PRIVACY-PRESERVING RANKED SEARCH OVER ENCRYPTED CLOUD DATA

by Cengiz Örencik

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Sabanci University
Spring, 2014

PRIVACY-PRESERVING RANKED SEARCH
OVER ENCRYPTED CLOUD DATA

APPROVED BY:

Assoc.Prof.Dr. Erkay SAVAŞ (Thesis Supervisor)	[illegible]
Assoc.Prof.Dr. Yücel SAYGIN	[illegible]
Assoc.Prof.Dr. Cem GÜNERİ	[illegible]
Assist.Prof.Dr. Alptekin KÜPÇÜ	[illegible]
Assoc.Prof.Dr. Albert LEVI	[illegible]

DATE OF APPROVAL: 22/05/2014

Acknowledgments

I wish to express my gratitude to my supervisor Erkay Savaş for his invaluable guidance, support and patience all through my thesis. I am also grateful to Murat Kantarcıoğlu for his guidance and valuable contributions to this thesis. I would like to thank him for welcoming me to the friendly and encouraging environment of Data Security and Privacy Lab, University of Texas at Dallas (UTD).

Special thanks to my friend Ayşe Selçuk, for her help in parallelisation of the code and her kind suggestions. I am grateful to all my friends from Cryptography and Information Security Lab. (i.e., FENS 2001), Sabanci University and Data Security and Privacy Lab, UTD for being very supportive.

I am indebted to the members of the committee of my thesis for reviewing my thesis and providing very useful feedback.

I am grateful to TÜBİTAK (The Scientific and Technological Research Council of Turkey), for the Ph.D. fellowship supports. This thesis is also partially supported by TÜBİTAK 1001 Project under Grant Number:113E537.

Especially, I would like to thank my family, for being there when I needed them to be. I owe acknowledgment to them for their encouragement, and love throughout difficult times in my graduate years.

PRIVACY-PRESERVING RANKED SEARCH OVER ENCRYPTED CLOUD DATA

Cengiz Örencik
Computer Science and Engineering
Ph.D. Thesis, 2014
Thesis Supervisor: Assoc.Prof.Dr. Erkay Savaş

Keywords: Searchable Encryption, Privacy, Cloud Computing, Ranking, Applied Cryptography, Homomorphic Encryption

Abstract

Search over encrypted data recently became a critical operation that raised a considerable amount of interest in both academia and industry, especially as outsourcing sensitive data to cloud proves to be a strong trend to benefit from the unmatched storage and computing capacities thereof. Indeed, privacy-preserving search over encrypted data, an apt term to address privacy related issues concomitant in outsourcing sensitive data, has been widely investigated in the literature under different models and assumptions. Although its benefits are welcomed, privacy is still a remaining concern that needs to be addressed. Some of those privacy issues can be summarized as: submitted search terms and their frequencies, returned responses and their relevancy to the query, and retrieved data items may all contain sensitive information about the users.

In this thesis, we propose two different multi-keyword search schemes that ensure users' privacy against both external adversaries including other authorized users and cloud server itself. The proposed schemes use cryptographic techniques as well as query and response randomization. Provided that the security and randomization parameters are appropriately chosen, both the search terms in the queries and the returned responses are protected against privacy violations. The scheme implements strict security and privacy requirements that essentially can hide similarities between the queries that include the same keywords.

One of the main advantages of all the proposed methods in this work is the capability of multi-keyword search in a single query. We also incorporate effective ranking capabilities in the proposed schemes that enable user to retrieve only the top matching results. Our comprehensive analytical study and extensive experiments using both real and synthetic data sets demonstrate that the proposed schemes are privacy-preserving, effective, and highly efficient.

ŞİFRELENMİŞ BULUT VERİSİ ÜZERİNDE MAHREMİYET KORUMALI ve SIRALAMALI KELİME ARAMA

Cengiz Örencik
Bilgisayar Bilimi ve Mühendisliği
Doktora Tezi, 2014
Tez Danışmanı: Erkay Savaş

Anahtar Sözcükler: Arama Yapılabilir Şifreleme, Mahremiyet, Bulut Bilişim, Sıralama, Uygulamalı Kriptografi, Homomorfik Şifreleme

Özet

Hem akademik hem de endüstri çevrelerinde, hassas bilgi içeren verilerin bulut hizmeti veren firmalara aktarılması akımının başlamasıyla, şifrelenmiş veri üzerinde arama yapmak çok kritik ve önemli bir işlem haline geldi. Bulut yapısının, çok yüksek depolama ve hesaplama kapasitesini uygun fiyatlarla kullanıcılara sunuyor olması, bu akımın temel çıkış noktasıdır. Problemin öneminden dolayı, şifrelenmiş veri üzerinde mahremiyet korumalı arama yapmak, literatürde farklı modeller altında geniş çaplı bir şekilde incelenmiştir. Bulut yapısının faydaları kabul edilmekle birlikte, aktarılan verilerin mahremiyeti konusu hala çözülmesi gereken bir problemdir. Sorgu sırasında gönderilen anahtar terimlerin içeriği, sorgu terimlerinin kullanım sıklığı, geri dönen verilerin içeriği, bu verilerin sorgu ile ne oranda örtüştüğü gibi bilgilerin tamamı kullanıcılarla ilgili hassas bilgiler olarak nitelendirilebilir. Mahremiyet korumalı arama metotları, bu hassas bilgilerin korunmasını hedeflemektedir.

Bu çalışmada iki farklı mahremiyet korumalı anahtar kelime arama yöntemi öneriyoruz. Her iki yöntem de, hem başka kullanıcılara karşı, hem de bulut sunucusunun kendisine karşı verilerin mahremiyetini sağlıyor. Mahremiyeti sağlamak için, kriptografik yöntemlerin yanı sıra, sorguları ve dönen cevapları rastgele hale getirme yöntemlerinden de faydalanıyoruz. Güvenlik parametrelerinin doğru bir şekilde ayarlanması sağlandığı taktirde, önerdiğimiz yöntemler hem sorguların hem de buluta aktarılan verilerin mahremiyetini koruyacak niteliktedir. Önerdiğimiz yöntemler arama yapmanın dışında, eşleşen verileri sorgu ile alakalarına göre sıralama özelliğine de sahiptir. Bu özellik sayesinde sadece sorgu ile en alakalı eşleşmeler döndürülebilmektedir. Hem gerçek, hem de sentetik olarak yaratılmış veri kümeleri üzerinde yaptığımız detaylı analizler, önerdiğimiz yöntemlerin mahremiyeti koruyan ve yüksek oranda doğru sonuçları hızlı bir şekilde döndürebilen yapılar olduğunu göstermektedir.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

INTRODUCTION

The data storage requirements increase as huge amounts of data need to be accessible for users. The associated storage and communication requirements are a huge burden on organizations, which show, a strong proclivity of outsourcing their data to remote servers. Outsourcing data to clouds provides effective solutions to users that have limited resource and expertise for storage and distribution of huge data at low costs. However, data outsourcing engenders serious privacy concerns. Protecting the privacy is an essential requirement, since the cloud providers are not necessarily trusted. Therefore, some precautions are required to protect the sensitive data from both the cloud server and any other non-authorized party.

Cloud computing has the potential of revolutionizing the computing landscape. Indeed, many organizations that need high storage and computational power tend to outsource their data and services to clouds. Clouds enable its customers to remotely store and access their data by lowering the cost of hardware ownership while providing robust and fast services [1]. It is expected that by 2015, more than half of Global 1000 enterprises will utilize external cloud computing services and by 2016, all Global 2000 will benefit from cloud computing to a certain extent [2].

1.1 Motivation

While its benefits are welcomed in many quarters, some issues remain to be solved before a wide acceptance of cloud computing technology. The security and privacy of remote data, are among the most important issues, if not the most important. Particularly, the importance and necessity of privacy-preserving search techniques are even more pronounced in the cloud applications. The large companies that operate the public clouds like Google Cloud Platform [3], Amazon Elastic Compute Cloud [4] or Microsoft Live Mesh [5] may access the sensitive data such as search and access patterns. Hence, hiding the query and the retrieved data has great importance in ensuring the privacy and security of those using cloud services. A trivial approach can be encrypting the data before sharing with the cloud. However, the advantage of the cloud data storage is completely lost if data cannot be selectively searched and retrieved. Unfortunately, off-the-shelf private key encryption methods are not suitable for applying search over cipher-text.

One of the most important operations on the remote data is the secure search operation. Although there are several approaches for searchable encryption, the basic setting is almost the same for all. There is a set of authorized users and a single or multiple semi-trusted servers. The data is assumed to be accessible to the authorized users. Due to the sensitive nature of the documents, the users do not want the server or other users to learn the content of their documents. Moreover, due to the number of users, the search operations can be executed very frequently. Hence, the search operation should not only protect the privacy of the users and the data but also should be highly efficient.

To facilitate search on encrypted data, an encrypted index structure (i.e., secure index) is stored in the server along with the encrypted data. The authorized users have access to a trapdoor generation function which enables them to generate valid trapdoors for any arbitrary keyword. This trapdoor is used in the server to search for the intended keyword. It is assumed that the server does not have access to the trapdoor generation function, and therefore, can not ascertain the keyword searched for. We assume all the entities in the system are semi-honest and do not collude with each other.

Considering the large data set sizes, a single keyword search query usually matches with lots of data items, where only few are relevant. Moreover, users need to apply several queries and take the intersection of the corresponding results, which impose a serious burden of both computation and time on the user. A multi-keyword search, instead can incorporate a conjunction of several keywords in a single query. Moreover, instead of returning undifferentiated results, the matching results can further be ranked according to the relevancy to the query. By increasing the search constraints and applying ranking, only the most relevant items will be returned to the user, which reduces both the computation and communication burden on user.

A typical scenario that benefits from our proposal is that a company outsources its document server to a cloud service provider. Authorized users or customers of the company can perform search operations using certain keywords on the cloud to retrieve the relevant documents. The documents may contain sensitive information about the company, and similarly, the keywords that the users search may give hints about the content of the documents hence, both must be hidden. Furthermore, the queried keywords themselves may reveal sensitive information about the users as well, which is considered to be a privacy violation by users if learned by others.

In this thesis, we propose two different novel privacy-preserving and efficient multi-keyword search methods. The both methods return the matching data items in a rank-ordered manner.

1.2 Contributions

This thesis presents two novel multi-keyword search methods for applying secure search over encrypted cloud data. The design of a secure search (i.e., searchable encryption) method is challenging since it must satisfy strict privacy requirements while still being highly efficient.

The major results of this thesis are summarized as follows:

- We adapt some of the existing formal definitions for the security and privacy requirements of keyword search on encrypted cloud data for our problem and also introduce some new privacy definitions.

- We propose two multi-keyword search schemes. The first one is based on keyed cryptographic hash functions. The second one is based on locality sensitive hashing (LSH) (i.e., MinHash), which ensures privacy and security requirements in the most strict sense.

- We utilize ranking approaches for the both search methods that base on term frequencies (tf) and inverse document frequencies (idf) of the keywords. The proposed ranking approaches prove to be efficient to implement and effective in returning documents highly relevant to the submitted queries.

- We apply the search method on a two server setting that averts correlation of a query with the corresponding matching document identifiers.

- For the *MinHash* based search method, we utilize a novel approach that reduces the number of encryption and the communication overhead by more than 50 times through combining several encryption in a single cipher-text.

- We provide formal proofs that the proposed methods are privacy-preserving in accordance with the defined requirements.

- We implement the proposed schemes and demonstrate that it is efficient and effective by experimenting with both real and synthetic data sets.

1.3 Outline

The organization of this thesis is as follows: The literature on secure search is reviewed in detail in Chapter 2, . In Chapter 3, we examine the related well known topics that are going to be used throughout the thesis. In Chapter 4, we introduce our first secure keyword search approach that is based on HMAC functions. The experimental results and security proofs of this approach are also provided in this chapter. In Chapter 5, we provide yet another secure search method which is based on locality sensitive hash (LSH) functions. We propose two different models in this LSH based method. The first one is a single server method which is very efficient but has some security flows. The second one, two server model, provides better security requirements but it is slower than the single server model due to required homomorphic encryption operations. The formal security analysis and extensive cost analysis of both single and two server models are provided in this section. Finally, in Chapter 6 we conclude the thesis.

Chapter 2

RELATED WORK

Privacy-preserving search over encrypted data and searchable encryption methods have been extensively studied in recent years. A trivial approach is sending a copy of the entire encrypted data set to the user and let the user does the search. This trivial approach provides information theoretic privacy since the server cannot learn any information about the searched keywords or accessed files. Nevertheless, this approach brings an enormous computation burden on the user and do not benefit from the utilities of cloud computing. Any useful method for search over encrypted data must provide better efficiency compared to the trivial approach.

There are three main models in search over encrypted data methods [6]. The first model is the *vendor system.* In this scenario, the data stored on a server is public, but the user wants to apply search without revealing the information on the data accessed, to the server administrator. Private Information Retrieval (PIR) protocols provide solutions for this scenario [7, 8, 9, 10]. The problem of PIR was first introduced by Chor et al. [7]. Later, Groth et al. [11] propose a multi-query PIR method with constant communication rate. However, the computational complexity of the server in this method is very inefficient to be used in large databases. On the other hand, PIR does not address as to how the user learns which data items are most relevant to his inquiries.

The second scenario is the *store and forward system*, where a user can apply search over the data which is encrypted under the user's public key. This scenario is suitable for secure email applications, where the senders know the receivers' public keys. A public key encryption with keyword searching (PEKS) scheme for this scenario, was first proposed by Boneh et al. [12]. Several subsequent improvements on the PEKS method are proposed [6, 13, 14, 15]; both for single and conjunctive keyword search settings.

The third model is the *public storage system* (i.e., database outsourcing scenario), where a user outsources his sensitive data to a remote server in an encrypted form. Several authorized users can then apply search over the encrypted data, without leaking any sensitive information about the queried keywords to the remote database administrator. In this thesis, we consider the public storage system scenario.

Related work for this scenario can be analyzed in two major groups: single keyword and multi-keyword search. While the user can only search for a single feature (e.g., keyword) per query in the former, the latter enables search for a conjunction of several keywords in a single query.

Most of the privacy-preserving keyword search protocols existing in the literature provide solutions for single keyword search. Goh [16] proposes a security definition for formalization of the security requirements of searchable symmetric encryption schemes. One of the first privacy-preserving search protocols is proposed by Ogata and Kurosawa [17] using RSA blind signatures. The scheme is not very practical due to the heavyweight public key operations per database entry that should be performed on the user side.

Later, Curtmola et al. [18] provides adaptive security definitions for privacy-preserving keyword search protocols and proposes a scheme that satisfies the requirements given in the definitions. Another single keyword search scheme is proposed by Wang et al. [19] that keeps an encrypted inverted index together with relevancy scores for each keyword-document pair. This method is one of the first work that is capable of ranking the results according to their relevancy with the search term. Recently, Kuzu et al. [20] proposed another single keyword search method that uses locality sensitive hashes (LSH) and satisfies adaptive semantic security. Different from the other work, this scheme is a similarity search scheme, which means that matching algorithm works even if typos exist in the query.

All the work that are given above, are only capable of conducting single keyword search. However, in the typical case of search over encrypted data for public storage system scenario, the size of the outsourced data set is usually huge and single keyword search will inevitably return an excessive number of matches, where most will be irrelevant for the user. Multi-keyword search allows more constraints in the search query and enables the user to access only the most relevant data. Raykova et al. [21] proposed a solution using a protocol called re-routable encryption. They introduce a new agent called query router (QR) between the user and the server. User sends the queries to the server through the QR to protect his anonymity with respect to the server. Security of the user's message with respect to the QR is satisfied by confidentiality (i.e., encryption). They utilize bloom filters for efficient search. Although this work is presented as a single keyword search method, the authors also show a trivial multi-keyword extension. Wang et al. [22] proposed a multi-keyword search scheme, which is secure under the random oracle model. The method uses a hash function to map keywords into a fixed length binary array. Later, Cao et al. [23] proposed another multi-keyword search scheme that encodes the searchable database index into two binary matrices and uses inner product similarity during matching. This method is inefficient due to huge matrix operations and it is not suitable for ranking.

Bilinear pairing based solutions for privacy-preserving multi-keyword search are also presented in the literature [15, 24, 25]. In contrast to other multi-keyword search solutions that are based on either hashing or matrix multiplications, the results returning from bilinear pairing based solutions are free from false negatives and false positives (i.e., only the correct results return). However, computation costs of pairing based solutions are prohibitively high both on the server and on the user side. Moreover, bilinear pairing based schemes provide neither any additional privacy for hiding access or search patterns of users, nor any solution for ranking the matching results according to their relevancy with the queries. Therefore, pairing based solutions are not practical for many applications.

The privacy definition for almost all of the existing efficient privacy-preserving search schemes, proposed for the *public storage system*, allows the server to learn some information due to efficiency concerns. Although the data is encrypted, it may not always ensure privacy. If an adversary can observe a user's access pattern (i.e., which items are accessed) to an encrypted storage, some information about the user can still be learned. In the case, there is a need for hiding the access patterns, Oblivious RAM [26] methods can be utilized for the document retrieval process. Oblivious RAM hides the access pattern by continuously applying a re-order process on the memory as it is being accessed. Since in each access, the memory location of the same data is different and independent of any previous access, the access pattern is not leaked. However the Oblivious RAM methods are not practical even for medium sized data sets due to incurred polylogarithmic overhead. Specifically, in real world setups ORAM yields execution times of hundreds to thousands of seconds per single data access [26]. Recently Stefanov et al. [27] present a simple Oblivious RAM protocol with a small amount of client storage, named Path ORAM. The method Path ORAM requires $\log^2 N / \log X$ bandwidth overhead for block size $B = X \log N$, which is asymptotically better than the best known ORAM scheme with small client storage for block sizes bigger than $\Omega(\log^2 N)$.

Chapter 3

PRELIMINARIES

The fundamental problem of search over encrypted data is examining the similarity between queries and encrypted data items. We use two different encryption methods, homomorphic encryption and PCPA-secure encryption, for ensuring the privacy of the data. Similarly two different hash functions, MinHash and HMAC, are used to deduce a similarity between secure index entries of the sensitive data and an encrypted query. We also utilize some of the well-known metrics used in information systems to estimate the order of relevancy of the matching results. In this Chapter, we present the definitions and the basics of these techniques.

3.1 Homomorphic Encryption

Homomorphic encryption is a type of encryption that allows some operations on the ciphertext, where the result of the operation is an encrypted version of the actual result. For instance, two numbers, encrypted with homomorphic property, can be securely added or multiplied without revealing the unencrypted individual numbers.

Homomorphic encryption schemes are suitable for various applications such as e-voting, multi-party computation and secure search. Due to the importance of the homomorphic property, several partially or fully homomorphic cryptosystems are proposed in the literature. While partially homomorphic encryptions provide either addition or multiplication operation, fully homomorphic systems can provide both at the same time but less efficiently. We present some homomorphic cryptosystems in the following sections.

3.1.1 Unpadded RSA

In the RSA encryption [28] method, if the public key modulus is m, the exponent is e and the private message is $x \in \mathbb{Z}_m$, the encryption is defined as:

$$Enc(x) = x^e \mod m$$

The homomorphic property is then,

$$\begin{aligned} Enc(x_1) \cdot Enc(x_2) &= x_1^e x_2^e \mod m \\ &= (x_1 \cdot x_2)^e \mod m \\ &= Enc(x_1 \cdot x_2). \end{aligned}$$

3.1.2 Paillier

In the Paillier cryptosystem [29], if the public key modulus is m and the base is g and the private message is $x \in \mathbb{Z}_m$, the encryption is defined as:

$$Enc(x) = g^x \cdot r^c \mod m^2,$$

where $r \in \mathbb{Z}_m^*$ is randomly chosen.

The Paillier cryptosystem has the following two homomorphic properties:

- $Enc(x_1) \cdot Enc(x_2) = Enc(x_1 + x_2)$
- $Enc(x_1)^{x_2} = Enc(x_1 \cdot x_2)$

These homomorphic properties can be shown as,

$$\begin{aligned} Enc(x_1) \cdot Enc(x_2) &= (g^{x_1} \cdot r_1^m)(g^{x_2} \cdot r_2^m) \mod m^2 \\ &= g^{x_1+x_2} \cdot (r_1 r_2)^m \mod m^2 \\ &= Enc(x_1 + x_2 \mod m). \end{aligned}$$

$$\begin{aligned} Enc(x_1)^{x_2} &= (g^{x_1} \cdot r_1^m)^{x_2} \mod m^2 \\ &= g^{x_1 \cdot x_2} \cdot (r_1^{x_2})^m \mod m^2 \\ &= g^{x_1 \cdot x_2} \cdot (r_3)^m \mod m^2 \\ &= Enc(x_1 \cdot x_2 \mod m). \end{aligned}$$

The Paillier cryptosystem provides semantic security against chosen-plaintext attacks. Intuitively, given the knowledge of the ciphertext (and length) of some unknown message, it is not feasible to extract any additional information on the message.

3.1.3 Damgard-Jurik

The Damgard-Jurik [30] cryptosystem is a generalization of the Paillier cryptosytem, where the modulus is m^{s+1} instead of m^2 for some $s \geq 1$. If the public key modulus is m and the base is g and the private message is $x \in \mathbb{Z}_{m^s}$, the encryption is defined as:

$$Enc(x) = g^x \cdot r^{m^s} \mod m^{s+1},$$

where $r \in \mathbb{Z}^*_{m^{s+1}}$ is randomly chosen.

The homomorphic property is then,

$$\begin{aligned} Enc(x_1) \cdot Enc(x_2) &= (g^{x_1} \cdot r_1^{m^s})(g^{x_2} \cdot r_2^{m^s}) \mod m^{s+1} \\ &= g^{x_1+x_2} \cdot (r_1 r_2)^{m^s} \mod m^{s+1} \\ &= Enc(x_1 + x_2 \mod m^s). \end{aligned}$$

3.1.4 Fully Homomorphic Encryption

The homomorphic encryption methods given above provide either additive or multiplicative homomorphic property. The cryptosystems that supports both additive and multiplicative homomorphic encryption are known as fully homomorphic encryption. These methods are very powerful such that any circuit can be homomorphicly evaluated without revealing any of the the unencrypted parameters.

The first fully homomorphic encryption system is proposed by Craig Gentry [31] which utilizes lattice-based cryptography. Later some subsequent work [32, 33] are proposed on fully homomorphic encryption systems, however, any of the proposed fully homomorphic encryption methods is very costly and not suitable for many practical applications.

In this thesis, we use the Paillier cryptosystem (Section 3.1.2) as the homomorphic encryption method.

3.2 PCPA-Secure Encryption

A symmetric encryption method is secure against chosen plaintext attacks if the encrypted outputs (i.e., ciphertexts) do not reveal any useful information on the unencrypted messages (i.e., plaintexts). Curtmola et al. [18] defines a stronger security notion as pseudo-randomness against chosen plaintext attacks (PCPA), that guarantees the ciphertexts are indistinguishable from random numbers. Formally, PCPA-security is defined as follows[18].

Definition 1. ***PCPA-security***

Let two ciphertexts c_0 and c_1 are generated as follows:

$$c_0 = Enc(msg)$$
$$c_1 \in_R \mathcal{C},$$

where $\mathcal{C}$ denotes the ciphertext space.

A bit b is chosen at random, given msg and c_b, adversary A guesses the value of b as b'.

The encryption method is said to be PCPA-secure if for all polynomial-size adversaries A,

$$Pr[b' = b] \leq \frac{1}{2} + negl,$$

where $negl$ is a negligible value.

PCPA-security satisfies a slightly stronger security compared to indistinguishability against chosen-keyword attacks (IND2-CKA), introduced by Goh [16]. While IND2-CKA provides indistinguishability between two ciphertexts, PCPA provides indistinguishability between a ciphertext and a random number.

3.3 Hash Functions

In secure search concept, the search is applied on a secure index instead of the actual documents, where the details are explained in the subsequent sections. We utilize special hash functions to deduce a similarity between the secure index entries of the sensitive data and an encrypted query. Each data item is represented by an entry in the secure index. The important property of the secure index is that, it should be possible to compare two index elements and estimate a distance between them without leaking any other information. Although the exact similarity cannot be deduced, they still provide a good approximation. Moreover, the accuracy of the similarity further increases as hash functions with larger output size are used. We utilize the Hash-based Message Authentication Code (HMAC) and the MinHash functions in this thesis.

3.3.1 Hash-based Message Authentication Code

In cryptography a hash-based message authentication code (HMAC) [34] is used for constructing a fix sized message authentication code utilizing a cryptographic hash function and a secret cryptographic key. The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function, the size of its hash output, and the size of the secret key. In this thesis, we use SHA based HMAC functions for the HMAC based secure search method.

3.3.2 MinHash

In the MinHash based method we proposed, a well-known technique, called locality sensitive hashing [35] is used. Each document is represented by a small set called signature. The important property of signatures is that, it should be possible to compare two signatures and estimate a distance between the underlying documents from the signatures alone. The signatures are composed of several elements, each of which is constructed using the MinHash functions. They provide close estimates and the larger the signatures the more accurate the estimates.

To MinHash a set, pick a permutation of the rows. The MinHash value is the number of the first row in the permuted order, in which the corresponding element is in the set. The formal definition is as follows.

Definition 2. ***MinHash:*** *Let Δ be a finite set of elements, P be a permutation on Δ and $P[i]$ be the i^{th} element in the permutation P. MinHash of a set $D \subseteq \Delta$ under permutation P is defined as:*

$$h_P(D) = min(\{i \mid 1 \leq i \leq |\Delta| \ \wedge \ P[i] \in D\})$$

In the proposed MinHash based method, for each signature, λ different random permutations on Δ are used so the final signature of a set D is:

$$Sig(D) = \{h_{P_1}(D), \ldots, h_{P_\lambda}(D)\},$$

where h_{P_j} is the MinHash function under permutation P_j. We use the MinHash signatures as an approximation method that maps the given items into several buckets (λ) using different hash functions. The functions are chosen such that while similar items are likely to be mapped into the same buckets, dissimilar items are mapped to different buckets with high probability.

3.4 Distance Functions

A distance function is a metric used for describing the notion of closeness for elements of some space. A distance function d, on a set X is a function

$$X \times X \to \mathbb{R}.$$

For all $x, y, z \in X$, this function is required to satisfy the following conditions:

1. $d(x, y) \geq 0$ (non-negativity)

2. $d(x, y) = 0 \Leftrightarrow x = y$ (identity of indiscernibles)

3. $d(x, y) = d(y, x)$ (symmetry)

4. $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

We use two well-known distance functions in this thesis.

3.4.1 Hamming Distance

The Hamming distance between two strings of equal length, is defined as the number of symbols in which they differ [36]. Intuitively, it measures the minimum number of substitutions required to change one string into the other one.

In this thesis we use the Hamming distance on binary strings.

Example 1. *Let x and y are "1011101" and "1001001" correspondingly. Then the Hamming distance between x and y,* $d(\text{“}10\underline{1}1\underline{1}01\text{”}, \text{“}10\underline{0}1\underline{0}01\text{”}) = 2$

3.4.2 Jaccard Distance

The Jaccard distance is a metric that measures the dissimilarity between two sets. Intuitively, the Jaccard distance is the ratio of the number of different elements in the two sets to the union.

Formally, the Jaccard distance between the sets A and B is defined as:

$$\begin{aligned} J_d(A, B) &= 1 - \frac{|A \cap B|}{|A \cup B|} \\ &= \frac{|A \cup B| - |A \cap B|}{|A \cup B|}. \end{aligned} \tag{3.1}$$

3.5 Relevancy Score

In order to sort the matching results according to their relevancy with the query, a similarity function is required. The similarity function assigns a relevancy score to each of the matching results corresponding to a given search query.

Four fundamental metrics are widely used in information systems for calculating relevancy [37]:

- **Term frequency**($\mathrm{tf}_{w,D}$) is defined as the number of times a keyword w appears in a document D. Higher term frequency implies that the document is more relevant to queries that contains the corresponding keyword w.

