DISSERTATION

TOWARDS A SECURE AND EFFICIENT SEARCH OVER ENCRYPTED CLOUD DATA

Submitted by

Mikhail Strizhov

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2016

ABSTRACT

TOWARDS A SECURE AND EFFICIENT SEARCH OVER ENCRYPTED CLOUD DATA

Cloud computing enables new types of services where the computational and network resources are available online through the Internet. One of the most popular services of cloud computing is data outsourcing. For reasons of cost and convenience, public as well as private organizations can now outsource their large amounts of data to the cloud and enjoy the benefits of remote storage and management. At the same time, confidentiality of remotely stored data on untrusted cloud server is a big concern. In order to reduce these concerns, sensitive data, such as, personal health records, emails, income tax and financial reports, are usually outsourced in encrypted form using well-known cryptographic techniques. Although encrypted data storage protects remote data from unauthorized access, it complicates some basic, yet essential data utilization services such as plaintext keyword search. A simple solution of downloading the data, decrypting and searching locally is clearly inefficient since storing data in the cloud is meaningless unless it can be easily searched and utilized. Thus, cloud services should enable efficient search on encrypted data to provide the benefits of a first-class cloud computing environment.

This dissertation is concerned with developing novel searchable encryption techniques that allow the cloud server to perform multi-keyword ranked search as well as substring search incorporating position information. We present results that we have accomplished in this area, including a comprehensive evaluation of existing solutions and searchable encryption schemes for ranked search and substring position search.

ACKNOWLEDGEMENTS

I would like to convey my gratitude to my adviser, Dr. Indrajit Ray, for providing me invaluable guidance in my research as well as publishing. I am especially thankful to him for providing me the freedom to choose my own research direction.

I would like to thank my commettee members Dr. Indrakshi Ray, Dr. Ross M. McConnell, Dr. James M. Bieman and Dr. Stephen C. Hayne for providing valuable advice throughout different stages of my research. Together, they provided helpful comments that improved this work.

I would like to thank Wayne Trzyna, Kim Judith, Sharon Van Gorder for providing me a financial support and help to pursue my studies. I would also like to thank all the members of DBsec group at Colorado State University.

I would like to thank my parents Vasiliy and Tatiana for their continued love, support, and encouragement over all these years, specifically for demonstrating to me the most important lesson, with hard work there are no limits. To Sanja, thank you for sharing this journey with me. Thank you for all the encouragement and understanding for the hours and late nights it took to write this dissertation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Cloud computing enables new types of services where the computational and network resources are available online through the Internet. One of the most popular services of cloud computing is data outsourcing. For reasons of cost and convenience, public as well as private organizations can now outsource their large amounts of data to the cloud and enjoy the benefits of remote storage. At the same time, confidentiality of remotely stored data on untrusted cloud server is a big concern. In order to reduce these concerns, sensitive data, such as, personal health records, emails, income tax and financial reports, etc. are usually outsourced in encrypted form using well-known cryptographic techniques. Although encrypted data storage protects remote data from unauthorized access, it complicates some basic, yet essential data utilization services such as plaintext keyword search. A simple solution of downloading the data, decrypting and searching locally is clearly inefficient since storing data in the cloud is meaningless unless it can be easily searched and utilized. Considering the potentially large number of on-demand data users and huge amount of outsourced data documents in the cloud, this problem is particularly challenging as it is extremely difficult to meet at the same time the requirements of performance, system usability and scalability.

In order to enable search over encrypted data, many Searchable Encryption (SE) schemes have been proposed in recent years [1–18]. Generally, SE solutions involve building an *searchable index* such that its content is hidden from the remote cloud server, yet allowing document search. The index is a data structure that keeps track of a stored document collection while supporting efficient keyword search, i.e., given a keyword, the index returns a pointer to the documents that contain the keyword. These solutions differ mostly in terms of whether they allow single keyword search or multi-keyword search and types of techniques they use to build the search query. A few of them, most notably [8–11], allow the notion of similarity search. The similarity search problem consists of a collection of data items that are characterized by some features, a query that specifies a value for a particular feature, and a similarity metric to measure the relevance between the query and data items. However, these techniques either do not allow searching on multiple keywords and ranking retrieved documents in terms of similarity scores, or are very computationally intensive. Moreover, none of similarity search schemes are protected against an adaptive adversary[5], that takes into consideration the history of cloud user's queries and set of matching documents retrieved back to the cloud user.

SE solutions differ in the level of efficiency and security guarantees they offer; however, most of them support only *exact* keyword search. As a result, there is no tolerance of format inconsistencies which are part of typical cloud user behavior; and they happen frequently. It is quite common that the search queries do not exactly match the pre-set keywords due to lack of exact knowledge about the data. SE solutions (for instance, Curtmola[5]) can be adopted to allow the substring search over encrypted data. To do this, the solution requires generating all possible substrings of each keyword extracted from the document collection and storing these substrings as keywords in the *searchable index.* However, this approach induces a very large storage requirement and thus makes solution impractical in the real world cloud environment. The significant drawback of existing SE schemes underlines an important need for new techniques that support search flexibility over encrypted documents.

Table 1.1. Comparison of Security Properties of Most Popular Cloud Storage Platforms.

Security Property	Google Drive	Microsoft OneDrive	Apple iCloud	Dropbox
Minimum Password Requirement	8 characters	8 characters	8 characters	6 characters
Two-step Verification	Yes	Yes	Yes	Yes
Transport Security	2048-bit SSL	2048-bit SSL	2048-bit SSL	2048-bit SSL
Data Encryption	256-bit AES	256-bit AES	128-bit AES	256-bit AES
Client-side Encryption	No	No	No	No
Searchable Encryption	No	No	No	No

1.1. Security of Existing Cloud Storage Platforms

Traditional cloud storage technology allows multiple people to take advantage of a set of networked servers at a data center. In the past, if the user wanted to store the data at the data center, he/she would have to spend a lot of money to rent a server from the data center - meaning it wasn't feasible for most people. The cloud storage changed this by allowing a number of users to securely share a series of servers in the data center. Also, it allows a user to access files from any compatible device, and can decrease the risks of hardware failure. However, it is important to note the cloud storage adds risks. When the data is stored in the cloud, the user does not have direct control over the data (or application). If an attacker manages to find the way to hack in the cloud, all the data could be compromised. Today, many large organizations are particularly vulnerable, as they store large amounts of sensitive data using cloud based technology. If an attacker manages to find a security hole, it can get the vast amounts of sensitive data, such as social security numbers, medical records and credit card information.

In Table 1.1 we compare the most popular cloud storage platforms using several security metrics. Our cloud storage platforms include Google Drive[19], Microsoft OneDrive[20], Apple iCloud[21] and Dropbox[22]. While Drive, OneDrive, iCloud are connected to the their own application ecosystem, Dropbox is a lightweight, simple alternative for file storage. We use a set of security requirements and appropriate measures to characterize each service so it can be considered sufficiently secure for the usage.

Our comparison shows that selected cloud storage platforms provide the following security properties: a requirement to have a password that is at least 8 characters long for new registration (except 6 characters in Dropbox), a support of two-step verification of the password, a support of data transfer via SSL using 2048-bit keys and an option to encrypt the stored data using AES encryption. However, our analysis shows that none of the platforms support the client-side encryption (where the user creates its own key, encrypts the data locally and stores encrypted files in the cloud), thus making user's information insecure because it remains easily accessible to the unauthorized person (i.e., attacker) in the cloud or the cloud provider itself. Only Apple iCloud provide a key generation service that guarantees that the encryption keys are created locally (at the user's device) and Apple can't access those keys. However, recent study[23] shows that Apple holds the master key and it can potentially decrypt and access all data stored on iCloud servers. Finally, our analysis shows that none of the existing cloud storage platforms support the search over encrypted data.

1.2. Applications of Searchable Encryption

Searchable encryption can be used to support a large number of diverse applications. For instance, in human resource management, one may want to look for a series of keywords that assess the performance of an employee. In hospital record management, a doctor may want to retrieve all records that match a given patient disease. At an educational institution an instructor may want to search for student information based on keywords related to the course performance. All of these applications share the common need of querying for keywords that are not necessarily pre-known.

An important application of searchable encryption is in the area of searching a substring within the large textual databases. For instance, a researcher may want to search a genome sequence (substring) against a large genome sequence database. Such search can be used in the analysis of genetic diseases, genetic fingerprinting or genetic genealogy, and requires returning as result not only the matching genome but also the position of the sequence within the genome. At the tax record service, a tax accountant may issue a search of "mcd", which describes multiple keywords such as "mcdaniel", "mcdavid", "mcdonald", "mcdunn", and she wants to find a set of documents that match the substring as well as first occurrence of the substring in each document. All the applications presented share common needs: confidentiality of data, query privacy, and query result privacy. Thus, they are perfect for the application of searchable encryption.

1.3. Overview of Results

This dissertation is concerned with developing novel searchable encryption techniques that include (1) multi-keyword similarity searchable encryption scheme and (2) substring position searchable encryption scheme. Both solutions are designed in such way that they allow a client to execute a search over encrypted documents outsourced to the cloud server.

Our first result is a novel secure and efficient multi-keyword similarity searchable encryption scheme that returns matching documents in a ranked order. More specifically, we develop scheme that allows multi-keyword query execution over an encrypted document corpus and it retrieves relevant documents ranked based on a similarity score. We present a construction that achieves an optimal search time. Unlike all previous schemes that are tied to the linear search complexity, our search is sublinear to the total number of documents that contain the queried set of keywords. We show that this type of searchable encryption scheme can be extremely efficient. We show that our construction is secure against an adaptive adversary[24, 5].

Our second result is a substring position searchable encryption scheme that allows substring queries over encrypted documents in the cloud. Cloud users can query remote untrusted server for a set of encrypted documents that contain a substring of characters. We present a construction that is very efficient and does not require large ciphertext space. Similarly to previous scheme, we define the security model and prove that our scheme is secure against an adaptive adversary.

Our last result is a natural extension of both techniques, where an arbitrary group of cloud users can submit their queries to search the encrypted document collection in the cloud. Specifically, our result is a group multi-keyword similarity searchable encryption scheme and a multi-user substring position searchable symmetric encryption scheme. Our extension solves the problem of managing access privileges within the cloud environment with multiple cloud users.

1.4. Dissertation Organization

The rest of this dissertation document is organized as follows. In Chapter 2 we give a brief overview of cryptography. In Chapter 3 we give an overview of the searchable encryption. We also present a systematic literature review and comparison on searchable encryption techniques. In Chapter 4 is devoted for a multi-keyword similarity searchable encryption solution. In Chapter 5 we present a group multi-keyword similarity searchable encryption scheme. Chapter 6 presents a substring position searchable symmetric encryption scheme. In Chapter 7 is devoted for a multi-user substring position searchable symmetric encryption. We conclude in Chapter 8 by discussing future directions based on the results presented.

CHAPTER 2

Background

2.1. Probability Theory

Probability theory plays a central role in cryptography. In fact, probability theory is essential to start the discussion of information or lack of information. In this chapter we represent the necessary probabilistic notations that are used throughout this dissertation.

We begin with the definition of negligible functions. Cryptography does not require that the adversary will always fail, but rather the adversary will have successful outcome with some small non-zero probability. We call this non-zero probability negligible. We define the negligible functions as follows.

Definition 1. ***(Negligible Functions[25]).*** *We call a function $\mu : \mathbb{N} \rightarrow \mathbb{R}$ negligible if for every positive polynomial $p(\cdot)$ there exist an N such that for all $n > N$,*

$$\mu(n) < \tfrac{1}{p(n)} \tag{1}$$

We now focus on probability distribution. We define the probability in terms of *sample space* S, which elements are called *elementary events*. An event A is a subset of the sample space S.

Definition 2. ***(Probability Distribution[26]).*** *A probability distribution $Pr[]$ on a sample space S is a mapping from events of S to real number such that the following probability axioms are satisfied:*

- *$Pr[A] \geq 0$ for any event A.*
- *$Pr[S] = 1$.*

- *$Pr[A \cup B] = Pr[A] + Pr[B]$ for any two events A and B. More generally, for any sequence of events A_1, A_2,... that are pairwise mutually exclusive:*

$$Pr\left[\bigcup_i A_i\right] = \sum_i Pr[A_i]. \tag{2}$$

$Pr[A]$ is defined as probability of the event A.

A random variable is a function $X : S \to \mathbb{R}$, where S is a sample space. We now introduce the notion of computation indistinguishability. Informally, two probability distributions are computationally indistinguishable if no efficient algorithm can tell them apart (*distinguish* them). We formalize it the definition in following way. Let D be some efficient algorithm, or distinguisher. D is provided with a sample from the first distribution and second distribution. We say that the distributions are computationally indistinguishable if every such D outputs 1 with almost the same probability upon receiving a sample from the first or second distribution. We define the computational indistinguishability using the notion of probability ensembles.

Definition 3. ***(Probability Ensemble[27]).*** *Let I be a countable index set. A probability ensemble indexed by I is a sequence of random variables indexed by I.*

Usually, the set I will either be $\mathbb{N}$ or an efficiently computable subset of $\{0,1\}^\star$. We will refer to an ensemble $X = \{X_n\}_{n \in \mathbb{N}}$, where X_n ranges over strings of length *poly*(n). This means that there is a single polynomial $p(\cdot)$ such that X_n ranges over string of length $p(n)$, for every n.

Definition 4. ***(Computational Indistinguishability[27]).*** *Two probability ensembles $X = \{X_n\}_{n \in \mathbb{N}}$ and $Y = \{Y_n\}_{n \in \mathbb{N}}$ are computationally indistinguishable, denoted $X \equiv cY$,* *if for every probabilistic polynomial-time distinguisher D, every positive polynomial $p(\cdot)$ and all sufficiently large n's*

$$Pr[D(X_n, 1^n) = 1] - Pr[D(Y_n, 1^n) = 1] < \frac{1}{p(n)} \tag{3}$$

Given the definition of computational indistinguishability, it is easy to define the pseudorandomness:

DEFINITION 5. ***(Pseudorandom Ensembles[27]).*** *An ensemble $X = \{X_n\}_{n\in\mathbb{N}}$ is called pseudorandom if there exist a polynomial $l(n)$ such that X is computationally indistinguishable from the uniform ensemble $U = \{U_{l(n)}\}_{n\in\mathbb{N}}$.*

2.2. CRYPTOGRAPHY

We formulate the notion of cryptography in following way:

DEFINITION 6. ***(Cryptography [28]).*** *Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication. The fundamental goals of cryptography are to adequately address the following four areas in both theory and practice:*

- Confidentiality*: is a service used to keep the content of information from all but those authorized to have it.*
- Data integrity*: is a service which addresses the unauthorized alteration of data.*
- Authentication*: is a service related to identification.*
- Non-repudiation*: is a service which prevents an entity from denying previous commitments or actions.*

Searchable encryption solutions rely heavily on several cryptography primitives. This includes pseudorandom generators, pseudorandom functions, pseudorandom permutations, symmetric and asymmetric encryption, and cryptographic hash functions. We refer the reader [25, 24, 28, 27] for more detailed discussion.

