher & murmurhash. https://code.google.com/p/smhasher/. Accessed: 30-April-2015.

[4] XML 1.0 Specification. http://www.w3.org/TR/REC-xml/. Accessed: 1-March-2015.

[5] A. Adelsbach, S. Katzenbeisser and A. Sadeghi. Cryptography meets watermarking: Detecting watermarks with minimal or zero knowledge disclosure. In *11th European Signal Processing Conference*, pages 446–449, 2003.

[6] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 563–574, 2004.

[7] G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson, and D. Song. Remote data checking using provable data possession. *ACM Trans. Inf. Syst. Secur.*, 14(1), 2011.

[8] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 598–609, 2007.

[9] M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: The case of hashing and signing. In *International Cryptology Conference Santa Barbara*, pages 216–233, 1994.

[10] M. Bellare and D. Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *International Conference on the Theory and Application of Cryptographic Techniques*, pages 163–192, 1997.

[11] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-homomorphic encryption and multiparty computation. In *Proceedings of the 30th Annual International Conference on Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT'11, pages 169–188. Springer-Verlag, 2011.

[12] E. L. Bird, Steven and E. Klein. *Natural Language Processing with Python*. O'Reilly Media Inc, 2009.

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

[14] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.

[15] K. Cai, C. Hong, M. Zhang, D. Feng, and Z. Lv. A secure conjunctive keywords search over encrypted cloud data against inclusion-relation attack. In *IEEE International Conference on Cloud Computing Technology and Science*, pages 339–346, 2013.

[16] R. Calderbank, S. Jafarpour, and R. Schapire. Compressed learning: Universal sparse dimensionality reduction and learning in the measurement domain. Technical report, 2009.

[17] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 41–50, 1995.

[18] C. L. A. Clarke, G. V. Cormack, and E. A. Tudhope. Relevance ranking for one to three term queries. *Information Processing and Management: an International Journal*, 36(2):291–311, Jan. 2000.

[19] I. Damgård, M. Geisler, and M. Kroigard. Homomorphic encryption and secure comparison. *International Journal of Applied Cryptology*, 1(1):22–31, 2008.

[20] M. Ding, F. Gao, Z. Jin, and H. Zhang. An efficient public key encryption with conjunctive keyword search scheme based on pairings. In *IEEE International Conference onNetwork Infrastructure and Digital Content*, pages 526–530, 2012.

[21] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, April 2006.

[22] L. Egghe. Untangling herdan's law and heaps' law: Mathematical and informetric arguments. *Journal of the American Society for Information Science and Technology*, 58:702–709, 2007.

[23] F. Kerschbaum, D. Biswas, and S. Hoogh. Peformance comparison of secure comparison protocols. In *20th International Workshop Database Expert System Applications*, pages 88–100, 2000.

[24] Z. Fu, X. Sun, N. Linge, and L. Zhou. Achieving effective cloud search services: multi-keyword ranked search over encrypted cloud data supporting synonym query. *IEEE Transactions on Consumer Electronics*, 60:164–172, 2014.

[25] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, pages 169–178, 2009.

[26] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikinen. On private scalar product computation for privacy-preserving data mining. In *International Conference in Information Security and Cryptology*, pages 104–120. Springer-Verlag, 2004.

[27] E.-J. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003.

[28] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia. Practical oblivious storage. In *Proceedings of the Second ACM Conference on Data and Application Security and Privacy*, pages 13–24, 2012.

[29] H. Guo and N. Georganas. A novel approach to digital image watermarking based on a generalized secret sharing scheme. *ACM Multimedia Systems Journal*, 9(3):249–260, 2003.

[30] M. Gymrek, A. L. McGuire, D. Golan, E. Halperin, and Y. Erlich. Identifying personal genomes by surname inference. *Science*, 339(6117):321–324, 2013.

[31] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg. Proofs of ownership in remote storage systems. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pages 491–500, 2011.

[32] S. Halevi and V. Shoup. HElib. `https://github.com/shaih/HElib`, 2014. Accessed: 9-Feb-2018.

[33] C. Hu and P. Liu. Public key encryption with ranked multi-keyword search. In *International Conference on Intelligent Networking and Collaborative Systems*, pages 109–113, 2013.

[34] G. Hu, D. Xiao, T. Xiang, S. Bai, and Y. Zhang. A compressive sensing based privacy preserving outsourcing of image storage and identity authentication service in cloud. *Information Sciences*, 387:132–145, 2017.