- **Inverse document frequency** measures rarity of a keyword within the database collection. Intuitively a keyword that is rare within the database but common in a document results in a higher relevancy. The inverse document frequency of a keyword w is obtained as:

  $$\mathrm{idf}_w = \log\left(\frac{|\mathcal{D}|}{\mathrm{df}_w}\right)$$

  where $|\mathcal{D}|$ is the total number of document entries and df_w is document frequency of w (i.e., total number of documents containing w).

- **Document length (Density)**, results in a higher score for the shorter of the two documents which contain equal number of keywords.

- **Completeness** results in a higher score for the documents that contain more keywords.

A commonly used weighting factor for information retrieval is the *tf-idf weighting* [37]. Intuitively, it measures the importance of a keyword within a document for a database collection. The weight of each keyword in each document is calculated using the tf-idf weighting scheme that assigns a composite weight using both term frequency (tf) and inverse document frequency (idf) information. The tf-idf of a keyword w in a document D is given by:

$$\text{tf-idf}_{w,D} = \text{tf}_{w,D} \times \text{idf}_w.$$

Note that, the ratio inside the idf's log function is always greater than or equal to 1, hence, the value of idf is greater than or equal to 0. Consequently, the resulting tf-idf is a real number greater than or equal to 0.

3.6 Success Rate

Two of the best metrics for analyzing the success of a search method are the precision and recall metrics, which are widely used in the secure search literature [20, 23, 38]. Let $R(F)$ be the set of items retrieved for a query with feature set F and $R^*(F)$ be a subset of $R(F)$ such that, the elements of $R^*(F)$ include all the features in F. Further let $D(F)$ be the set of items in the data set that contains all the features in F. Note that $R^*(F) \subseteq R(F)$ and $R^*(F) \subseteq D(F)$. Precision ($prec(F)$), recall ($rec(F)$), average precision ($aprec(F)$) and average recall ($arec(F)$) for a set $\mathcal{F} = \{F_1, \ldots, F_n\}$ are defined as follows:

$$prec(F) = \tfrac{|R^*(F)|}{|R(F)|}, \quad aprec(\mathcal{F}) = \textstyle\sum_{i=1}^{n} \tfrac{prec(F_i)}{n} \tag{3.2}$$

$$rec(F) = \tfrac{|R^*(F)|}{|D(F)|}, \quad arec(\mathcal{F}) = \textstyle\sum_{i=1}^{n} \tfrac{rec(F_i)}{n} \tag{3.3}$$

The methods compare the expected and the actual results of the evaluated system. Intuitively, precision measures the ratio of correctly found matches over the total number of returned matches. Similarly recall measures the ratio of correctly found matches over the total number of expected results.

Chapter 4

HMAC-BASED SECURE SEARCH METHOD

In this chapter, we propose an efficient system where any authorized user can perform a search on an encrypted remote database with multiple keywords, without revealing neither the queried keywords, nor the information of the documents that match with the query. The only information that the proposed scheme leaks is the access pattern which is also leaked by almost all of the practical encrypted search schemes due to efficiency reasons.

Wang et al. [22] propose a trapdoorless private multi-keyword search scheme that is proven to be secure under the random oracle model. The scheme uses only binary comparison to test whether the secure index contains the queried keywords, therefore, the search can be performed very efficiently. However, there are some security issues that are not addressed in the work of Wang et al. [22]. We adapt their indexing method to our scheme, but we use a different encryption methodology to increase the security and address the security issues that are not considered in [22]. While a preliminary version of the work introduced in this chapter, is presented in the EDBT/ICDT conference [39], the full version of the work is published in the journal of Distributed and Parallel Databases [40].

4.1 System and Privacy Requirements

The problem that we consider is privacy-preserving keyword search on *public storage system*, where the documents are simply encrypted with the secret keys unknown to the actual holder of the database (e.g., cloud server). We consider three roles consistent with the previous works [23, 22]:

- *Data Controller* is the actual entity that is responsible for the establishment of the database. The data controller collects and/or generates the information in the database and lacks the means (or is unwilling) to maintain/operate the database.
- *Users* are the members in a group who are entitled to access (part of) the information of the database.
- *Server* is a professional entity (e.g., cloud server) that offers information services to authorized users. It is often required that the server be oblivious to the content of the database it maintains, the search terms in queries and the documents retrieved.

Let D_i be a document in the sensitive database $\mathcal{D}$, and $F_i = \{w_1, \ldots, w_m\}$ be the set of features (i.e., keywords) that characterizes D_i. Initially, the data controller generates a searchable secure index $\mathcal{I}$, using the feature sets of the documents in $\mathcal{D}$ and sends $\mathcal{I}$ to the server. Given a query from the user, the server applies search over $\mathcal{I}$ and returns a list of ordered items. Note that this list does not contain any useful information to the third parties. Upon receiving the list of ordered items, the user selects the most relevant data items and retrieves them. The details of the framework are presented in Section 4.2.

The privacy definition for search methods in the related literature is that, the server should not learn the searched terms [23]. We further tighten the privacy over this general privacy definition and establish a set of privacy requirements for privacy-preserving search protocols. A privacy preserving multi-keyword search method should provide the following user and data privacy properties (first intuitions and then formal definitions are given):

1. (Query Privacy) The query should not leak information of the corresponding search terms it contains.

2. (Search Pattern Privacy) Equality between two search requests (i.e., queries) should not be verifiable by analyzing the queries or the returned list of ordered matching results.

3. (Access Control) No one can impersonate a legitimate user.

4. (Adaptive Semantic Security) All the information that an adversary can access, can be simulated using the information that is allowed to leak. Hence, it is guaranteed that the only information leaks in the proposed method, is the one that is is told to be leaked.

An algorithm A is probabilistic polynomial time (PPT) if it uses randomness (i.e, flips coins) and its running time is bounded by some polynomial in the input size or a polynomial in a security parameter. In cryptography, an adversary's advantage is a measure of how successfully it can attack a cryptographic algorithm, by distinguishing it from an idealized version of that type of algorithm.

Definition 3. ***Query Privacy:*** *A multi-keyword search protocol has query privacy, if for all probabilistic polynomial time adversaries A that, given two different feature sets F_0 and F_1 and a query Q_b generated from the feature set F_b, where $b \in_R \{0, 1\}$, the advantage of A in finding b is negligible.*

Definition 4. ***Access Control:*** *A multi-keyword search protocol provides access control, if there is no adversary A that can impersonate a legitimate user with probability greater than ϵ, where ϵ is the probability of breaking the underlying signature scheme.*

Definition 5. ***Search Pattern** (S_p) is the frequency of the queries searched, which is found by checking the equality between two queries. Formally, let $\{Q_1, \ldots, Q_n\}$ be a set of queries and $\{F_1, \ldots, F_n\}$ be the corresponding search feature sets. Search pattern S_p is an $n \times n$ binary matrix, where*

$$S_p(i, j) = \begin{cases} 1, & \textit{if } F_i = F_j, \\ 0, & \textit{otherwise} \end{cases}$$

for $i, j \leq n$.

Intuitively, any deterministic query generation method reveals the search pattern.

Definition 6. ***Search Pattern Privacy:*** *A multi-keyword search protocol has search pattern privacy, if for all polynomial time adversaries A that, given a query Q, a set of queries, $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$ and the corresponding match results that returns, the adversary cannot find the queries in $\mathcal{Q}$ that are equivalent with Q.*

Definition 7. ***Access Pattern** (A_p) is the collection of data identifiers that contains search results of a user query. Let F_i be the feature set of Q_i*

and $R(F_i)$ be the collection of identifiers of data elements that matches with the feature set F_i, then $A_p(Q_i) = R(F_i)$.

Intuitively, if access pattern is leaked, given a query Q of a feature set F, an attacker does not learn the content of F but learns which are the documents in the data set that contains the features in F.

Definition 8. ***History** (H_n)**:*** *Let $\mathcal{D}$ be the collection of documents in the data set and $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$ be a collection of n queries. The n-query history is defined as $H_n(\mathcal{D}, \mathcal{Q})$.*

Definition 9. ***Trace** ($\gamma(H_n)$)**:*** *Let $C = \{C_1, \ldots, C_l\}$ be the set of encrypted user profiles, $id(C_i)$ be the identifier of C_i and $|C_i|$ be the size of C_i. Furthermore, let $\texttt{Dsig}(Q_i)$ be the digital signature of query Q_i, $|\texttt{Dsig}(Q_i)|$ be the size of $\texttt{Dsig}(Q_i)$, $\mathcal{I}$ be the searchable index and $|\mathcal{I}|$ be the number of all elements, fake and genuine, in $\mathcal{I}$.*

The trace of H_n is defined as:

$$\gamma(H_n) = \{(id(C_1), \ldots, id(C_l)), (|C_1|, \ldots, |C_l|), |\texttt{Dsig}(Q)|, |\mathcal{I}|, A_p(H_n)\}. \tag{4.1}$$

We allow to leak the trace to an adversary and guarantee no other information is leaked.

Definition 10. ***View** ($v(H_n)$)* *is the information that is accessible to an adversary. Let $\texttt{Dsig}(\mathcal{Q})$ be the list of digital signatures of queries in $\mathcal{Q}$ and, $id(C_i)$, C, $\mathcal{Q}$ and $\mathcal{I}$ are as defined above. The view of H_n is defined as:*

$$v(H_n) = \{(id(C_1), \ldots, id(C_l)), C, \mathcal{I}, \mathcal{Q}, \texttt{Dsig}(\mathcal{Q})\}. \tag{4.2}$$

Definition 11. ***Adaptive Semantic Security:*** *[18]*

A cryptosystem is adaptive semantically secure, if for all probabilistic polynomial time algorithms (PPTA), there exists a simulator $\mathcal{S}$ such that, given the trace of a history H_n, $\mathcal{S}$ can simulate the view of H_n with probability $1 - \epsilon$, where ϵ is a negligible probability.

Intuitively, all the information accessible to an adversary (i.e., view $(v(H_n))$) can be constructed from the trace $(\gamma(H_n))$ that is allowed to leak.

4.2 Framework of the HMAC-based Method

In this section, we provide the interactions between the three entities that we consider: Data Controller, Users and Server, which are introduced in Section 4.1. Due to the privacy concerns that are explained in Section 4.3.4, we utilize two servers namely: search server and file server. The overview of the proposed system is illustrated in Figure 4.1. We assume that the parties are semi-honest ("honest but curious") and do not collude with each other to bypass the security measures; two assumptions which are consistent with most of the previous work.

In Figure 4.1, steps and typical interactions between the participants of the system are illustrated. In an off-line stage, the data controller creates a search index element for each document. The searchable index file $\mathcal{I}$ is created using a secret key based trapdoor generation function where the secret keys[1] are only known by the data controller. Then, the data controller uploads the searchable index file to the search server and the actual encrypted documents to the file server. We use symmetric-key encryption as the encryption method since it can handle large document sizes efficiently. This process is referred as the *index generation* henceforth and the *trapdoor generation* is

[1]More than one key can be used in trapdoors for the search terms.

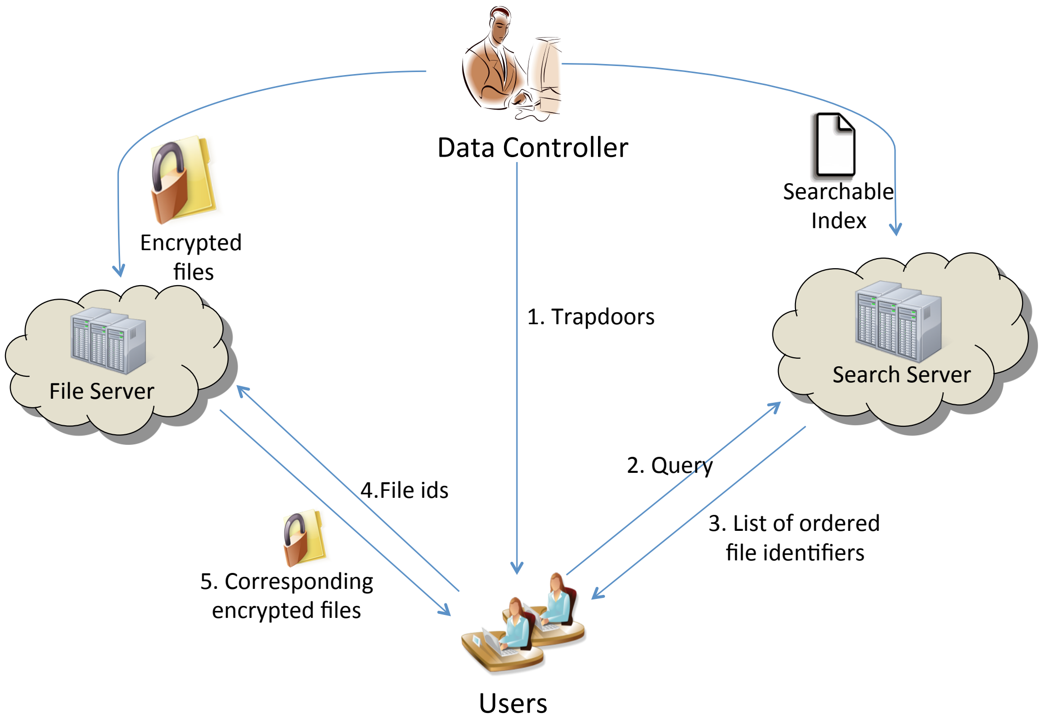


Figure 4.1: Architecture of the search method

considered as one of its steps.

When a user wants to perform a search, he first connects to the data controller. He learns the trapdoors (cf. Step 1 in Figure 4.1) for the keywords (i.e., features) he wants to search for, without revealing the keyword information to the data controller. Since the user can use the same trapdoor for many queries containing the corresponding features, this operation does not need to be performed every time the user performs a query. Alternatively, the user can request all the trapdoors in advance and never connects again to the data controller for the trapdoors. One of these two methods can be selected depending on the application and the users' requirements. After learning the trapdoor information, the user generates the query (referred as *query generation* henceforth) and submits it to the search server (cf. step 2 in Figure 4.1). In return, he receives meta data[2] for the matched documents in a rank ordered manner as will be explained in subsequent sections. Then the user retrieves the encrypted documents from the file server after analyzing the meta data that basically conveys a relevancy level of the each matched document, where the number of documents returned is specified by the user.

The proposed scheme satisfies the privacy requirements as defined in Section 4.1 provided that the parameters are set accordingly. For an appropriate setting of the parameters, the data controller needs to know only the frequencies of the most commonly queried search terms for a given database. By performing a worst case analysis for these search terms, the data controller can estimate the effectiveness of an attack and take appropriate countermeasures. The necessary parameters and the methods for their optimal selections are elaborated in the subsequent sections.

[2]Metadata does not contain useful information about the content of the matched documents.

4.3 The HMAC-based Ranked Multi-Keyword Search

In this section, we provide the details for the crucial steps in the proposed HMAC-based secure search method, namely index generation, trapdoor generation, query generation and document retrieval.

4.3.1 Index Generation (basic scheme)

Recently Wang et al. [22] proposed a conjunctive keyword search scheme that allows multiple-keyword search in a single query. We inspire from the scheme in [22] and develop an index construction scheme with better privacy properties.

The original scheme uses forward indexing, which means that a searchable index file element for each document is maintained to indicate the search terms existing in the document. In the scheme of Wang et al. [22], a secret cryptographic hash function, that is shared between all authorized users, is used to generate the searchable index. Using a single hash function shared by several users forms a security risk since it can easily leak to the server. Once the server learns the hash function, the security of the model can be broken, if the input set is small. The following example illustrates a simple attack against queries with few search terms.

Example 2. *There are approximately* 25000 *commonly used words in English [41] and users usually search for a single or two keywords. For such small input sets, given the hashed trapdoor for a query, it will be easy for the server to identify the queried keywords by performing a brute-force attack. For instance, assuming that there are approximately* 25000 *possible keywords in a database and a query submitted by a user involves two keywords, there* *will be* $25000^2 < 2^{28}$ *possible keyword pairs. Therefore, approximately* 2^{27} *trials will be sufficient to break the system and learn the queried keywords, if the underlying trapdoor generation function is known.*

We instead propose a trapdoor based system where the trapdoors can only be generated by the data controller through the utilization of multiple secret keys. The keywords are mapped to a secret key using a public mapping function named `GetBin` which is defined in Section 4.3.2. The usage of secret keys eliminates the feasibility of a brute force attack. The details of the index generation algorithm which is adopted from [22] are explained in the following and formalized in Algorithm 1.

Let $\mathcal{D}$ be the document collection where $|\mathcal{D}| = \sigma$. While generating the search index entry for a document $D \in \mathcal{D}$ that contains the keywords $\{w_1, \ldots, w_m\}$, we take HMAC (Hash-based Message Authentication Code) of each keyword with the corresponding secret key K_{id} which produces an $l = rd$ bit output (HMAC: $\{0,1\}^* \rightarrow \{0,1\}^l$). Let x_i be the output of the HMAC function for an input w_i and the trapdoor of a keyword w_i be denoted as I_i, where I_i^j represents the j^{th} bit of I_i, (i.e., $I_i^j \in GF(2)$, where GF stands for Galois field [42]). The trapdoor of a keyword w_i, $I_i = (I_i^{r-1}, \ldots, I_i^j, \ldots, I_i^1, I_i^0)$ is calculated as follows.

The l-bit output of HMAC, x_i can be seen as an r-digit number in base-d, where each digit is d bits. Also let $x_i^j \in GF(2^d)$ denotes the j^{th} digit of x_i and we can write

$$x_i = x_i^{r-1}, \ldots x_i^1, x_i^0.$$

After this, each r-digit output is reduced to r-bit output with the mapping from $GF(2^d)$ to $GF(2)$ as shown in Equation (4.3).

$$I_i^j = \begin{cases} 0, & \text{if } x_i^j = 0, \\ 1, & \text{otherwise.} \end{cases} \tag{4.3}$$

As a last step in the index entry generation, the bit-wise product of trapdoors of all keywords (I_i, $\forall i \in \{1, \ldots, m\}$) in the document D is used to obtain the final searchable index entry $\mathcal{I}_D$ for the document D as shown in Equation (4.4)

$$\mathcal{I}_D = \odot_{i=1}^{m} I_i, \tag{4.4}$$

where $\odot$ is the bit-wise product operation. The resulting index entry $\mathcal{I}_D$ is an r-bit binary sequence and its j^{th} bit is 1, if for all i, j^{th} bit of I_i is 1, and 0 otherwise.

Algorithm 1 Index Generation

```
Require: $\mathcal{D}$ : the document collection,
  $K_{id}$: secret key for the bin with label $id$
  for all documents $D_i \in \mathcal{D}$ do
    for all keywords $w_{i_j} \in D_i$ do
      $id \leftarrow \texttt{GetBin}(w_{i_j})$
      $x_{i_j} \leftarrow \text{HMAC}_{K_{id}}(w_{i_j})$
      $I_{i_j} \leftarrow Reduce(x_{i_j})$
    end for
    index entry $\mathcal{I}_{D_i} \leftarrow \odot_j I_{i_j}$
  end for
  return $\mathcal{I} = \{\mathcal{I}_{D_1}, \ldots, \mathcal{I}_{D_\sigma}\}$
```

In the following section, we explain the technique used to generate queries from the trapdoors of feature sets.

4.3.2 Query Generation

The searchable index file of the database is generated by the data controller using secret keys. A user who wants to include a search term in his query, needs the corresponding trapdoor from the data controller since he does not know the secret keys used in the index generation. Asking for the trapdoor openly would violate the privacy of the user against the data controller, therefore a technique is needed to hide the trapdoor asked by the user from the data controller.

Bucketization is a well-known data partitioning technique that is frequently used in the literature [43, 44, 45, 46]. We adopt this idea to distribute keywords into a fixed number of bins depending on their hash values. More precisely, every keyword is hashed by a public hash function, and certain number of bits in the hash value is used to map the keywords into one of the bins. The number of bins and the number of keywords in each bin can be adjusted according to the security and efficiency requirements of the system.

In our proposal for obtaining trapdoors, we utilize a public hash function with uniform distribution, named `GetBin`, that takes a keyword and returns a value in $\{0, \ldots, (\delta - 1)\}$ where δ is the number of bins. All the keywords that exist in a document are mapped by the data controller to one of those bins using the `GetBin` function. Note that, δ is smaller than the number of keywords so that each bin contains several elements, which provides obfuscation. The `GetBin` function has uniform distribution, therefore each bin will have approximately equal number of items in it. Moreover, δ must be chosen deliberately such that there are at least ϖ items in each bin where ϖ is a security parameter. Each bin in the index generation phase has a unique secret key used for all keywords in that bin.

The query generation method, which is given in Algorithm 2, works as follows. When an authorized user connects to the data controller to obtain the trapdoors for a set of keywords, he first calculates the bin IDs of the keywords and sends these values to the data controller. The data controller then returns the secret keys of the bins requested for, which can be used by the user to generate the trapdoors[3] for all keywords in those bins. Alternatively, the data controller can send the trapdoors of all the keywords in the corresponding bins resulting in an increase in the communication overhead. However, the latter method relieves the user from computing the trapdoors. Subsequent to obtaining the trapdoors, the user can calculate the query in a similar manner to the method used by the data controller to compute the searchable index. More precisely, if there are n keywords in a user query, the following formula is used to calculate the privacy-preserving query, given that the corresponding trapdoors (i.e., $I_1, \ldots, I_n$) are available to the user:

$$Q = \odot_{j=1}^{n} I_j.$$

Finally, the user sends this r-bit query Q to the search server. The users' keywords are protected against disclosure since the secret keys used in trapdoor generation are chosen by the data controller and never revealed to the search server. In order to avoid impersonation, the user signs the messages using a digital signature method.

4.3.3 Oblivious Search on the Database

A user's query, in fact, is just an r-bit binary sequence (independent of the number of search terms in it) and therefore, searching consists of as simple operations as binary comparison only. If the search index entry of the

[3]In fact, I_i, which is calculated for the search term w_i as explained in Section 4.3.1 is the trapdoor for the keyword w_i.

Algorithm 2 Query Generation

```
Require: a set of query features $F = \{w'_1, \ldots, w'_n\}$
  for all $w'_i \in F$ do
    $id \leftarrow \texttt{GetBin}(w'_i)$
    if $K_{id} \notin$ previously received keys then
      send $id$ to Data Controller
      get $K_{id}$ from Data Controller
    end if
    $x_i \leftarrow \mathrm{HMAC}_{K_{id}}(w'_i)$
    $I_i \leftarrow \mathit{Reduce}(x_i)$
  end for
  query $Q \leftarrow \odot_i I_i$
  return $Q$
```

document ($\mathcal{I}_R$) has 0 for all the bits, for which the query (Q) has also 0, then the query matches to that document as shown in Equation (4.5).

$$\mathrm{result}(Q, \mathcal{I}_R) = \begin{cases} \text{match}, & \text{if } \forall j \; Q^j = 0 \Rightarrow \mathcal{I}_R^j = 0, \\ \text{not match}, & \text{otherwise.} \end{cases} \tag{4.5}$$

Note that, the given query should be compared with the search index entry of each document in the database. The following example clarifies the matching process.

Example 3. *Let the user's query be* $Q = [011101]$ *and two document index entries be* $I_1 = [001100]$ *and* $I_2 = [101101]$*. The query has the 0 bit in* 0^{th} *and* 4^{th} *bits therefore, those bits must be 0 in the index entry of a document in order to be a match. Here the query matches with* I_1*, but does not match with* I_2 *since* 0^{th} *bit of* I_2 *is not 0.*

Subsequent to the search operation, the search server sends a rank ordered list of meta data of the matching documents to the user, where the underlying rank operation is explained in Section 4.6. The meta data is the search index entry of that document, which the user can analyze further to learn more about the relevancy of the document. After analyzing the meta data, the user retrieves ciphertexts of the matching documents of his choice from the file server.

To improve security, the data controller can change the HMAC keys periodically whereby each trapdoor will have an expiration time. After the expiration, the user needs to get the new trapdoors for the keywords he wants to use in his queries. This will alleviate the risk when the HMAC keys are compromised.

4.3.4 Document Retrieval

The search server returns the list of pseudo identifiers of the matching documents. If a single server is used for both search and file retrieval, it can be possible to correlate the pseudo identifiers of the matching documents and the identifiers of the retrieved encrypted files. Furthermore, this may also leak the search pattern that the proposed method hides. Therefore, we use a two-server system similar to the one proposed in [20], where the two servers are both semi-honest and do not collude. This method leaks the access pattern only to the file server and not to the search server, hence prevents any possible correlation between search results and encrypted documents retrieved.

Subsequent to the analyzes of the meta data retrieved from the search server, the user requests a set of encrypted files from the file server. The file server returns the requested encrypted files. Finally the user decrypts the files and learns the actual documents. The distribution of the document decryption keys can be performed using state-of-the-art key distribution methods and is not within the scope of this thesis.

In case access pattern also needs to be hidden, Oblivious RAM [47, 27] methods can be utilized for the document retrieval process instead. However, these methods are too expensive to be practical even on medium sized datasets, due to incurred polylogarithmic overhead.

4.4 Query Randomization

Search pattern is the information of equality among the keywords of two queries that can be inferred by linking one query to another. If an adversary can test the equality among two queries, he may learn the most frequent queries and correlate with the frequently searched real keywords that may be learned from statistics such as Google Trends [48]. The proposed basic scheme fails to hide the search pattern since the search index entries are generated in a deterministic way. Any two query generated from the identical feature sets, will be exactly the same. In order to hide the search pattern of a user, we introduce randomness into the query generation phase. This process is known as *query randomization*, which should be carefully implemented so that the queries do not leak information about the search patterns. In this section, we analytically demonstrate the effectiveness of the proposed query randomization method. Note that, the query randomization does not change the response to a given query.

For introducing non-determinacy into the search index generation, we generate a set $\mathcal{U}$ with $|\mathcal{U}| = U$, whereby the elements of $\mathcal{U}$ are dummy keywords that do not exist in the dictionary (i.e., they are simply random strings). The U dummy keywords are added in every index entry along with the genuine keywords. While generating a query, the user first randomly creates a set $\mathcal{V}$ where $|\mathcal{V}| = V$ and $\mathcal{V} \subset \mathcal{U}$. Then the query is composed using all elements of $\mathcal{V}$ together with the genuine search terms. The number of different choices of $\mathcal{V}$ from $\mathcal{U}$ is calculated as C_V^U, where C_k^n is the number of k-combinations from a set with n elements.

We can formalize the discussion as follows. Let

$$\mathcal{Q}_i = \{Q_{i1}, Q_{i2}, \ \ldots, Q_{i\mu}\}$$

be the set of queries that are generated from the same search features using different dummy keywords. Furthermore, let $\mathcal{Q}_x$ be the set of all possible other queries. Given two queries $Q_i \in \mathcal{Q}_i$ and Q_j, identifying whether $Q_j \in \mathcal{Q}_i$ or $Q_j \in \mathcal{Q}_x$ must be hard.

We use the Hamming distance metric for evaluating the similarity of two queries, which is defined in Section 3.4.1. We define two new functions to analytically calculate the expected Hamming distance.

Definition 12. ***Scarcity Function*** ***(**$F(x)$**)*** *The scarcity function $F(x)$ is the expected number of 0's in a query, where x is the number of keywords.*

Definition 13. ***Overlap Function*** ***(**$C(x)$**)*** *The overlap function $C(x)$ is the expected number of 0's that coincide in the corresponding bit positions of an x keyword query (Q_a) and a single keyword query (Q_b).*

Recall that r is the size of a query, d is the reduction value (cf. Section 4.3.1) and $Q[i]$ is the i^{th} bit of Q. The functions are calculated as follows:

Proposition 1. *For the Scarcity and Overlap functions we can write,*

$$F(x) = F(x-1) + F(1) - C(x-1)$$
$$C(x) = \sum_{i=0}^{r-1} P(Q_a[i] = 0, Q_b[i] = 0),$$

where $P(a,b)$ is a joint probability distribution.

Note that, the initial case for $F(x)$ is $F(1) = \frac{r}{2^d}$ and $C(x)$ is calculated as follows:

$$\begin{aligned} C(x) &= \sum_{i=0}^{r-1} P(Q_a[i] = 0, Q_b[i] = 0) \\ &= r\frac{F(x)}{r}\frac{F(1)}{r} \\ &= \frac{F(x)}{2^d}. \end{aligned}$$

The expected Hamming distance between two queries (i.e., Q_1 and Q_2) with x keywords each, where they have $\bar{x} \leq x$ common keywords, is calculated as in the following.