2.2.1. Pseudorandom Primitives. We start with discussion on pseudorandom generator. Speaking informally, a pseudorandom generator (PRG) is an efficient deterministic algorithm G that stretches a short random seed into a long pseudorandom string.

Definition 7. ***(Pseudorandom Generators[29]).*** *A pseudorandom generator is a deterministic polynomial-time algorithm G satisfying the following two conditions:*

- Expansion*: There exists a function $l : \mathbb{N} \to \mathbb{N}$ such that $l(n) > n$ for all $n \in \mathbb{N}$, and*
- Pseudorandomness*: The ensemble $\{G(U_n)\}_{n\in\mathbb{N}}$ is pseudorandom.*

Intuitively, a pseudorandom function (PRF) is one that cannot be distinguished from a random one. However, this notion is non-trivial since it is not possible to hand a distinguisher a function description and ask it to decide whether or not it is random. Therefore the distinguisher is provided with *oracle access* to a function that is either random or one that we have constructed. An efficient function f is pseudorandom if no distinguisher with an oracle access can tell whether its oracle computes a truly random function or the function f. $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^l$ is a keyed pseudorandom function, where k,n, $l > 1$. Formally, a pseudorandom function is defined as follows:

Definition 8. ***(Pseudorandom Function (PRF)[24]).*** *A keyed function $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^l$ is pseudorandom if for any probabilistic polynomial-time distinguisher D, given oracle access to $F_k = F(k, \cdot)$, there exists a negligible function, $negl(n)$ such that*

$$|Pr[D^{F_k(\cdot)}(1^n) = 1] - Pr[D^{f(\cdot)}(1^n) = 1]| \leq negl(n), \tag{4}$$

where $K \overset{R}{\leftarrow} \{0,1\}^k$ is chosen uniformly at random and f is chosen uniformly at random from all functions that map $\{0,1\}^n$ to $\{0,1\}^l$.

More specifically, notation $D^{f(\cdot)(\cdot)}$ means that the distinguisher D uses f as an oracle and D able to query f a polynomial number of times.

If $l = n$ then we get the pseudorandom permutation (PRP), as follows:

Definition 9. ***(Pseudorandom Permutation (PRP)[24]).*** *Let $F : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ be an efficient, length-preserving keyed function. We say that F is a pseudorandom permutation if for any probabilistic polynomial-time distinguisher D, there exist a negligible function $negl(n)$ such that*

$$|Pr[D^{F_K(\cdot)}(1^n) = 1] - Pr[D^{f(\cdot)}(1^n) = 1]| \leq negl(n), \tag{5}$$

where $K \overset{R}{\leftarrow} \{0,1\}^n$ is chosen uniformly at random and f is chosen uniformly at random from the set of functions mapping $\{0,1\}^n$ to $\{0,1\}^n$.

2.2.2. Encryption Schemes. In this section, we consider the problem of secure encryption. More specifically, we present the definitions of private-key and public-key encryption. We also focus on essential definitions of *semantic security.*

We first begin with the notion of encryption scheme.

Definition 10. ***(Encryption Scheme[24]).*** *An encryption scheme consist of a triple of probabilistic polynomial-time algorithms (G, E, D) satisfying the following conditions:*

- *On input* 1^n, *the* key-generator *algorithm* G *outputs a pair of keys* (e, d).
- *For every pair* (e, d) *in the range of* $G(1^n)$ *and for every* $\alpha \in \{0,1\}^\star$, *the encryption and decryption algorithms* E *and* D *satisfy*

$$Pr[D(d, E(e, \alpha)) = \alpha] = 1 \tag{6}$$

where the probability is taken over the internal coin tosses of algorithms E *and* D.

The integer n *serves as the* security parameter *of the scheme. The key* e *is called the* encryption key *and the key* d *is called the* decryption key. *The string* α *is the* plaintext *and* $E(e, \alpha)$ *is the* ciphertext. *For simplicity, we will denote* $E_e(\alpha) = E(e, \alpha)$ *and* $D_b(\beta) = D(d, \beta)$.

We now show how to construct private-key and public-key encryption schemes.

Definition 11. ***(Private-key Encryption Scheme[24]).*** *Let* $F = \{F_n\}$ *be an efficiently computable function ensemble, and let* I *and* V *be the sampling and evaluation functions, respectively. Then, define* (G, E, D) *as follows:*

- *Key generation:* $G(1^n) = (k, k)$, *where* $k \leftarrow I(1^n)$.
- *Encryption of* $x \in \{0,1\}^n$ *using key* k: $E_k(x) = (r, V(k, r) \oplus x)$ *where* $r \in_R \{0,1\}^n$.
- *Decryption of* (r, y) *using key* k: $D_k(r, y) = V(k, r) \oplus y$.

Definition 12. ***(Public-key Encryption Scheme[24]).*** *Let* (I, S, F, F^{-1}) *be a collection of trapdoor permutations and let* B *be a predicate. Then, define* (G, E, D) *as follows:*

- *Key generation:* $G(1^n) = (i, t)$, *where* $i = I_1(1^n)$ *is the first element of the output of* $I(1^n)$ *and* t *is the second element of the output of* $I(1^n)$, *or the "trapdoor".*

- *Encryption of $\sigma \in \{0,1\}$ using key i: Sample a random element x according to $S(i)$ and compute $y = F(i,x)$. Output (y,τ) where $\tau = B(i,x) \oplus \sigma$.*
- *Decryption of (y,τ) using key (i,t): Compute $x = F^{-1}(t,y)$ and output $\sigma = B(i,x) \oplus \tau$.*

An private-key or public-key encryption scheme is secure if the output ciphtertext reveals no information about the encrypted plaintext. Speaking formally, an encryption scheme is secure under *semantic security* if everything the adversary can learn about the plaintext given the ciphtertext, it could learn about the plaintext using its a priori knowledge alone. In other words, the probability of finding the plaintext from the ciphtertext is no much different from guessing the plaintext without the ciphtertext. Formally, this defined as follows.

Definition 13. ***(Semantic Security Private-key Model[24]).*** *An encryption scheme (G,E,D) is* semantically secure in the private-key model *if for every probabilistic polynomial-time algorithm A there exist a probabilistic polynomial-time algorithm $A^{'}$ such that for every polynomially-bounded probabilistic ensemble $\{X_n\}_{n\in\mathbb{N}}$, every pair of polynomially-bounded functions $f,h : \{0,1\}^\star \rightarrow \{0,1\}^\star$, every positive polynomial $p(\cdot)$ and all sufficient large n's*

$$\begin{aligned} &Pr[A(1^n, E_{G_1(1^n)}(X_n), 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)] \\ &< Pr[A^{'}(1^n, 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)] + \frac{1}{p(n)} \end{aligned} \tag{7}$$

where the probabilities are taken over X_n and the internal coin tosses of G, E, A and $A^{'}$.

The semantic security definition for public-key model is almost identical. In this definition $G_1(1^n)$ denotes the first key output by key-generator algorithm G. We include it only for reason of completeness.

Definition 14. ***(Semantic Security Public-key Model[24]).*** *An encryption scheme* (G, E, D) *is* semantically secure in the public-key model *if for every probabilistic polynomial-time algorithm A there exist a probabilistic polynomial-time algorithm $A^{'}$ such that for every polynomially-bounded probabilistic ensemble $X_{n n \in \mathbb{N}}$, every pair of polynomially-bounded functions $f, h : \{0,1\}^{\star} \rightarrow \{0,1\}^{\star}$ every positive polynomial $p(\cdot)$ and all sufficient large n's*

$$\begin{aligned} & Pr[A(1^n, G_1(1^n), E_{G_1(1^n)}(X_n), 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)] \\ & \quad < Pr[A^{'}(1^n, 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)] + \frac{1}{p(n)} \end{aligned} \tag{8}$$

where the probabilities are taken over X_n and the internal coin tosses of G, E, A and $A^{'}$.

CHAPTER 3

Related Work

In this chapter we present the result of a systematic literature review we conducted on existing research in the searchable encryption area. A systematic review is important for research activities since it summarizes existing techniques concerning a research interest and identifies further research directions. The purpose of the review described in this chapter is to compare current searchable encryption solutions and identify their limitations through a systematic evaluation. The review follows a meticulously designed paper selection procedure, and identifies different techniques in scientific journals and conferences published from 2000 to 2015.

We begin with an overview of the searchable encryption framework in Section 3.1. Next, we describe the state-of-the-art survey on symmetric searchable encryption techniques available in the literature in Section 3.2. We outline the comparison criteria that we will use to review existing solutions in Section 3.2.1. Section 3.3 is devoted to describe existing security models. In Section 3.4 we evaluate and compare current solutions to answer the research questions identified in Section 3.2.1. We review some other related solutions in Section 3.5.

3.1. Searchable Encryption System Model

Consider a cloud data hosting service show in Figure 3.1 that involves three entities: data owner, cloud server and data user. The data owner may be an individual or an enterprise, who wishes to outsource a collection of documents $D = (D_1, D_2, \ldots, D_n)$ in encrypted form $C = (C_1, C_2, \ldots, C_n)$ to the cloud server and still preserve the search functionality on outsourced data. Here, the data owner uses semantically secure encryption scheme E with a secret key

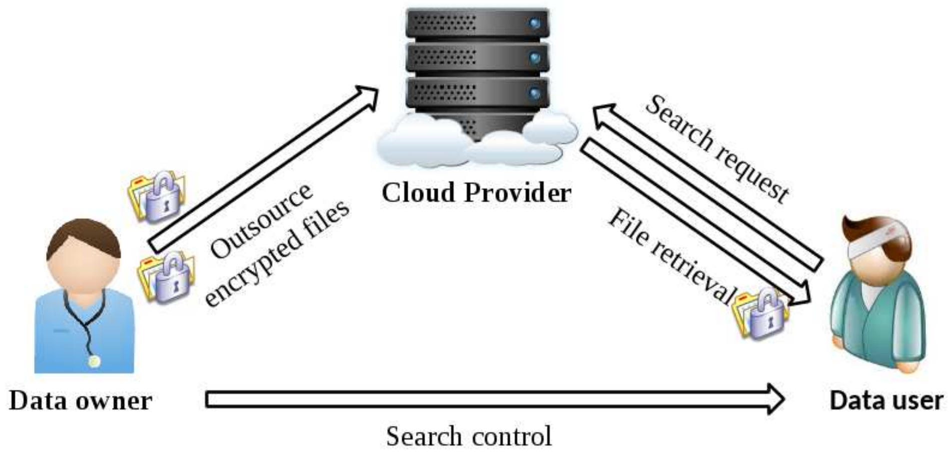


FIGURE 3.1. Searchable Encryption Framework.

S to encrypt each document D_i to form C_i, i.e. $C_i = E_S[D_i]$. To enable search, the data owner constructs an index I from collection D. The index is a data structure that allows an efficient keyword search on encrypted collection C without revealing any meaningful information about the original document content. The index is built on m distinct keywords $K = (k_1, k_2, \ldots, k_m)$ extracted from the original dataset D. Now, the data owner uploads both the index I and encrypted document collection C to the cloud server.

To securely search the document collection for one or more keywords $\bar{K} \in K$, the data user contacts the data owner to obtain a search control. The search control includes a search trapdoor that serves to input a set of keywords $\bar{K}$ and output a search query Q which is sent to the cloud server. Note, the trapdoor learning process is one-time operation and thus the data user does not need to contact the data owner anymore. Once the cloud server receives the search request Q, it invokes search on stored index I and returns a set of matching encrypted documents $L \subseteq C$ back to the data user. The data user then uses the secret key S to decrypt received documents L to original view.

The reminder of the this chapter uses the model defined by Curtmola *et al.*[5] that uses a tuple of four algorithms that enable search over encrypted document collection. We define these algorithms as follows:

- $KeyGen(1^s)$: a probabilistic key generation algorithm that is run by the data owner. This algorithm inputs the security parameter s and output a secret key S.
- $BuildIndex(K, D)$: a probabilistic algorithm run by the data owner that takes as input a collection of documents D, the secret key S and it outputs the index I.
- $Trapdoor(S, \bar{K})$: is a deterministic algorithm run by the data user that inputs keywords of interest $\bar{K}$, the secret key S and outputs search request Q.
- $Search(I, Q)$: is a deterministic algorithm run by the cloud server to execute search. The algorithm inputs the index I and search request Q (received from the data user). It outputs a set of $L \subseteq C$ matching encrypted documents.

Before discussing any existing solution we need to define the model of the adversary (or attacker). We assume that the adversary resides at the cloud server. In cryptographic protocols the *honest-but-curious* model has been traditionally used to model the adversarial behavior. We outline *honest-but-curious* model as follows:

- The cloud server is *honest*, that is, it is always available to the data user and it correctly follows the designated protocol specification, and it provides all services that are expected.
- The *curious* cloud server may try to perform some additional analysis to breach the confidentiality of the stored data (both index and encrypted documents) and search queries.

In this dissertation proposal we assume that the cloud server and the adversary are the same entity. That way, the adversary has access to the same set of information as the cloud server. In addition, we are not concerned about the cloud server being able to link a query to a specific user; nor are we concerned about any denial-of-service attacks.

3.2. Previous Work

3.2.1. Review Scope and Comparison Criteria.

Searchable encryption has been an active research area and many quality works have been published [1–18]. Traditional searchable encryption solutions build an index data structure such that its content is hidden, yet allowing document search with given search query. The scope of the systematic review can thus be restricted to research on searchable encryption techniques. In the rest of this survey we will focus only on symmetric searchable encryption, and detail the most renowned schemes. Our systematic review uses the following comparison criteria:

- **Search Type**: What kind of techniques are available to search an encrypted data in the cloud? What kind of efficient algorithms and data structures are used in modern solutions?
- **Search Expressiveness**: What types of search features can be executed? What is the granularity of search queries? It is desirable to support single as well as multi-keyword capability in the search.
- **Security**: What kind of security is provided in existing solutions? What kind of information is leaked to the adversary? It is important to protect the content of stored data (as well as the content of search queries) without leaking any meaningful information to the untrusted server.

- **Performance/Efficiency**: How do existing solutions scale over large data sets? What are the search and storage algorithm complexities? Pushing any existing solution to the real-world cloud storage requires a thorough performance evaluation.