[35] T. L. I.J. Cox, J. Kilian and T. Shamoon. Secure spread spectrum watermarking for multimedia. *IEEE Trans. on Image Processing*, 6(12):1673–1687, 1997.

[36] I.J. Cox, M.L. Miller and J.A. Bloom. *Digital Watermarking*. Morgan Kaufmann, 2001.

[37] J. J. Eggers, J. K. Su and B. Girod. Public key watermarking by eigenvectors of linear transforms. In *European Signal Processing Conference*, 2000.

[38] R. C. Jammalamadaka and S. Mehrotra. Querying encrypted XML documents. In *International Database Engineering and Applications Symposium*, pages 129–136, 2006.

[39] L. Juan and M. De-ting. Research and application on the query processing for encrypted XML data. In *IEEE International Conference on Advanced Management Science*, pages 707–711, 2010.

[40] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics*. Prentice Hall, 2009.

[41] F. Kerschbaum. Secure conjunctive keyword searches for unstructured text. In *International Conference on Network and System Security*, pages 285–289, 2011.

[42] C. Li. Digital fragile watermarking scheme for authentication of jpeg images. In *IEE Proceedings - Vision, Image and Signal Processing*, volume 151, pages 460–466, 2004.

[43] Y. Li, B. Song, R. Cao, Y. Zhang, and H. Qin. Image encryption based on compressive sensing and scrambled index for secure multimedia transmission. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 12(4s):62, 2016.

[44] C. Liu, L. Zhu, L. Li, and Y. Tan. Fuzzy keyword search on encrypted cloud storage data with small index. In *2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, pages 269–273, 2011.

[45] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision*, 1999.

[46] M. Loytynoja, N. Cvejic, E. Lahetkangas and T. Seppanen. Audio encryption using fragile watermarking. In *International Conference on Information, Communications and Signal Processing*, pages 881–885, 2005.

[47] J. Menn. Exclusive: Russian antivirus firm faked malware to harm rivals - Ex-employees. `http://www.reuters.com/article/us-kaspersky-rivals-idUSKCN0QJ1CR20150814`, 2015. Accessed: 20-May-2016.

[48] J. Menn. Kaspersky says it obtained suspected NSA hacking code from U.S. computer. `https://www.reuters.com/article/us-usa-security-kaspersky-russia/kaspersky-says-it-obtained-suspected-nsa-hacking-code-from-us-computer-idUSKBN1CU0TN`, 2017. Accessed: 5-Mar-2018.

[49] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse data set. In *2008 IEEE Symposium on Security and Privacy*, pages 111–125, 2008.

[50] A. Orsdemir, H. O. Altun, G. Sharma, and M. F. Bocko. On the security and robustness of encryption via compressed sensing. In *MILCOM 2008-2008 IEEE Military Communications Conference*, pages 1–7. IEEE, 2008.

[51] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. *Lecture Notes in Computer Science*, 1592:223–238, 1999.

[52] H. Poon and A. Miri. Computation and search over encrypted XML documents. In *IEEE International Congress on Big Data*, pages 631–634, 2015.

[53] H. Poon and A. Miri. An efficient conjunctive keyword and phrase search scheme for encrypted cloud storage systems. In *IEEE International Conference on Cloud Computing*, pages 508–515, 2015.

[54] H. Poon and A. Miri. A low storage phrase search scheme based on bloom filters for encrypted cloud services. In *IEEE International Conference on Cyber Security and Cloud Computing*, pages 253–259, 2015.

[55] H. Poon and A. Miri. A combined solution for conjunctive keyword search, phrase search and auditing for encrypted cloud storage. In *IEEE Conference on Advanced and Trusted Computing*, pages 938–941, 2016.

[56] H. Poon and A. Miri. Scanning for viruses on encrypted cloud storage. In *IEEE Conference on Cloud and Big Data Computing*, pages 954–959, 2016.

[57] H. Poon and A. Miri. Privacy-aware search and computation over encrypted data stores. In S. Srinivasan, editor, *Guide to Big Data Applications*, chapter 11, pages 273–293. Springers International, 2018.

[58] H. Poon and A. Miri. Fast phrase search for encrypted cloud storage. *IEEE Transactions on Cloud Computing*, DOI: 10.1109/TCC.2017.2709316, to appear.

[59] D. M. W. Powers. Applications and explanations of zipf's law. In *Proceedings of the Joint Conferences on New Methods in Language Processing and Computational Natural Language Learning*, pages 151–160, 1998.