Proposition 2. *The Expected Hamming distance between two queries with $\bar{x}$ common keywords, can be calculated as follows:*

$$\Delta(Q_1, Q_2) \quad = \quad \frac{(F(x) - F(\bar{x}))(r - F(x))}{r} + \frac{F(x)(r - F(x))}{r}.$$

This can be seen by the simple derivation:

$$\begin{aligned} &\Delta(Q_1, Q_2) = \sum_{i=0}^{r-1} P(Q_1[i] \neq Q_2[i]) \\ &= r \cdot P(Q_1[1] \neq Q_2[1]) \\ &= r \cdot [P(Q_1[1] = 0) P(Q_2[1] = 1 | Q_1[1] = 0) + P(Q_1[1] = 1) P(Q_2[1] = 0 | Q_1[1] = 1)] \\ &= r \cdot \left[\frac{F(x)}{r}\left(\frac{F(\bar{x})}{F(x)} \cdot 0 + \frac{F(x) - F(\bar{x})}{F(x)} \cdot \frac{r - F(x)}{r}\right) + \frac{r - F(x)}{r}\left(\frac{F(x)}{r}\right)\right] \\ &= \frac{(F(x) - F(\bar{x}))(r - F(x))}{r} + \frac{F(x)(r - F(x))}{r}, \end{aligned} \tag{4.6}$$

where $P(A|B)$ is the conditional probability of A given B.

Each query chooses V keywords out of U dummy keywords. While comparing two arbitrary queries, the expected number of dummy keywords that exists in the both queries (E_O) is calculated as in the following.

Proposition 3. *The expected number of dummy keywords that both queries contain can be calculated as follows:*

$$E_O\left(V\right) = \sum_{i=0}^{V} \frac{\binom{V}{i}\binom{U-V}{V-i}}{\binom{U}{V}}\, i. \tag{4.7}$$

The first query chooses V keywords. The probability that i ($i \leq V$) keywords that is chosen by the second query also exist in the first one, is calculated as follows: i keywords are chosen from the set of keywords that are also selected by the first query and $(V - i)$ keywords are chosen from the set of not-selected keywords. Then we use summation to calculate the expected value in (Equation (4.7)). Note that, $E_O(V)$ is a monotonically increasing function (i.e., $V \geq V' \Leftrightarrow E_O(V) \geq E_O(V')$).

A possible way of choosing an optimum parameter setting is shown in the following example.

Example 4. *We use 448 bits as the query size (r) and the largest U for this query size that provides sufficient accuracy (i.e., high precision rate; cf. Section 4.4.4) is found as 60. Any further increase in U necessitates increasing the query size, which causes an increase in communication, computation and storage requirements (cf. Section 4.8). Using the formulae in (Equation* (4.6) *and* (4.7)*), the normalized differences between $\Delta(Q_i, Q_j)$ for two arbitrary queries $\{Q_i, Q_j\}$ and $\Delta(Q_{i\alpha}, Q_{i\beta})$, where $\{Q_{i\alpha}, Q_{i\beta}\} \in \mathcal{Q}_i$ are given* *in Figure 4.2. The normalized difference is calculated using the equation,*

$$\frac{\Delta(Q_i, Q_j) - \Delta(Q_{i\alpha}, Q_{i\beta})}{\Delta(Q_i, Q_j)}.$$

One can observe from Figure 4.2 that, when U is fixed as 60, $V = 30$ is the smallest value, which ensures that the distance between the two queries $Q_i \in \mathcal{Q}_i$ and $Q_j \notin \mathcal{Q}_i$ is sufficiently close to the distance between $Q_{i\alpha} \in \mathcal{Q}_i$ and $Q_{i\beta} \in \mathcal{Q}_i$. Note that, any $V \geq 30$ can also be used. The parameter setting that we used in our tests, is discussed more formally in Section 4.8.

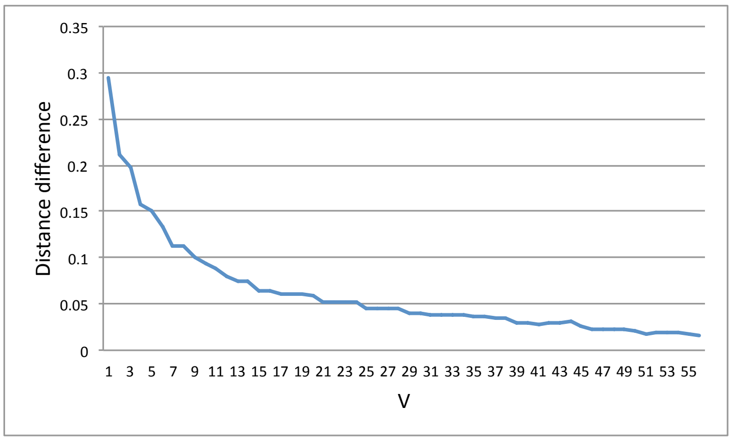


Figure 4.2: Normalized difference of the Hamming Distances between two arbitrary queries and two queries with the same genuine search features, where $U = 60$.

In order to demonstrate the usefulness of our analysis, we conducted an experiment using synthetic query data for the case, where adversary does not know the number of genuine search terms in a query. We generate a synthetic data set for a set of queries with the parameters $V = 30$ and $U = 60$ being fixed. The set contains a total of 250 queries, where the first 50 queries contain 2 genuine search terms each, the second 50 queries contain 3 genuine search terms each, and so on. And finally, the last set of 50 queries contains

6 genuine search terms each. We create another set that contains only 5 queries, which includes 2, 3, 4, 5 and 6 genuine search terms, respectively. The distances between pairs of queries, in which one query is chosen from the former set and the second one from the latter, are measured to obtain a histogram as shown in Figure 4.3(a). Consequently, a total of $250 \times 5 = 1250$ distances are measured. We also obtain another histogram featuring a total of 1250 distances between pairs of queries, whereby queries in a pair contain the same genuine search terms with different dummy keywords. Both histograms are given in Figure 4.3(a), where it can be observed that adversary cannot do better than a random guess to identify whether given two queries contain the same genuine search terms or not.

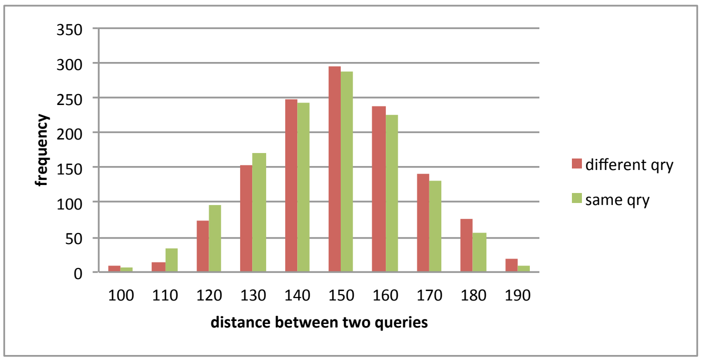
We also conduct a similar experiment, in which we assume that the adversary has the knowledge of the number of search terms in a query. A set containing a total of 1000 queries is generated, whose subsets with 200 queries each contain 2, 3, 4, 5 and 6 genuine search terms, respectively. A single query is then created using 5 genuine search terms. We measure the distances of the single query to all 1000 queries in the former set of queries and create the corresponding histogram (i.e., a total of $200 \times 5 = 1000$ distances are measured). We compared this with the histogram for 1000 measurements of the distance between two queries with five identical search terms as shown in Figure 4.3(b). As can be observed from the histogram in Figure 4.3(b), 20% of the time, distances between two queries are 150 and they are totally indistinguishable. In 45% of the time, the distances are smaller than 150, where the adversary can guess $Q_j \in \mathcal{Q}_i$ with 0.6 confidence. In 35% of the time, the distances are greater than 150 and the adversary can guess $Q_j \notin \mathcal{Q}_i$ with 0.7 confidence. In accordance with these results, one can guess whether the queries are from the same search terms or not correctly with 0.6 confidence under the assumption that the number of genuine search terms in the query is known to be 5. The tests with different number of genuine search terms also provide analogous results, where the confidence of distinguishing equivalent queries increase as the number of genuine search terms increase. This observation is in parallel with the expectation. As the number of genuine search terms in a query increase, the number of terms that are identical for the equivalent queries also increase, which make them vulnerable for such attacks.

Hence, the information of the number of genuine search terms should be kept secret, which is the case in our proposed method.

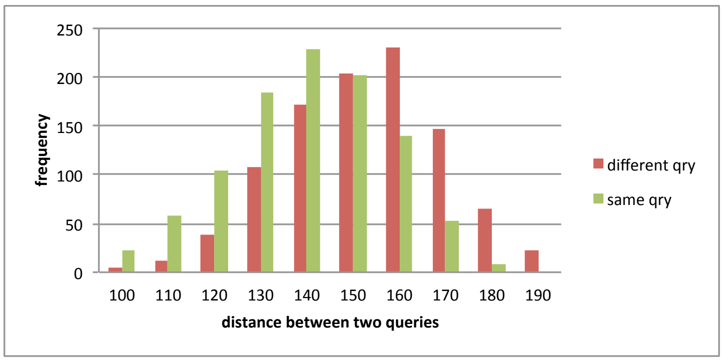
4.4.1 Correlation Attack

It is possible that the attacker may have some prior knowledge on the statistical model of the search terms in the queries (e.g., search frequency of the most frequently queried search terms). In this case, the attacker may use this information to deduce a set of queries that all include a search term w. Then, the trapdoor for w may be revealed with some error, provided that the adversary obtains a sufficient number of queries that all features the search term w. In this section, the proposed method is analyzed against this attack that is referred as correlation attack. Note that the wisest choice of w for the adversary to attack, will be the most commonly used search term which has the highest occurrence rate in the previous queries. The adversary may have prior knowledge of the most frequently queried search terms or may guess using the public statistics such as [48].

In order to analyze whether the attacker can identify a group of equivalent queries from other queries, we define a Distinguisher function $H(\mathcal{A}_k, Q_{k+1})$. This function takes two parameters; a set $\mathcal{A}_k = \{Q_1, \ldots, Q_k\}$ with k queries



(a) The distances for two arbitrary queries verses two queries that are generated from the same search terms



(b) The distances for two arbitrary queries verses two queries that are generated from the same search terms, where the number of search terms in the queries is 5

Figure 4.3: Histograms for the Hamming distances between queries

and a single query Q_{k+1}, and returns the number of 0's that coincides with all elements of the set $\mathcal{A}_k$ and Q_{k+1} (i.e., the number of query bit positions where all $k+1$ queries has 0 in that bit position). Let Q_{k+1} have x_{k+1} search terms, where $\bar{x}$ of them are common with all the queries in $\mathcal{A}_k$. The expected number of common 0 bits is estimated with the distinguisher function which is defined as follows.

Definition 14. *(Distinguisher Function) The distinguisher function, $H(\mathcal{A}_k, Q_{k+1})$,* *is the number of bits with value 0, that coincide in the corresponding bit positions of each element of the set $\mathcal{A}_k$ and the query Q_{k+1}.*

Proposition 4. *Expected Value of Distinguisher Function can recursively be estimated as:*

$$H(\mathcal{A}_k, Q_{k+1}) = \begin{cases} F(\bar{x}) + \dfrac{(F(x_1) - F(\bar{x}))(F(x_2) - F(\bar{x}))}{r} & \mathit{if}\, k = 1 \\ F(\bar{x}) + \dfrac{(H(\mathcal{A}_{k-1}, Q_k) - F(\bar{x}))(F(x_{k+1}) - F(\bar{x}))}{r} & \mathit{otherwise.} \end{cases} \tag{4.8}$$

Let a search term w be an element of all the queries in $\mathcal{A}_k$ and further let $w \in Q_{k+1}$ and $w \notin Q'_{k+1}$. Using the distinguisher function in equation (4.8), we define a dissimilarity function $\Omega(\mathcal{A}_k, Q_{k+1}, Q'_{k+1})$, that compares the dissimilarity between Q_{k+1} and Q'_{k+1} as in the following.

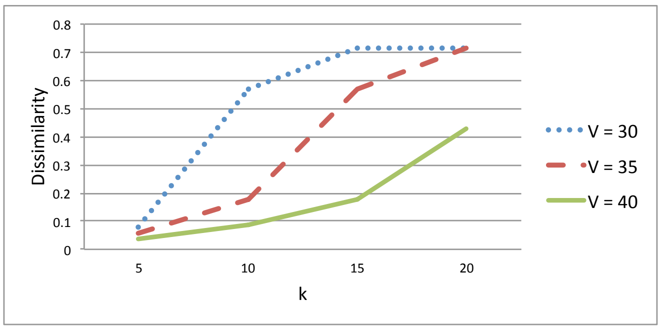
Definition 15. *(Dissimilarity Function)*

$$\Omega(\mathcal{A}_k, Q_{k+1}, Q'_{k+1}) = \frac{|H(\mathcal{A}_k, Q_{k+1}) - H(\mathcal{A}_k, Q'_{k+1})|}{H(\mathcal{A}_k, Q_{k+1})} \tag{4.9}$$

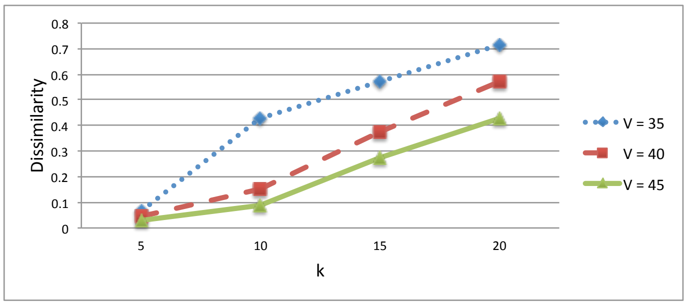
If $\Omega(\mathcal{A}_k, Q_{k+1}, Q'_{k+1}) \leq z$, where z is a sufficiently small security parameter, we say that distinguishing the set of queries that all contain the same keyword w from the other queries, is hard.

The Dissimilarity function is analyzed for various values of inputs in order to find the optimum choice for the parameters of randomness (i.e. U, V) that minimize $\Omega(\mathcal{A}_k, Q_{k+1}, Q'_{k+1})$ given in equation (4.9). We present the results in Figure 4.4.

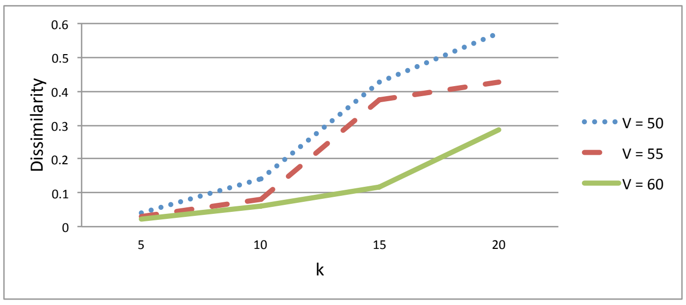
Figures 4.4(a), 4.4(b), and 4.4(c) indicate that for a fixed value of U, increasing V decreases the dissimilarity of queries when compared with the set $\mathcal{A}_k$. In other words, it will be difficult to distinguish queries that possess w from those that do not. Note that, since V is introduced for obfuscating the queries, increasing V also increases the similarity between unrelated queries



(a) U=50



(b) U=60



(c) U=80

Figure 4.4: Values of Dissimilarity Function (Equation (4.9)) for different parameters

as expected. However, increasing V also increase the probability that two equivalent queries pick exactly the same set of dummy terms which reveal they are equivalent. Hence, choosing the value of V slightly larger than $U/2$ provides an optimum choice. Another issue that can be observed from the figures is that, increasing U, which enables larger values for V, also decreases the dissimilarity of queries.

Let the adversary be able to access all the search history (i.e., all previous queries from all users). If the adversary can find k queries that all feature an arbitrary search term w, where the dissimilarity function $\Omega(\mathcal{A}_k, Q_{k+1}, Q'_{k+1})$ is greater than z for that k, then the adversary can identify with a high confidence level that all that k queries include the same search term and may learn the trapdoor of the search term w with a small error. Therefore, the probability of finding such k queries should be negligible.

We provide an example using the Reuters news dataset [49] that shows the difficulty of finding a trapdoor of a search term. Without loss of generality, we assume the adversary tries to find the trapdoor of the most commonly queried search term.

Example 5. *In large databases, the occurrence frequencies of the real keywords are considerably small. For instance, in the Reuters dataset we use, after the stop words are removed, the most frequent keyword occurs in only 7% of all the documents in the dataset. We assume that the same statistics apply to the real search patterns, which implies one keyword w can occur at most 7% of all the queries.*

Let there be 1000 queries in the history H_n, where 70 of them are expected to feature the keyword w. The probability of finding such k queries where the most frequent keyword occurs in $p\%$ of the queries in a database of n queries, is

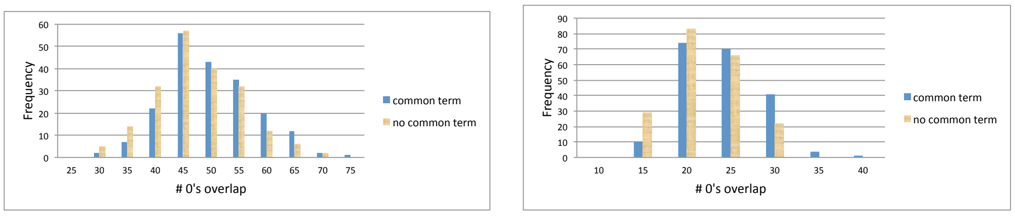
$$\frac{C_k^{np/100}}{C_k^n},$$

which is approximately 2^{-19} for $k = 5$, 2^{-39} for $k = 10$, 2^{-60} for $k = 15$ and 2^{-81} for $k = 20$ in a database, where $n = 1000$ and $p = 7$. In other words, when $k = 20$, adversary has to try 2^{81} combinations of queries to find a *correct set. Note that increasing n has a very minimal effect on the calculated probability. For practical purposes,* $k > 15$ *satisfies sufficient security level, which implies that it is not feasible to find k queries that all feature the same keyword. The attacker may find queries featuring the same keyword for smaller values of k, but this time identifying whether they all include the same search term or not will be hard, as shown in the following section.*

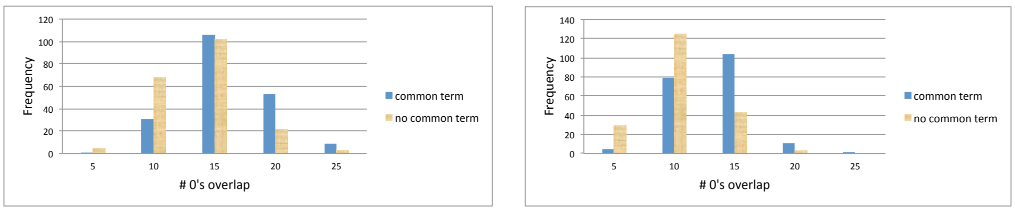
4.4.2 Experiments on Correlation Attack

In order to demonstrate that our analyses are valid, we conducted experiments using synthetic data. Given a set of queries, we want to analyze the probability of identifying whether all the set elements contain a common genuine search term or not. We generate histograms that compare the number of 0's coinciding in two sets of queries. All queries in both sets have k genuine and V dummy search terms. While the first set has a common genuine search term w that exists in all the queries, the second set does not have any common genuine search term. We further compute the confidence levels indicating the reliability of the guess. While a confidence level of 0.5 means that one cannot do better than a random guess out of the two sets, a confidence level of 1.0 means that one can certainly identify the correct set from the given two sets. For the case where $U = 60$ and $V = 40$, the histograms that compare the number of coinciding 0's for a set of k queries are given in Figure 4.5. In Table 4.1 we enumerate the confidence levels calculated from the histograms.

We want the confidence level of the attacker to be less than 0.6. Our experiments indicate that, setting the security threshold z for the dissimilarity function Ω (equation (4.9)), as $z = 0.4$ gives sufficient level of obfuscation that satisfies the required low confidence level. From Figure 4.4 that



(a) k=5 (b) k=10



(c) k=15 (d) k=20

Figure 4.5: Histograms that compare the number of 0's coinciding in k queries with a common search term and those with no common search term where U=60 and V=40

shows the dissimilarity values for different U and V values, the candidates that satisfy the required security level are found as, $\{U = 50, V = 40\}$, $\{U = 60, V = 40\}$ and $\{U = 80, V = 55\}$.

4.4.3 Hiding Dummy Elements

During the query randomization process, we add $U \in \mathcal{U}$ dummy keywords in all the entries in the index. Similar to the genuine keywords, those dummy keywords are processed with the HMAC and *Reduce* functions following the steps of Algorithm 1, which eventually maps some of the d-bit digits to single 0 bits. Since these U dummy keywords exist in all the entries, the bits that are assigned by those keywords are 0 in all the index entries $\mathcal{I}_{R_i} \in \mathcal{I}$. If the adversary has access to the searchable index file (e.g., cloud service provider), he can trivially identify bits set by dummy keywords by just marking bits

k	Confidence level
5	0,55
10	0,57
15	0,59
20	0,67

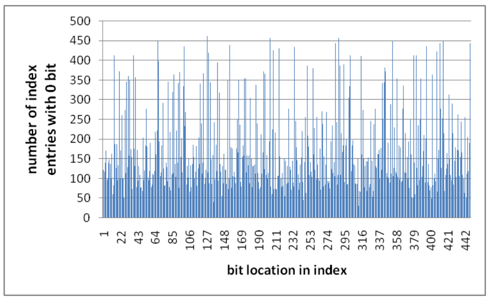
Table 4.1: Confidence levels of identifying queries featuring the same search term

that are 0 in all the index entries.

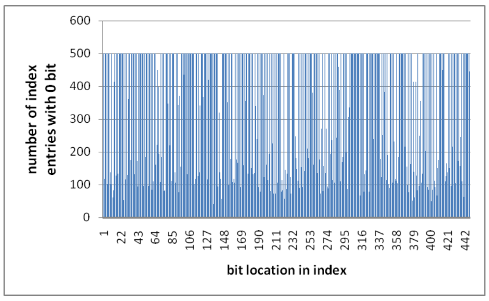
In order to hide those bits set by the dummy keywords, we add fake index entries, where their bits are deliberately set. The adversary cannot distinguish the bits that are set by genuine keywords, from the bits set by dummy keywords, if the distributions of the number of 0's are equivalent in both cases. Figure 4.6 shows the number of 0's in each bit location for the cases;

1. only genuine search terms (cf. Figure 4.6(a)),
2. after adding U dummy keywords (cf. Figure 4.6(b)),
3. after the addition of fake index entries (cf. Figure 4.6(c)).

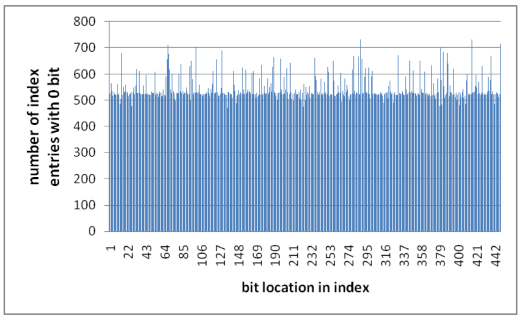
Figure 4.6 indicates that prior to the addition of fake elements (cf. Figure 4.6(b)), the bits set by the dummy keywords are obvious, since all the index entries contain the 0 bit in the same bit locations. However, after the addition of fake entries (cf. Figure 4.6(c)), they become indistinguishable from the other bits. The number of fake entries is chosen to be equal to the number of genuine entries leading to doubling index size. However, since size of an index entry is very small (constant r bits) this is not a burden for the cloud server. Note that, the additional fake entries do not have any effect on the number of false positives (i.e., false accept rates) and hence, precision of the method is not affected. Although, the search time increases due to the increased number of index entries, the increase in search time is not significant due to the fact that it can be performed very efficiently in a parallel fashion.



(a) before addition of dummy keywords



(b) with dummy keywords



(c) after addition of fake index entries

Figure 4.6: Number of 0's in each bit location for 500 genuine and 500 fake entries (for (c))

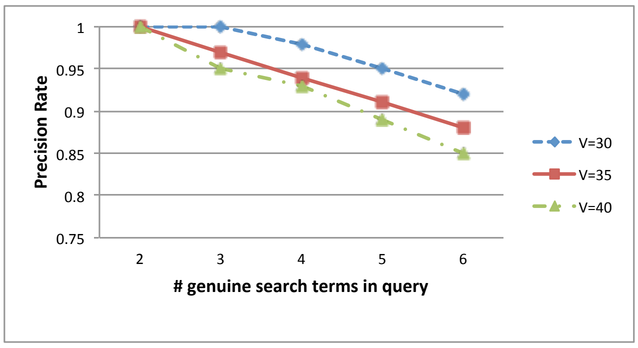
The search server may have access to excessive number of search results from various users. Utilizing the search results, the server can identify some of the fake entries with high confidence via analysis of the number of matches with each index entry. Note that the expected number of matches with fake index entries is smaller than the genuine index entries. We propose two methods to prevent the correlation of fake index entries. Firstly, the data controller can change the HMAC keys and the pseudo identifiers of index entries periodically. This will alleviate the risk by limiting the search server's access to search results. Alternatively, a trusted proxy can be utilized to occasionally send fake queries that match with the fake index entries. If number of matches of the fake index entries has a similar distribution with the genuine index entries, then fake index entries are indistinguishable from the genuine index entries.

4.4.4 Success Rates

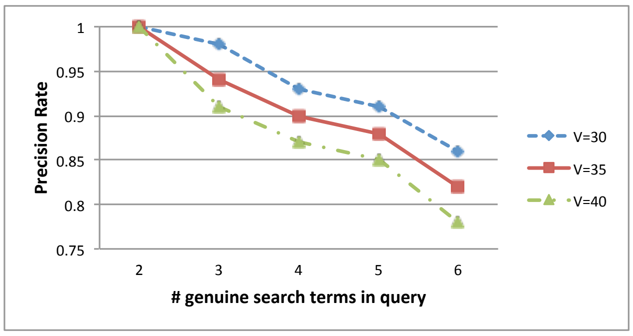
The indexing method that we employ includes all the information on search terms in a single r-bit index file per document. Despite the fact that the hash function employed in the implementations is collision-free, after the reduction and bitwise product operations there is a possibility that index of a query may wrongly match with an irrelevant document, which is called as a *false positive* (i.e., false accept).

As defined in Section 3.6, the success of a secure search method is measured with precision and recall metrics. The recall of the proposed HMAC based scheme is always one, which means that if a document contains all the queried features, then it will definitely be a match with the query. Although all matching documents can be identified by the method, there may also be some false positives, which affects the precision.

Let m be the number of genuine features in a document. In Figure 4.7, precision is measured for queries with 2, 3, 4, 5 and 6 search terms for index size $r = 448$ bits and $U = 60$, whereby in Figures 4.7(a) and 4.7(b) each document in the database has 30 and 40 genuine features (i.e., $m = 30$ and $m = 40$), respectively. When the number of genuine search terms in a query is small (3 or less), the noise in the query is limited, which results in



(a) number of genuine search terms per doc is 30



(b) number of genuine search terms per doc is 40

Figure 4.7: Effect of V in precision rate, where $U = 60$

almost no false positives (fp), hence precision rates are very close to 1. When the number of search terms is higher (4 or more), the number of matching documents gets very small. As the number of matching documents is small, the effect of the false positive matches is higher on the precision.