3.2.2. Keyword Encryption Construction. The scheme proposed by Song *et al.*[1] was the first attempt to develop new techniques for keyword search over outsourced encrypted document collection. The authors begin with idea to store a set of plaintext documents on data storage server such as mail servers and file servers in encrypted form to reduce security and privacy risks. The work presents a cryptographic scheme that enables indexed search on encrypted data without leaking any sensitive information to the untrusted remote server.

To illustrate proposed scheme, consider the following example. A data owner Alice wishes to store a set of sensitive documents on a remote server Bob. Since Bob is untrusted, Alice wishes to encrypt her documents and only store set of ciphertexts on Bob. Whereas Alice want to utilize storage and computational resources of Bob, she wishes to retrieve only documents that contain a keyword w. To do so, Alice builds an index that provides desired keyword search and also leaks as little information as possible about the content or the original documents to Bob. In order to achieve this goal, for each word w Alice constructs a *search trapdoor* that allows her to retrieve matching documents from Bob.

We now describe the proposed scheme in details. Let the document D consist of the sequence of words $\{w_1, \ldots, w_m\}$. The index is created by computing the bitwise XOR of the keyword plaintext with a sequence of pseudorandom bits that Alice generated using a stream cipher. Figure 3.2 shows details of proposed technique.

First, Alice generates a sequence of pseudorandom values $\{s_1, \ldots, s_m\}$ using a stream cipher, where each s_i is $n-m$ bit long. For each sequence s_i, Alice computes a pseudorandom

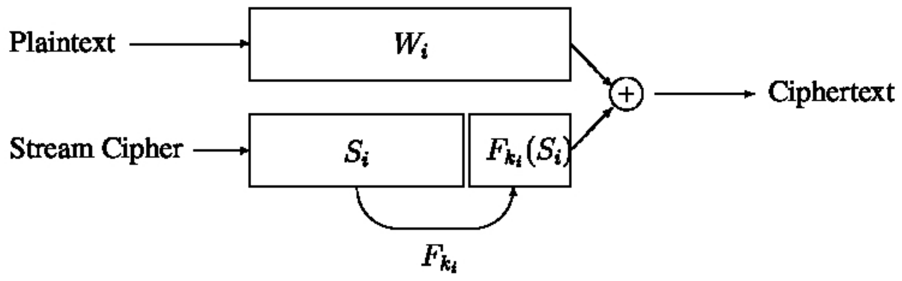


FIGURE 3.2. Basic Search Scheme. Taken from [1].

function $F_k(s_i)$ using a secret key k. Using the result of $F_k(s_i)$, Alice computes a n-bit sequence $t_i = < s_i, F_k(s_i) >$, where $< x, y >$ is concatenation of the strings x and y. To encrypt n-bit keyword, Alice calculates the ciphertext $c_i = w_i \oplus t_i$ with XOR operator. Since Alice is the only who generates the pseudorandom stream $\{t_1, \ldots, t_m\}$, no one else can decrypt c_i.

To execute a search on outsourced documents, Alice sends a keyword w and secret key k to remote server Bob. Bob, using existing index structure, analyzes incoming keyword w and checks whether $c_i \oplus w$ makes a form of $< s_i, F_k(s_i) >$. Once matching items are found, Bob sends back documents that contain the keyword w. Alice later can decrypt founded document back to the original view.

Proposed scheme is a pioneer work in the area (proposed in early 2000') of remote searching on encrypted document collection. Experiments showed that for a document of length n, the encryption and search algorithms need $O(n)$ stream cipher and block cipher operations. We can clearly see that proposed scheme requires one key for each keyword in every document and thus introduces an extremely large storage and computational overhead. From security point of view, this scheme reveals plaintext keywords to the remote server and thus does not provide an adequate security and privacy protection of the search query.

3.2.3. Secure Index Construction. The solution proposed by Song *et al.*[1] was greatly improved by Goh *et al.*[2]. New proposed scheme "secure index" greatly improved search complexity compared to previous approach. The search complexity is reduced to $O(D)$, where D is total number of documents in the collection, which makes this scheme as the first practical searchable encryption scheme. The scheme uses an index that is constructed using a probabilistic data structure called the Bloom filter[30]. We first provide details on the Bloom filter data structure and then we describe the searchable encryption scheme in details.

Bloom filter is a probabilistic data structure that represent sets while allowing efficient membership testing. Bloom filter allow verification of whether or not an element exist in a given set, storing all elements in an efficient way. The most important feature of Bloom filter is that the time of the element verification is constant, $O(1)$. We now describe the formal definition of Bloom filter.

Definition 15. ***(Bloom Filter).*** *A Bloom filter represents a set of n elements $S = \{a_1, \ldots, a_n\}$ and r independent hash function $h_i : \{0,1\}^\star \to \{0,m\}$ where $i \in [1,r]$. Then, for each $a \in S$, the array bits at positions $h_1(a), \ldots, h_r(s)$ are set of 1. To determine if an element $a^\star$ belongs to the set S, we check the bits at positions $h_1(a^\star), \ldots, h_r(a^\star)$. If all the bits are set to 1, then $a^\star$ is considered a member of the set.*

A bloom filter does not store the elements themselves, in fact it stores the bit output generated by hash functions. However, due to the nature of hashing, the false positive occurrences may appear in the Bloom filter. Note that Bloom filter does not require rebuilding in order to add the new elements of the original set.

"Secure index" scheme proposed by Goh *et al.*[2] uses Bloom filter to store keywords extracted from the document collection. Specifically, the solution uses Bloom filter to map each document to a set of unique keywords, i.e. for each document D_i in the collection there exist a Bloom filter, constructed using keywords from D_i. Before describing the details of proposed technique, we first give cryptographic notations used in this work:

- There exists a pseudorandom function $f : \{0,1\}^n \times \{0,1\}^s \rightarrow \{0,1\}^s$, and a key set $K = \{k_i \overset{R}{\leftarrow} \{0,1\}^s | 1 \leq i \leq r\}$, where s is a security parameter and r s the number of hash functions used by Bloom filter.
- For every document d in the collection of documents D, there is a unique document identifier $id(d)$.
- The index is constructed as a result of process involving the generation of code-word and a trapdoor.

We now describe the details of framework.

- KeyGen(k): an algorithm that inputs a secret parameter k. The algorithm outputs a secret key K_{priv}, which is decomposed into r other keys $K_{priv} = (k_1, \ldots, k_r)$.
- Trapdoor(w, K_{priv}): an algorithm that inputs the keyword w, secret key K_{priv}. The output of the algorithm is defined as: $T_w = (f_{k_1}(w), \ldots, f_{k_r}(w))$
- BuildIndex(d, K_{priv}): an algorithm that inputs a single document d and outputs a Bloom filter $BF(d)$. Let w_i represents a keyword in document d. The index is constructed using trapdoor of w_i and document identifier $id(d)$ as follows: $C_{w_i} = (f_{T_{w_1}}(id(d)), \ldots, f_{T_{w_r}}(id(d)))$. Lastly, the algorithm inserts each result of C_{w_i} into the Bloom filter $BF(d)$. Output index $I = (id(D), BF(d))$

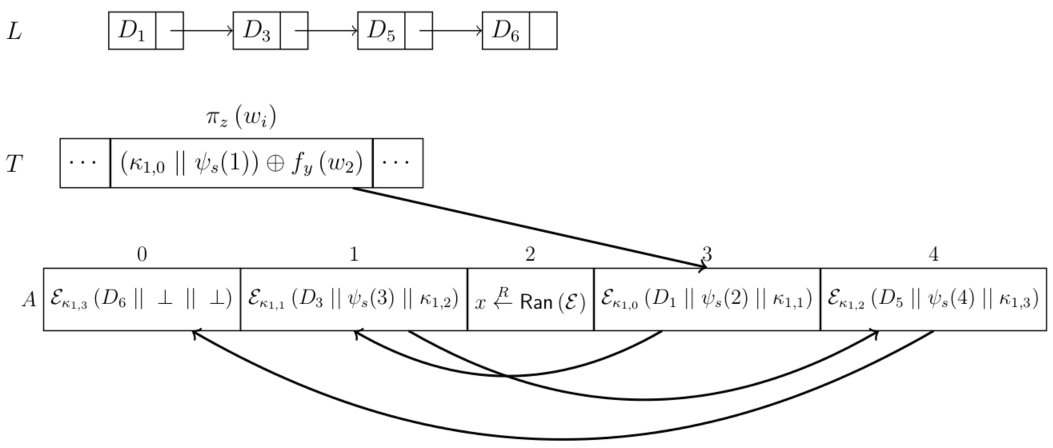
- Search(T, I): an algorithm that inputs a trapdoor T_w and index I. The algorithm calculates $C_w = (f_{T_{w_1}}(id(d)), \ldots, f_{T_{w_r}}(id(d)))$ and checks $BF(d)$ if such entry exists in Bloom filter. The algorithm outputs a set of documents that contain a keyword w back to requester.

Proposed scheme is very efficient since the use of Bloom filter enables pre-processing of search on the client side and consequently the search on the server becomes linear in the number of documents. The pre-processing requires computation of $O(n)$ hash functions, where n is the number of keywords for a single document, hence the construction is performed in $O(n \times D)$, where D is the number of documents. While proposed solution is efficient, it suffers from the possibility of false positives. In this sense the remote server may return identifiers for documents that do not contain keywords of interest.

3.2.4. Searchable Symmetric Encryption (SSE-1) Construction.

Previous searchable encryption scheme[2] remained the most efficient scheme until 2006, when Curtmola *et al.*[5] proposed the first searchable symmetric encryption scheme (SSE). SSE scheme allows a party to outsource the storage of his data to another party in a private manner, while maintaining the ability to selectively search over it. The this work, the authors describe a non-adaptively secure SSE system (SSE-1) and adaptively secure SSE system (SSE-2). Both SSE-1 and SSE-2 are index-based schemes and they allow a keyword search over encrypted document collection. In this section we preset non-adaptive SSE-1 solution. Section 3.2.5 describes adaptive SSE-2 scheme.

Now we describe the properties of non-adaptive adversary and give the definition of history that is used in non-adaptive security. The history is defined as the interaction between the cloud client and the cloud server by a document collection and a sequence of



L
D_1
D_3
D_5
D_6
$\pi_z(w_i)$
T
$\cdots$
$(\kappa_{1,0} \mid\mid \psi_s(1)) \oplus f_y(w_2)$
$\cdots$
0
1
2
3
4
A
$\mathcal{E}_{\kappa_{1,3}}(D_6 \mid\mid \perp \mid\mid \perp)$
$\mathcal{E}_{\kappa_{1,1}}(D_3 \mid\mid \psi_s(3) \mid\mid \kappa_{1,2})$
$x \overset{R}{\leftarrow} \mathsf{Ran}(\mathcal{E})$
$\mathcal{E}_{\kappa_{1,0}}(D_1 \mid\mid \psi_s(2) \mid\mid \kappa_{1,1})$
$\mathcal{E}_{\kappa_{1,2}}(D_5 \mid\mid \psi_s(4) \mid\mid \kappa_{1,3})$

FIGURE 3.3. SSE-1 Index.

keywords that the client wants to search for and that we want to hide from the adversary. The authors assume that the adversary generates the histories at once, i.e. the adversary is not allowed to see the index of the document collection or the trapdoors of any keywords it chooses before it has finished generating the history. In non-adaptively secure SSE approach the index is based on encrypted array and permuted linked list data structures.

We now give some cryptographic notations used in non-adaptively secure SSE scheme. Let k and l be security parameters, and there exist semantically secure encryption system, one pseudorandom function f, and two pseudorandom permutations π, ψ. The pseudorandom function has following parameters $f : \{\}^k \times \{0,1\}^p \rightarrow \{0,1\}^{l+lg(m)}$, where m is the total size of the plaintext document collection and p is the size of a keyword. Two pseudo-random permutations are defined as $\pi : \{0,1\}^k \times \{0,1\}^p \rightarrow \{0,1\}^p$ and $\psi : \{0,1\}^k \times \{0,1\}^{lg(m)} \rightarrow \{0,1\}^{lg(m)}$. The semantically secure encryption scheme (G, E, D) has an encryption function $E : \{0,1\}^l \times \{0,1\}^r \rightarrow \{0,1\}^r$, where r is the block size.

Non-adaptively secure SSE scheme is defined as follows:

- KeyGen(k): an algorithm that generates key as a triple of random bit strings needed for the system: $s, y, z \overset{R}{\leftarrow} \{0,1\}^k$.
- BuildIndex(D, s, y, z): an algorithm that constructs an index that consists of encrypted array with permuted linked list, as follows:
  1. Scan the document collection to construct $id(D)$, where $id()$ assigns the identifier of the document D. Build a keyword dictionary Δ that contains all the distinct keyword in D. Initialize the counter c to 1.
  2. For each keyword $w \in \Delta$ build the document set $D(w)$, which includes the set of all document with keyword w.
  3. For each $w_i \in \Delta$ build an encrypted permuted linked list that contains $D(w_i)$ and store it in array A. Select $\kappa_{i,0} \overset{R}{\leftarrow} \{0,1\}^l$ for each i.
  4. For each j^{th} identifier in $D(w_i)$:
     - Select $\kappa_{i,j} \overset{R}{\leftarrow} \{0,1\}^l$ and create $N_{i,j} = id(D_{i,j})||\kappa_{i,j}||\psi_s(c+1)$, where $id(D_{i,j})$ is the j-th identifier in $D(w_i)$.
     - Compute $E_{\kappa_{i,j-1}}(N_{i,j})$ and store it in A at location $\psi_s(c)$, i.e. $A[\psi_s(c)] = E_{\kappa_{i,j-1}}(N_{i,j})$.
     - Increase counter c by 1.
  5. To make the start of the list in array A, a lookup table is constructed for each $w_i \in \Delta$
     - Let $v = (addr(A(N_{i,1}))||\kappa_{i,0}) \oplus f_y(w_i)$, where $addr(A(N_{i,j}))$ denotes the address of $N_{i,j}$ in A.
     - Set location $\pi_z(w_i)$ of T to v, i.e. $T[\pi_z(w_i)] = v$.
  6. Set any empty locations in table T to random bit strings.

- Trapdoor(w, z, y): for a given keyword w, the search trapdoor is calculated as a tuple $T_w = (\pi_z(w),\ f_y(w))$.
- Search(T, I): the remote server first locates the document set for a keyword w using $\pi_z(w)$ in index I. Next, the algorithms walks the list and uses the κ values it finds at each node to decrypt the subsequent node values. The resulting document identifiers are then sent back to the data user.