[60] Y. Rachlin and D. Baron. The secrecy of compressed sensing measurements. In *46th Annual Allerton Conference on Communication, Control, and Computing*, pages 813–817, Sept 2008.

[61] S. Ruj, M. Stojmenovic, and A. Nayak. Privacy preserving access control with authentication for securing data in clouds. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 556–563, 2012.

[62] S. Craver. Zero knowledge watermark detection. In *Information Hiding: Third International Workshop*, pages 101–116, 2000.

[63] S. Ghaffaripour, F. Younis, H. Poon, and A. Miri. An analysis of the security of compressed sensing using an artificial neural network. In *Privacy, Security and Trust*, pages 1–3, 2017.

[64] H. Shacham and B. Waters. Compact proofs of retrievability. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security*, pages 90–107, 2008.

[65] H. Shacham and B. Waters. Compact proofs of retrievability. *Journal of Cryptology*, 26(3):442–483, 2013.

[66] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[67] K.-S. Sung, H. Ko, and H.-S. Oh. XML document encrypt implementation using elliptic curve cryptosystem. In *International Conference on Convergence Information Technology*, pages 2473–2478, 2007.

[68] T. Furon and P. Duhamel. An asymmetric public detection watermarking technique. In *Information Hiding: Third International Workshop*, pages 88–100, 2000.

[69] Y. Tang, D. Gu, N. Ding, and H. Lu. Phrase search over encrypted data with symmetric encryption scheme. In *International Conference on Distributed Computing Systems Workshops*, pages 471–480, 2012.

[70] H. Tuo and M. Wenping. An effective fuzzy keyword search scheme in cloud computing. In *International Conference on Intelligent Networking and Collaborative Systems*, pages 786–789, 2013.

[71] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou. Secure ranked keyword search over encrypted cloud data. In *International Conference on Distributed Computing Systems*, pages 253–262, 2010.

[72] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *Proceedings of the 14th European Conference on Research in Computer Security*, pages 355–370, 2009.

[73] Q. Wang, W. Zeng, and J. Tian. A compressive sensing based secure watermark detection and privacy preserving storage framework. *IEEE transactions on image processing*, 23(3):1317–1328, 2014.

[74] B. Waters, D. Balfanz, G. Durfee, and D. K. Smetters. Building an encrypted and searchable audit log. In *Network and Distributed System Security Symposium*, pages 205–214, 2004.

[75] Y. Yang, H. Lu, and J. Weng. Multi-user private keyword search for cloud computing. In *IEEE Third International Conference on Cloud Computing Technology and Science*, pages 264–271, 2011.

[76] J. Zhang, B. Deng, and X. Li. *Additive Order Preserving Encryption Based Encrypted Documents Ranking in Secure Cloud Storage*, pages 58–65. Springer Berlin Heidelberg, 2012.

[77] M. Zheng and H. Zhou. An efficient attack on a fuzzy keyword search scheme over encrypted data. In *International Conference on High Performance Computing and Communications and Embedded and Ubiquitous Computing*, pages 1647–1651, 2013.

[78] N. Zhou, S. Pan, S. Cheng, and Z. Zhou. Image compression–encryption scheme based on hyper-chaotic system and 2d compressive sensing. *Optics & Laser Technology*, 82:121–133, 2016.

[79] S. Zittrower and C. C. Zou. Encrypted phrase searching in the cloud. In *IEEE Global Communications Conference*, pages 764–770, 2012.

Glossary

IR	Inclusion Relation
DRM	Digital Rights Management
IBE	Identity-Based Encryption
TF-IDF	Term Frequency $\times$ Inverse Document Frequency
LUT	Lookup Table
TP	True Positive
FP	False Positive
NLTK	Natural Language Toolkit
PDP	Provable Data Possession
POR	Proof of Retrievability
CBC	Cipher-Block Chaining
CTR	Counter Mode
SIFT	Scale-Invariant Feature Transform
PEKS	Public-Key Encryption with Keyword Search
DMCA	Digital Millennium Copyright Act
RIAA	Recording Industry Association of America
DCT	Discrete Cosine Transform
CS	Compressed Sensing
RIP	Restricted Isometric Property
MAC	Message Authentication Code
AMPC	Arithmetic Multi-Party Computation
SURF	Speeded Up Robust Features
HOG	Histogram of Oriented Gradients
P2P	peer-to-peer
IV	Initialization Vector