Figure 4.7 also indicates that an increase in the number of search terms in documents (m) also decreases precision. Note that precision decreases from Figure 4.7(a) to Figure 4.7(b) as m increases from 30 to 40. This is a result of the increase in the number of 0's in the index entries of the documents. If larger number of keywords is required for a document, a longer HMAC function can be used with a larger reduction parameter d. Recall that, as shown in Equation (4.3) in Section 4.3.1, reduction maps a d-bit digit to a single bit where output is 0, with probability $1/2^d$ and 1, with probability $1-1/2^d$. Therefore, the ratio of the number of zeros in an index entry with m genuine and U dummy keywords, with respect to index size, can be estimated as $\frac{m+U}{2^d}$. Similarly each keyword is approximately represented with $\frac{r}{2^d}$ zero bits in an index entry. If d gets larger and r is kept constant, the number of zeros in the index decreases, which may cause some keywords being not represented in the index. Provided that the ratios $\frac{m+U}{2^d}$ and $\frac{r}{2^d}$ do not change, the precision will expectedly remain constant. If the number of genuine keywords in a document (m) doubles, the number of dummy keywords (U) also doubles and d is incremented by 1 to keep the ratio of $\frac{m+U}{2^d}$ constant. Due to the increase in d, r is also doubled to keep $\frac{r}{2^d}$ constant. In Figure 4.8, we present the required output sizes for the HMAC functions (l) and the index entries (r) with respect to the total number of keywords (genuine and dummy) in documents such that, minimum precision does not fall behind 75%.

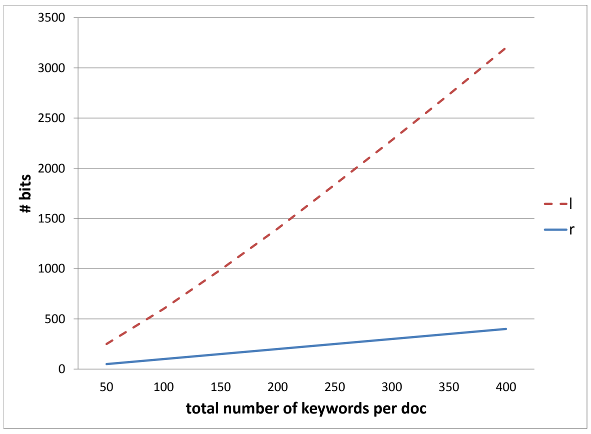


Figure 4.8: Effect of increase in the total number of keywords $(m + U)$ per document on HMAC size (l) and index entry size (r)

Figure 4.8 indicates that the increase in the index entry size, r, is quite limited and the proposed method can still be efficiently applied for large number of keywords per document. Although the usage of longer HMAC functions increases the cost of the index generation, since the increase in the index size, r, is limited, the communication cost, storage requirements and search time will remain at acceptable levels, without affecting the overall efficiency of the proposed scheme. Optimized value for the index size should be determined considering the requirements of the applications.

Increasing the level of obfuscation (i.e., addition of more dummy keywords) increases the security of the method against an adversary trying to identify queries that feature the same genuine search term. Nevertheless, this will also decrease precision rate. The results presented in Figure 4.9 compare the precision rates of the three settings found in Section 4.4.2 that satisfy the security requirements.

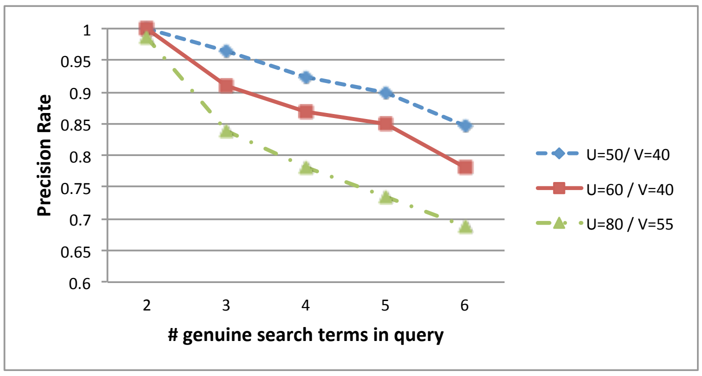


Figure 4.9: Precision comparison, where number of genuine search terms per document is $m = 40$

Figure 4.9 indicates that the precision decrease with the number of dummy keywords (U) added. While precision for $U \leq 60$ are acceptable, the precision for $U = 80$ is not suitable for various applications since it immediately incurs additional communication cost.

For a fixed U, the level of obfuscation increases as V gets larger (Figure 4.4), however precision decreases with the increase of V (Figure 4.7). Therefore, we use the smallest V that satisfies sufficient obfuscation (i.e., satisfy $\Omega(\mathcal{A}_k, Q_{k+1}, Q'_{k+1}) < z$) as the optimum choice for V.

Given two queries with the same genuine search terms, the probability of having exactly the same set of dummy keywords must be very small. Otherwise the generated queries may be exactly the same, which leaks the information that the genuine search terms for the both queries are the same. This probability is $\left(C^U_V\right)^{-1}$, which is minimized if $U = 2V$. For the two parameter settings that satisfy both high precision and sufficient obfuscation, the probability of having the same set of dummy keywords is 2^{-52} for $U = 60, V = 40$ and it is 2^{-33} for $U = 50, V = 40$. Hence, the optimum parameters, for the data set we consider, are set as $U = 60, V = 40$. Note that the number of genuine keywords in each document is set as $m = 40$ in the experiments. Utilizing these tests on the Reuters data set [49], we can generalize an optimum setting for U, V and m in the corresponding data set, as follows:

$$\frac{m}{U} = \frac{2}{3} \ \& \ \frac{V}{U} = \frac{2}{3}.$$

4.5 Hiding Response Pattern

In the previous section, it is demonstrated that, linking queries that feature the same genuine search terms is not feasible, provided that the randomization parameters (i.e. U and V) are set appropriately. However, if the attacker has access to the database, (e.g., the cloud service provider) it may be possible to correlate queries with the same search terms since the list of matching results will be almost the same except for some false positives. Note that different queries can match with the same index entries due to different keywords in those documents. Nevertheless, if the list of matching documents is the same, then the attacker can guess with a high confidence that the queries are also the same. Similar to the randomization method we use in Section 4.4.3, we propose to add some fake index entries[4] to the database such that the lists of matching documents for the two queries with the same genuine search terms will be different.

In the basic scheme given in Section 4.3, in addition to the genuine keywords, index entry of each document possesses U random dummy strings, where random V of them are added to each query index. In the modified method, similar to the real documents, the fake entries include both genuine search terms and dummy strings. The genuine search terms are placed in the fake entries according to the distribution in the real data set, but with a constant factor more frequent. We define a frequency enhancer constant c as in the following.

Definition 16. *(Frequency Enhancer (c)) Fake entries in the index file include genuine search terms more frequently than real documents with a factor of c (i.e., frequency enhancer).*

Intuitively, if a genuine search term w occurs in $p\%$ of the real documents in the database, it also occurs in $c \cdot p\%$ of the fake entries.

While the dummy strings are chosen with uniform distribution from the set $\mathcal{U}$, the number of dummy strings selected for a fake entry, which we denote as V', must be carefully set. The subsequent sections provide the analysis on as to how the values of V' and c are set.

[4]Users, but not the server, can identify the fake index entries. Since there is no document corresponding to fake entries, they will be discarded by the user.

Note that, the fake entries that are added for hiding the response pattern are generated in a different way from the ones we use for hiding positions of dummy keywords in Section 4.4.3. The fake entries generated in this section (i.e., Section 4.5) include both genuine search terms and dummy strings hence, can match with queries. However, the fake entries generated in Section 4.4.3 do not possess the dummy keywords and cannot match with the queries.

4.5.1 Analysis on Selecting Number of Fake Entries

Note that, the search index entry of each genuine document contains all the U dummy strings, while the queries have only V out of those U strings. Let the fake document entries possess the dummy strings in a set $\mathcal{V}' \subseteq \mathcal{U}$, where $|\mathcal{V}'| = V'$. In order to match a query with a fake document entry, all the genuine search terms and the dummy strings in the query should also exist in that fake document entry which implies, V should be smaller than V'. If $V' = U$ as in the real document index entries, the lists of matching index entries of documents for two queries with same genuine search terms would be identical, a case which we want to avoid. Hence, the inequality, $V < V' < U$, must be satisfied. Small V' reduces the number of fake document entries that match with a query. However, it increases the probability that the sets of matching fake entries to queries with same genuine search terms are different.

Given a query and a fake entry that possesses all the genuine search terms of that query, we denote the probability that the query matches with the fake entry as p_F, where the definition is as follows.

$$p_F = \prod_{i=0}^{U-V'-1} \frac{U-V-i}{U-i}. \tag{4.10}$$

In Section 4.4.1, we show that an optimal choice for the parameters of dummy strings in our setting is $U = 60$ and $V = 40$. In Figure 4.10, we plot the values of p_F with respect to V', while U and V are fixed. The figure shows that, for values of $V' \leq 57$, p_F is very low, which drastically reduces the number of fake matches and thus decreases obfuscation. Therefore, the only two possible choices of V' for the setting used, are 58 and 59. This can also be generalized as V' should be chosen as very close but smaller than U.

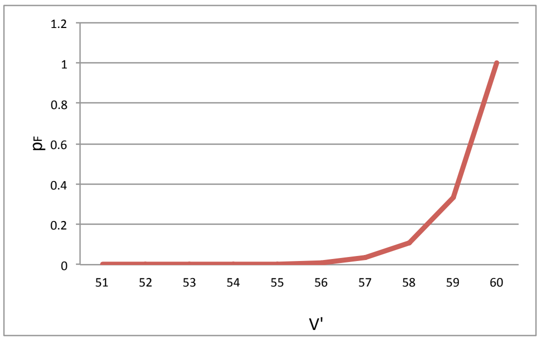


Figure 4.10: p_F values with respect to V', where $U = 60$ and $V = 40$

The security of the system is analyzed in the following section.

4.5.2 Correlating Match Results

Given a query, the number of matching fake document entries should be larger than the number of matching real documents. Otherwise, correlating two queries with the same genuine search terms can be possible. Let σ be the number of real documents in the database, we add $q \cdot \sigma$ fake entries to the index file. Also let f be the fake match enhancer such that if a query matches to a real document with probability p, it matches with a fake entry with probability $f \cdot p$, which is calculated as $f = c \cdot p \cdot p_F$, where p_F is as defined in Equation (4.10) and c is the frequency enhancer as defined in Section 4.5. Then, the number of matching fake document entries will approximately be

$f \cdot \sigma$. While this method increases the storage requirement for the database index by a factor of q, since the index entry size is very small (constant r bits) this is not a burden for the cloud server. Nevertheless, search and index generation times also increase with a factor of q, therefore q needs to be minimized. We set $q = 1$ and increase c to satisfy $c \cdot q \cdot p_F = f$.

The server can correlate two queries with the same genuine search terms if the number of index entries that the both queries match, is significantly larger than the average number of entries that any two arbitrary queries both match. We provide a theoretical analysis on the number of common matching entries for a given pair of queries. Note that there are three ways a document index entry can match to a query:

1. A real document entry can match if the document possesses all the genuine search terms in the query,
2. A real document entry can falsely as a false positive,
3. A fake document entry can match if the fake document entry possesses all the genuine search terms and dummy strings in the query.

Let x be the number of genuine search terms in the query and p_i is the frequency of the i^{th} search term in the database, assuming that the occurrences of search terms in a document are independent events, the expected number of index entries that match from case 1 ($E(M_1)$) is:

$$E(M_1) = \sigma \prod_{i=1}^{x} p_i$$

Let FP_x be the false positive rate of a query containing x genuine search terms and q be the multiplicative factor for the number of fake index entries as defined in this section. Then, the expected number of index entries that match from case 2 ($E(M_2)$) is:

$$E(M_2) = \mathrm{FP}_x \cdot (q+1)\sigma$$

Let p_F be as defined in Equation (4.10), the expected number of index entries that match from case 3 ($E(M_3)$) is:

$$E(M_3) = c\, q\, \sigma\, p_F \prod_{i=1}^{x} p_i = f\ \sigma\ \prod_{i=1}^{x} p_i$$

$E(M_1)$ is the expected number of true positive matches while $E(M_2)$ and $E(M_3)$ are the expected numbers of false positives (accidental and intentional respectively). Therefore, we denote $E(M_1)$ as $E(T^+)$ and $E(M_2) + E(M_3)$ as $E(F^+)$.

The expected total number of index entries that match to a query with x genuine search terms ($E(M)$) is:

$$E(M) = E(T^+) + E(F^+)$$

Note that, $E(T^+) < \dfrac{E(F^+)}{f}$ and $E(M_2)$ is reasonably small due to the small false accept rates given in Figure 4.9. This implies that

$$E(M) \leq (f+1)E(T^+). \tag{4.11}$$

Given two arbitrary queries Q and Q', the expected number of common index entries that both queries match, denoted as $E(C_{arb})$, is estimated as:

$$\begin{aligned} E(C_{arb}) &= \frac{E(T_Q^+)}{\sigma}\frac{E(T_{Q'}^+)}{\sigma}\sigma + \frac{E(F_Q^+)}{q\,\sigma}\frac{E(F_{Q'}^+)}{q\,\sigma} q\,\sigma \\ &\approx \frac{E(T_Q^+)\,E(T_{Q'}^+)}{\sigma} + \frac{f^2\,E(T_Q^+)\,E(T_{Q'}^+)}{q\,\sigma}. \end{aligned} \tag{4.12}$$

Given two queries Q and Q' that have the same genuine search terms, the expected number of common index entries that both queries match, is estimated as:

$$\begin{aligned} E(C_{same}) &= E(T^+) + \frac{E(F_Q^+)}{q\,\sigma}\frac{E(F_{Q'}^+)}{q\,\sigma}\,q\,\sigma \\ &\approx E(T^+) + \frac{f^2\,E(T^+)^2}{q\,\sigma} \end{aligned} \tag{4.13}$$

We assume that the two queries have equal number of search terms in the queries compared (i.e., they have similar number of index entries matched) and therefore, $E(T_Q^+) \approx E(T_{Q'}^+)$. Otherwise, they can easily be identified as different queries. We define an identifiability function $\mathcal{S}(Q_1, Q_1', Q_2)$ that takes three queries where Q_1 and Q_1' have the same genuine search terms and Q_2 is an arbitrary query and returns a value indicating the identifiability of Q_2 from Q_1.

Definition 17. *(Identifiability Function)*

$$\mathcal{S}(Q_1, Q_1', Q_2) = \frac{E(C_{same}) - E(C_{arb})}{E(C_{same})}$$

The identifiability function can be calculated as:

$$
\begin{aligned}
\mathcal{S}(Q_1, Q_1', Q_2) &= \frac{E(C_{same}) - E(C_{arb})}{E(C_{same})} \\
&= \frac{\left(E(T^+) + \dfrac{f^2\, E(T^+)^2}{q\,\sigma}\right) - \left(\dfrac{E(T_Q^+)\, E(T_{Q'}^+)}{\sigma} + \dfrac{f^2\, E(T_Q^+)\, E(T_{Q'}^+)}{q\,\sigma}\right)}{\left(E(T^+) + \dfrac{f^2\, E(T^+)^2}{q\,\sigma}\right)} \\
&= \frac{E(T^+) - \dfrac{E(T^+)^2}{\sigma}}{\left(E(T^+) + \dfrac{f^2\, E(T^+)^2}{q\,\sigma}\right)} \\
&= \frac{1 - \dfrac{E(T^+)}{\sigma}}{\left(1 + \dfrac{f^2\, E(T^+)}{q\,\sigma}\right)} \\
&= \frac{1 - \dfrac{\sigma\ \prod_{i=1}^{x} p_i}{\sigma}}{\left(1 + \dfrac{f^2\, \sigma\ \prod_{i=1}^{x} p_i}{q\,\sigma}\right)} \\
&= \frac{1 - \prod_{i=1}^{x} p_i}{\left(1 + \dfrac{f^2\ \prod_{i=1}^{x} p_i}{q}\right)} \\
&= \frac{1 - \prod_{i=1}^{x} p_i}{(1 + f^2\ \prod_{i=1}^{x} p_i)} \qquad \text{since we set } q = 1 \qquad (4.14)
\end{aligned}
$$

If we have the following inequality,

$$\mathcal{S}(Q_1, Q_1', Q_2) \leq \check{\epsilon}$$

where $\check{\epsilon}$ is a security threshold, we say that the attacker cannot identify whether two queries are from same search terms or not, from the information of matching index entry ids. Note that f and $\mathcal{S}(Q_1, Q_1', Q_2)$ are inversely proportional and therefore, we set f as large as possible by adjusting parameters q and c.

4.5.3 Experimental Results

We conducted tests on the real data set [49] of $30,000$ database index entries (10,000 real, 10,000 fake from Section 4.4.3 and 10,000 fake from Section 4.5) to demonstrate the success in hiding the response patterns. We randomly generate four groups of genuine search term sets which contain from 2 up to 5 search terms. Each group has 200 elements with total of 800 sets of search terms. For each search term set, we generate 2 queries (i.e., same genuine search terms with different dummy keywords) and measure the similarity between the sets of matching index entry ids. Similarly we also generate another test group. This time, we apply 200 tests for each group by generating two different queries within the same group and measure the similarity between the sets of matching index entry ids. The results are illustrated in Figure **??** where $U = 60, V = 40$ and $V' = 59$. In order to decrease c, we use the largest possible V' which is $U - 1$. Note that although higher f implies lower identifiability which is desirable, increasing f necessitates increasing c, which also increases number of genuine search terms in fake index entries. In our tests we set $f = 5$, which is the largest f that keeps the number of keywords in fake index entries the same as the number of keywords in genuine index entries. Note that the proposed method also provides search with single term, however we applied multiple terms in our experiments to emphasize the multi-keyword search property of the proposed method and the privacy issues in this setting.

4.6 Ranked Search

The multi-keyword search method, explained in Section 4.3, checks whether queried keywords exist in an index entry or not. If the user searches for a single or a few keywords, there will possibly be many correct matches, where some of them may not be useful for the user at all. Therefore, it is difficult to decide as to which documents are the most relevant. We add ranking capability to the system by adding extra index entries for frequently occurring keywords in a document. With ranking, the user can retrieve only the top τ matches where τ is chosen by the user.

In order to rank the documents, a ranking function is required, which assigns relevancy scores to each index entry corresponding to a given search query. In this thesis, we utilize the tf-idf metric for ranking the results, where the details of the relevancy metrics used in information systems are summarized in Section 3.5.

We assign tf-idf weights to each search term in each document. Instead of using these weights directly, we assign relevancy levels based on the weights of search terms. The proposed search scheme is conjunctive and requires the document to contain all the queried search terms for a match.

We assume that there are Λ levels of ranking in our proposed method for some integer $\Lambda \geq 1$. For each document, each level stores an index entry for search terms with higher weights of that document in a cumulative way in descending order. This basically means that the i^{th} level entry includes all the keywords in the $(i+1)^{th}$ level and also the keywords that have sufficient weight for the i^{th} level. The higher the level, the higher the weight of the search term is. For instance, if $\Lambda = 3$, level 1 index entry includes keywords that occur at least once in the document while levels 2 and 3 include keywords that have tf-idf$_{w,R}$ values at least, say 0.1 and 0.2 [5], respectively. There are several variations for relevancy score calculations [50] and we use a very basic

[5]The number of levels and the weights of each level can be chosen in any convenient way.

method. The relevancy score of a document is calculated as the number representing the highest level search index entry that the query matches.

All the keywords that exist in a document are included in the first level search index entry of that document as explained in Section 4.3.1. The other higher level entries include the frequent keywords that also occur in its previous level, but this time they have to occur the number of times, which should at least be equal to the tf-idf of the corresponding level. The highest level includes only the keywords that have the highest tf-idf values. In the oblivious search phase, the server starts comparing the user query against the first level index entries of each document. The matching documents found as a result of the comparison in the first level are then compared with the index entries in the other levels as shown in Algorithm 3.

Algorithm 3 Ranked Search

```
for all documents $D_i \in \mathcal{D}$ do
    Compare(level$_1$ entry of $D_i$ , query)
    $j = 1$
    while match do
        increment $j$
        Compare (level$_j$ entries of $D_i$, query)
    end while
    rank of $D_i$ = highest level that match with query
end for
```

In this method, some information may be lost due to the ranking method employed here. Rank of two documents will be the same if one involves all the queried keywords infrequently and the other involves all the queried keywords frequently except one infrequent one. The rank of the document is identified with the least frequent keyword of the query. We tested the correctness of our ranking method by comparing with a commonly used formula for relevance score calculation [19], given in the following:

$$Score(\mathcal{W}, R) = \sum_{w \in \mathcal{W}} \frac{1}{|R|} \text{tf-idf}_{w,R} \tag{4.15}$$

where $\mathcal{W}$ is the set of search terms in a query and $|R|$ is the length of the document R.

We use the Reuters [49] dataset to compare the two ranking methods. We generate a database of 10.000 documents and test with 100 queries of 2, 3, 4 and 5 genuine search terms each. In our proposed ranking there is no ordering within the matches from the same level. The number of elements matched in each level set is correlated with the number of levels.

- For the case where $\Lambda = 5$, 94% of the time the top match for the given relevance score, is also in the top match level (i.e., matches with highest level) for our proposed ranking method. Additionally, in 82% of the time, at least 4 of the top 5 matches for the given relevance score are in the top match level in our method.

- For the case where $\Lambda = 6$, 90% of the time the top match for the given relevance score, is also in the top match level (i.e., matches with highest level) for our proposed ranking method. Additionally, in 79% of the time, at least 4 of the top 5 matches for the given relevance score is in the top match level in our method.

Note that as Λ gets larger, number of genuine search terms in each level decreases which causes less number of matches in top levels. Since a user asks for top k rank levels, higher Λ provides user to retrieve accurate information with less communication. Nevertheless, reducing the number of matches per level slightly increase the probability of missing some top relevant documents in the top match level. For the given experiment on 10000 documents, average number of matching document entries for each level is given in Table 4.2. The table shows that while in the lowest level, the average match rate is approximately 0.5%, for the top level the match rate is about 0.03%.

$\Lambda \setminus$ lvl	6	5	4	3	2	1
5	NA	3.8	11.7	21.9	28.2	59.2
6	2.7	9.5	15.6	23.4	25.3	46.7

Table 4.2: Number of matching documents per level

While this new method necessitates an additional r-bit storage per level for a document, it reduces the communication overhead of the users since matches with low rank documents will not be retrieved unless the user requests. Considering Λ search entries are stored instead of a single search index entry per document, storage overhead for indexing mechanism increases Λ times due to ranking. This additional cost is not a burden for the server since the size of an index entry is usually negligibly small compared to actual document sizes.

4.7 Privacy of the Method

The privacy-preserving multi-keyword search (MKS) method must provide the user and data privacy requirements specified by definitions in Section 4.1. This section is devoted to the proofs that the proposed method satisfies these privacy requirements. In the proofs, we assume that the randomization parameters are selected appropriately by taking into consideration of the database or search statistics.

The proposed method is semantically secure against chosen keyword attacks under indistinguishability of ciphertext from ciphertext (ICC). The formal proof is provided in Theorem 1 of [22] from which we adopt their indexing method; therefore, we omit this proof here. Intuitively, the proof is based on the property that, since the HMAC function is a pseudo random function (PRF), the hash values of any two different keywords will be two random numbers and be independent of each other. Therefore, given two keyword lists L_0, L_1 and an index $\mathcal{I}_b$ for the keyword list L_b, where $b \in \{0, 1\}$, it is not possible for an adversary to guess the value b with probability greater than 1/2.

The security against chosen keyword attack is required, but not sufficient for the privacy-preserving search scheme we define in Section 4.1. We consider further privacy requirements that the work in [22] does not satisfy. The major difference of our work from [22] is capability of hiding the search pattern.

Lemma 1. *Given three queries Q_0, Q_1 and Q'_b where $b \in \{0, 1\}$ and Q_b and Q'_b are generated from the same genuine search terms, for a suitable parameter choice, the advantage of an attacker in finding b is very small.*

Proof. In Section 4.4, it is shown that $\Delta(Q_0, Q'_b) \approx \Delta(Q_1, Q'_b)$ for the selected values of U, V. Therefore, an attacker cannot do better than a random guess, to find b, if the parameters U and V in the index generation are securely chosen. $\square$

Note that, the values of the security parameters depend on the structure of the database and any $V \geq 30$, where $U = 60$, is a secure candidate pair for our test database.

Theorem 1 (Query Privacy)**.** *The proposed MKS method, for a suitable parameter choice, satisfies query privacy in accordance with Definition 3.*

Proof. Let the adversary A be an authorized user. Given the search term lists L_1 and L_2, the attacker can get the corresponding trapdoors from the data controller and generate corresponding queries. By Lemma 1, it is not possible to correlate the generated query with the given query Q_b unless all the V random dummy keywords are also same. There are C_V^U possible random choices of V for each set of search terms, where C_V^U is the number of V-combinations from a set of U elements. Therefore, A needs to generate and try C_V^U queries and compare with Q_b. Generating a single query requires choosing a set of V dummy keywords from the set $\mathcal{U}$, combining the trapdoors of those chosen keywords by bitwise product operation and finally comparing with the given query Q_b. For appropriate choice of U and V, generating C_V^U queries and applying bitwise comparison for each of them is infeasible. □

A numerical example that demonstrates the difficulty of the attack, for the parameters that we use in our implementation is shown in the following example.

Example 6. *In our setting this operation should be repeated $C_V^U = C_{40}^{60} = 2^{52}$ times. Note that assuming generating a single index from the trapdoors that are given in advance followed with a binary comparison requires 0.1 ms, this brute force search takes 10^4 years which is infeasible for all practical purposes.*

Theorem 2 (Access Control)**.** *The proposed MKS method satisfies access control in accordance with Definition 4.*

Proof. All communication from the user to the data controller is authenticated by a signature with the user's private key. We assume that the private key information of the authorized users is not compromised and we further assume that the server is semi-honest. In order to impersonate an authorized user with an RSA public-key e_u, A needs to learn the private key d_u where

$e_u \cdot d_u = 1 \mod \phi(N)$. Therefore, the probability of impersonating an authorized user is ϵ where ϵ is the probability of breaking the underlying RSA signature scheme. □

Lemma 2. *Given the list of all previous queries $\mathcal{Q}$ and a single query Q, it is not possible to find the list of queries from the set of queries $\mathcal{Q}$ that are generated from exactly the same genuine keywords with Q.*

Proof. The trivial approach will be one-by-one comparison of Q with each element of $\mathcal{Q}$. However, by Lemma 1, it is proven that adversary cannot identify equality of queries with one-by-one comparison.

An advanced approach will be applying correlation attack. The adversary may try to find a set of k queries that all possess a genuine search term w. If the adversary can extract genuine search term information from a set of queries, he can correlate Q with other queries that are generated from exactly the same genuine keywords with Q. However, we have shown in Section 4.4.1 that it is not possible to apply correlation attack in our proposed scheme. □

Lemma 3. *Given the database (i.e., the searchable secure database index), the list of all previous queries $\mathcal{Q}$ and a single query Q, it is not possible to find the list of queries from the set of queries $\mathcal{Q}$ that are generated from exactly the same genuine keywords with Q .*

Proof. With the additional index file access information, the adversary (e.g., cloud server) can also use the information of the list of ordered matching items with the query Q while comparing with other queries in $\mathcal{Q}$. As shown in Section 4.5, due to the additional fake document index entries, it is not possible to correlate two queries using one-by-one comparison between the lists of matching documents. □

Theorem 3 (Search Pattern Privacy)**.** *The proposed MKS method satisfies search pattern privacy in accordance with Definition 5.*

Proof. By lemma 2, it is shown that the query information do not leak useful search pattern information. By lemma 3, it is shown that the information of retrieved matching entries with a given query does not leak search pattern information. Therefore, the proposed scheme satisfies search pattern privacy. □

Theorem 4 (Adaptive Semantic Security)**.** *The proposed MKS method provides adaptive semantic security in accordance with Definition 11.*

Proof. Let the original view $v(H_n)$ and the trace $\gamma(H_n)$ be

$$\begin{aligned} v(H_n) =& \{(id(C_1), ..., id(C_l)), C, \mathcal{I}, \mathcal{Q}, \texttt{Dsig}(\mathcal{Q})\} \\ \gamma(H_n) =& \{(id(C_1), ..., id(C_l)), (|C_1|, ..., |C_l|), |\texttt{Dsig}(\mathcal{Q})|, |\mathcal{I}|, A_p(H_n)\}. \end{aligned}$$

Further let,
$v^*(H_n) = \{(id^*(C_1), \ldots, id^*(C_l)), C^*, \mathcal{I}^*, \mathcal{Q}^*, \texttt{Dsig}(\mathcal{Q})^*\}$ be the view simulated by the simulator S. The proposed method is adaptive semantically secure if $v(H_n)$ is indistinguishable from $v^*(H_n)$.