An illustrative example of SSE-1 index construction for document corpus $\{D_1, D_3, D_5, D_6\}$ is shown on the Figure 3.3.

3.2.5. Searchable Symmetric Encryption (SSE-2) Construction. Now we describe the adaptively secure SSE scheme (SSE-2), which is significantly simpler in all algorithms than non-adaptive form. SSE-2 scheme uses the stronger security model (adaptive security), which takes into consideration the adaptive attacker. Here, the adaptive attacker can take into consideration the history of all queries sent to it from the first data user's query. In SSE-2 scheme, the index is constructed from an encrypted lookup table of keyword/document identifier pair. SSE-2 relies on the following four algorithms:

- KeyGen(k): an algorithm that generates key $s \overset{R}{\leftarrow} \{0,1\}^k$.
- BuildIndex(s, D): an algorithm that constructs the index as follows:
  1. Scan the entire document collection and construct a dictionary Δ that contains all the distinct keywords in D.
  2. For each keyword $w \in \Delta$ build the document set $D(w)$.
  3. For each keyword $w_i \in \Delta$, apply $\pi_s(w_i||j)$ and set the value in table $T[\pi_s(w_i||j)] = id(D_{i,j})$. Here, π is pseudorandom permutation and $id(D)$ is document identifier (name).

(4) Set all empty entries in the table T to random values with correct length.

- *Trapdoor*: an algorithm that constructs trapdoor $T_w = (\pi_s(w||1), \ldots, \pi_s(w||max))$, where max is the size of the longest plaintext document in D.
- *Search*: an algorithm inputs the trapdoor T_w and finds values in the table T. The result identifiers are collected and returned to the querier.

3.2.6. Ranked Search Construction. In this section we present a multi-keyword ranked search over encrypted data proposed by Cao *et al.*[9]. Their searchable encryption scheme is based on information retrieval techniques to measure the similarity between the given search query and the each encrypted document stored at the cloud server. To efficiently achieve multi-keyword ranked search, the scheme uses "inner product similarity"[31] to quantitatively evaluate the efficient similarity measure. More specifically, let D_i be the binary data vector for document F_i, let $D_i[j] \in \{0, 1\}$ defines the existence of a corresponding keyword w_j in that document, let Q be the binary query vector that indicate the keyword of interested (submitted by the data user). The similarity score of the document F_i to the search query Q is defined as inner product of their binary column vectors, i.e. $D_i \times Q$. The ranking is done at the cloud server in a way that the server must rank the documents with the highest inner product output and return the documents back to the data user.

We now describe the scheme in details:

- *KeyGen*: an algorithm that generates a $(n + 2)$-bit vector S, two $(n + 2) \times (n + 2)$ invertible matrices $\{M_1, M_2\}$, where n is the number of keywords in the document collection. The output is tuple $\{S, M_1, M_2\}$.

- *BuildIndex*: an algorithm that generates the binary data vector D_i for each document F_i. Every D_i is expanded to the view $\bar{D}_i$ by applying dimension extension and including random variables. The output of the algorithm is index $I_i = \{M_1^T \bar{D}_i, M_2^T \bar{D}_i\}$ for each encrypted document C_i.
- *Trapdoor*: with t keywords of the interest, the trapdoor is generated as vector Q, where each bit $Q[i]$ indicates if keyword exists in the dictionary (1 or 0 values). In similar way vector Q is extended and scaled to the random value $r \neq 0$ to form $\bar{Q}$. The trapdoor output is a tuple $\{M_1^{-1}\bar{Q}, M_2^{-1}\bar{Q}\}$.
- *Search*: the remote server inputs the trapdoor T and compute an inner score between each index I_i and the trapdoor T: $score = I_i \times T$. After sorting all scores, the remote server returns the top-k ranked document list back to the data user.

Proposed scheme sorts documents using the score based on "inner product similarity" where a document score is simply the number of matches of query keywords in each document. Therefore, this ranking loses information about keyword importance to the document collection w.r.t. document lengths and other keywords (e.g., document which contain all query keywords are ranked equally). Also, proposed scheme uses a heuristic to hide the search and access patterns by adding dummy keywords and noise. As a result, the returned document list may contain false negatives and false positives, which is not desired property. In addition, the authors did not define a security model beyond the general data privacy model.

3.2.7. Fuzzy Keyword Search Construction. As we have seen above, many effective symmetric searchable encryption schemes have been proposed. Most of the time, these schemes consist of building up some indexing data structure and associate constructed index with document collection. By integrating the trapdoors of keyword within the index information, effective keyword search can be executed while both document content and keyword privacy are well-preserved. Although allowing for performing searches securely and effectively, existing searchable encryption techniques do not work well for cloud computing scenario since they support only *exact* keyword search, i.e. there is no tolerance of minor typos and format inconsistencies. It is quite common that cloud users' input might not match pre-defined set of keywords and may contain typos, such as "Britney" and "Brittaney", "Illinois" and "Ilinois", and/or just lack of exact knowledge about the data. Li *et al.*[18] were the first to focus on enabling effective yet privacy preserving fuzzy keyword search on encrypted data outsourced to the cloud. Fuzzy keyword search greatly enhances system usability by returning the matching documents when cloud users' searching inputs exactly match the predefined keyword or the closest possible matching files based on keyword similarity semantics, when exact match fails. The authors use edit distance (number of operations (substitution, deletion, insertion) required to transform one work into another) to quantify keywords similarity and develop a novel technique - fuzzy keyword symmetric searchable encryption that is based on wildcard fuzzy sets. We first provide details on the wildcard-based fuzzy set construction and then we describe the main searchable encryption scheme.

Wildcard-based fuzzy set construction is a technique to list all possible variants of the keyword with given edit distance. The construction works as follows: the set of a keyword w_i with edit distance d is denoted as $S_{w_i,d} = \{S'_{w_i,0}, S'_{w_i,1}, \ldots, S'_{w_i,d}\}$, where $S'_{w_i,\tau}$ denotes the set of words w'_i with τ wildcards. For example, for the keyword "CAT" with the pre-set edit distance 1, its wildcard-based fuzzy keyword set can be constructed as $S_{CAT,1} = \{CAT,$

$*CAT$, $*AT$, $C*AT$, $C*T$, $CA*T$, $CA*$, $CAT*\}$, where $*$ represents an edit operation on that position. From this example we can see that with larger pre-set edit distance value (and keyword dictionary), more storage overhead is required.

We now describe the details of proposed fuzzy keyword symmetric searchable encryption scheme:

- KeyGen(k): an algorithm that generates key $sk \overset{R}{\leftarrow} \{0,1\}^k$.
- BuildIndex(sk, D): an algorithm that first computes a trapdoor for each keyword w_i as follows: $T_{w_i'} = f_{sk}(w_i')$ for all $w_i' \in S_{w_i,d}$ with secret key sk and pseudorandom permutation f. Second, the algorithm uses symmetric key encryption to encrypt document identifier and trapdoor output as follows: $SKE.Enc_{sk}(id(D)_{w_i}||w_i)$. The output of algorithm is index table $\{\{T_{w_i'}\}_{w_i' \in S_{w_i,d}}, SKE.Enc_{sk}(id(D)_{w_i}||w_i)\}$.
- Trapdoor(sk, w): an algorithm that computes trapdoor for a given keyword w and pre-set edit distance k. First, the algorithm forms a fuzzy set $S_{w,k}$. Second, the algorithm computes trapdoor $\{T_{w'}\}_{w' \in S_{w,k}}$ for each element $w' \in S_{w,k}$. The trapdoor output is submitted to the cloud server.
- Search(T, I): an algorithm that checks if given trapdoor exist in index stored at the cloud server. The algorithm returns the search result $SKE.Enc_{sk}(id(D)_w||w)$ if there exists exact match. Otherwise, the algorithm compares all elements of $\{T_{w'}\}_{w' \in S'_{w,\tau}}$ $(1 \leq \tau \leq k)$ with index for the document collection and return matching results $\{SKE.Enc_{sk}(id(D)_{w_i}||w_i)\}$ back to requester.

3.3. Existing Security Models

In order to talk about the security of any system we must model the abilities of the attacker. In this case, the attacker is the cloud server shown on Figure 3.1. In this section we focus on formal definitions of security properties in the searchable encryption. There have been proposed many different security models in the literature . We focus on following four security models:

- Indistinguishability under Chosen Keyword Attacks (IND-CKA and IND2-CKA) defined by Goh *et al.*[2].
- Non-Adaptive and Adaptive Indistinguishability against Chosen Keyword Attacks (CKA-1 and CKA-2) defined by Curtmola *et al.*[5].

We should note the pioneer work by Song *et al.*[1] did not define any security models beyond the normal cryptographic assumptions.

We begin our discussion with two security models: IND-CKA and IND2-CKA, first security models defined by Goh *et al.*[2]. IND-CKA model aims to capture the notion that an adversary A cannot deduce a document's contents from its index. The model involves the challenger C and the adversary A, and uses the game-defined security. Suppose the challenger C gives the adversary A two equal length documents D_0 and D_1, each containing some number of keyword together with an index. Adversary's challenge here is to determine which document is encoded in the index. If A cannot determine which document is encoded in the index with probability non-negligibly different from 1/2, then the index reveals nothing about its contents. Note, that the index does not hide information such as document size that can be obtained by examining the encrypted documents.

We are now ready to give the formal definition of IND-CKA security model:

Definition 16. ***(IND-CKA Game[2]).*** *An IND-CKA game between the challenger C and the adversary A is a game that includes the following four rounds:*

- *Setup: The challenger C creates a set S of q words and gives this to the adversary A. The adversary then chooses a number of subsets from $S^\star$. Once C receives $S^\star$ it runs BuildIndex algorithm to create index I for each document D. The challenger C concludes by sending all indexes with their associated subsets to A.*
- *Queries: The adversary A is allowed to query C on the keyword k and receive the trapdoor T for k. With T, the adversary A can invoke Search on an index I and determine if k exist in I.*
- *Challenge: After making some trapdoor T queries, A decides on a challenge by picking a non-empty subset $V_0 \in S^\star$, and generates another non-empty subset V_1 from S such that $|V_0 - V_1| \neq 0$, and the total length of the words in V_0 is equal to the total length of words in V_1. Finally, A must not have queries C for the trapdoor of any work in $V_0 \cup V_1$. Next, A gives V_0 and V_1 to C who chooses $b \xrightarrow{R} \{0,1\}$, invokes BuildIndex to returns index I_{V_b} for V_b to the adversary A. The challenge for A is to determine b. After the challenge is issued, the adversary A is not allowed to query C for the trapdoors of any keyword $k \in (V_0 - V_1)$.*
- Response*: The adversary A eventually outputs a bit b', representing its guess for b. The advantage of A in winning this game is defined by $Adv_A = |Pr[b = b'] - \frac{1}{2}|$. This probability is taken over all the internal coin tosses of A and C.*

We say that the adversary A (t, ϵ, q)-breaks the index if Adv_A is at least ϵ after A takes at most t time and makes q trapdoor queries to the challenger. The index I is (t, ϵ, q)-IND-CKA secure if no adversary can (t, ϵ, q)-break it.

IND2-CKA security game has slightly stronger security than IND-CKA. More specifically, in IND2-CKA game, with the given access to a set of indexes, the adversary is not able to learn any partial information about the encrypted document that cannot be learned from possessing the trapdoor. Here, possessing the trapdoor only provides the knowledge of whether or not a keyword occurs in the index.

Taking into account IND-CKA and IND2-CKA security models, they have a set deficiencies. For instance, both IND-CKA and IND2-CKA models *do not require* trapdoors to be kept secure. Namely, the trapdoor T for a keyword k may reveal k entirely because this property is not necessary for all applications[2]. This issue can easily lead to scenario where an adversary can recover the keyword content from the trapdoor.

To address the issues above Curtmola *et al.*[5] introduced the set of formal definitions for non-adaptive and adaptive indistinguishability notions, and showed a reduction to a form of non-adaptive and adaptive indistinguishability notions for semantic security. The interaction between the cloud user and cloud server is determined by a document collection and a sequence of keyword that the user wants to search for and that we wish to hide from the adversary. In non-adaptive security model, the security is only guaranteed when cloud users generate all queries at once (in one batch). This might not be feasible for certain (practical) scenarios[5]. In the case of adaptive security, the security is guaranteed even if the users generate queries as a function of previous search outcomes. It should be noted that until work by Curtmola *et al.*[5], any searchable encryption schemes were classified in the non-adaptive sense at best. We now give more details to understand the definitions of non-adaptive and adaptive security guarantees.

The strongest definition of searchable encryption scheme can be characterized as the requirement that nothing is leaked beyond the outcome of the search execution that includes *search pattern* and *access pattern.* Search pattern refers to the information that can be derived in the following sense: given that two searches return the same results, determine whether two searches use the same keyword. Access pattern refers to the information that is implied by the query results. For example, one query can return a document x, while the other query could return x and another 5 documents. This implies that the predicate used in the first query is more restrictive than that in the second query. We say that searchable encryption scheme is secure if nothing is leaked beyond the search pattern and access pattern.

Curtmola *et al.*[5] defined three sources of information over an interaction between the client and the remote server: the history, the view and the trace. The *history* is the combination between a given query and set of resulted documents matching the set of keyword. The *view* consist of the encrypted documents, the index and incoming trapdoors, i.e. all information given to the remote server. The *trace* defines the information about all of the structure of the interaction. More specifically, the trace includes the length of the documents, the search outcomes, and the search patterns. We now formalize these notions:

DEFINITION 17. ***(History).*** *Let D be a document collection of n documents and δ is a keyword dictionary. A* history *H_q is an interaction between the client and the server over q queries, denoted as $H_q = (D, k_1, \ldots, k_q)$, where k_i is the queried keyword.*

DEFINITION 18. ***(View).*** *Let D be a document collection of n documents and δ is a keyword dictionary. The* view *of the adversary using history H_q is defined as: $V(H_q) =$*

$(id(D_1), \ldots, id(D_n)), (C_1, \ldots, C_n), I, (T_1, \ldots, T_q)$, where $id(D_i)$ is the document identifier (i.e. name), C_i is encrypted form of document D_i, I is the index, T_i is the trapdoor constructed from keyword k_i.

Definition 19. ***(Trace).*** *Let D be a document collection of n documents and δ is a keyword dictionary. The* trace *Tr is the following tuple: $Tr(H_q) = (id(D_1), \ldots, id(D_n))$, $(|D|, \ldots, |D_n|)$, $(D(k_1), \ldots, D(k_q))$, π_q, where $|D_i|$ denote the length of document D_i, D_{k_i} specifies the set of document identifiers that contain the keyword k_i and π_q is the search pattern of the cloud user.*

We now give the definition of non-adaptive semantic security.