- The first component of the view $v(H_n)$ is the document identifiers $id(C_i)$ which are also available in trace. Hence, S can trivially simulate document identifiers as $id^*(C) = id(C)$. Since $id^*(C) = id(C)$, they are indistinguishable.

- Each user identifier is encrypted using a pseudo-randomness against chosen plaintext attack (PCPA) secure encryption method (e.g., AES in CTR mode). The output of a PCPA-secure encryption method [18] is by definition indistinguishable from a random number that has the same size with ciphertext. For simulating the ciphertexts C, S assigns l random numbers to C^* such that $C^* = \{C_1^*, \ldots, C_l^*\}$, where $\forall i, |C_i^*| = |C_i|$. Note that size of each ciphertext is available in the trace. Considering for all i, C_i and C_i^* are indistinguishable, C and C^* are also indistinguishable.

- The secure searchable index $\mathcal{I}$ is composed of index entries for each genuine profile and fake profiles. Note that, each index entry is generated by applying "bit-wise product" operation on the HMAC outputs of each attribute value in the profile. S simulates the searchable index $\mathcal{I}$, by generating $|\mathcal{I}|$ index entries using the public HMAC function with a random key S chooses. For each entry in simulated $\mathcal{I}^*$, S randomly selects a set of U attributes (e.g., random strings) and apply the HMAC and the reduction methods accordingly, where U is the number of dummy elements introduced in the index entries as defined in Section 4.4. An index entry in $\mathcal{I}^*$ is generated with exactly the same method as an entry in $\mathcal{I}$ other than the HMAC key. Hence, an entry of $\mathcal{I}^*$ is indistinguishable from an entry in $\mathcal{I}$. Moreover, both real and simulated index has the same number of index entries (i.e., $|\mathcal{I}| = |\mathcal{I}^*|$). Therefore, $\mathcal{I}$ is indistinguishable from $\mathcal{I}^*$.

- $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$ is a set of n consecutive queries where each query Q_i is a bitwise product of trapdoors for the search keywords of Q_i. $|Q_i|$ is the number of keywords in query Q_i. Note that each trapdoor is an output of the HMAC function which is a random bit string of size r, where $\frac{r}{2^d}$ bits are expected to be zero.

  Simulator S generates $\mathcal{Q}^*$ as follows. For each Q_i^*, S generates $|Q_i|$ random binary strings where the number of zero bits is a random number with mean $\frac{r}{2^d}$ and take the bit-wise product of those $|Q_i|$ strings to generate Q_i^*. The number of search terms $|Q_i|$ is available in the trace. Both Q_i and Q_i^* are bit-wise products of same number of bit stings of length r with same number of expected zeros. Therefore Q_i and Q_i^* are indistinguishable. Since $\forall i$ Q_i and Q_i^* are indistinguishable, $\mathcal{Q}$ and $\mathcal{Q}^*$ are also indistinguishable.

- The RSA signature of a query $\texttt{Dsig}(Q_i)$ is a random looking number of size $|\texttt{Dsig}(Q_i)|$ (i.e., size of public key N) which is available in trace $\gamma(H_n)$. To simulate $\texttt{Dsig}(\mathcal{Q})$, S assign n random numbers of size $|\texttt{Dsig}(Q_i)|$ to $\texttt{Dsig}(\mathcal{Q})^*$. $\forall i$ $\texttt{Dsig}(Q_i)$ and $\texttt{Dsig}(Q_i)^*$ are two random numbers of same length and indistinguishable. Hence, $\texttt{Dsig}(\mathcal{Q})$ and $\texttt{Dsig}(\mathcal{Q})^*$ are also indistinguishable.

The simulated view v^* is indistinguishable from genuine view v since each component of v and v^* are indistinguishable. Hence, the proposed method satisfies adaptive semantic security. $\square$

4.8 Complexity

In this section, we present an extensive cost analysis of the proposed technique. The communication and computation costs will be analyzed separately. Especially, low costs on the user side are crucial for rendering the proposed technique feasible for mobile applications, where the users usually perform the search through resource-constrained devices such as smart phones. We use both real and synthetic data sets in our analysis. The used real data set is a part (10.000 documents) of the RCV1 (Reuters Corpus Volume 1), which is a corpus of newswire stories made available by Reuters, Ltd. [49].

- Communication Costs:

  Two steps in the proposed method are identified, where communication is required: i) for learning the trapdoor and ii) for sending the query and receiving the results.

  1. *Between the user and the data controller, for learning the trapdoor*: To build a query, the user first determines the bin IDs of the keywords he wants to search for and send these values to the data controller. Let γ be the number of genuine search terms the user queries. Then the user sends at most $32 \cdot \gamma$ bits to the data controller together with a signature since each bin ID is represented by a 32 bit integer. The data controller replies with the HMAC keys that belong to those bins. The reply is encrypted with the user's public-key, so the size of the result is $\log N$. Note that if two search terms happen to map to the same bin, then sending only one of them will be sufficient since their responses will be the same.

  2. *Between the user and the search server, for query*: After learning the trapdoor keys, the user calculates the query and transmits it to the server. The size of the query is r bits, independent from γ, so the user transmits only r bits. Let α be the number of index entries matched with the query. The server returns the index entries of the matching documents whose size is $\alpha \cdot r$ bits in total. Note that size of the encrypted document depends on the size of the actual document. Independent of the used scheme, all privacy-preserving search methods return the encrypted document, therefore the communication cost of document retrieval is

not considered here. In the case where the ranking is used, only the top τ matches are returned to the user by the server instead of α where $\tau \leq \alpha$.

The communication costs are summarized in Table 4.3.

	Trapdoor	Search
User	$32 \cdot \gamma + \log N$	r
Data Controller	$\log N$	0
search server	0	$\alpha \cdot r$

Table 4.3: Communication costs incurred by each party (in bits)

- Computation Costs:

  Among the three parties participated in the protocol, computation cost of the user is the most crucial one. The data controller and the server can be implemented on quite powerful machines, however the users may be using resource-constrained devices.

  1. *User:* After receiving trapdoor keys from the data controller, query is generated as explained in Section 4.3, which is essentially equivalent to performing hash operations[6].
  2. *search server:* The search server performs only the search operation, which is binary comparison of r-bit query with $(q + 2) \cdot \sigma$ database entries, each of which is again an r-bit binary sequence. Note that there are σ genuine database entries, σ fake entries

[6]Computing bitwise product is negligible compared the overall operations the user performs.

added for hiding bit positions of dummy keywords (Section 4.4.3), and $q \cdot \sigma$ fake entries added for hiding response pattern (Section 4.5), which add up to a total of $(q+2) \cdot \sigma$ database entries. If the ranking is used, the query should also be compared with higher level index entries of the matching documents. So the server performs Λ additional binary comparison of r-bit index entries for each matching document, where Λ is the number of levels.

3. *Data Controller:* The data controller creates the index file and symmetric-key encryption operations of all documents; but these operations are performed only once in the initialization phase. Other than this, data controller is active while the user learns the trapdoors, which requires one encryption and one signature. This is equivalent to 2 modular exponentiation operations.

The computation costs are summarized in Table 4.4.

User	1 hash and bit-wise product
Data Controller	initialization phase 2 modular exponentiation per search
Index Server	$(q+2) \cdot \sigma \cdot \Lambda$ binary comparison over r-bit index entries

Table 4.4: Computation costs incurred by each party

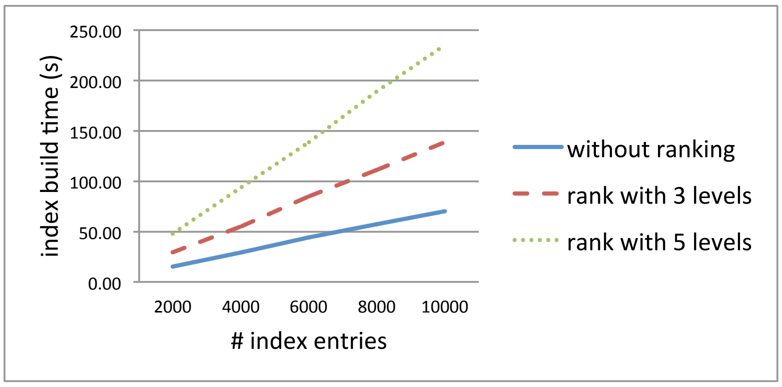
4.8.1 Implementation Results

The entire system is implemented by Java language using socket programming on an iMac with Intel Core i7 processor of 2.93 GHz. Considering that initial document analysis for finding the keywords in the document is out of the scope of this work, a synthetic database is created by assigning random keywords with random term frequencies for each document. The HMAC function produces outputs, whose size (l) is 336 bytes (2688 bit), which is generated by concatenating different SHA2-based HMAC functions. We choose $d = 6$ so that after the reduction phase the result is reduced to one-sixth of the original result; therefore the size of each database index entry and (r) is 56 bytes (448 bits).

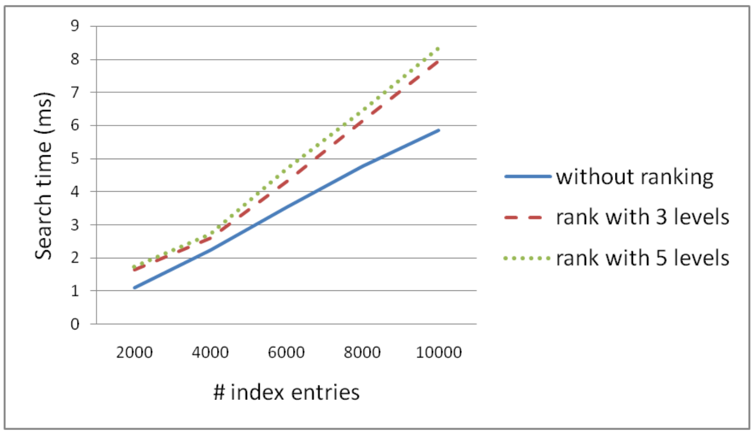
In our experiments, we used different data sets with different number of documents (from 2000 to 10000 documents). The timing results for creating the queries are obtained for documents with 30 genuine search terms and 60 random keywords each using ranking technique with different rank levels for parameters $q = 1$ and $f = 5$ in Figure 4.11(a). Considering that index generation is performed only occasionally (if not once) by the data controller and that index generation problem is of highly paralleled nature, the proposed technique presents a highly efficient and practical solution to the described problem.

Figure 4.11(b) demonstrates the server timings for a search with different rank levels. As can be observed from the graphic in Figure 4.11(b), time spent by the server per query is quite low, rendering high-throughput for the processing of user queries possible. By parallelization and native language support, the throughput can be increased by several orders of magnitude.

Most of the privacy-preserving search methods that exist in the literature are only capable of single keyword search. The problem that we consider is multi-keyword search; therefore, we did not provide a comparison with the works that consider only single keyword search. A very recent work by Cao et al. [23] is the closest work to our proposed method. Our implementations show that our method is one to two orders of magnitude faster than the method in [23] in both off line and on line operations. The index con-



(a) Timings for index construction with 30 genuine and 60 random keywords per document (on data controller side)



(b) Timings for query search (on server side)

Figure 4.11: Timing results

struction method of [23], takes about 4500 s for 6000 documents while we need only 140 s in the highest rank level. Similarly the work in [23] requires 600 ms to perform a search over 6000 documents where we need only 4.7 ms. The tests in [23] were done on an equivalent computer, Intel Xeon processor 2.93 GHz. Among the other existing multi-keyword solutions, bilinear pairing based methods such as [15] provide only theoretical solutions. The method in [15] is not implemented due to its excessive computational requirements hence, cannot be compared with our proposed work. The work of Wang et. al. [22], which is the inspiration for our proposed method, provides a faster solution than our work since they do not use additional fake entries or dummy keywords. However, that work does not satisfy some of the privacy requirements that we are interested in (cf. Section 4.7), such as hiding search pattern privacy.

The low time requirements on the data controller side enable processing multiple requests with high-throughput. Note that the programs used in the experiments are developed in Java language for portability reasons and unoptimized. Further optimization or support of native code or parallel implementation will further increase the performance of the proposed system.

4.9 Chapter Summary

The solution proposed in this chapter, addresses the problem of privacy-preserving ranked multi-keyword search, where the database is outsourced to a semi-honest remote server. Our formal definitions pertaining to the privacy requirements of a secure search method are based on a comprehensive analysis of possible attack scenarios. One particular privacy issue concerning linking of queries featuring the identical search terms is often overlooked in the literature. When an attacker is able to identify queries featuring the same search terms by inspecting the queries, their responses and database and search term statistics, he can mount successful attacks. Therefore, the proposed privacy-preserving search scheme essentially implements an efficient method to satisfy query unlinkability based on query and response randomization and cryptographic techniques. Query randomization cost is negligible for the data controller and even less for the user. Response randomization, on the other hand, results in a communication overhead when the response to a query is returned to the user since some fake matches are included in the response. However, we show that the overhead can be minimized with the optimal choice of parameters. The true cost is due to the additional storage for extended index file and the actual search time. This can also be minimized by proper selection of the parameters (i.e., the ratio of fake index entries to real index entries). On the other hand, the storage is usually not a real concern for the cloud servers, considering that index file is relatively small compared to the document sizes. As for the search time, the proposed technique is extremely efficient that a relative increase in search time can easily be tolerated. Our implementation results confirm this claim by demonstrating search time over a database of $10,000$ documents, including ranking, takes only a couple of milliseconds. Considering that the search algorithm easily yields to the most straightforward parallelization technique such as MapReduce, the overhead in search time due to the proposed randomization method effectively raises no difficulty.

Selection of parameters involves some knowledge about the database and therefore, a priori analysis is required. However, our proposal needs only the frequency of the most used search terms and number of search terms used in queries. The formulation for parameter selection is simple and easy to calculate. Furthermore, we do not need to repeat the calculation process for different datasets. One can easily specify an upper bound on the frequency of the most used search terms and number of search terms that can be used for many cases.

Ranking capability is incorporated to the scheme which enables the user to retrieve only the most relevant matches. The accuracy of the proposed ranking method is compared with a commonly used relevance calculation method where privacy is not an issue. The comparison shows that the proposed method is successful to return highly relevant documents.

We implement the entire scheme and extensive experimental results using both real and synthetic datasets demonstrate the effectiveness and efficiency of our solution.

Chapter 5

MINHASH-BASED SECURE SEARCH METHOD

The work presented in Chapter 4 provides a secure and efficient search method that hides the search pattern. The drawback of this method is the provided ranking approach. As explained in Section 4.6, the work presented in Chapter 4 supports only a fixed number of rank levels, where all the documents that match with the same level have the same rank such that, ranks of two documents will be the same even if one contains all the queried terms infrequently and the other one contains all the queried terms very frequently except only one infrequent term. Moreover, in the work given in Chapter 4, some fake document index entries are introduced in the searchable index as explained in Section 4.5. Those fake documents may match with the query with some probability given in equation (4.10), which is used for hiding the search pattern. However, the user then needs to sanitize the final result from the fake matches, which brings an extra burden on the user. In this chapter, we propose an alternative secure multi-keyword search scheme, that provides better ranking capability together with enhanced security requirements. The MinHash-based secure search model given in this thesis, has two versions. The first version, which we refer as the single server model, is very fast but some sensitive information is allowed to leak due to efficiency reasons. The single server model is presented in the IEEE CLOUD 2013 conference [38]. In the second version, which is referred as the two server model, the sensitive information that is allowed to leak in the single server model, is also hidden. However, due to the cryptographic primitives, the two server case, is not as efficient as the first one. There is a strict trade off between privacy and efficiency, and in the two server model, we managed to increase the privacy of the method while keeping the increase in the computational cost at an acceptable level. Both the single and the two server models provide multi-keyword search with ranking capability.

The both models proposed in this chapter consider the same problem of privacy-preserving keyword search over encrypted cloud data for the database outsourcing scenario as the model proposed in Chapter 4.

5.1 Single Server Framework

In the setting for the single server model, we assume the data owner does not have sufficient resources or is unwilling to store the whole database. He outsources the data to an untrusted, semi-honest server, but maintains the ability to search without revealing anything except the access and search patterns.

The data owner encrypts the sensitive documents to be outsourced and generates a searchable index using the features of these sensitive documents. In an offline stage, both searchable index and the encrypted documents are outsourced to a semi-honest cloud. Then, authorized users can perform

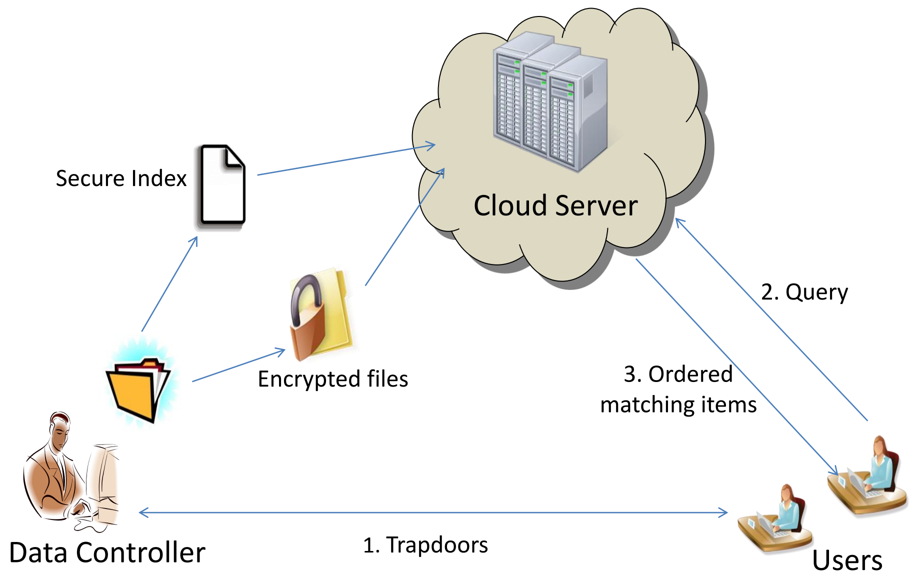


Figure 5.1: Framework of the model with a single server

search on the cloud and receive the encrypted documents that match with their queries. During this process, the cloud server should not learn anything other than what the data owner allows to leak. Finally, user decrypts the retrieved documents using the decryption keys. The steps and typical interactions between the participants of the system are illustrated in Figure 5.1.

The method is formalized as follows. Let $\mathcal{D}$ be the set of sensitive documents and F_i be the set of features (i.e., keywords) of $D_i \in \mathcal{D}$. There are four algorithms in the scheme, namely: setup, index generation, query generation and search.

1. $Setup(\Psi)$: Given a security parameter Ψ, it generates a secret key $K \in \{0,1\}^{\Psi}$.

2. $IndexGeneration(K, \mathcal{D})$: Given the collection of sensitive documents

$\mathcal{D}$, it extracts the feature set F_i for each document $D_i \in \mathcal{D}$ and generates a searchable secure index $\mathcal{I}$ via encryption with the key K.

3. *QueryGeneration*(K, F): Generates a query Q for the given set of features F with key K.
4. *Search*$(\mathcal{I}, Q)$: Query Q is compared with the searchable index $\mathcal{I}$ and returns encrypted versions C_i of the matching documents D_i.

The details of these algorithms are given in Section 5.2.

5.1.1 Security Model

The privacy definition for almost all of the existing efficient privacy-preserving search schemes allows the server to learn some information such as the search and access patterns. Therefore, due to efficiency concerns, the proposed single server search method, also leaks search and access pattern, but nothing else.

The definitions of Search pattern (S_p), Access pattern (A_p), History (H_n) and Adaptive Semantic Security, are as given in Section 4.1. The definitions of Trace and View, are redefined in this section, since the contents of trace and view are different in this model.

Definition 18. ***Trace*** ***(*$\gamma(H_n)$*)*** *Let* $C = \{C_1, \ldots, C_{|\mathcal{D}|}\}$ *be the set of encrypted documents,* $|\mathcal{D}|$ *be the number of documents in the dataset,* $id(C_i)$ *be the identifier of* C_i *and* $|C_i|$ *be the size of* C_i*. The trace of* H_n *is defined as* $\gamma(H_n) = \{(id(C_1), \ldots, id(C_{|\mathcal{D}|})), (|C_1|, \ldots, |C_{|\mathcal{D}|}|), S_p(H_n), A_p(H_n)\}$*. We allow to leak the trace to an adversary and guarantee no other information is leaked.*

Definition 19. ***View** (v) is the information that is accessible to an adversary. Let $\mathcal{I}$ be the secure searchable index and, $id(C_i)$ and $\mathcal{Q}$ are as defined above. The view of H_n is defined as $v(H_n) = \{(id(C_1), \ldots, id(C_{|\mathcal{D}|})), C, \mathcal{I}, \mathcal{Q}\}$.*

5.2 Single Server MinHash-based Method

In this section, we provide the crucial steps of our proposed method. Search over encrypted cloud data is performed through an encrypted searchable index that is generated by the data owner and outsourced to the cloud server. Given a query, the server compares the query with the searchable index and returns the results without learning anything other than the information that is allowed to be leaked due to efficiency concerns.

5.2.1 Secure Index Generation

Our proposed method utilizes the idea of bucketization which is a data partitioning technique widely used in the literature [43, 45, 20]. Here, each object is distributed into several buckets via *MinHash* functions introduced in Section 3.3.2 and the bucket-id is used as an identifier for each object in that bucket. This method maps objects such that the number of buckets, in which two objects collide, increases as the similarity between those objects increases. In other words, while two identical objects collide in all of the buckets, number of common buckets decreases as similarity between objects decreases. The proposed secure index is generated by the data owner utilizing the following phases, namely: feature extraction, bucket index construction and bucket index encryption. These three phases are explained in the following.

1) **Feature Extraction:** For each document $D_i \in \mathcal{D}$, the set of features

$F_i = \{f_{i1}, \ldots, f_{iz}\}$ that characterize the document is extracted. In our case, those features are composed of two values $f_{ij} = (w_{ij}, rs_{ij})$. The first one is a keyword w_{ij} of the sensitive document D_i. The second one is the relevancy score (rs), which is based on the tf-idf value of the keyword w_{ij} for the document D_i as explained in Section 3.5. This relevancy score is later used in the search method (cf. Section 5.7.3) while ranking the matching results.

2) **Bucket Index Construction:** We first construct a *MinHash* structure by selecting λ random permutations on the set of all possible keywords (Δ). We then apply the *MinHash* on the first values of each feature set $F_i[0] = \{w_{i1}, \ldots, w_{iz}\}$ as shown in Section 3.3.2 and generate a signature for each document as:

$$Sig(D_i) = \{h_{P_1}(F_i[0]), \ldots, h_{P_\lambda}(F_i[0])\}.$$

Note that $\forall i \in \{1, \ldots, \lambda\},\ h_{P_i}(F_j[0]) \in F_j[0]$. In other words, each signature element of a document is a keyword for that document.

Then, feature set of each document is mapped to λ buckets using the elements of the signature. Suppose $h_{P_i}(F_j[0]) = w_k$, then we create a bucket with bucket identifier B_k^i, and identifiers and relevancy scores of all the documents that satisfy this property are added to this bucket. The bucket content is a vector of integer elements of size $|\mathcal{D}|$, where $|\mathcal{D}|$ is the number of documents in the outsourced data set. Let B_k^i be a bucket identifier and $V_{B_k^i}$ be the integer vector,

$$V_{B_k^i}[id(D_j)] = \begin{cases} rs_{jk}, & \text{if } h_{P_i}(F_j[0]) = B_k^i \text{ and } V_{B_k^i}[id(D_j)] = 0, \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

3) **Bucket Index Encryption:** In this step, we hide the bucket identifiers and bucket contents due to privacy requirements.

Bucket identifier B^i_k is a sensitive information since it may reveal a search term in a query that matches with a bucket, so it cannot be kept as plaintext. Moreover, the server should be able to map the given encrypted bucket id to the one kept in the server, without knowing the decryption keys. Hence, the encryption method used for hiding the bucket identifier must be a deterministic scheme. One of the most efficient methods that hides a value in a deterministic way is HMAC functions which are essentially cryptographic hash functions that utilize secret keys. In our proposed scheme, decryption of the encrypted bucket identifier is not required so an HMAC function is used for hiding the bucket identifiers. Any pseudo random function (PRF) can also be used, but we preferred HMAC functions due to their efficiency. The secret key of HMAC function (K_{id}) is only known by the data owner and never revealed to the server.

The content of a bucket (i.e., $V_{B^i_k}$) possesses sensitive information such as the pseudo identifiers of the documents in that bucket and their relevancy scores. These information must also be protected from the untrusted server, hence should be outsourced to the server only after encryption. A proper approach for encrypting bucket contents would be using a PCPA-secure [18] (Pseudorandomness against chosen plaintext attacks) encryption method such as AES in CTR mode with a secret key ($K_{content}$).

Let max be the maximum size of a searchable index $\mathcal{I}$ and cnt be the number of real elements (i.e., number of buckets) in the index. We add $max - cnt$ dummy elements to the index in order to hide the number of buckets. The dummy elements $(\pi_{dum_i}, \mathcal{V}_{dum_i})$ are randomly generated with the condition that

$$|\pi_{B^j_k}| = |\pi_{dum_i}| \quad \text{and} \quad |\mathcal{V}_{B^j_k}| = |\mathcal{V}_{dum_i}|.$$

The secure index generation method is summarized in Algorithm 4.

Algorithm 4 Single Server Index Generation

```
Require: $\Delta$:set of possible keywords, $\mathcal{D}$: collection of documents, $h$: $\lambda$ *MinHash* functions, $\Psi$: security parameter

$K_{id} = Setup(\Psi)$, $K_{content} = Setup(\Psi)$
for all $D_i \in \mathcal{D}$ do
  $F_i \leftarrow$ extract features of $D_i$
  $Sig(D_i) = \{h_{P_1}(F_i[0]), \ldots, h_{P_\lambda}(F_i[0])\}$
  for $j = 1 \rightarrow \lambda$ do
    $B_k^j = Sig(D_i)[j-1]$
    if $B_k^j \notin$ bucket identifier list then
      add $B_k^j$ to bucket identifier list
      create $V_{B_k^j}$
    end if
    add $rs_{ik}$ to vector $V_{B_k^j}[id(D_i)]$
  end for
end for
for all $B_k^j \in$ bucket identifier list do
  $\pi_{B_k^j} \leftarrow HMAC_{K_{id}}(B_k^j)$
  $\mathcal{V}_{B_k^j} \leftarrow Enc_{K_{content}}(V_{B_k^j})$
  add $(\pi_{B_k^j}, \mathcal{V}_{B_k^j})$ to secure index $\mathcal{I}$
end for
add $max - cnt$ dummy elements $(\pi_{dum_i}, \mathcal{V}_{dum_i})$
return $\mathcal{I}$
```

Subsequent to the index generation, data owner encrypts each document in the dataset $\mathcal{D}$ as $\Omega_{id(D_i)} = Enc_{K_{data}}(D_i)$ and outsources this set of encrypted documents E_{Doc} to the server with the $\mathcal{I}$, where

$$E_{Doc} = \{(id(D_1), \Omega_{id(D_1)}), \ldots, (id(D_{|\mathcal{D}|}), \Omega_{id(D_{|\mathcal{D}|})})\}.$$

5.2.2 Query Generation and Search

The query generation is constructed in a similar way to the index generation phase (Section 5.2.1) and exact steps are detailed in Algorithm 5. Given a feature set of n keywords to be queried (i.e., $F = \{w'_1, \ldots, w'_n\}$), the user first creates the query signature from this feature set using the same *MinHash* functions that are used in the index generation phase. Then, for each signature element, the corresponding bucket identifier is hashed with the key K_{id}. The query Q is this list of hashed bucket identifiers (i.e., $Q = \{\pi_1, \ldots, \pi_\lambda\}$). Note that independent of the number of search terms in a query (n), the query signature has λ elements and therefore, the information of n is not leaked to the server.

Algorithm 5 Single Server Query Generation

```
Require: $F$: feature set of the keywords to be queried,
  $h$: $\lambda$ minhash functions, $K_{id}$: encryption key

  $Sig(F) = \{h_{P_1}(F[0]), \ldots, h_{P_\lambda}(F[0])\}$
  for $j = 1 \to \lambda$ do
    $B_k^j = Sig(F)[j-1]$
    $\pi_{B_k^j} \leftarrow HMAC_{K_{id}}(B_k^j)$
    $Q[j-1] = \pi_{B_k^j}$
  end for
  return $Q$
```

Given a query Q, the server finds the encrypted vectors ($\mathcal{V}_{B_k^j}$) corresponding to the bucket identifiers in Q. The server then sends back the λ encrypted vectors $E_V = \{\mathcal{V}_1, \ldots, \mathcal{V}_\lambda\}$ to the user. After receiving the buckets, user decrypts the vectors and ranks the data identifiers as it is detailed in Section 5.2.3.

5.2.3 Document Retrieval

The user wants to avoid returning unrelated documents since this immediately bring forth an unnecessary communication burden. Hence, user tends to retrieve only the top t matches, instead of returning all documents that share at least one bucket with the query. The standard formulation for calculating the document-term weights is tf-idf (cf. Section 3.5) which is commonly used for relevance score calculation in search methods. Therefore, we also utilize the tf-idf values for ranking the matching results.

Upon receiving the requested encrypted vectors $E_V = \{\mathcal{V}_1, \ldots, \mathcal{V}_\lambda\}$, the user decrypts those vectors and get the plain vectors as $V_i = Dec_{K_{content}}(\mathcal{V}_i)$. Then the documents are sorted according to their scores. Note that $V_i[id(D_j)]$ is the tf-idf value of document D_j for i^{th} bucket.

In the index generation phase, each document is mapped to a certain number of buckets using the output of the *MinHash* functions and tf-idf value of the *MinHash* output is assigned as the relevancy score of that document for that bucket. Similarly query Q is also mapped to some λ buckets. The score of a document D_j (i.e., $score(id(D_j))$) is the summation of the relevancy scores for the buckets that both document and query share, which is defined as follows:

$$score(id(D_j)) = \sum_{i=1}^{\lambda} V_i[id(D_j)]. \tag{5.2}$$

As the $score(id(D_j))$ gets higher, the relevancy of the document to the query is expected to increase.

After the ranking phase, the user retrieves the top t matches from the server. The document retrieval method is summarized in Algorithm 6. As the database is updated by adding or removing documents, tf-idf values need to be recalculated and indices should be updated accordingly. However, we assume the database is highly static, hence the update is done infrequently.

5.3 Privacy for the Single Server Model

The privacy-preserving search scheme that we propose is adaptive semantically secure according to Definition 11.

Theorem 5. *The proposed method satisfies adaptive semantic security in accordance with Definition 11.*

Proof. Let the original view $v(H_n)$ and the trace $\gamma(H_n)$ be

$$v(H_n) = \{(id(C_1), \ldots, id(C_{|\mathcal{D}|})), C, \mathcal{I}, \mathcal{Q}\},$$
$$\gamma(H_n) = \{(id(C_1), \ldots, id(C_{|\mathcal{D}|})), (|C_1|, \ldots, |C_{|\mathcal{D}|}|), Sim_p(H_n), A_p(H_n)\}.$$

Further let $v^*(H_n) = \{(id^*(C_1), \ldots, id^*(C_{|\mathcal{D}|})), C^*, \mathcal{I}^*, \mathcal{Q}^*\}$ be the view simulated by the simulator S. The proposed method is adaptive semantically secure if $v(H_n)$ is indistinguishable from $v^*(H_n)$.

- The first component of the view $view(H_n)$ is the document identifiers $id(C_i)$ which are also available in trace. Hence, S can trivially simulate document identifiers as $id^*(C) = id(C)$. Since $id^*(C) = id(C)$, they are indistinguishable.

Algorithm 6 Single Server Document Retrieval

```
  USER:
Require: $E_V$: encrypted vectors, $K_{content}$: secret key,
  $t$: limit for number of documents to retrieve

  for all $\mathcal{V}_i \in E_V$ do
    $V_i \leftarrow Dec_{K_{content}}(\mathcal{V}_i)$
  end for
  for $j = 1 \rightarrow |V_i|$ do
    $score(j) = \sum_{i=1}^{\lambda} V_i[j]$
  end for
  sort $score$ list
  idList $\leftarrow$ identifiers of top $t$ $scores$
  send idList to Server

  SERVER
Require: idList: requested document identifiers, $E_{Doc}$: outsourced en-
  crypted documents

  for all $id$ $\in$idList do
    if $(id, \Omega_{id}) \in E_{Doc}$ then
      send $(id, \Omega_{id})$ to user
    end if
  end for

  USER:
  $D_{id} \leftarrow Dec_{K_{data}}(\Omega_{id})$
```

- Each document is encrypted using a PCPA-secure[1] encryption method (e.g., AES in CTR mode). The output of a PCPA-secure encryption method [18] is by definition indistinguishable from a random number that has the same size with ciphertext. To simulate ciphertexts C, S assigns l random numbers to C^* such that $C^* = \{C_1^*, \ldots, C_{|\mathcal{D}|}^*\}$, where $\forall i, |C_i^*| = |C_i|$. Note that, size of each ciphertext is available in the trace. Considering for all i, C_i and C_i^* are indistinguishable, C and C^* are also indistinguishable.

- Note that $\mathcal{I}$ is composed of encrypted bucket identifiers and corresponding encrypted bucket content vectors. Let $size_B$ and $size_V$ be the sizes of bucket identifier and bucket content, respectively. Further let max be the maximum number of buckets that may occur in $\mathcal{I}$. Simulator S generates max index elements, $\mathcal{I}^*[i] = (\pi_i^*, \mathcal{V}_i^*)$ such that π_i^* is a random number, where $|\pi_i^*| = size_B$ and $\mathcal{V}_i^*$ is another random number, where $|\mathcal{V}_i^*| = size_V$. Note that π_i^* and π_i are indistinguishable since π_i is the output of a random function (i.e., HMAC) where the output is indistinguishable from a random number. Similarly, $\mathcal{V}_i^*$ and $\mathcal{V}_i$ are indistinguishable since $\mathcal{V}_i$ is a cipher of a PCPA-secure encryption method. Hence, $\mathcal{I}$ is indistinguishable from $\mathcal{I}^*$.

- $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$ is a set of n consecutive queries where each query Q_i is composed of λ encrypted bucket identifiers (i.e., $Q = \{\pi_1, \ldots, \pi_\lambda\}$). S can simulate the queries using the similarity pattern (Sim_p). Let $Q_i[j]$ be the j^{th} element of Q_i where $size_B$ is the size of bucket identifier.

[1]We used a PCPA-secure encryption in our analysis which is in parallel with the literature. However, an encryption that is secure against chosen keyword attack (IND2-CKA) [16] can also be used, but the corresponding security proof should be modified accordingly.

If $\exists p, r$ $1 \leq p \leq i$ and $1 \leq r \leq \lambda$ such that $Sim_p[i[j], p[r]] = 1$ set $Q_i^*[j] = Q_p^*[r]$. Otherwise, set $Q_i^*[j]$ to a random value R_i^j where $|R_i^j| = size_B$. Note that for all i, Q_i is indistinguishable from Q_i^* since Q_i is the output of a pseudorandom permutation and Q_i^* is a random number, and they are of the same length.

The simulated view v^* is indistinguishable from genuine view v since each component of v and v^* are indistinguishable. Hence, the proposed method satisfies adaptive semantic security. $\square$

5.4 Experiments (Single Server)

In this section, we extensively analyze the proposed method in order to demonstrate the efficiency and effectiveness of the scheme. The entire system is implemented by Java language using a 32-bit Windows 7 operating system with Intel Pentium Dual-Core processor of 2.30GHz. In our experiments we use the publicly available Enron dataset [51].

The success of the search method is analyzed using the precision and recall metrics (cf. Section3.6).

The matching items are ordered according to the relevancy scores (cf. Section 3.5) and only items with top t scores are retrieved. We analyzed the effect of the number of *MinHash* functions (λ) on the accuracy of the method for a fixed threshold $t = 15$, by taking the average of 1500 queries with number of features differ from 2 to 6 (i.e., 300 queries per each feature size). As Figure 5.2 demonstrates, recall of the proposed scheme is 1 for any $\lambda \geq 150$ implying that all of the items that contain all the features in the given query are retrieved by the user. For the database outsourcing scenario that we consider, it is crucial that the user retrieves all the documents matching with

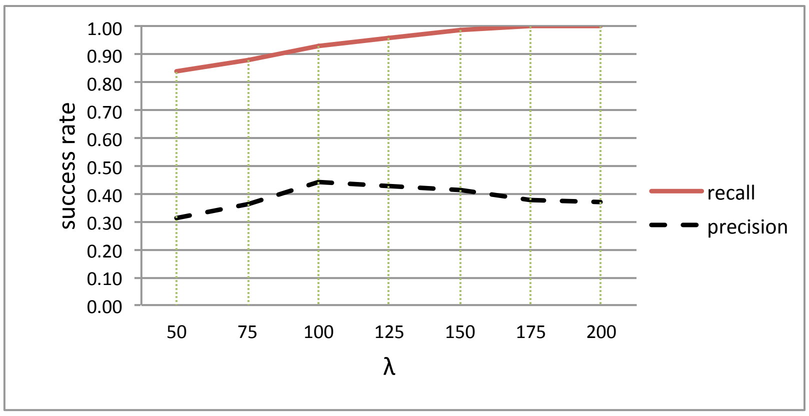
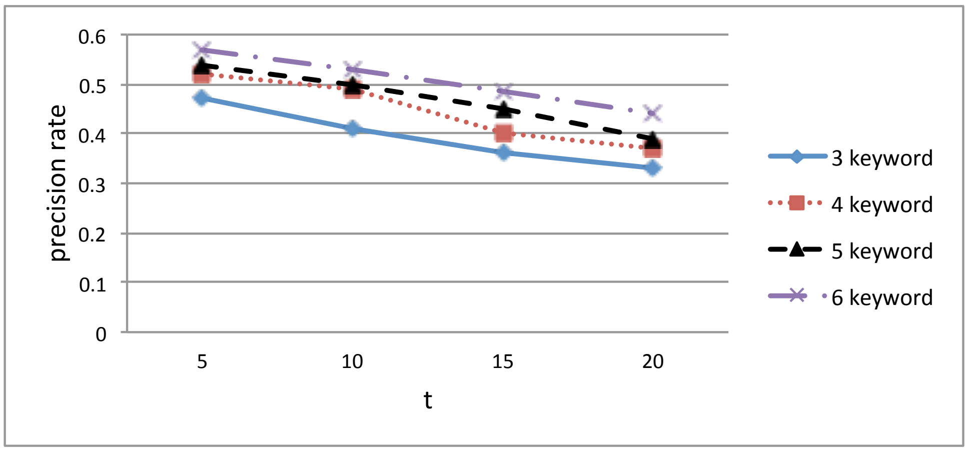


Figure 5.2: Success Rates as λ change for $t = 15$

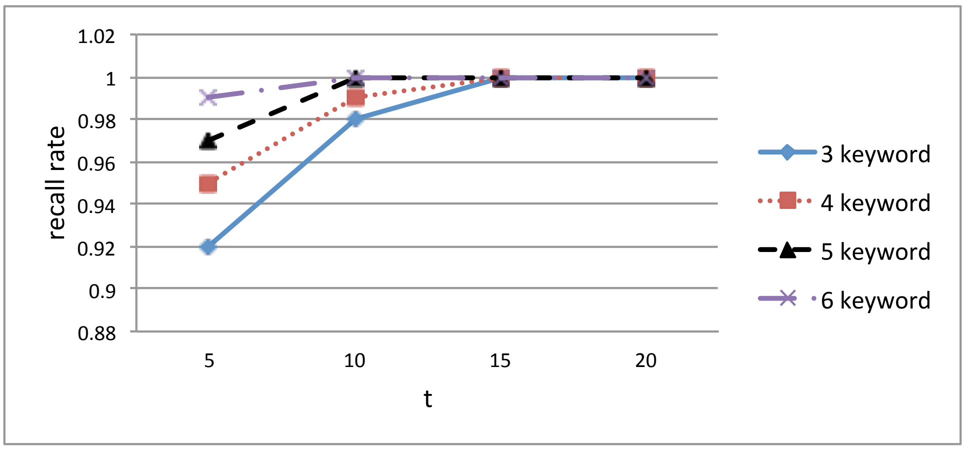
the queried feature set. Precision is rather small, which indicates about 40% of the retrieved documents contain all the queried features. Nevertheless, the other retrieved items are still relevant with the query. Those items contain a subset of the query features and the matching features have high relevancy scores indicating that the matching item is highly relevant to the query even when not all the features are captured. Note that, an item that has no matching feature with a query has zero relevancy score, hence cannot match with the query. We set $\lambda = 150$ since it satisfies the best precision rate while ensuring full recall.

We analyze the impact of the number of keywords in a query on the precision and recall rates and present the results in Figure 5.3. The similarity between query and document signatures increases as the number of common keywords increases. Hence, both the precision and recall rates of the method increase as the number of keywords in a query increases. The increase in success rate indicates our proposed method is even more useful for searches with more than 5 keywords.

We test the efficiency of our proposed method using various dataset sizes from 4000 to 10000 documents. The most costly operation of our method is



(a) precision



(b) recall

Figure 5.3: Impact of number of keywords in a query and t on the precision (a) and recall (b) rates

the index generation. Figure 5.4 shows that the index generation operation takes about a few minutes and linearly increases as the number of documents increases. Considering this operation is only performed in an offline stage by the data owner, the method is practical. One of the most important parameters of privacy-preserving search is the query response time since this operation is used very frequently and the users want to access their search results as fast as possible. Search operation does not depend on the number of documents since, in the proposed method search is performed by retrieving λ requested buckets which is constant. The average query response time for the single server search method, where $\lambda = 150$ is 210 ms, independent of the number of documents in the dataset.

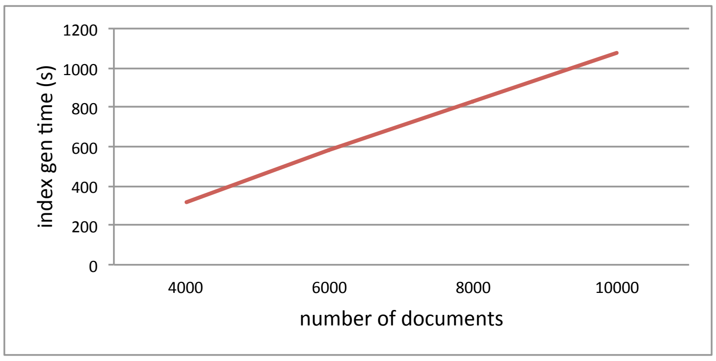


Figure 5.4: Timings for index construction for $\lambda = 150$

The user then needs to decrypt the encrypted content vectors and request the documents with the highest relevancy scores.

The communication cost of the user for the single server case has two phases. First, the encrypted matching vectors ($|E_V| = \lambda|\mathcal{V}_i|$bits) are received and in the next phase matching encrypted documents are received.

Most of the secure search methods in the literature do not support multiple features in queries. We do not provide any comparison with those single keyword search methods but compare our proposed method with the existing secure multi-keyword search methods instead. Some of the multi-keyword search methods utilize bilinear mapping such as [15]. This approach has similar security requirements with our proposed method, such that it reveals search and access pattern but nothing else. Unfortunately, this work is not implemented by the authors due to excessive computational requirements. In this work, each search operation does about $2l$ bilinear mapping operation where l is the number of features in a document, which is not practical due to the cost of bilinear map operations. A recent work by Cao et al. [23] utilizes matrix multiplication operations where the number of rows is determined by the size of the complete feature set. This method performs index construction for 6000 documents in about 4500 s, while we perform the same operation in less than 600 s. Similarly, the search operation over 6000 documents in [23] requires 600 ms, while we perform in about 210 ms. Our HMAC-based method [39] performs efficiently in both index construction and search operations. However the ranking of that method is not as accurate as the *MinHash*-based method and its security is ad-hoc, where random elements are added for hiding the properties of the genuine features.

5.5 Two Server Framework

In this work, we consider privacy-preserving keyword search over encrypted cloud data for the public storage system model. In our setting, we assume there are three entities in the system, namely: Data owner, two non-colluding semi-honest servers and users. The steps and typical interactions between the participants of the system are illustrated in Figure 5.5.

- *Data Owner* is the actual owner and provider of the data. We assume that, the data owner does not have sufficient expertise or resource to store the whole database, hence outsources it to the cloud. While the data owner and other authorized users retain the ability to search over the data, no sensitive information is leaked to the cloud server provider. In order to hide the sensitive data from the servers, the data owner encrypts it before outsourcing and generates a searchable secure index using the features of the sensitive data.

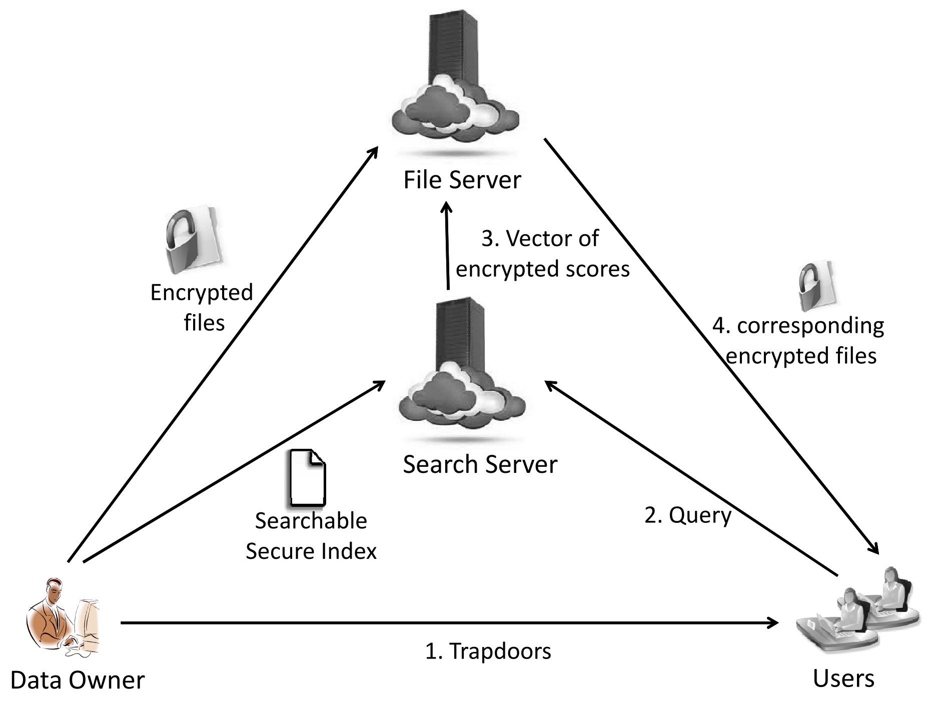


Figure 5.5: The framework of the method with two non-colluding servers

- *Cloud Server* is a semi-trusted professional entity that offers storage and computation services. In our setting, we utilize two non-colluding servers; namely the *search server* and the *file server*. The data owner outsources the searchable secure index to the search server and the actual encrypted documents are outsourced to the file server. Given a query to the search server, the encrypted search results (i.e., scores) are sent to the file server from the search server. The file server then decrypts the results and sends the corresponding encrypted documents with top t scores, to the user in a relevancy ordered way. The main reason of the necessity of two servers is that, in the single server case, the server can correlate a query with the matching document identifiers [38], which may cause some important information leakage such as revealing the search pattern (cf. Section 5.6).

- *Users* can apply secure search over the cloud server by generating a query using the search features. The query, which is also encrypted, is sent to the search server and the corresponding encrypted documents that match with the query is received from the file server. During this process, neither of the servers learn anything other than what the data owner allows to leak. Finally, user decrypts the retrieved documents using the decryption key.

The method is formalized as follows. Let $\mathcal{D}$ be the set of documents and F_i be the set of features (i.e., keywords) of $D_i \in \mathcal{D}$. There are four algorithms in the scheme, namely: setup, index generation, query generation and search.

1. *Setup*(Ψ): Given a security parameter Ψ, it generates a secret key $K \in \{0,1\}^{\Psi}$.

2. *IndexGeneration*$(K, \mathcal{D})$: Given the collection of sensitive documents $\mathcal{D}$, it extracts the feature set F_i for each document $D_i \in \mathcal{D}$ and generates a searchable secure index $\mathcal{I}$ via encryption with the key K.

3. *QueryGeneration*(K, F): Generates a query Q for the given set of features F using the key K.

4. *Search*$(\mathcal{I}, Q)$: Query Q is compared with the searchable secure index $\mathcal{I}$ and returns encrypted versions of the matching documents D_i.

The details of these algorithms are given in Section 5.7.

5.6 Two Server Security Model

The two server setting can provide enhanced security requirements compared to the single server setting such as the search pattern is not leaked. The server may have some background information such as the search frequency statistics of some keywords. If the search pattern is leaked, this information can be combined with the search frequency statistics to deduce certain keywords in the query. Hence, avoiding the leakage of search pattern is of paramount importance. In this section we provide the privacy definitions, which are partially borrowed from, but not limited to, the ones provided in Section 4.1.

The definitions of Search pattern (S_p), Access pattern (A_p), History (H_n) and Adaptive Semantic Security, are as given in Section 4.1. The definitions of Trace and View, are redefined in this section since the contents of trace and view are different in this model.

Definition 20. ***ϵ-Probability Distinguishability:*** *Let Q_a and Q_b be two queries that are generated from the feature sets F_a and F_b, respectively. Further let there exists a keyword w that both queries contain (i.e., $w \in F_a$ and $w \in F_b$). Also let a query Q be represented as a set of encrypted signature elements as given in Section 3.3.2 (i.e., $Q = \{\pi_1, \ldots, \pi_{|Q|}\}$).*

A privacy-preserving keyword search method provides ϵ-probability distinguishability if, the probability that the both queries Q_a and Q_b, have a common element π_w due to the keyword w, is less than ϵ. Specifically, the privacy preserving search scheme satisfies ϵ-probability distinguishability if,

$$prob(Q_a \cap Q_b \neq \emptyset) \leq \epsilon, \tag{5.3}$$

where ϵ is a security parameter.

Definition 21. ***δ-Mean Query Obfuscation:*** *Let Q, Q_0 and Q_1 be three queries such that for $b \in_R \{0, 1\}$, Q_b be generated from the same feature set,*

F that is used in Q and Q_{1-b} be generated from a different feature set. A privacy-preserving keyword search method provides δ-mean query obfuscation if, given Q, Q_0 and Q_1, the difference of the expected distances between, two equivalent queries and two arbitrary queries, is less than δ. Specifically, the privacy preserving search scheme satisfies δ-mean query obfuscation if,

$$Exp\left[|d(Q, Q_0) - d(Q, Q_1)|\right] \leq \delta, \tag{5.4}$$

where δ is a security parameter and $d(x, y)$ is any proper distance metric.

Definition 22. ***Similarity Pattern** (Sim_p) is the same with S_p with the extension for multiple features. Let feature set of Q_i be $F_i = \{f_i^1, \ldots, f_i^y\}$ and $(F_1, \ldots, F_n)$ be the feature sets of n queries. $Sim_p[i[j], p[r]]$ =1 if $f_i^j = f_p^r$ and 0, otherwise, for $1 \leq i, p \leq n$ and $1 \leq j, r \leq y$. Intuitively, similarity pattern is the information of the number of common features between two queries.*

Definition 23. ***Trace** ($\gamma(H_n)$) Let $C = \{C_1, \ldots, C_{|\mathcal{D}|}\}$ be the set of encrypted documents, $id(C_i)$ be the identifier of C_i, $|C_i|$ be the size of C_i and $|\mathcal{I}|$ be the number of buckets in the secure index $\mathcal{I}$. The trace of H_n is defined as:*

$$\gamma(H_n) = \{(id(C_1), \ldots, id(C_{|\mathcal{D}|})), (|C_1|, \ldots, |C_{|\mathcal{D}|}|), A_p(H_n), |\mathcal{I}|\}.$$

We allow to leak the trace to an adversary.

Definition 24. ***View** ($v(H_n)$) is the information that is accessible by an adversary. Let $\mathcal{I}$ be the searchable secure index and, $id(C_i)$ and $\mathcal{Q}$ be as defined above. The view of H_n is defined as:*

$$v(H_n) = \{(id(C_1), \ldots, id(C_{|\mathcal{D}|})), C, \mathcal{I}, \mathcal{Q}\}.$$

5.7 The Two Server MinHash-based Method

In this section we explain the construction of the proposed method. The privacy-preserving search is applied utilizing the searchable secure index that is generated by the data owner. Given a query from a user, the search server performs the search on the secure index without learning anything about the query and returns the encrypted intermediate results to the file server, which then sends the final results to the user.

5.7.1 Secure Index Generation with Query Obfuscation

Our proposed method utilizes the idea of bucketization; a data partitioning technique widely used in the literature [20, 43, 45]. Here, each object is distributed into a constant number of buckets via the *MinHash* functions introduced in Section 3.3.2 and the bucket-id is used as an identifier for each object in that bucket. Note that, this method maps similar objects (i.e., documents) into the same buckets with high probability. Consequently, the number of buckets shared by any two documents increases as the similarity between those two documents increases. Similarly, two documents without any common keyword do not share any bucket.

In the generation of the secure index, each document is represented by a set called signature (*cf.* Section 3.3.2 for signature generation). The signatures are comparable such that the distance between two documents can be estimated by comparing their signatures. However, the signature generation process for the single server setting given in Section 5.2.1 is deterministic, which means that for any two queries generated from exactly the same set of keywords, their corresponding signatures will be identical. This inevitably leaks the search pattern.

In this Section, we now modify the index construction and query generation methods by introducing randomness in query signature generation phase in order to obfuscate the search pattern. The proposed secure index is generated by the data owner utilizing the following phases, namely: feature extraction, bucket index construction and bucket index encryption. These three phases are explained in the following.

1. **Feature Extraction:** For each document $D_i \in \mathcal{D}$, the set of features $F_i = \{f_i^1, \ldots, f_i^y\}$ that characterizes the document is extracted. In our case, those features are composed of two values $f_i^j = (w_{ij}, rs_{ij})$. The first value is a keyword w_{ij} in the document. The second one is the relevancy score (rs_{ij}), which is based on tf-idf value of the keyword w_{ij} for the document D_i as explained in Section 3.5. This relevancy score is later used in the search method (cf. Section 5.7.3) while ranking the matching results.

2. **Bucket Index Construction with Obfuscation:** In [38], the *MinHash* structure is constructed by selecting λ random permutations on the set of all possible features (Δ). In the randomized method, instead, λ set of random permutations are used for document signatures, where each set is composed of ϕ permutations. Hence, in the proposed method each signature is composed of $\phi\lambda$ elements.

   The *MinHash* functions are applied on the first values (i.e., w_{ij}) of each feature set. Let F_i^* be the list of first elements in F_i as, $F_i^* = \{w_{i1}, \ldots, w_{iz}\}$, then the signature for each document $D_i \in \mathcal{D}$ is calculated as:

$$Sig(D_i) = \{\ \left(h_{P_{11}}(F_i^*), \ldots, h_{P_{1\phi}}(F_i^*)\right), \ldots, \left(h_{P_{\lambda 1}}(F_i^*), \ldots, h_{P_{\lambda\phi}}(F_i^*)\right)\ \}\,.$$

Note that, each signature element of a document is a feature (i.e., keyword) of that document.