Definition 20. ***(Non-Adaptive Semantic Security (CKA-1)[5]).*** *A searchable encryption scheme is non-adaptively semantically secure if $\forall q \in \mathbb{N}$ for all (non-uniform) probabilistic polynomial-time adversaries A, there exists a (non-uniform) probabilistic polynomial-time algorithm (simulator) S such that for all traces Tr_q of length q, all polynomially sampleable distribution $\mathbb{H}_{\shortparallel}$ over $H_q \in 2^{2^\delta} \times \delta^q : Tr(H_q) = Tr_q$, for all functions $f : \{0,1\}^m \rightarrow \{0,1\}^{l(m)}$, for all polynomials p, and sufficiently large s, we have:*

$$|Pr[A(V_k(H_q)) = f(H_q)] - Pr[S(Tr(H_q)) = f(H_q)]| < negl(s) \tag{9}$$

where $H_q \overset{R}{\leftarrow} \mathbb{H}_{\shortparallel}$ and the probabilities are taken over $\mathbb{H}_{\shortparallel}$ and the internal coin tossing over the algorithms KeyGen, BuildIndex, A and S.

In other words, the searchable encryption scheme is secure if the adversary is unable to learn anything more than what he can learn from the index. The definition above describes the scenario if the simulator can simulate some function of the history that the adversary cannot distinguish with negligible probability, where the simulator is given access to only the trace of the history and the adversary is given the view of the history of previous queries.

The stronger security can occurs if the simulator is given access to only the partial trace of the history and the adversary is only given an access to a partial view of the history. The partial trace of the history is denoted as H_q^t where t is number of elements of the q-length history. The partial view V_q^t is composed of the t elements of the q-length view. We now define the adaptive semantic security for searchable encryption schemes.

Definition 21. ***(Adaptive Semantic Security (CKA-2)[5]).*** *A searchable encryption scheme is adaptively semantically secure if for all $q \in \mathbb{N}$ and for all (non-uniform) probabilistic polynomial-time adversaries A, there exist a (non-uniform) probabilistic polynomial-time algorithm (simulator) S such that for all traces Tr_q of length q, all polynomially sampleable distribution $\mathbb{H}_{||}$ over $H_q \in 2^{2^\delta} \times \delta^q : Tr(H_q) = Tr_q$, all functions $f : \{0,1\}^m \rightarrow \{0,1\}^{l(m)}$, all $0 \leq t \leq q$, and all polynomials p, and sufficiently large s, we have:*

$$|Pr[A(V_k^t(H_q^t)) = f(H_q^t)] - Pr[S(Tr(H_q^t)) = f(H_q^t)]| < negl(s) \tag{10}$$

where $H_q \overset{R}{\leftarrow} \mathbb{H}_{||}$ and the probabilities are taken over $\mathbb{H}_{||}$ and the internal coin tossing over the algorithms KeyGen, BuildIndex, A and S.

We say that the searchable encryption is secure if the simulator can simulate a view of the partial history that the adversary cannot distinguish with more than negligible probability from the actual view.

Table 3.1. Comparison of search features and security analysis of existing searchable encryption schemes.

Scheme	Search type	Query type	Security level	Adaptive adversary
Song [1]	Linear	Single keyword	CPA	No
Goh [2]	Index	Single keyword	CKA-1	No
SSE-1 [5]	Index	Single keyword	CKA-1	No
SSE-2 [5]	Index	Single keyword	CKA-2	Yes
Cao [9]	Index	Multiple keywords	CKA-1	No
Li [18]	Index	Single keyword	CKA-2	Yes

3.4. Evaluation and Open Issues

In this section we present an evaluation of the symmetric searchable encryption solutions using previously introduced comparison criteria in Section 3.2.1. We split the comparison into two subsets. Table 3.1 deals with search types, query types and security features, namely the security level and ability to handle adaptive adversaries. Table 3.2 shows a comparison that deals with search performance by detailing the search, query execution complexity and index storage. We use the same terminology for both tables: d is the number of encrypted documents, $d(w)$ is the number of documents containing the keyword w, Δ is the keyword dictionary in d. We use security notations from [2, 5]: $CKA1$ refers to the security against chosen-keyword attack, $CKA2$ is the security model against adaptive chosen-keyword attacks.

We now discuss the open issues with existing solutions in the area of searchable encryption. From Table 3.1 we can see that almost all existing symmetric searchable encryption solutions are index-based and support single keyword search semantics. Cao's solution [9] is the first work that supports multi-keyword semantics and ranks documents using similarity metric, however it clearly lacks strong security model against an adaptive adversary (which was introduced in SSE-2 scheme by Curtmola *et al.*[5]). Moreover, Cao's scheme has a linear

TABLE 3.2. Performance comparison of existing searchable encryption solutions. With d we denote the size of document collection, with $d(w)$ we denote the size of document collection that contain keyword w, with Δ the size of the keyword dictionary, with τ the number of fuzzy sets for a keyword w.

Scheme	Search complexity	Index storage	Number of rounds	Query
Song [1]	$O(d \times \Delta)$	N/A	1	O(1)
Goh [2]	$O(d)$	$O(d)$	1	O(1)
SSE-1 [5]	$O(d(w))$	$O(d + \Delta)$	1	O(1)
SSE-2 [5]	$O(d(w))$	$O(d \times \Delta)$	1	O(1)
Cao [9]	$O(d)$	$O(d \times \Delta)$	2	O(1)
Li [18]	$O(\tau)$	$O(d \times \Delta \times \tau)$	2	O(1)

search complexity according to the Table 3.2, and thus the search is executed on the whole document collection (even if only one document contain the queried keyword), which makes this solution inefficient over the large data collection.

Finally, looking at both Table 3.1 and Table 3.2, almost all existing solutions support only *exact* keyword search and there is no tolerance of minor typos in the search query. Li *et al.*[18] is the only work that allows fuzzy keyword search over encrypted document collection. However, this work requires a very large storage and search time according to Table 3.2 (τ is the number of fuzzy sets for a single keyword). With this, we conclude that all existing searchable encryption techniques are not ready to efficiently handle different types of search (other than exact match) in secure and efficient manner. We believe that there is a need for more advanced techniques in the searchable encryption area.

Only few papers[9, 18] provide an implementation of these schemes including performance numbers. Most implementations are not publicly available, which makes it hard to compare the schemes on the same hardware with the same dataset. Moreover, it is difficult to provide a direct performance comparison since existing protocols for searchable encryption address different scenarios and security models.

3.5. Other Solutions

To complete the systematic review, we describe some related topics, which are often addressed with conjunction with symmetric searchable encryption.

3.5.1. Asymmetric Searchable Encryption.

In asymmetric searchable encryption, any user with a public key can store the document collection in the cloud server, but only the one possessing the corresponding private key can generate encrypted queries and generate search queries for a keywords of interest in the documents stored by other users. Such schemes may be useful in an email scenario. For example, consider a person who wants to retrieve encrypted email messages containing a given keyword from remote mail server. Any sender should be able to encrypt email with the public key, while only the receiver should have the ability to query for a given keyword using the private key.

The security goals of asymmetric searchable encryption are the same as in symmetric searchable encryption with the additional property that any sender should not be able to gain access to keywords included in documents sent by another sender. The first solution was introduced in [32] and later, improved schemes were proposed in [14, 13, 33], where the authors focus on improving either the search complexity or the search features (for example, multiple keywords in the search query). As with the symmetric setting, there have been several enhancements to search and storage complexity in the public settings as well in [34–37].

3.5.2. Private Information Retrieval Approach.

Private Information Retrieval (PIR) focuses solely on allowing users to retrieve documents without revealing the access pattern, i.e. which documents are being retrieved. Confidentiality of documents is not the focus and thus documents can be stored at the server unencrypted. The examples of such application are services that provide the public access to the information, where the users wants to preserve the anonymity and the search interests. PIR presents a great deal of complexity, particularly in terms of communication cost. The first solution was address by [38] and later by [39], where the authors proposed the scheme with communication cost equal to n^a, where n is the number of bits in the database and $a < 1$. Next, [40] proposed solution that uses φ-hiding assumption for its security model and achieves a poly-logarithmic communication cost. The most recent approach by [41, 42] reduce the communication overhead even further.

3.5.3. Oblivious RAM Approach. Oblivious RAM (O-RAM) solves both searchable encryption and PIR issues by continuously shuffling the memory to prevent the cloud server from learning any information about the search query and previously issues queries by the user. O-RAM was initially introduced in [43, 44] in order to enable efficient protection from reverse engineering by hiding program access patterns from the computer memory. O-RAM have many other applications including symmetric searchable encryption and secure operations between central processing unit (CPU) and an external memory. O-RAM focuses on more ambitious security goal than searchable encryption by hiding not only the content of search queries and documents, but also the search pattern and access pattern. The main disadvantage of O-RAM is the search complexity, storage and communication, which are in order of magnitude higher than in searchable encryption. For this reason, the recent O-RAM solutions [45–47] focus on reducing the search complexity as well as storage and communication complexity.

CHAPTER 4

Multi-keyword Similarity Searchable Encryption Scheme (MKSim)

In this chapter we investigate the problem of finding mechanisms to enable efficient multi-keyword similarity search in a collection of encrypted documents. Researchers have investigated this problem quite extensively in the context of encrypted documents [1, 2, 6, 8–11, 5, 13–16, 3, 4, 48–50]. Solutions generally involve building an encrypted searchable index such that its content is hidden from the remote server yet allowing the corresponding documents to be searched. These solutions differ from each other mostly in terms of whether they allow single keyword search or multi-keyword search, and in terms of the types of techniques they use to build the trapdoor function that facilitates the search. A few of them, most notably [8–10], allow the notion of similarity search. The similarity search problem consists of a collection of data items that are characterized by some features, a query that specifies a value for a particular feature, and a similarity metric to measure the relevance between the query and the data items. However, these techniques either do not allow searching on multiple keywords and ranking the retrieved document in terms of similarity scores or are very computationally intensive. Moreover, none of these schemes are resistant against *adaptive* adversaries [5]. Taking into account large volumes of data available today, there is need for efficient methods to perform secure similarity search over encrypted data outsourced into the cloud. Finally, these works are mostly confined to the single user setting – the owner of the encrypted document corpus is the one to search it. If an arbitrary group of data users need to search the encrypted document corpus, existing schemes fail to manage their access privileges. In this work, we propose a novel secure and efficient multi-keyword similarity searchable encryption scheme that returns the matching data items in a ranked order.

Our contributions can be summarized as follows:

- We present a secure searchable encryption scheme that allows multi-keyword query over an encrypted document corpus and retrieves the relevant documents ranked based on a similarity score.
- We construct the searchable encryption scheme that is CKA2-secure in the random oracle model[24, 5]. Our scheme achieves semantic security against *adaptive* adversaries that choose their search queries as a function of previously obtained trapdoors and search outcomes.
- We present a construction that achieves the *optimal* search time. Unlike many previous schemes that are glued to the linear search complexity, our search is sublinear to the total number of documents that contain the queried set of keywords. We perform a thorough experimental evaluation of our solution on a real-world dataset.

In Section 4.1 we briefly review preliminaries and cryptographic notations used in our solution. Section 4.2 describes the important details of building blocks that are used in constructing our scheme. In Section 4.3 we propose algorithm definitions, security model as well as details of the proposed searchable encryption scheme. The security analysis and perfomance comparison to other existing schemes is given in Section 4.4. Finally, Section 4.5 is devoted for the concluding remarks.

4.1. Notations

In this section we present the set of preliminaries and set of cryptographic notations.

4.1.1. Preliminaries. We first introduce the notations and preliminary concepts that will be used in the proposed searchable encryption scheme.

Let $D = (D_1, D_2, \ldots, D_n)$ be a set of documents and $K = (k_1, k_2, \ldots, k_m)$ be the dictionary consisting of unique keywords in all documents in D, where $\forall$ i $\in$ [1,m] $k_i \in \{0, 1\}^*$. $C = \{C_1, C_2, \ldots, C_n\}$ is an encrypted document collection stored in the cloud server. I_i is a searchable index associated with the corresponding encrypted document C_i. If A is an algorithm then $a \leftarrow A(\ldots)$ represents the result of applying the algorithm A to given arguments. Let R be an operational ring, we write vectors in bold, e.g. $\mathbf{v} \in R$. The notation $\mathbf{v}[\mathbf{i}]$ refers to the i-th element of $\mathbf{v}$. We denote the dot product of $\mathbf{u}$, $\mathbf{v} \in R$ as $\mathbf{u} \otimes \mathbf{v} = \sum_{i=1} \mathbf{u}[\mathbf{i}] \cdot \mathbf{v}[\mathbf{i}] \in R$. We use $\lfloor x \rceil$ to indicate rounding x to the nearest integer, and $\lfloor x \rfloor$, $\lceil x \rceil$ (for $x \geq 0$)to indicate rounding down or up.

4.1.2. Cryptographic Notations.

Definition 22. ***(Symmetric Key Encryption Scheme (SKE)).*** *A symmetric encryption scheme $SKE = (Gen, Enc, Dec)$ consists of three algorithms, as follows:*

- *Gen: the* key generation algorithm, *is a probabilistic algorithm that returns a string K. Let $Keys(SKE)$ denote the set of all strings that have non-zero probability of being output by Gen. The members of this set are called keys. We denote $K \leftarrow Gen$ for the operation of executing Gen and letting K denote the key returned.*
- *Enc: the* encryption algorithm, *is a probabilistic algorithm that takes a key $K \in Keys(SKE)$ and a plaintext $M \in \{0, 1\}^\star$. It returns a ciphertext $C \in \{0, 1\}^\star$. We write $C \leftarrow Enc(K, M)$ for the operation of executing Enc on K and M and letting C denote the ciphertext returned.*

- *Dec: the* decryption algorithm*, a deterministic algorithm that takes a key $K \in Keys(SKE)$ and a ciphertext $C \in \{0,1\}^\star$. It returns some $M \in \{0,1\}^\star$. We write $M \leftarrow Dec(K,C)$ for the operation of executing Dec on K and C and letting M denote the message returned.*

The SKE scheme is correct if for any key $K \in Keys(SKE)$, any sequence of messages $(M_1, \ldots, M_Q) \in \{0,1\}^\star$, and any sequence of ciphertexts $(C_1 \leftarrow Enc(K,M_1), \ldots, C_Q \leftarrow Enc(K,M_Q))$ that may arise in encrypting $(M_1, \ldots, M_Q)$, it is the case that $Dec(K,C_i) = M_i$ for each $C_i \neq \perp$ $(i \in [1;Q])$. A symmetric-key encryption scheme SKE is secure against chosen-plaintext attacks (CPA) (executed by the adversary with an access to encryption oracle) If produced ciphertexts do not leak any useful information about the original plaintexts[27].

We also make use of pseudo-random permutation (see Definition 9), which is a polynomial-time computable function that cannot be distinguished from random functions by any probabilistic polynomial-time adversary.

4.2. Background

4.2.1. Term Frequency-Inverse Document Frequency.

One of the main problems of information retrieval is to determine the relevance of documents with respect to the user information needs. The most commonly used technique to represent the relevance score in the information retrieval community is Term Frequency-Inverse Documents Frequency measure[51, 52, 31]. It is computed based on two independent measures - the Term Frequency and the Inverse Document Frequency. The Term Frequency (TF) is a statistical measure that represents the frequency of repeated terms in documents. The TF value calculates the number of times a given term appears within a single document. The Inverse Documents Frequency (IDF) is a measure of a term's importance across the whole document collection. It is defined as the logarithm of the ratio of the number of documents in a given collection to the number of documents containing a given term. The consequence of IDF definition is that rarely occurring terms have high IDF values and common terms have low IDF value. In judging the value of a term for ranking representation, two different statistical criteria come into consideration. A term appearing often in one single document is assumed to carry more importance for content representation than a rarely occurring term. On the other hand, if that same term occurs as frequently in other documents of the collection, the term is possibly not as valuable as some other terms that occur less frequently in the remaining documents. An example of this conundrum is illustrated by the occurrence of prepositions in English language documents. This suggests that the ranking of a given term as applied to a given document can be measured by a combination of its frequency of occurrence and an inverse function of the number of documents in the collection. Therefore, we use the product of term frequency and the inverse documents frequency (TF-IDF) for the ranking function in our search. A term with higher TF-IDF value is more relevant to the user's query than the term with lower TF-IDF value.

We adopt the following equation for TF-IDF measure from [31]:

$$WT_{i,j} = \log\left(f_{i,j} + 1\right) \times \log\left(1 + \frac{n}{\sum\limits_{k=1}^{n} \chi(f_{i,k})}\right) \tag{11}$$

where $f_{i,j}$ specifies the TF value of term j in the document D_i, n is the total number of documents in the corpus and $\frac{n}{\sum_{k=1}^{n} \chi(f_{i,k})}$ denotes the IDF value of term j among the entire collection D.

To provide the ranked results to user's queries we choose to use the dot product as similarity metric. We use vector space model[51], where the documents and search query are represented as high dimensional vectors. The similarity metric is measured by applying the dot product between each document vector and the search query vector as follows:

$$dotprod(D_i, Q) = D_i \otimes Q, \tag{12}$$

where D_i is a vector that represents i-th document and Q is a query vector.

4.2.2. Homomorphic Cryptosystem. We now review definitions related to homomorphic cryptosystem. Our definitions are based on Gentry's works [53] and [54], but we slightly relax the definition of decryption correctness, to allow a negligible probability of error.

Definition 23. ***(Homomorphic encryption (Hom)).*** *Let s denote the security parameter. Fix a function $l = l(s)$. An l-homomorphic encryption Hom for a class of circuits $\{\mathbb{C}_s\}_{s\in\mathbb{N}}$ consists of four polynomial-time algorithms Gen, Enc, Dec, and Eval such that:*

- *Gen: The* key generation algorithm*, is a probabilistic algorithm that takes the security parameter s as input and outputs a public key PK and secret key SK.*
- *Enc: The* encryption algorithm*, is a probabilistic algorithm that takes a public key PK and a message $m \in \{0, 1\}$ as input, and outputs a ciphertext c.*

- *Dec: The* decryption algorithm*, is a deterministic algorithm that takes the secret key SK and a ciphertext c as input, and outputs a message $m \in \{0, 1\}$.*
- *Eval: The* homomorphic evaluation algorithm*, takes as input a public key PK a circuit $C \in \mathbb{C}_s$, and a list of ciphertexts $c_1, \ldots, c_{l(s)}$, and outputs a ciphertext $c^\star$.*

The following correctness properties are required to hold:

- *For any s, any $m \in \{0, 1\}$, and any (PK, SK) output by $Gen(s)$, we have $m = Dec(SK, Enc(PK, m))$.*
- *For any s, any $m, \ldots, m_l$, and any $C \in \mathbb{C}_s$, we have*

$$C(m_1, \ldots, m_l) = Dec(SK, Eval(PK, (C, Enc(PK, m), \ldots, Enc(PK, m_l)))) \tag{13}$$

We use the standard notion of security against chosen-plaintext attacks (CPA-security).

Definition 24. ***(CPA security).*** *Let $Hom = Gen$, Enc, Dec, $Eval$ be a homomorphic encryption scheme. Let s be the security parameter, A be an adversary and there is a probabilistic experiment $CPA_{Hom,A}(s)$ that is executed between the challenger and the adversary:*

- *Generate $(PK, SK) \leftarrow Gen(s)$.*
- *The adversary outputs $(m_0, m_1, st_A) \leftarrow A_1^{(\cdot)}(PK)$.*
- *The bit is chosen at random, i.e $b \leftarrow \{0, 1\}$.*
- *The adversary runs number of polynomial queries $c \leftarrow Enc(PK, m_b)$.*
- *The adversary outputs bit $b' \leftarrow A_2^{(\cdot)}(c, st_A)$.*

Homomorphic encryption scheme Hom is CPA-secure if for all polynomial-size adversaries A,

$$Pr[CPA_{Hom,A}(s) = 1] \leq \frac{1}{2} + negl(s), \tag{14}$$

where the probability is over the choice of bit b and the coins of Gen and Enc.

There are many homomorphic encryption scheme available in the literature. However, in our solution we use Brakerski *et al.*[55, 56] homomorphic cryptosystem since it provides the efficient "batching mode" property where a single ciphertext represent a vector of encrypted values and single homomorphic operation on two such ciphertexts applies the homomorphic operation component-wise to the entire vector.

4.3. Basic Construction

Recall that we are targeting the following scenario: the data owner creates secure searchable index and sends it along with encrypted data files to the cloud server. The index is constructed in such a way that it provides enough information to perform the search on the outsourced data, but does not give away any information about the original data. Once the server receives the index and encrypted document files, it performs a search on the index and retrieves the most relevant documents according to data user's query.

4.3.1. Algorithm Definitions.

Definition 25. ***(Multi-keyword Similarity Searchable Encryption (MKSim)).*** *An index-based MKSim scheme over a set of documents D is a tuple of five polynomial-time algorithms (Gen, $BuildIndex$, $MakeQuery$, $Evaluate$, $Decrypt$), as follows:*

- $S_1,\ S_2,\ PK,\ SK \leftarrow Gen(1^s)$: *a probabilistic algorithm that is run by the data owner to setup the scheme. The algorithm outputs a set of secret keys* $S_1,\ S_2,\ SK$ *and public key* PK.
- $(I, C) \leftarrow BuildIndex(S_1, S_2, PK, D, K)$: *a probabilistic algorithm run by the data owner that takes as input a collection of documents* D, *a keyword dictionary* K *and keys* S_1, S_2 *and* PK, *and outputs a collection of encrypted documents* $C = \{C_1, C_2, \ldots, C_n\}$ *and searchable index* I.
- $\Omega \leftarrow MakeQuery(S_2, PK, K, \bar{K})$: *a probabilistic algorithm run by the data user that inputs keys* S_2 *and* PK, *keyword dictionary* K *and multiple keywords of interest* $\bar{K}$, *and outputs the search query* Ω.
- $L \leftarrow Evaluate(PK, I, \Omega)$: *a deterministic algorithm run by the cloud server. The algorithm inputs a public key* PK, *searchable index* I *and search query* Ω. *The algorithm outputs a sequence of identifiers* $L \subseteq C$ *matching the search query.*
- $D_i \leftarrow Decrypt(S_1, SK, C_i)$: *a deterministic algorithm that takes as input secret keys* S_1 *and* SK, *and a ciphertext* C_i *and outputs a document* D_i.

An index-based MKSim scheme is correct if $\forall\ s \in \mathbb{N}$, $\forall\ S_1,\ S_2,\ PK,\ SK$ *generated by* $Gen(1^s)$, $\forall\ D$, $\forall\ (I, C)$ *output by* $BuildIndex(S_1, S_2, PK, D, K)$, $\forall\ \bar{K} \in K$, *and* $1 \le i \le n$,

$$Evaluate(PK, I, MakeQuery(S_2, PK, K, \bar{K})) = L \bigwedge Decrypt(S_1, SK, C_i) = D_i \tag{15}$$

4.3.2. Security Model. In this section we focus on security definitions for our scheme. Security goal of any searchable encryption scheme is to reveal nothing (no meaningful information) to the adversary. Our goal is to provide two following security guarantees:

- Given a searchable index I and a set of encrypted document $C = (C_1, C_2, \ldots, C_n)$ to the adversary, no valuable information about the original documents $D = (D_1, D_2, \ldots, D_n)$ is leaked to the adversary.
- Given a set of search queries $Q = (Q_1, Q_2, \ldots, Q_m)$ generated by the data user, the adversary cannot learn any information about the content of the search query Q_i or the content of the documents in the search outcome.

To hide the plaintext document collection we require the symmetric key encryption scheme SKE (see Definition 22) to have the pseudo-randomness against chosen-plaintext attacks (PCPA) security guarantee which assures that the ciphertexts are indistinguishable from random[1]. We outline the PCPA-security of SKE scheme as following experiment:

Definition 26. ***(PCPA security).*** *Let $SKE = (Gen, Enc, Dec)$ be a symmetric key encryption scheme, s be the security parameter, A be an adversary and there is a probabilistic experiment $PCPA_{SKE,A}(s)$ that is run as follows:*

- *Output the secret key $S_1 \leftarrow Gen(1^s)$.*
- *The adversary A is given oracle access to $Enc(S_1, \cdot)$.*
- *The adversary A outputs a message M (e.g., plaintext document D).*
- *Let $C_0 \leftarrow Enc(S_1, M)$ and $C_1 \overset{R}{\leftarrow} \mathbb{C}$, where $\mathbb{C}$ denotes the set of all possible ciphertexts. A bit b is chosen at random and C_b is given to the adversary A.*
- *The adversary A is again given to the oracle access to $Enc(S_1, \cdot)$, and A runs number of polynomial queries to output a bit $b^{'}$.*
- *The experiment outputs 1 if $b = b^{'}$, otherwise 0.*

[1]Note that symmetric key encryption schemes such as AES in counter mode satisfy the PCPA-security definition.

Symmetric key encryption scheme SKE is PCPA-secure if for all polynomial-size adversaries A,

$$Pr[PCPA_{SKE,A}(s) = 1] \leq \frac{1}{2} + negl(s), \tag{16}$$

where the probability is over the choice of bit b and the coins of Gen and Enc.

Achieving the second security property is difficult and and most known searchable encryption solutions [3, 57, 5, 2, 48] reveal some information from the search queries, namely *access pattern* and *search pattern.* In MKSim scheme we follow the similar approach to weaken the security guarantees and allow some limited information to the adversary. We begin with definitions of the *history*, *access pattern* and *search pattern*, and then we give the definition of *trace* that combines definitions of history, access and search patterns.

Definition 27. ***(History).*** *Let $\bar{K}$ be a collection of keywords of interest, D be a collection of documents and $\Omega = \{\Omega_1, \Omega_2, \ldots, \Omega_q\}$ be a vector of q search queries. The history of search queries is defined as $H(D, \Omega)$.*

Definition 28. ***(Access Pattern).*** *Let $\bar{K}$ be a collection of keywords of interest, D be a collection of documents and $\Omega = \{\Omega_1, \Omega_2, \ldots, \Omega_q\}$ be a vector of q search queries. The access pattern induced by a q-query history $H(D, \Omega)$, is defined as follows: $\alpha(H) = (D(\Omega_1), D(\Omega_2), \ldots, D(\Omega_q))$, where $D(\Omega_i)$ denotes the set of documents that match search query Ω_i.*

Definition 29. ***(Search Pattern).*** *Let $\bar{K}$ be a collection of keywords of interest, D be a collection of documents and $\Omega = \{\Omega_1, \Omega_2, \ldots, \Omega_q\}$ be a vector of q search queries, The search pattern of the history $H(D, \Omega)$ is a symmetric binary matrix $\sigma(H)$ such that for $1 \leq i, j \leq q$, the element in the i^{th} row and j^{th} column is 1 if $\Omega_i = \Omega_j$, and 0 otherwise.*

The access pattern represents the results of search queries Q sent to the cloud server, and specifically, the document identifiers corresponding to each query. The search pattern represents a historical observation of queries searched for. We combine both access pattern and search pattern to form the definition of *trace*, as follows:

Definition 30. ***(Trace).*** *Let $\bar{K}$ be a collection of keywords of interest, D be a collection of documents and $\Omega = \{\Omega_1, \Omega_2, \ldots, \Omega_q\}$ be a vector of q search queries. The trace induced by the history $H(D, \Omega)$ is a sequence $\tau(H) = (|D_1|, |D_2|, \ldots, |D_n|, \alpha(H), \sigma(H))$ comprised of the lengths of document collection $D = (D_1, D_2, \ldots, D_n)$, and the access and search patterns induced by the history $H(D, \Omega)$.*

Unlike security definitions originally presented in [5], our searchable encryption scheme introduces a randomization of a search query that enables the data user to hide the search pattern. Thus, queries are different even if they are generated for the same set of keywords of interest. The definition of randomized queries is given as:

Definition 31. ***(Randomized query).*** *Let $\Omega_{1 \leq i \leq q}$ be a sequence of q generated search queries with the same set of keywords $\bar{K}$. We say that the scheme has (q, ϵ)-randomized query if: $\forall i, j \in 1, q \;\; Pr(\Omega_i = \Omega_j) < \epsilon$.*

Our security model uses the simulator-based definition approach that includes the view from adversary and the simulator. Adaptive security means that the adversary generates the history adaptively, that means that the results of previous search queries are taken into account. We outline simulation-based experiments in the following definition:

Definition 32. ***(Adaptive Semantic Security).*** *Let $MKSim = (Gen, BuildIndex, MakeQuery, Evaluate, Decrypt)$ be an index-based similarity searchable encryption scheme, s be the security parameter, and $A = (A_0, \ldots, A_q)$ be an adversary such that $q \in \mathbb{N}$ and $\mathbb{S} =$*

$(\mathbb{S}_0, \ldots, \mathbb{S}_q)$ *be a simulator. Consider the following probabilistic experiments* $Real^{\star}_{MKSim,A}(s)$ *that is executed between the challenger and the adversary, and* $Sim^{\star}_{MKSim,A,\mathbb{S}}(s)$ *that is executed between the adversary and the simulator:*