Then, each document identifier is mapped to $\phi\lambda$ buckets using the elements of the signature of the document. Suppose $h_{P_i}(F_j^*) = w_k$, then a bucket vector is created with a bucket identifier B_k^i. The bucket content vector, $V_{B_k^i}$, is a vector of integers of the size of the outsourced data set, $|\mathcal{D}|$, where initially all the values in $V_{B_k^i}$ are set to 0. The relevancy scores of all the documents that are mapped to a bucket are then inserted to the corresponding bucket content vector. Let B_k^i be a bucket identifier and $V_{B_k^i}$ be the corresponding content vector then,

$$V_{B_k^i}[id(D_j)] = \begin{cases} rs_{jk}, & \text{iff } h_{P_i}(F_j^*) = B_k^i, \\ 0, & \text{otherwise.} \end{cases}$$

3. **Bucket Index Encryption:** Bucket identifier B_k^i is a sensitive information since it may reveal a search term in a query that matches with a bucket, so it must be encrypted. Moreover, the server should be able to map the given encrypted bucket id to the one kept in the server without knowing the decryption keys. Hence, the encryption method used for hiding the bucket identifier must be a deterministic scheme. One of the most efficient methods that hides a value in a deterministic way is the HMAC functions, which are essentially cryptographic hash functions that utilize secret keys. In our proposed scheme, decryption of the encrypted bucket identifier is not required so an HMAC function

is used for hiding the bucket identifiers. The secret key of the HMAC function (K_{id}) is not revealed to the servers. We denote the encrypted bucket identifier as $\pi_{B_k^i} = HMAC_{K_{id}}(B_k^i)$.

The content of a bucket ($V_{B_k^i}$) also possesses sensitive information such as the relevancy score of each document in a bucket. Moreover, the content vector also contains the information of pseudo identifiers of the documents that are not mapped to that bucket since their relevancy score will be zero. Those information must be protected from the cloud server, hence should be outsourced to the server only after encryption. The search over the encrypted data will be applied by the search server and due to the security requirements, the search server cannot apply decryption on the bucket content vectors. The homomorphic encryption schemes provide a solution for this problem since they allow operations such as addition over the ciphertext without applying decryption. We use the Paillier encryption [29], a well known additive homomorphic encryption method, for the encryption of the bucket content vectors $V_{B_k^i}$. The Paillier encryption algorithm satisfies the following homomorphic property;

$$Enc(m_1) \cdot Enc(m_2) = Enc(m_1 + m_2).$$

Prior to outsourcing to the search server, each element of the content vector is encrypted using the Paillier encryption with a secret key ($K_{content}$). We denote the encrypted content vector as $\mathcal{V}_{B_k^i} = Enc_{K_{content}}(V_{B_k^i})$. Note that, Paillier encryption provides semantic security against chosen plaintext attacks, hence, different encrypted outputs of the same message will be different and the encrypted values will be indistinguishable.

The secure index generation method is summarized in Algorithm 7.

Algorithm 7 Two Server Index Generation

```
Require: $\Delta$:set of possible keywords, $\mathcal{D}$: collection of documents, $h$: $\lambda$ Min-Hash functions, $\Psi$: security parameter

$K_{id} = Setup(\Psi)$, $K_{content} = Setup(\Psi)$
for all $D_i \in \mathcal{D}$ do
  $F_i \leftarrow$ extract features of $D_i$
  $Sig(D_i) = \{\left(h_{P_{11}}(F_i^*), \ldots, h_{P_{1\phi}}(F_i^*)\right), \ldots, \left(h_{P_{\lambda 1}}(F_i^*), \ldots, h_{P_{\lambda\phi}}(F_i^*)\right)\}$
  for $j = 1 \rightarrow \phi\lambda$ do
    $B_k^j = Sig(D_i)[j-1]$
    if $B_k^j \notin$ bucket identifier list then
      add $B_k^j$ to bucket identifier list
      create $V_{B_k^j}$, where all elements are 0
    end if
    set $V_{B_k^j}[id(D_i)] = rs_{ik}$
  end for
end for
for all $B_k^j \in$ bucket identifier list do
  $\pi_{B_k^j} = HMAC_{K_{id}}(B_k^j)$
  $\mathcal{V}_{B_k^j} = Enc_{K_{content}}(V_{B_k^j})$
  add $(\pi_{B_k^j}, \mathcal{V}_{B_k^j})$ to secure index $\mathcal{I}$
end for
return $\mathcal{I}$
```

Subsequent to the index generation, the data owner encrypts each document in the data set $\mathcal{D}$ as $\Omega_{id(D_i)} = Enc_{K_{data}}(D_i)$, using an encryption scheme that satisfies pseudo-randomness against chosen plaintext attacks

(PCPA) [18] (e.g., AES in CTR mode). The output of a PCPA-secure encryption is indistinguishable from any random bit sequence of the same length as the cipher text. Finally, the data owner outsources this set of encrypted documents E_{Doc} to the file server and the searchable index $\mathcal{I}$ to the search server.

5.7.2 Randomized Query Generation

The query generation is constructed in a similar way to the index generation phase (Section 5.7.1) and exact steps are detailed in Algorithm 8. Given a feature set of η keywords in a query (i.e., $F = \{w'_1, \ldots, w'_\eta\}$), the user first creates the query signature from this feature set using the *MinHash* functions that are used in the index generation phase. Different from the index generation, in the signature generation phase of a query Q, only a randomly chosen subset with c elements among the ϕ MinHash functions is used as:

$$Sig(Q) = \{ \left(h_{P_{1j_1}}(F), \ldots, h_{P_{1j_c}}(F)\right), \ldots, \left(h_{P_{\lambda j_1}}(F), \ldots, h_{P_{\lambda j_c}}(F)\right) \},$$

where $c < \phi$ and $j_i \in_R \{1, \ldots, \phi\}$.

Then, for each signature element, the corresponding bucket identifiers are hashed with the key K_{id}. The query Q is therefore, the list of hashed bucket identifiers (i.e., $Q = \{\pi_1, \ldots, \pi_{c\lambda}\}$). Note that independent of the number of keywords in a query (i.e., η), the query signature has always $c\lambda$ elements and therefore, the information of η is not leaked to the server.

5.7.3 Secure Search

Given a query Q, the search server finds the encrypted vectors ($\mathcal{V}_{B_k^j}$) corresponding to the bucket identifiers in Q. Utilizing the homomorphic properties

Algorithm 8 Two Server Query Generation

```
Require: $F$: feature set of keywords to be queried,
  $h$: $\lambda$ MinHash functions, $K_{id}$: encryption key

  for $i = 1 \rightarrow c$ do
    $j_i \in_R \{1, \ldots, \phi\}$
    for $l = 1 \rightarrow \lambda$ do
      add $h_{P_{lj_i}}(F)$ to $Sig(Q)$
    end for
  end for

  for $j = 1 \rightarrow c\lambda$ do
    $B_k^j = Sig(Q)[j-1]$
    $\pi_{B_k^j} = HMAC_{K_{id}}(B_k^j)$
    $Q[j-1] = \pi_{B_k^j}$
  end for
  return $Q$
```

of the encryption, the search server then computes an encrypted score vector using the $c\lambda$ encrypted vectors as $E_V = \prod_{i=1}^{c\lambda} \mathcal{V}_i$ and sends this single encrypted score vector E_V to the file server. After receiving E_V, the file server decrypts the vector and sorts the data identifiers. Finally, the encrypted documents with top t relevancy scores are sent to the user, where t is an arbitrary number, which is set in any convenient way. The two server search method is described in Algorithm 9.

Algorithm 9 Two-Server Secure Search
and Document Retrieval

```
  SEARCH SERVER:
Require: $\mathcal{I}$: secure index, $Q$: query
  $t$: limit for the number of documents to retrieve
  set $E_V := 1$
  for all $\pi_i \in Q$ do
    if $(\pi_i, \mathcal{V}_i = \{e_{i_1}, \ldots, e_{i_\mathcal{D}}\}) \in \mathcal{I}$ then
      for all $j = 1 \rightarrow \mathcal{D}$ do
        $E_V[j] = E_V[j] \cdot e_{i_j}$
      end for
    end if
  end for
  send $E_V$ and $t$ to File Server

  FILE SERVER:
Require: $S_L$ : encrypted scores, $K_{content}$: secret key,
  $K_{priv}$ : Paillier private key
  for all $i, E_V[i] \in S_L$ do
    $score(i) = Dec_{K_{priv}}(E_V[i])$
  end for
  sort all scores
  send the encrypted documents with the highest $t$ scores, to the user
```

5.8 Analysis of the Method of the Search Pattern Hiding

The search pattern is intuitively the search frequency of the queries, which can be found by checking the equality between the given query and the previous queries. Therefore, in order to hide the search pattern, identifying equality of two queries should be infeasible, i.e., δ-mean query obfuscation and ϵ-probability distinguishability should be satisfied.

Note that a query Q_a, is a set of encrypted bucket identifiers generated by the outputs of the chosen MinHash functions, where $|Q_a| = c\lambda$ (i.e., $Q_a = \{\pi_1, \ldots, \pi_{c\lambda}\}$).

A well-known metric for finding the similarity between two sets is the Jaccard distance, as given in Section 3.4.2 in the preliminaries.

In this work, the Jaccard distance is utilized in order to analyze the difference between the signatures of two queries. Hence, the Jaccard distance function J_d is used as the distance function d given in the definition of the *δ-mean query obfuscation* (cf. Definition 21).

Throughout our analysis, we use fundamental probability concepts such as permutation and combination. The number of permutations of length k from a set of n elements is denoted as P^n_k, which is equal to,

$$P^n_k = \frac{n!}{(n-k)!}.$$

Similar to permutation, combination is the number of different ways of selecting a set of k elements out of a group of n elements, where, unlike permutations, order does not matter. The number of k-combinations from a set with n elements is denoted as C^n_k and equals to,

$$C^n_k = \frac{n!}{k!(n-k)!}.$$

Given two queries Q_a and Q_b, generated from the same feature set F, the probability that they are not the same (i.e., $J_d(Q_a, Q_b) > 0$) can be formulated as:

$$prob(Q_a \neq Q_b) = 1 - (C_c^{\phi})^{-1} = 1 - \frac{c!(\phi - c)!}{\phi!}.$$

This probability is almost 1, especially if the parameters are chosen as $\phi = 2c$, which maximizes the combination C_c^{ϕ}. However, inequality of two queries does not imply indistinguishability. Let there be $\bar{c}$ common elements among the randomly chosen c *MinHash* functions, where $0 \leq \bar{c} \leq c$. Given the same set of keywords, the outputs of these $\bar{c}$ *MinHash* functions will be the same for all λ sets. Therefore, it can still be possible to distinguish equivalent queries, if there exist common *MinHash* functions such that $\bar{c} > 0$.

The ultimate obfuscation is satisfied when $\bar{c} = 0$, namely there are no common *MinHash* functions chosen in the generation of two queries. The probability that the second query randomly chooses the *MinHash* functions from the set of $\phi - c$ *MinHash* functions that are not chosen by the first query, is calculated as:

$$\begin{aligned} prob\left(Q_a \cap Q_b = \emptyset\right) &= \prod_{i=0}^{c-1} \left(\frac{\phi - c - i}{\phi - i}\right) \\ &= \prod_{i=0}^{c-1} \left(1 - \frac{c}{\phi - i}\right) \\ &= \frac{C_c^{\phi - c}}{C_c^{\phi}}. \end{aligned} \tag{5.5}$$

Note that $prob\left(Q_a \cap Q_b = \emptyset\right)$ increases as ϕ increases and c decreases. Therefore, the query obfuscation can be satisfied by utilizing a large ϕ and a small c pair.

Example 7. *Let ϕ be 20 and c be 2. Given two queries Q_a and Q_b generated from the same feature set, the probability that they share no common element* *(i.e., $J_d(Q_a, Q_b) = 1$) is calculated as:*

$$\begin{aligned} prob\left(Q_a \cap Q_b = \emptyset\right) &= \frac{C_c^{\phi-c}}{C_c^{\phi}} \\ &= \frac{C_2^{18}}{C_2^{20}} \\ &= \frac{18}{20} \cdot \frac{17}{19} \\ &= 0.805. \end{aligned}$$

With this parameter setting, the probability that two equivalent queries have totally unrelated signatures is greater than 80%, and this probability can further be increased by increasing ϕ.

Query obfuscation provides, same or similar queries look unrelated. However, if the server can correlate queries with the matching document identifiers, then similar queries can still be distinguished by comparing the common matching documents. Therefore, the two server search setting, which hides correlation between queries and the matching document identifiers, has paramount importance for the security of the method.

5.8.1 Expected Jaccard Distance

In the case, where two queries are generated from the same feature set F, the two queries have a common element if and only if the same *MinHash* function is used. The intuitive proof is as follows. If all the *MinHash* functions are different, by definition the outputs will be different since the *MinHash* functions identify the matching bucket identifiers and hence, the two queries will not have any common element. If there exists a *MinHash* function common for both queries, since the inputs for that function are the same, the outputs will also be the same.

Let Q_a and Q_b be two queries generated from the same feature set F. Using Equation (3.1), the Jaccard distance between the two queries is calculated as:

$$J_d(Q_a, Q_b) = 1 - \frac{\bar{c}\lambda}{2c\lambda - \bar{c}\lambda}, \tag{5.6}$$

where $\bar{c}$ is the number of common MinHash functions chosen by Q_a and Q_b.

Note that if $\bar{c} = 0$, the distance is $J_d(Q_a, Q_b) = 1 - 0 = 1$, which leaks no information that the two queries are related. However, if $\bar{c} = c$ then the distance is $J_d(Q_a, Q_b) = 1 - 1 = 0$, which leaks the information that the two queries are equivalent.

In the case, where two queries are generated from different feature sets, the corresponding signatures may still have common elements if there are some common keywords in their feature sets. Let two queries, Q_a and Q_b, have η_a and η_b keywords in their feature sets, respectively and $\bar{\eta}$ be the number of common keywords in the two feature sets. Further, let $\bar{c}$ be as defined above, then the Jaccard distance between two different queries Q_a and Q_b can be estimated as:

$$J_d(Q_a, Q_b) = 1 - \frac{\frac{\bar{\eta}^2}{\eta_a \eta_b}\bar{c}\lambda}{2c\lambda - \left(\frac{\bar{\eta}^2}{\eta_a \eta_b}\bar{c}\lambda\right)}. \tag{5.7}$$

Equation (5.7) is almost the same as the case, where queries are equivalent as given in Equation (5.6), except for the factor $(\bar{\eta}^2/\eta_a\eta_b)$. This factor is the probability that a MinHash function gives the same output for two input sets with lengths η_a and η_b, where $\bar{\eta}$ elements are common in both sets. Note that, the MinHash functions are random permutations and the output is one of the elements in the input with uniform distribution. The probability that a MinHash function gives the same output for the two sets with η_a and η_b elements, where $\bar{\eta}$ of them are common, is calculated in the following way.

With probability $\bar{\eta}/\eta_a$, one of the common elements is chosen from the set of the query, Q_a. For the second set, Q_b, due to the MinHash property of the functions, the minimum value of the common elements will be the same as the first set, but different for non-common elements. Hence, the same element will be the output of the function with probability $\bar{\eta}/\eta_b$. Therefore, the probability that a MinHash function gives the same output for the two input sets is calculated as $\bar{\eta}^2/\eta_a\eta_b$. Note that if $\bar{\eta} = 0$, the distance is $J_d(Q_a, Q_b) = 1 - 0 = 1$, which hides any possible correlation as in the case of equivalent queries. Similarly, if $\bar{\eta} = \eta_a = \eta_b$, then equations (5.6) and (5.7) become the same.

The Jaccard distance between two queries, depends on the number of MinHash functions shared in their signatures; the lower the functions shared, the higher the distance. The expected value of the Jaccard distance can be estimated by calculating the probability that the number of common MinHash functions in the two query signatures is i, (i.e., $\bar{c} = i$). Let $P_{\bar{c}}(i)$ be the probability that $\bar{c} = i$, the expected Jaccard distance between any two queries Q_a and Q_b is estimated as:

$$Exp\left[J_d(Q_a, Q_b)\right] = \sum_{i=0}^{c} P_{\bar{c}}(i) J_{d_i}(Q_a, Q_b), \tag{5.8}$$

where $J_{d_i}(Q_a, Q_b)$ is the Jaccard distance for $\bar{c} = i$.

The probability $P_{\bar{c}}(i)$ is equal to the probability that i elements are chosen from c elements and $c - i$ are chosen from $\phi - c$ elements. First, consider the ordered case such that the first i elements are chosen from c, followed by $c - i$ elements are chosen from $\phi - c$ elements. This probability is calculated as:

$$\frac{P_i^c P_{c-i}^{\phi-c}}{P_c^{\phi}}$$

However, in our case the order is not important, so there are C_i^c different ways of choosing these elements. Therefore, the probability $P_{\bar{c}}(i)$ is calculated as:

$$\begin{aligned} P_{\bar{c}}(i) &= C_i^c \frac{P_i^c P_{c-i}^{\phi-c}}{P_c^\phi} \\ &= \frac{c!}{i!(c-i)!} \frac{c!}{(c-i)!} \frac{(\phi-c)!}{(\phi-2c+i)!} \frac{(\phi-c)!}{\phi!}. \end{aligned} \tag{5.9}$$

In Table 5.1, the values of $P_{\bar{c}}(i)$ are presented for $\phi = 15$ to 25, where c is 2. Note that $\sum_{i=0}^{c} P_{\bar{c}}(i) = 1$.

ϕ	$i = 0$	$i = 1$	$i = 2$	sum
15	0,7429	0,2476	0,0095	1
16	0,7583	0,2333	0,0083	1
17	0,7721	0,2206	0,0074	1
18	0,7843	0,2092	0,0065	1
19	0,7953	0,1988	0,0058	1
20	0,8053	0,1895	0,0053	1
21	0,8143	0,1810	0,0048	1
22	0,8225	0,1732	0,0043	1
23	0,83	0,1660	0,004	1
24	0,837	0,1594	0,0036	1
25	0,8433	0,1533	0,0033	1

Table 5.1: $P_{\bar{c}}(i)$ for different ϕ values

The distance between equivalent queries converges to 1 (i.e., ultimate obfuscation) as ϕ increases; however, this also increases the number of buckets in the secure index $\mathcal{I}$, which inevitably increases the storage requirements and the search time. Therefore, we set $\phi = 20$, which satisfies sufficient security while still keeping the size of $\mathcal{I}$ within practical limits.

Example 8. *Let ϕ be 20 and c be 2, the expected distance between two equivalent queries Q_a and Q_b that are generated using the same feature set, is calculated as follows.*

$P_{\bar{c}}(i)$ will be as:

$$\begin{aligned}
P_{\bar{c}}(0) &= \frac{2!}{0!2!}\frac{2!}{2!}\frac{18!}{16!}\frac{18!}{20!} = \frac{17\cdot 18}{19\cdot 20} = 0.8053,\\
P_{\bar{c}}(1) &= \frac{2!}{1!1!}\frac{2!}{1!}\frac{18!}{17!}\frac{18!}{20!} = \frac{2\cdot 2\cdot 18}{19\cdot 20} = 0.1895,\\
P_{\bar{c}}(2) &= \frac{2!}{2!0!}\frac{2!}{0!}\frac{18!}{18!}\frac{18!}{20!} = \frac{2}{19\cdot 20} = 0.0052.
\end{aligned}$$

Note that $\sum_{i=0}^{c} P_{\bar{c}}(i) = 0.8053 + 0.1895 + 0.0052 = 1$.

The Jaccard distance is calculated as:

$$\begin{aligned}
J_{d_0}(Q_a, Q_b) &= 1 - \frac{0}{4\lambda} = 1,\\
J_{d_1}(Q_a, Q_b) &= 1 - \frac{\lambda}{3\lambda} = \frac{2}{3},\\
J_{d_2}(Q_a, Q_b) &= 1 - \frac{2\lambda}{2\lambda} = 0.
\end{aligned}$$

Then the expected Jaccard distance between Q_a and Q_b is:

$$Exp\left[J_d(Q_a, Q_b)\right] = 0.8053\cdot 1 + 0.1895\cdot\frac{2}{3} + 0.0052\cdot 0 = 0.932,$$

which shows, even if two queries are generated from identical feature sets, they have a significantly large distance in-between that makes it difficult to distinguish them from the queries generated from different feature sets.

5.9 Security Analysis of the Two Server Method

In this section, we provide the formal definitions that show the proposed scheme satisfies the privacy requirements defined in Section 5.6.

Theorem 6. *The proposed method satisfies ϵ-probability distinguishability in accordance with Definition 20.*

Proof. An upper bound for the probability that, two queries, Q_a and Q_b, that have a common keyword w in the corresponding feature sets, have a common bucket identifier, can be found as:

$$\begin{aligned} prob(Q_a \cap Q_b \neq \emptyset) &= 1 - prob(Q_a \cap Q_b = \emptyset) \\ &\leq 1 - P_{\bar{c}}(0) \\ &\leq \epsilon, \end{aligned} \tag{5.10}$$

where ϵ is a security parameter. □

Theorem 7. *The proposed query randomization method satisfies δ-mean query obfuscation in accordance with Definition 21.*

Proof. Recall that, three queries Q, Q_b and Q_{1-b} are defined in Definition 21, where Q_b and Q are generated from the same set of features and $b \in_R \{0, 1\}$. Without loss of generality, let $b = 0$. For the case, where two queries are different, we assume the worst case, in which they do not share any common keyword (i.e., $\bar{\eta} = 0$). In this case $J_d(Q, Q_1)$ will be 1.

For the case, where two queries are equivalent, with probability $P_{\bar{c}}(0)$, Q and Q_0 will not share any common MinHash functions and $J_d(Q, Q_0)$ will be 1. With probability $1 - P_{\bar{c}}(0)$, which is small but non-negligible, the equivalent queries will share $\bar{c}$ common MinHash functions and the Jaccard distance between those two queries will be less than one. In this case, the distance between the two queries can be estimated using the expected value given in Equation (5.8) as:

$$Exp[J_d(Q, Q_b)] = \sum_{i=0}^{c} P_{\bar{c}}(i) J_{d_i}(Q, Q_b).$$

This expected distance converges to 1 as $\phi \to \infty$ and as the analysis given in Section 5.8.1 shows, $Exp\left[J_d(Q, Q_b)\right]$ gets very close to 1 for reasonable values of ϕ. Therefore, if the security parameters c and ϕ are set appropriately, then

$$Exp[|J_d(Q,Q_0) - J_d(Q,Q_1)|] = |(\approx 1) - 1| < \delta, \tag{5.11}$$

where δ is a security parameter.

$\square$

For the case, where the Jaccard distance between any two queries is not one, we leak the information that the corresponding feature sets of the two queries share at least one keyword. This is especially true since two queries with no common keyword in their feature sets, will have always Jaccard distance one. However, the method is still good since the server or the adversary cannot attack using background information on the statistics of the search terms used. The properties of the method that thwart background attacks, are clarified below.

We assume an adversary may have some background information on the data set, such that the search frequencies of the most frequently queried keywords are known. There are two main properties of the proposed method that avert attacks using background information. Firstly, most of the statistical information is obfuscated using the randomized query generation method, explained in Section 5.7.2. Secondly, the probability in Equation (5.10) depends on η, which is the number of keywords in the feature sets of the queries. As the distribution of the number of keywords in each query may significantly vary, the correlation of the actual search frequencies with the similarities between the collection of queries is hard to distinguish.

The correlation of the original search statistics and the statistics gathered from the queries is formalized in the following proposition.

Proposition 5. *Let $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_n\}$ be a collection of n queries and w be a keyword that occurs in p_w percent of the queries in $\mathcal{Q}$. Given $\mathcal{Q}$, w*

and p_w, in the proposed scheme, it is hard to obtain the set of queries in $\mathcal{Q}$ that contains the keyword w.

Proof. We assume that the adversary A, having observed a collection of queries, $\mathcal{Q}$, knows that a keyword w occurs in p_w percent of the set $\mathcal{Q}$. Let there be n queries in $\mathcal{Q}$.

The probability that a query $Q_i \in \mathcal{Q}$, where $w \in F_i$ contains a specific bucket identifier π_w, is $\frac{c}{\phi\eta_i}$ since only a single MinHash function can produce the π_w output and that function exists in Q_i with probability c/ϕ. There are $p_w|\mathcal{Q}|$ queries that contain the keyword w, hence the expected number of queries that contain a specific encrypted identifier π_w, in their signatures is,

$$\begin{aligned} \#_i(Q_i \supset \pi_w) &= \forall i,\ w \in F_i \sum_i \frac{c}{\phi\eta_i} \\ &= p_w|\mathcal{Q}| \cdot \frac{c}{\phi} \cdot Avg\left(\frac{1}{\eta_i}\right). \end{aligned} \tag{5.12}$$

Let p_w and $p_{w'}$ be the frequencies of the two most frequent keywords, w and w' respectively, in $\mathcal{Q}$, (i.e., $p_w > p_{w'}$). Depending on the values of η corresponding to the queries containing w and w', the expected number of queries containing the encrypted bucket identifier $\pi_{w'}$ may be greater than the number of queries containing π_w. As there are several different keywords with various occurrence frequencies in the queries, where η values of the queries are unknown, it is hard to correlate the occurrence frequencies of the keywords with the cardinality of the sets of queries that share a specific encrypted bucket identifier. □

Considering the fact that, with high probability, the Jaccard distance between two queries generated from the same feature set is 1 and distribution of the number of keywords in queries is unknown to the adversary, correlating the original keyword search frequency with $\mathcal{Q}$ is hard.

We illustrate the arguments about the hardness of correlating statistics with $\mathcal{Q}$ for the proposed method in the following example.

Example 9. *For an extreme case, let the adversary learn* 10^6 *queries (i.e.,* $|\mathcal{Q}| = 10^6$*). Further let,* $\phi = 20$, $c = 2$, *the frequency of the most frequent keyword* w *be* 10% *and let there be a set of other keywords* w'_i *that each occurs in* 8% *of the queries in* $\mathcal{Q}$*. We assume the number of keywords in a query,* η*, is chosen uniformly random in the range* $[2, 5]$*.*

As, η *is chosen uniformly randomly in the range* $[2, 5]$*, the mean (i.e., expected) value of* $(1/\eta)$ *is*

$$\begin{aligned} Avg\left(\frac{1}{\eta}\right) &= \frac{1}{4}\left(\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}\right) \\ &= 0.32. \end{aligned}$$

Hence, on average, the number of queries that share a common encrypted bucket identifier due to the keyword w *(i.e., the most frequent keyword) is:*

$$\begin{aligned} &= p_w|\mathcal{Q}| \cdot \frac{c}{\phi} \cdot Avg\left(\frac{1}{\eta_i}\right) \\ &= 0.1 \cdot 10^6 \cdot \frac{2}{20} \cdot 0.32 \\ &= 3200 \end{aligned}$$

On the other hand, for each i*, depending on the values of* η*, the expected number of queries that share a common encrypted bucket identifier due to a less frequent keyword* w'_i *will be distributed in the range:*

$$\begin{aligned} &= [0.08 \cdot 10^6 \cdot \frac{2}{20} \cdot 0.2, 0.08 \cdot 10^6 \cdot \frac{2}{20} \cdot 0.5] \\ &= [1600, 4000]. \end{aligned}$$

The number of keywords in each query is assumed to be uniformly random, hence there may be several bucket identifiers, $\pi_{w'_i}$, such that although the occurrence frequency of w'_i is less than the occurrence frequency of w, the number of queries that share a common bucket due to the keyword w'_i is greater than the number of queries that share a bucket identifier due to w.

This example shows that, it is very hard to correlate two equivalent queries with exactly the same set of search terms, due to the randomization phase we propose.

We also confirm this claim with an experiment using the statistics of the real data set [49]. The occurrence frequencies of the keywords in the queries, $\mathcal{Q}$, are chosen according to the term frequencies of the corresponding keywords in the data set. It is observed that on the average, among the sets of queries that share a bucket identifier, only in 20% of the experiments, the set with the largest cardinality corresponds to the keyword with greatest occurrence frequency. Moreover, the correlation between the cardinality of the sets and the term frequencies of keywords is further corrupted as the term frequencies decrease. This is due to the existence of several keywords with very similar occurrence frequencies.

Theorem 8. *The proposed method satisfies adaptive semantic security in accordance with Definition 11.*

Proof. Let the original view $v(H_n)$ and the trace $\gamma(H_n)$ be

$$v(H_n) = \{(id(C_1), \ldots, id(C_{|\mathcal{D}|})), C, \mathcal{I}, \mathcal{Q}\}$$
$$\gamma(H_n) = \{(id(C_1), \ldots, id(C_{|\mathcal{D}|})), (|C_1|, \ldots, |C_{|\mathcal{D}|}|), A_p(H_n), |\mathcal{I}|\}.$$

Further let $v^*(H_n) = \{(id^*(C_1), \ldots, id^*(C_{|\mathcal{D}|})), C^*, \mathcal{I}^*, \mathcal{Q}^*\}$ be the view simulated by a simulator S. The proposed method is adaptive semantically secure if $v(H_n)$ is indistinguishable from $v^*(H_n)$.