$Real^{\star}_{MKsim,A}(s)$:	$Sim^{\star}_{MKSim,A,\mathbb{S}}(s)$:
$(D, K, st_A) \leftarrow A_0(1^s)$	$(D, K, st_A) \leftarrow A_0(1^s)$
$(S_1,\ S_2,\ PK,\ SK) \leftarrow Gen(1^s)$	$(I, C, st_{\mathbb{S}}) \leftarrow \mathbb{S}_0(\tau(D, K))$
$(I, C) \leftarrow BuildIndex(S_1, S_2, PK, D, K)$	$(\bar{K}_1, st_A) \leftarrow A_1(st_A, I, C)$
$(\bar{K}_1,\ st_A) \leftarrow A_1(st_A, I, C)$	$(\Omega_1, st_{\mathbb{S}}) \leftarrow \mathbb{S}_1(st_{\mathbb{S}}, \tau(D, \bar{K}_1))$
$\Omega_1 \leftarrow MakeQuery(S_2, PK, K, \bar{K}_1)$	*for* $2 \leq i \leq q$
for $2 \leq i \leq q$	$(\bar{K}_i, st_A) \leftarrow A_i(st_A, I, C, \Omega_1, \ldots, \Omega_{i-1})$
$(\bar{K}_i, st_A) \leftarrow A_i(st_A, I, C, \Omega_1, \ldots, \Omega_{i-1})$	$(\Omega_i, st_{\mathbb{S}}) \leftarrow \mathbb{S}_i(st_{\mathbb{S}}, \tau(D, \bar{K}_1, \ldots, \bar{K}_i))$
$\Omega_i \leftarrow MakeQuery(S_2, PK, K, \bar{K}_i)$	*let* $\Omega = \{\Omega_1,\ \Omega_2,\ \ldots,\ \Omega_q\}$
let $\Omega = \{\Omega_1,\ \Omega_2,\ \ldots,\ \Omega_q\}$	*output* $o = (I, C, \Omega)$ *and* st_A
output $o = (I, C, \Omega)$ *and* st_A	

where st_A *is the state of adversary,* $st_{\mathbb{S}}$ *is the state of the simulator. We say that MKSim is adaptively semantically secure if for all polynomial-size adversaries* $A = (A_0,\ \ldots,\ A_q)$ *such that* q *is polynomial in* s*, there exists a non-uniform polynomial-size simulator* $\mathbb{S} = (\mathbb{S}_0,\ \ldots,\ \mathbb{S}_q)$ *such that for all polynomial-size* $\mathbb{R}$*:*

$$|Pr[\mathbb{R}(o, st_A) = 1] - Pr[\mathbb{R}(\bar{o}, \bar{st_A})]| \leq \epsilon, \tag{17}$$

where $(o, st_A) \leftarrow Real^{\star}_{MKSim,A}(s)$, $(\bar{o}, \bar{st_A}) \leftarrow Sim^{\star}_{MKSim,A,\mathbb{S}}(s)$ *and the probabilities are over the coins of Gen and* $BuildIndex$ *and* $MakeQuery$.

```
$\underline{Gen(1^s):}$ generate $S_1 \leftarrow SKE.Gen(1^s)$ and $S_2 \overset{R}{\leftarrow} \{0,1\}^s$. Sample $SK$, $PK \leftarrow$
$Hom.KeyGen()$. Output $S_1$, $S_2$, $SK$ and $PK$.
$\underline{BuildIndex(S_1, S_2, PK, D, K):}$
Initialization:
    (1) scan document corpus $D$, extract $k_1$, $k_2$, ..., $k_p \leftarrow$ from $D_i$.
    (2) construct dictionary $K$ with dummy $Z$.
    (3) for each $k \in K$, build $D(k)$(i.e., the sequence of documents with $k$).
Build lookup filter $T$:
    (1) for each $k_i \in K$:
        • for $1 \leq j \leq |D(k_i)|$:
            – value = id($D_{i,j}$), where id($D_{i,j}$) is the $j^{th}$ identifier in $D(k_i)$.
            – set $T[\pi_{S_2}(k_i||j)]$ = value.
    (2) let $\bar{p} = \sum_{k_i \in K} |D(k_i)|$. If $\bar{p}<p$, assign $value = id(D)$ for all $D_i \in D$ for exactly $max$
        entries, set the $address$ to random values.
Build TF-IDF table $\Phi$:
    (1) for each $D_i \in D$
        • for each $k_j \in K$
            – $WT_{i,j} \leftarrow$ TFIDF($k_j$) (i.e., calculate TF-IDF value for keyword $k_j$).
        • set $\boldsymbol{\Phi_i} = Hom.Enc(PK, \mathbf{WT_i})$ (i.e., apply homomorphic encryption $Hom$ with
          public key $PK$ on vector $WT_i$).
Output:
    (1) for each $D_i \in D$, let $C_i \leftarrow SKE.Enc(S_1, D_i)$.
    (2) output $(I, C)$, where $I = (T, \Phi)$ and $C = \{C_1, C_2, \ldots, C_n\}$.
```

FIGURE 4.1. MKSim Scheme Setup Phase.

This completes our security model requirement for MKSim solution, and we are now ready to present scheme details.

4.3.3. SCHEME CONSTRUCTION. **Setup Phase.** Our searchable scheme is based on SSE-2 inverted index data construction previously introduced in [5]. We enhance SSE-2 scheme with addition of TF-IDF statistical measurement and dot product for ranked search. We show that our construction is very efficient and it achieves the same semantic security guarantees as SSE-2 scheme. Fig. 4.1 shows an outline of MKSim setup phase.

Our searchable index consists of two main algorithms: building the lookup filter T, based on SSE-2 construction and building the TF-IDF table Φ, based on TF-IDF word importance. We create a lookup filter T whose entries have of the form $(keyword, value)$. For each keyword $k \in K$ we add an entry in T whose $value$ is the document identifier with the instance of keyword k. Note, for a given keyword k and the set of documents that contains the keyword k, we derive a label $k||j$ for keyword k with j-th document identifier. For example, if "colorado" is contained in document D_5, then $k||j$ is "colorado1". Formally, we represent the family of k with matching j-th documents as follows: $F_k = \{k||j : 1 \leq j \leq |D(k)|\}$, where $|D(k)|$ represents the list of matching documents. For instance, if "state" is contained in set of four document $(D_2, D_4, D_{10}, D_{13})$, then family F_k is {"state1", "state2", "state3", "state4"} and we add the following entries in lookup table T: $(state1, 2)$, $(state2, 4)$, $(state3, 10)$ and $(state4, 13)$. In our construction, searching for keyword k becomes equal to searching for all labels in a form of $k||j$ in the family F_k.

We guard the unique number of words in each document by adopting the idea of padding the lookup filter T such that the identifier of each document appears in the same number of entries. To protect the keyword content in the table T, we use the pseudo-random permutation π with secret parameter S_2 such as $\{0,1\}^{S_2} \times \{0,1\}^{l+\log_2(n+max)} \rightarrow \{0,1\}^{l+\log_2(n+max)}$, where max denote the maximum number of distinct keywords in the largest document in D, n is the number of documents in D and each keyword is represented using at most l bits. Our lookup table T is $(\{0,1\}^{l+\log_2(n+max)} \times \{0,1\}^{\log_2(n)} \times \{p\})$, where $p = max \star n$.

In our second step, for each distinct keyword k_j in a document D_i, we calculate the TF-IDF measure using Equation 11. We then construct the TF-IDF table Φ where each row corresponds to the document identifier D_i and each column is the keyword in the dictionary K. Each cell element in Φ contains the TF-IDF value of a keyword k_j. Unfortunately, outsourcing the table elements to the cloud leaks some important information. It is well known fact[36] that an adversary (in our case, the cloud server) may know some of keywords and their TF distributions. Using this information, an adversary can infer the keyword index or even the document content. Based on this observation, we decided to improve the security of our solution. Our table includes the set of dummy keywords Z that are added to the keyword dictionary K. This gives us the randomness that hides the original keyword distribution of TF-IDF values. Finally, we use the CPA-secure homomorphic cryptosystem Hom to protect the values of TF-IDF table Φ. We apply $Hom.Enc()$ with public key PK on each row of TF-IDF table Φ.

Once both the lookup filter T and TF-IDF table Φ are constructed, we use PCPA-secure symmetric key encryption scheme SKE with secret key S_1 to encrypt each document D_i, i.e. $C_i = SKE.Enc(S_1, D_i)$. We outsource searchable index $I = (T, \Phi)$ and encrypted collection $C = \{C_1, C_2, \ldots, C_n\}$ to the cloud server. Now the collection is available for selective retrieval from the cloud server.

Search Phase. We outline the search phase in Figure 4.2. To search the keywords of interest $\bar{K}$, the data user contacts the data owner to receive the *search trapdoor*. The trapdoor includes the keyword dictionary K with IDF values for each keyword in K, the set of (S_1, S_2, PK, SK) keys, the pseudo-random permutation π, homomorphic cryptosystem Hom and symmetric-key encryption scheme SKE. Note, the trapdoor learning process is a one-time operation and the data user does not need to contact the data owner anymore. The data owner inputs the set of keywords of interest $\bar{K}$ to the search trapdoor. The trapdoor generates a search query in form of $\Omega = (t, x)$, where t corresponds to the lookup search query and x corresponds to the TF-IDF search query. The lookup search query t is the output of the pseudo-random permutation π with secret key S_2. The TF-IDF search query

```
$\underline{MakeQuery(S_2, PK, K, \bar{K})}$ :
    (1) for each keyword $k_j \in \bar{K}$, set $t_j = (\pi_{S_2}(k_j||1), \ldots, \pi_{S_1}(k_j||n))$.
    (2) for each keyword $k_j \in \bar{K}$, calculate $WT_j \leftarrow \text{TFIDF}(k_j)$.
    (3) set $\mathbf{x} = Hom.Enc(PK, \mathbf{WT})$ (i.e., encrypt vector $\mathbf{WT}$ using homomorphic encryption $Hom$ with public key $PK$).
    (4) output search query $\Omega = (t, \mathbf{x})$.
$\underline{Evaluate(PK, I, \Omega)}$ :
    (1) Find all matching identifiers $id \leftarrow T[t]$.
    (2) for all $j \in [1; |id|]$, calculate $score_{C_j} = Hom.Eval(PK, (\mathbf{\Phi}_{C_j}, \mathbf{x}))$. (i.e. with public key $PK$, execute homomorphic dot product (multiplication and addition) between $\mathbf{\Phi}_{\boldsymbol{C_j}}$ and $\mathbf{x}$).
    (3) output results $\{score_{C_1}, \ldots, score_{C_{id}}\}$.
$\underline{Decrypt(S_1, SK, C_i)}$ :
    (1) set $score_{D_i} = Hom.Dec(SK, score_{C_i})$ (i.e., decrypt $score_{C_i}$ using $Hom$ with secret key $SK$).
    (2) select $(score_{D_i})_m \xleftarrow{\text{top-m}} \{score_{D_1}, \ldots, score_{D_{id}}\}$ (i.e. select top-$m$ documents with highest scores).
    (3) output $(D_i)_m \leftarrow SKE.Dec(S_1, C_i)$, where $i \in [1; m]$.
```

FIGURE 4.2. MKSim Scheme Search Phase.

x is the result of applying the homomorphic encryption $Hom.Enc$ with public key PK on the output of TF-IDF measure calculated using Equation 11 on set of keywords of interest $\bar{K}$. Finally, the data user sends resulted search query $\Omega = (t, x)$ to the cloud server.

Once Ω received, the server locates the matching document identifiers in the lookup filter T as $T[t]$. Now the cloud server executes the homomorphic dot product $Hom.Eval$ (that includes homomorphic multiplication and homomorphic addition) between the TF-IDF search query x and rows in the TF-IDF table Φ that correspond to the matching document identifiers. The cloud server sends the set of resulted ciphertexts $\{score_{C_1}, \ldots, score_{C_{id}}\}$ back to the data user.

The data user decrypts each polynomial $score_{C_i}$ using $Hom.Dec$ with secret key SK to form $score_{D_i}$. The data user retrieves the top-m documents with highest similarity scores from the cloud server. Using $SKE.Dec$ with secret key S_1, the data user decrypts matching documents to the original view.

4.4. Evaluation

4.4.1. Security Analysis.

We now focus on the security evaluation of proposed searchable encryption scheme. We focus on two security properties:

- We prove that our scheme is CKA2 secure[24, 5] and achieves the strongest semantic security against adaptive adversaries.
- We demonstrate that inserting dummy keywords provides randomized search queries. It is important to show that the search queries generated by our scheme are different for the same set of keywords of interest, making the adversary difficult to collect and distinguish the document outcomes.

Theorem 1. If π is a pseudo-random permutation, Hom is CPA-secure and SKE is PCPA-secure, then $MKSim = (Gen, BuildIndex, MakeQuery, Evaluate)$ scheme is adaptively secure.

Proof. We are going to describe a polynomial-size simulator $\mathbb{S} = \{\mathbb{S}_0, \ldots, \mathbb{S}_q\}$ such that for all polynomial-size adversaries $A = \{A_0, \ldots, A_q\}$, the outputs of $Real^{\star}_{MKSim,A}(s)$ and $Sim^{\star}_{MKSim,A,\mathbb{S}}(s)$ are computationally indistinguishable. Consider the simulator $\mathbb{S} = \{\mathbb{S}_0, \ldots, \mathbb{S}_q\}$ that adaptively generates the output $o^{\star} = (I^{\star}, C^{\star}, \Omega^{\star})$ as follows:

- $\mathbb{S}_0(1^s, \tau(D))$: the simulator has a knowledge of history H that includes the number and the size of the documents. $\mathbb{S}_0$ starts with generating $C_i^{\star} \overset{R}{\leftarrow} \{0,1\}^{|D_i|}$ where $i \in [1, n]$ and searchable index $I^{\star} = (T^{\star}, \Phi^{\star})$. Here $T^{\star} \overset{R}{\leftarrow} \{0,1\}^{l+\log_2(n+max)}$ is a lookup filter, and $\Phi^{\star} \overset{R}{\leftarrow} \{0,1\}^{K\times n}$ is a TF-IDF table. $\mathbb{S}_0$ now includes $I^{\star}$ in $st_{\mathbb{S}}$ and outputs

$(I^\star, C^\star, st_{\mathbb{S}})$. Since, with all but negligible probability, st_A does not include secret S_2, $T^\star$ is indistinguishable from the real lookup table T. Otherwise $\mathbb{S}_0$ can distinguish between the output of pseudo-random permutation π and a random values of size $l + \log_2(n + max)$. Similarly, simulated TF-IDF table $\Phi^\star$ is indistinguishable from the real Φ due to the CPA security of homomorphic encryption Hom, otherwise one could distinguish between the output of $Hom.Enc$ and a random string of size $K \times n$. At the same time, st_A does not include secret S_1, thus the PCPA security of SKE scheme will guarantee that the output $C^\star$ is indistinguishable from the real ciphertext.