- The first component of the view $v(H_n)$ is the pseudo identifiers of the documents, $id(C_i)$, which are also available in the trace. Hence, S can trivially simulate document identifiers as $id^*(C_i) = id(C_i)$. Since for all possible values of i, $id^*(C_i) = id(C_i)$, they are indistinguishable.

- Each document is encrypted using a PCPA-secure encryption method. Note that, the output of a PCPA-secure encryption method [18] is by definition indistinguishable from a random number that has the same size as the ciphertext. To simulate ciphertexts C, S assigns l random numbers to C^* such that $C^* = \{C_1^*, \ldots, C_{|\mathcal{D}|}^*\}$, where $\forall i, |C_i^*| = |C_i|$ using the size information of each ciphertext, which is available in the trace. Considering that for all i, C_i and C_i^* are indistinguishable, C and C^* are also indistinguishable.

- Note that $\mathcal{I}$ is composed of encrypted bucket identifiers and corresponding encrypted bucket content vectors. Let $size_B$ and $size_V$ be the sizes of bucket identifier and bucket content, respectively. Simulator S generates $|\mathcal{I}|$ index elements, $\mathcal{I}^*[i] = (\pi_i^*, \mathcal{V}_i^*)$ such that π_i^* is a random number, where $|\pi_i^*| = size_B$ and $\mathcal{V}_i^*$ is a vector of size $size_V$ such that each element is a random number of size of the Paillier encryption modulus. Note that π_i^* and π_i are indistinguishable since π_i is the output of a cryptographic hash function (i.e., HMAC), where the output is indistinguishable from a random number. Similarly, $\mathcal{V}_i^*$ and $\mathcal{V}_i$ are indistinguishable since each element of $\mathcal{V}_i$ is a ciphertext of a Paillier encryption method, which provides semantic security. Hence, $\mathcal{I}$ is indistinguishable from $\mathcal{I}^*$.

- $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$ is a set of n queries, where each query Q_i is composed of $c\lambda$ encrypted bucket identifiers. Encrypted bucket identifiers

(π_i^*) can be simulated by the S as shown in the previous case. S can simulate the queries using the previously simulated bucket identifiers, π_i^*. For each Q_i^*, $c\lambda$ random simulated bucket identifiers are chosen from the simulated index $\mathcal{I}^*$. Note that real and simulated encrypted bucket identifiers are indistinguishable from each other. Hence, for all i, Q_i is indistinguishable from Q_i^* and following from this, $\mathcal{Q}$ is indistinguishable from $\mathcal{Q}^*$.

The simulated view v^* is indistinguishable from the genuine view v since components of v and v^* are indistinguishable. Henceforth, the proposed method satisfies adaptive semantic security. $\square$

5.10 Compressing Content Vector

The two server search method given in Section 5.7.3 requires the search server to send a vector of encrypted scores to the file server, where the size of the vector is in the order of the size of the data set (i.e., $|\mathcal{D}|$). The search server cannot know the scores of the documents, therefore an encrypted score of each document, including those with 0 score, should be sent. Although it is transparent to the users, creation and forwarding of this score vector necessitates a considerable communication and computation costs on both servers. In order to mitigate this cost, we propose combining several document scores in a single vector entry while still providing the correctness of the results and privacy of individual scores.

Note that each entry of an encrypted content vector $\mathcal{V}$ is encrypted one by one, using the Paillier encryption and the final encrypted scores vector, E_V, is the element-wise product of the encrypted content vectors corresponding to the query. Let μ be the Paillier modulus, then each message is an element of $\mathbb{Z}_{\mu}$. However, each score that is stored on the content vector is much smaller than μ, which brings forth an unnecessary increase in the size of the E_V. Instead of encrypting each single score separately, we propose encoding several score values on each $\log(\mu)$ bit message such that each score can still be summed up using the homomorphic addition property of the Paillier encryption method.

Let maximum possible score value be b_{max} bits. During the search process, the search server applies summation on the $c\lambda$ score vectors corresponding to the query of the user. Hence, in the final score vector, the score of a document can at most be $\log(c\lambda) + b_{max}$ bits after the summation of $c\lambda$ score values. In the proposed compressed score vector method, each score value is stored in a $\log(c\lambda)+b_{max}$ bit part of the $\log(\mu)$ bit message. With this method, each score vector element can keep up to $\log(\mu)/(\log(c\lambda) + b_{max})$ document score values and these plain score values are then encrypted with the Paillier encryption by the data owner as in the case without compression. Reserving $\log(c\lambda) + b_{max}$ bits for each document score eliminates the possibility of any overflow after the summation. Hence, it is guaranteed that the correct accumulated scores can be received by the file server after decryption.

The compression of the content vector is depicted in the following example.

Example 10. *The content vector is represented in base* 10 *instead of binary for visualization purposes. As a toy example, let there be only* 12 *documents in the data set, and each score is in the range* $[0 - 99]$ *(i.e.,* 2 *digits). Also let* $c\lambda \leq 10$*, hence 3 digits will be reserved for each score. Let* $\mathcal{V}_1$ *and* $\mathcal{V}_2$ *be two encrypted vectors for 12 document scores. The homomorphic addition of* *the two vectors is shown as follows.*

$$
\begin{aligned}
&\mathcal{V}_1 \cdot \mathcal{V}_2 = \\
&= \begin{pmatrix} E(015032035000) \\ E(000027052000) \\ E(000034000000) \end{pmatrix} \cdot \begin{pmatrix} E(000082000012) \\ E(000000042000) \\ E(000015000000) \end{pmatrix} \\
&= \begin{pmatrix} E(015114035012) \\ E(000027094000) \\ E(000049000000) \end{pmatrix}
\end{aligned}
$$

In this example, each encrypted vector element stores four scores instead of a single score. Although a single homomorphic addition is depicted here, in the actual model, the sum of $c\lambda$ encrypted vectors are calculated. The additional $\log(c\lambda)$ *bits (e.g., 1 digit in this case), eliminate the possibility of any overflow after the summation. Therefore, the accumulated score can be received without any corruption.*

Using the compressed score vector method, the size of the final accumulated score vector E_V is reduced by the order of $\log(\mu)/(\log(c\lambda) + b_{max})$, which decrease both the communication and storage costs at this order. In Section 5.11, possible values for b_{max}, μ and λ are presented, which shows that compressed score vector method provides significant increase in the efficiency of the keyword search scheme.

5.11 Experiments (Two Server)

In this section, we extensively analyze the proposed method in order to demonstrate and compare its efficiency and effectiveness. The entire system is implemented by Java language using a 64-bit Windows 7 operating system with Intel Xeon processor with 6 cores of 3.2 GHz. In our experiments we use both the RCV1 (Reuters Corpus Volume 1), which is a corpus of newswire stories that is made available by Reuters, Ltd. [49] and the Enron data set [51].

The success of the search method is analyzed using the precision and recall metrics (cf. Section3.6).

The matching items are ordered according to their relevancy with the given query (cf. Section 3.5) and only the top items that have the highest t scores are retrieved. In our experimental setting, we test the accuracy of the method with various number of *MinHash* functions (λ) using the precision and recall metrics. As Figures 5.6 and 5.7 demonstrate, both precision and recall increase as λ increases. Even though, higher λ provides better search accuracy, it also increases the storage requirements and the computation cost of the index generation. We therefore set λ as 125, which is the smallest value that provides sufficiently high recall rate.

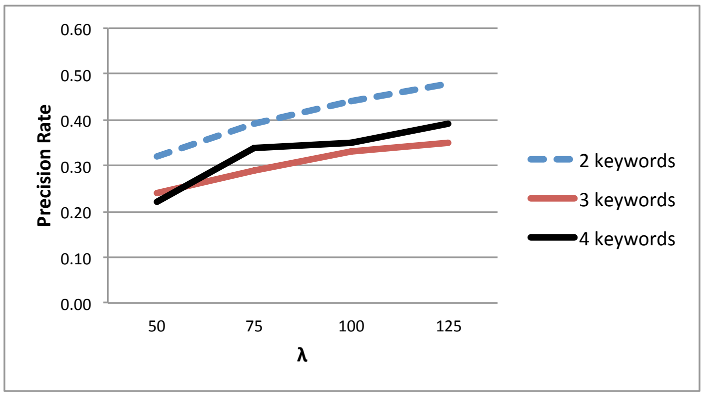


Figure 5.6: Precision rates, where η is 2, 3 and 4 with various λ values

The effect of the number of keywords in a query (i.e., η) on the precision rate depends on several parameters. As η increases, the number of

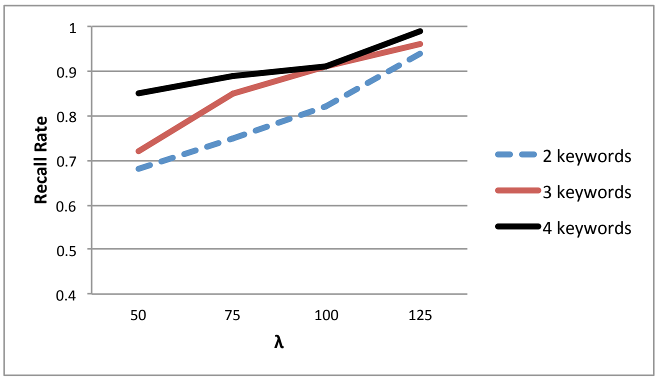


Figure 5.7: Recall rates, where η is 2, 3 and 4 with various λ values

common keywords with the matching documents also increases. The number of common buckets between a query and a document signature increases as the number of common keywords increases, which also increases their scores. Hence the increase in the number of keywords in a query has a positive effect on the precision rate. On the other hand, when the number of queried keywords increases, documents that contain a large subset of the queried keywords may also have a high score and therefore be retrieved by the user. However, those matches decrease precision unless they contain all the queried terms. Therefore, it is not possible to give a direct relation between η and precision rate. The correlation with the recall rate however, is straightforward. As the number of queried keywords increases, the number of elements in the set of documents that contain all the keywords (i.e., $|D(F)|$) significantly decreases, which increases the recall values.

It is important to note that, although the precision rates are not high, all the matches that are returned to the user contain at least a non-empty subset of the queried terms. The precision and recall metrics do not consider the parameters like term frequency and rarity that are used in the scoring of the proposed method. Hence, due to the tf-idf based scoring, some documents that have partial match with the query may get a higher rank compared to a document with a full match. Although the documents with partial match is counted as a false match in calculating the precision, we claim that documents that have partial match with the query may also be important for the user, due to the high relevancy score (i.e., high tf-idf values of the matching keywords). Moreover, the precision rate also depends on the threshold t. For the case, where only the top 10 matches are retrieved, then the average precision rate is $98,5\%$. Therefore, we also tested the precision and recall rates of the proposed method for the case only top t percent of the documents with non-zero scores are retrieved by the user. The results, given in Figure 5.8, demonstrate that as more documents are retrieved, the precision decreases as some of the retrieved items do not contain all the queried keywords. Similarly, as less number of documents are retrieved, the precision increases but recall drops since some of the matching documents are not retrieved. By setting the t variable appropriately, the user may either retrieve only a small number of documents with high precision or all the related documents where some only contain a non-empty subset of the queried features. Therefore, the accuracy of our method is suitable for most of the practical requirements.

Utilizing the compression method explained in Section 5.10, several individual scores can be stored in a single homomorphic encrypted value. In our experimental setting, each score is represented with 10 bits (i.e., $b_{max} = 10$), we set $\lambda = 125$ and $c = 2$ hence, after adding $c\lambda$ scores, the final score of each document is represented with $\log_2(250) + 10 = 18$ bits. In the Paillier encryption, we use 1024 bit messages, which enables keeping $1024/18 = 56$ scores in a single encryption instead of a single score. With this novel method, the storage requirements of the search server, the communication cost between the search server and the file server, the number of Paillier encryption per-

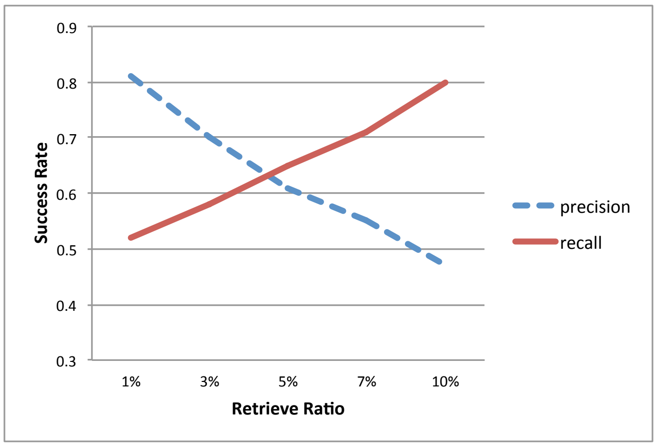


Figure 5.8: Precision and recall rates, where $t\%$ of documents with non-zero scores are retrieved

formed by the data owner and the number of decryption performed by the file server are all reduced 56 times. This improvement can further be extended by using smaller b_{max} or λ.

The most time consuming operation of the proposed method is the generation of the secure searchable index, where the bottleneck is the homomorphic encryption operations. Each element of all content vectors are encrypted using the Paillier encryption, where the number of content vectors depends on both the number of different features in the data set and number of *MinHash* functions used. The index generation timings for various data set sizes are presented in Figure 5.9. Although the index generation times are in the order of several minutes, since this operation is done only once in an offline stage, the efficiency of the proposed method is still high.

The most crucial operation of the proposed method is the search over the secure index. There can be several users in the system and the query rate can therefore be very high, but the users should still be able to rapidly retrieve the responses. The search operation explained in Section 5.7.3 has two time-

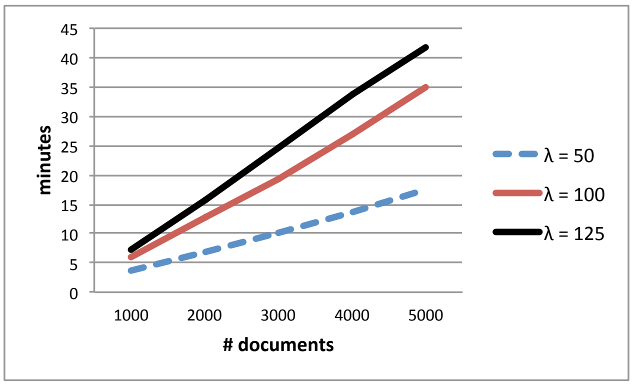


Figure 5.9: Timing results for index generation as data set size changes for various λ values

consuming operations. Firstly, the search server multiplies the encrypted scores of the queried bucket vectors for applying homomorphic summation of the scores and sends this single accumulated encrypted score vector E_V to the file server. Secondly, after receiving E_V, the file server decrypts each vector element and sorts the data identifiers. The search time for various data set sizes are presented in Figure 5.10. The search operation works in the order of a few seconds with parallel processing in the six core server used in our experiments. The timings can further be decreased utilizing parallel processing with more powerful servers. Note that, each decryption and homomorphic addition operation are independent and therefore can trivially be parallelized.

We do not provide any comparison with the work that are for vendor system such as private information retrieval methods or solutions for store and forward system such as the PEKS based methods. We also do not provide any comparison with single keyword search methods since all those works are not suitable for the problem we consider. Some of the existing work in the literature [21, 52] including the preliminary version of this work [38] that

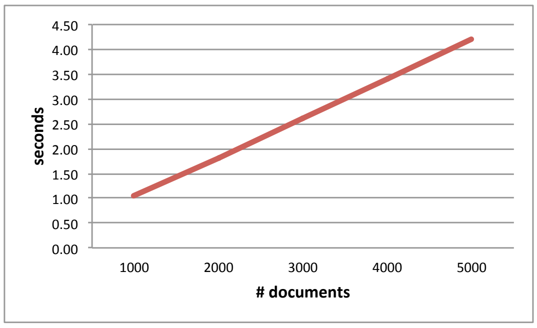


Figure 5.10: Timing results for the search operation as data set size changes for $\lambda = 125$

use single server, can provide much faster search operation. However, none of those methods provides query indistinguishability or obfuscation, which has crucial importance for hiding the content of a query. The work of Cao et al. [23] provides query indistinguishability but since the queries can be correlated with the scores of matching documents, the similarity between queries can still be revealed (i.e., reveals search pattern). Another recent work by Orencik and Savas [40] also provides a secure search method that hides the search pattern. The drawback of the method in [40] is the ranking approach used. The method supports only a fixed number of rank levels, where all the documents that match with the same level have the same rank such that, ranks of two documents will be the same even if one contains all the queried terms infrequently and the other one contains all the queried terms very frequently except only one infrequent term. Moreover, in [40], some fake document index entries are introduced in the searchable index. The fake documents may match with the query, which is used for hiding the search pattern. However, the user needs to sanitize the final result from the fake matches, which imposes an extra burden on the user. There is a strict tradeoff between privacy and efficiency, and we managed to increase the privacy of the method by an acceptable level of increase in the computational cost.

5.12 Chapter Summary

In this chapter, we addressed the privacy-preserving multi-keyword search over encrypted cloud data for the database outsourcing model (i.e., public storage system) using locality hash functions. We first utilize a single server scheme that use an efficient secure index method using *MinHash* functions. We also incorporate ranking capability to the proposed scheme utilizing well known tf-idf based relevancy scoring. This approach ensures that only the most relevant items are retrieved by the user, preventing unnecessary communication and computation burden on the user. However, this single approach leaks the search pattern for efficiency reasons. We then utilize the two server search method for hiding the correlation between the queries and corresponding matching identifiers. Utilizing the two server search method, the δ-mean query obfuscation and ϵ-probability distinguishability properties, the proposed method satisfies the requirement of hiding search pattern, which most of the work in the literature fail to provide. We also propose a novel method that enables storing more than 50 scores at each element of the encrypted content vector. This approach reduces both the number of decryption operations on the file server side and the communication between the search server and the file server more than 50 times compared to the standard approach, where each score is encrypted one by one.

We provide formal security definitions and prove that our proposed work satisfies adaptive semantic security, δ-mean query obfuscation and ϵ-probability distinguishability. We implement the entire system and demonstrate the effectiveness and efficiency of our solution through extensive experiments using the publicly available Reuters dataset [49].

Chapter 6

CONCLUSIONS and FUTURE WORK

Cloud computing is today's one of the most exciting computing paradigms in information technology. With the tremendous computation and storage capacity advantages, cloud computing creates a fundamental shift in delivering computing services. Enterprises are motivated to outsource the burden of computation to clouds such that they can avoid purchasing and managing software and hardware. However, security and privacy are perceived as primary obstacles to its wide adoption. Although designing security into the cloud benefits users and cloud server providers, it inevitably increases overhead for both.

Privacy preserving search over encrypted cloud data has been extensively studied in recent years. Although several methods are proposed in the literature, non of them can provide an optimal solution that is both fully privacy preserving and also very efficient. In this thesis, we aim to provide a secure keyword search method that provides the privacy of the sensitive data of the users in an efficient manner. We proposed three different secure search methods. The first method is a hash based search scheme that provides very efficient and accurate search over encrypted data. The proposed privacy-preserving search scheme essentially implements an efficient method to satisfy query unlinkability based on query and response randomization and cryptographic techniques. Query randomization cost is negligible for the data controller and even less for the user. Response randomization, on the other hand, results in a communication overhead when the response to a query is returned to the user since some fake matches are included in the response. However, we show that the overhead can be minimized with the optimal choice of parameters. The only flow of this method is the scoring method utilized. This work supports only a fixed number of rank levels, where all the documents that match with the same level have the same rank such that, ranks of two documents will be the same even if one contains all the queried terms infrequently and the other one contains all the queried terms very frequently except only one infrequent term.

The second method, single server *MinHash* based scheme, provides very efficient search and also incorporate ranking capability to the proposed scheme utilizing well known tf-idf based relevancy scoring. As several work in the literature do, we also allow to leak search pattern for efficiency concerns in the single server method. Thirdly we utilize a two server scheme that uses the same secure searchable index structure used in the single server method. With the addition of a second server and a novel query obfuscation method, we managed to enhance the privacy of the method such that the search pattern is also hidden.

6.1 Future Directions

The main issue we considered in this thesis is secure keyword search over encrypted cloud data. We aim to extend the research on secure data-mining over encrypted cloud data. The two of the techniques we want to cover is secure k-nearest neighbour and range queries. Another research interest is the BigData concept. The BigData has some challenges that is not yet considered for encrypted data. The challenges include capture, curation, storage, search, sharing, transfer, analysis and visualization. One of the most important cases in the BigData is that the data is changing very frequently, which makes keeping a static searchable index infeasible. Hence, data-mining on encrypted BigData is still an open problem that we want to consider.

Bibliography

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 50–55, December 2008.

[2] "Cloud computing innovation key initiative overview." http://my.gartner.com, May 2012.

[3] "Google cloud platform." http://cloud.google.com/, Jan. 2012.

[4] "Amazon elastic compute cloud (amazon ec2)." http://aws.amazon.com/ec2/, Jan. 2012.

[5] "Windows live mesh." http://windows.microsoft.com/en-US/windows-live/essentials-other-programs?T1=t4, Jan. 2012.

[6] Y. Zhao, X. Chen, H. Ma, Q. Tang, and H. Zhu, "A new trapdoor-indistinguishable public key encryption with keyword search," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 3, no. 1/2, pp. 72–81, 2012.

[7] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *J. ACM*, vol. 45, pp. 965–981, November 1998.

[8] C. Cachin, S. Micali, and M. Stadler, "Computationally private information retrieval with polylogarithmic communication," in *Lecture Notes in Computer Science*, pp. 402–414, Springer, 1999.

[9] B. Chor, N. Gilboa, and M. Naor, "Private information retrieval by keywords." Technical Report TR-CS0917, 1997.

[10] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. Skeith, "Public key encryption that allows pir queries," in *Advances in Cryptology - CRYPTO 2007*, vol. 4622 of *Lecture Notes in Computer Science*, pp. 50–67, Springer Berlin / Heidelberg, 2007.

[11] J. Groth, A. Kiayias, and H. Lipmaa, "Multi-query computationally-private information retrieval with constant communication rate.," in *PKC*, pp. 107–123, 2010.

[12] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *EUROCRYPT*, pp. 506–522, 2004.

[13] C. Gu, Y. Zhu, and H. Pan, "Efficient public key encryption with keyword search schemes from pairings," in *Information Security and Cryptology*, vol. 4990 of *Lecture Notes in Computer Science*, pp. 372–383, Springer Berlin Heidelberg, 2008.

[14] D. Park, K. Kim, and P. Lee, "Public key encryption with conjunctive field keyword search," in *Information Security Applications* (C. Lim and M. Yung, eds.), vol. 3325 of *Lecture Notes in Computer Science*, pp. 73–86, Springer Berlin Heidelberg, 2005.

[15] B. Zhang and F. Zhang, "An efficient public key encryption with conjunctive-subset keywords search," *J. Netw. Comput. Appl.*, vol. 34, pp. 262–267, Jan. 2011.

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

[17] W. Ogata and K. Kurosawa, "Oblivious keyword search," in *Journal of Complexity, Vol.20*, pp. 356–371, 2004.

[18] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pp. 79–88, ACM, 2006.

[19] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data.," in *ICDCS'10*, pp. 253–262, 2010.

[20] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering*, ICDE '12, (Washington, DC, USA), pp. 1156–1167, IEEE Computer Society, 2012.

[21] M. Raykova, B. Vo, S. M. Bellovin, and T. Malkin, "Secure anonymous database search," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, pp. 115–126, ACM, 2009.

[22] P. Wang, H. Wang, and J. Pieprzyk, "An efficient scheme of common secure indices for conjunctive keyword-based retrieval on encrypted data," in *Information Security Applications*, Lecture Notes in Computer Science, pp. 145–159, Springer, 2009.

[23] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in *IEEE INFOCOM*, 2011.

[24] Z. Chen, C. Wu, D. Wang, and S. Li, "Conjunctive keywords searchable encryption with efficient pairing, constant ciphertext and short trapdoor," in *PAISI*, pp. 176–189, 2012.

[25] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rou, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Advances in Cryptology, CRYPTO 2013*, vol. 8042 of *Lecture Notes in Computer Science*, pp. 353–373, Springer Berlin Heidelberg, 2013.

[26] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *J. ACM*, vol. 43, pp. 431–473, May 1996.

[27] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path oram: An extremely simple oblivious ram protocol," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*, CCS '13, (New York, NY, USA), pp. 299–310, ACM, 2013.

[28] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, pp. 120–126, Feb. 1978.

[29] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *ADVANCES IN CRYPTOLOGY - EUROCRYPT 1999*, pp. 223–238, Springer-Verlag, 1999.

[30] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of paillier's probabilistic public-key system," in *Public Key Cryptography*, pp. 119–136, 2001.

[31] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, STOC '09, pp. 169–178, 2009.

[32] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers." Cryptology ePrint Archive, Report 2009/616, 2009.

[33] J.-S. Coron, D. Naccache, and M. Tibouchi, "Public key compression and modulus switching for fully homomorphic encryption over the integers," in *Advances in Cryptology - EUROCRYPT 2012*, vol. 7237 of *Lecture Notes in Computer Science*, pp. 446–464, 2012.

[34] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," pp. 1–15, Springer-Verlag, 1996.

[35] A. Rajaraman and D. Ullman, Jeffrey, *Mining of massive datasets.* Cambridge University Press, 2011.

[36] R. W. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[37] H. S. Christopher D. Manning, Prabhakar Raghavan, *Introduction to Information Retrieval.* Cambridge University Press, 2008.

[38] C. Orencik, M. Kantarcioglu, and E. Savas, "A practical and secure multi-keyword search method over encrypted cloud data," in *CLOUD 2013*, pp. 390–398, IEEE, 2013.

[39] C. Orencik and E. Savas, "Efficient and secure ranked multi-keyword search on encrypted cloud data," in *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, pp. 186–195, ACM, 2012.

[40] C. Orencik and E. Savas, “An efficient privacy-preserving multi-keyword search over encrypted cloud data with ranking,” *Distributed and Parallel Databases*, vol. 32, no. 1, pp. 119–160, 2014.

[41] “Oxford dictionaries, the oec: Facts about the language.” http://oxforddictionaries.com/page/oecfactslanguage/the-oec-facts-about-the-language, June 2011.

[42] L. E. Dickson, *Linear Groups with an Exposition of Galois Field Theory.* New York: Dover Publications, 2003.

[43] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra, “Executing sql over encrypted data in the database-service-provider model,” in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, SIGMOD ’02, pp. 216–227, ACM, 2002.

[44] B. Hore, S. Mehrotra, and G. Tsudik, “A privacy-preserving index for range queries,” in *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB ’04, pp. 720–731, VLDB Endowment, 2004.

[45] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu, “Secure multi-dimensional range queries over outsourced data,” *The VLDB Journal*, vol. 21, pp. 333–358, June 2012.

[46] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold, “Keyword search and oblivious pseudorandom functions,” in *Theory of Cryptography Conference - TCC 2005*, pp. 303–324, 2005.

[47] B. Pinkas and T. Reinman, “Oblivious ram revisited,” in *Proceedings of the 30th annual conference on Advances in cryptology*, CRYPTO’10, (Berlin, Heidelberg), pp. 502–519, Springer-Verlag, 2010.

[48] "Google trends." http://www.google.com/trends/, June 2012.

[49] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "Rcv1: A new benchmark collection for text categorization research," *J. Mach. Learn. Res.*, vol. 5, pp. 361–397, Dec. 2004.

[50] J. Zobel and A. Moffat, "Exploring the similarity space," *SIGIR FORUM*, vol. 32, pp. 18–34, 1998.

[51] "Enron email dataset." http://www.cs.cmu.edu/enron, Jan. 2012.

[52] Q. Liu, C. C. Tan, J. Wu, and G. Wang, "Cooperative private searching in clouds," *J. Parallel Distrib. Comput.*, vol. 72, no. 8, pp. 1019–1031, 2012.