- $\mathbb{S}_1(st_{\mathbb{S}}, \tau(D, \Omega_1))$: now the simulator $\mathbb{S}_1$ has a knowledge of all document identifiers corresponding to the search query Ω_1. However the search query does not disclose its structure and the content. Recall that $D(\Omega_1)$ is the set of all matching document identifiers. For all $1 \le j \le |D(\Omega_1)|$, the simulator first makes an association between each document identifier $id(D_j)$ and a generated search query such that $(D(\Omega_1)_i, I_i^\star)$ are pairwise distinct. $\mathbb{S}_1$ then creates $\Omega_1^\star = (t^\star, x^\star)$, where $t^\star \overset{R}{\leftarrow} (id(D_1), \ldots, id(|D(\Omega_1)|))$ and $x^\star \overset{R}{\leftarrow} \{0,1\}^K$. $\mathbb{S}_1$ stores the association between $\Omega_1^\star$ and Ω_1 in $st_{\mathbb{S}}$, and outputs $(\Omega_1^\star, st_{\mathbb{S}})$. Since, with all but negligible probability, st_A does not include secret S_2, the output $t_1^\star$ is indistinguishable from the real generated query t_1 otherwise one could distinguish between the output of π and a random string of size $l + \log_2(n + max)$. Similarly, simulated $x^\star$ is indistinguishable from the real x due to the CPA security of homomorphic encryption Hom, otherwise one could distinguish between the output of $Hom.Enc()$ and a random string of a size K. Thus, $\Omega_1^\star$ is indistinguishable from Ω_1.

- $\mathbb{S}_i(st_{\mathbb{S}}, \tau(D, \Omega_1, \Omega_2, \dots, \Omega_q))$ for $2 \leq i \leq q$: first $\mathbb{S}_i$ checks if the search query Ω_i was executed before, that is, if it appeared in the trace $\sigma[i,j] = 1$, where $1 \leq j \leq i-1$. If $\sigma[i,j] = 0$, the search query has not appeared before and $\mathbb{S}_i$ generates the search query as $\mathbb{S}_1$. If $\sigma[i,j] = 1$, then $\mathbb{S}_i$ retrieves previously searched query, and constructs $\Omega_i^\star$. $\mathbb{S}_i$ outputs $(\Omega_i^\star, st_{\mathbb{S}})$, where $\Omega_i^\star$ is indistinguishable from real Ω_i. The final output $\Omega = (\Omega_1^\star, \dots, \Omega_q^\star)$ is indistinguishable from generated query $(\Omega = \Omega_1, \dots, \Omega_q)$, and outputs of experiments $Sim^\star_{MKSim,A,\mathbb{S}}(s)$ and $Real^\star_{MKSim,A}(s)$ are indistinguishable.

□

Theorem 2. Injection of dummy keywords provides randomized search queries.

Proof. Let us consider two search queries Ω_1 and Ω_2, both constructed from the same keyword set K and a randomly chosen set of dummy keywords Z_1 and Z_2 from a dictionary $\mathcal{Z}$ of a size $n = |\mathcal{Z}|$. We are aiming to prove that: $\forall i,j \;\; i \neq j \;\; Pr(\Omega_i = \Omega_j) < \epsilon$ where ϵ tends to zero as n increases.

We first estimate the probability Pr(k) that the intersection $\mathbf{Z}$ of two sets $\mathrm{Z}_1, \mathrm{Z}_2 \subseteq \mathcal{Z}$ is equal to some value k. We have:

$$Pr(k) = \frac{\#of\ ways\ of\ choosing\ Z_1, Z_2\ with\ |\mathbf{Z}| = k}{\#\ of\ ways\ of\ choosing\ Z_1, Z_2} \tag{18}$$

Note, there are $\binom{n}{k}$ choices for $\mathbf{Z}$. If Z_1 has size k, then there is one choice for Z_1, and we can choose Z_2 arbitrarily from 2^{n-k} possibilities. If Z_1 has size $\mathrm{k}+1$, then there are $\binom{n-k}{1}$ choices for Z_1 and 2^{n-k-1} choices for Z_2. Let $\mathrm{m} = \mathrm{n} - \mathrm{k}$, then there are $\sum_{j=0}^{m} 2^{m-j}\binom{m}{j} = 3^m$ possible choices for $\mathrm{Z}_1, \mathrm{Z}_2$ with intersection $\mathbf{Z}$. Thus, there are $3^{n-k}\binom{n}{k}$ possible ways of choosing the subsets. Since there are 4^n ways of choosing any two subsets of $\mathcal{Z}$, we have the following:

TABLE 4.1. Comparison of several searchable encryption schemes. n is size of the document collection D, m is the size of the keyword space K, δ is the number of documents containing keywords of interest $\bar{K}$.

Scheme	Matching	Query Randomization	Security	Search Time	Index Size
Song *et al.* [1]	Exact	no	CPA	O(n)	N/A
Goh *et al.* [2]	Exact	no	CKA1	O(n)	O(n)
Curtmola *et al.*[5] (SSE-1)	Exact	no	CKA1	O(δ)	O(n+m)
Curtmola *et al.*[5] (SSE-2)	Exact	no	CKA2	O(δ)	O(nm)
Cao *et al.* [9]	Similarity	yes	CKA1	O(n)	O(nm)
Moataz *et al.* [3]	Exact	yes	CKA2	O(n)	O(nm)
Orencik *et al.* [4]	Exact	no	CKA2	O(n)	O(nm)
MKSim	**Similarity**	**yes**	**CKA2**	**O(δ)**	**O(nm)**

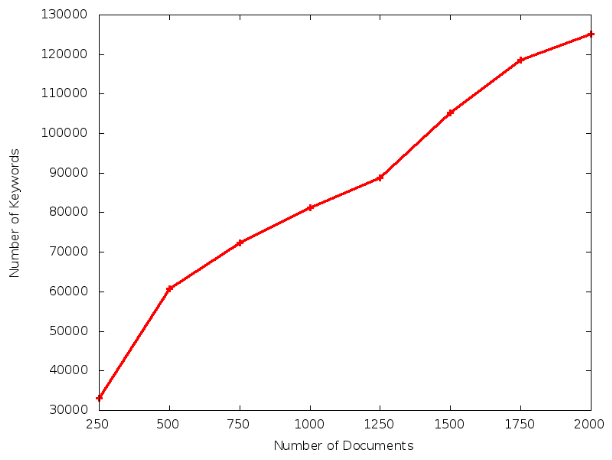
$Pr(k) = \frac{3^{n-k}\binom{n}{k}}{4^n}$. We now evaluate Pr(k) with input $k \to 0$: $\lim_{k\to 0} Pr(k) = \lim_{k\to 0} \frac{3^{n-k}\binom{n}{k}}{4^n}$ $= \left(\frac{3}{4}\right)^n$. As n increases, $lim_{k\to 0}Pr(k) \to 0$ and hence Theorem 2 is preserved . $\square$

4.4.2. PERFORMANCE EVALUATION. We compare $MKSim$ scheme with previous searchable encryption solutions in Table 4.1. Our comparison is based on the following metrics: matching technique, query randomization, security, search time and index size. Matching technique describes if solution supports *exact* match or *similarity* match. Query randomization describes the support of randomized search queries. We use security notations from [5] to describe the security of each solution. Search time is the time to invoke search on the index that is constructed from the document collection D of size n. Note that previous solutions are able to achieve the linear search complexity within the total number of documents in the collection. In contrast, the search in our solution is proportional to the number of documents that contain a set of keywords of interest. Finally, we measure the size of the searchable index. Our searchable index I consists of a lookup filter T of size $O(nm)$ (where n is the size of document collection and m is the size of keyword dictionary K) and a TF-IDF table Φ of size $O(nm)$, which makes the total ciphertext having the size of $O(nm)$.

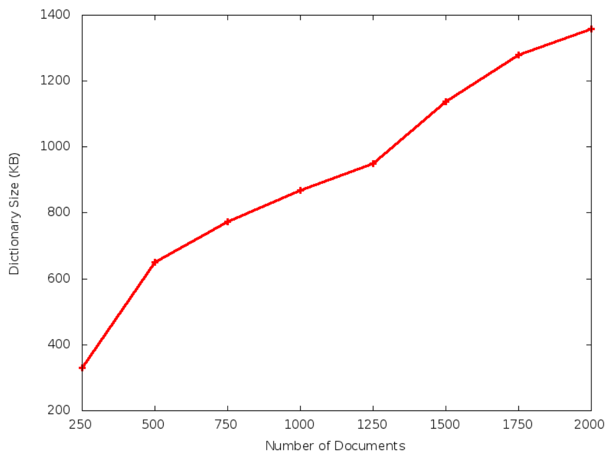
We have developed and implemented a multi-threaded proof-of-concept prototype of $MKSim$ scheme using C++ language. Our implementation includes 37 classes with a total of 10200 lines of code. Our prototype leverages two cryptographic libraries: *libtomcrypt* [2] and *HElib* [3]. *Libtomcrypt* is portable C cryptographic library that supports symmetric ciphers, one-way hashes, pseudo-random number generators, and a plethora of support routines. We use *libtomcrypt* to build the lookup filter T and encrypt the document collection. *HElib* is a C++ cryptographic library for homomorphic encryption available as an implementation of Brakerski *et al.* [58] scheme. We utilize *HElib* homomorphic cryptosystem since its one the few homomorphic toolkits that efficiently supports multiplication operation on the ciphertexts. Specifically, it provides the efficient "batching mode" property where a single ciphertext represents a vector of encrypted values. Thus, single homomorphic operation (i.e., multiplication) on two such ciphertexts applies the homomorphic operation component-wise to the entire vector. We use *HElib* to build the TF-IDF table Φ and compute the similarity measure of a search query and set of stored encrypted documents.

We show a thorough experimental evaluation of the MKSim scheme on a real-world dataset: the Internet Request for Comments (RFC) database [59], which is a collection of plaintext documents that consists of a large number of technical keywords describing different specifications, protocols, procedures and events in the Internet. All experiments have been performed on a 6 core Intel(R) Xeon(R) E5645 @ 2.40GHz processor and 98 GB memory running 64-bit Fedora 21 distro with the 3.19 kernel. We setup our prototype to use 20 pthread[60] threads in all experiments. The cloud server, data owner and data user applications were run on the same machine, as the network communication overhead was

[2]https://github.com/libtom/libtomcrypt
[3]https://github.com/shaih/HElib



(A)



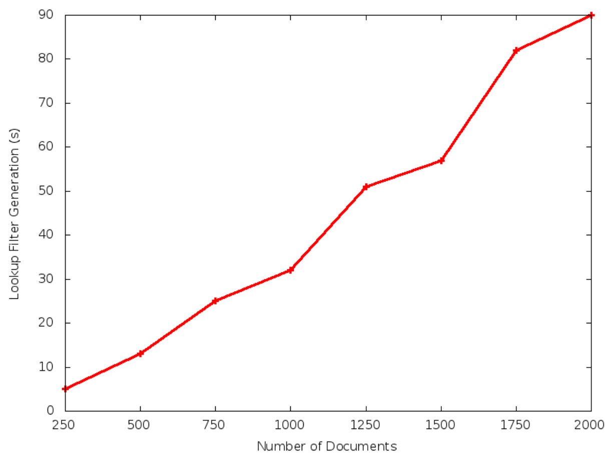
(B)

FIGURE 4.3. (**a**) RFC Keyword Dictionary. (**b**) RFC Keyword Dictionary Size.

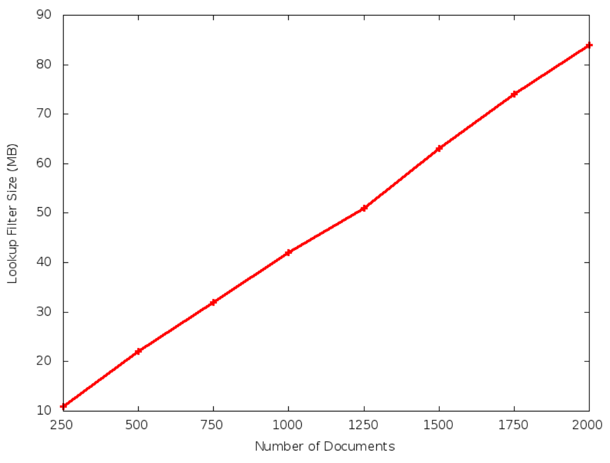
assumed to be negligible. In our evaluation we measure the computation and communication overheads of proposed solution. Specifically, we measure the overheads of the $BuildIndex$, $MakeQuery$ and $Evaluate$ algorithms presented in n Section 4.3.1.

Encryption Overhead. In this section, the overheads associated with the encryption process will be detailed. First, we study the size of keyword dictionary $K = (k_1, k_2, \ldots, k_m)$ extracted from RFC documents $D = (D_1, D_2, \ldots, D_n)$ of different sizes. Figure 4.3a shows the number of distinct keywords extracted from 250 to 2000 RFC documents. Figure 4.3b shows that the storage overhead of keyword dictionary is nearly linear with the size of the dataset.

Second, we study computation and communication overheads for $BuildIndex$ algorithm. Our building index algorithm consists of two phases: constructing lookup filter T and TF-IDF table Φ. Figure 4.4a shows the time cost of constructing the lookup filter T using RFC documents of different sizes. The lookup filter T consists of applying a pseudo-random permutation on each keyword from the dictionary and labeling the output with document



(A)



(B)

FIGURE 4.4. (**a**) Lookup Filter T Generation. (**b**) Lookup Filter T Size.

TABLE 4.2. Keyword Search Queries.

Search Query 1 (SQ-1)	OSPF BGP MIME XML iSCSI
Search Query 2 (SQ-2)	Encryption Framework Certificate Authorization Authentication

identifier where the keyword appeared. Figure 4.4b shows the storage requirements of lookup filter T.

Next, we measure the overhead related to TF-IDF table Φ. Our TF-IDF table Φ construction consists of computing the TFIDF table for each extracted keyword from RFC dataset and applying Brakerski's homomorphic cryptosystem in "batching mode" on each document row. We show the time cost of generating the TF-IDF table Φ in Figure 4.5a and the storage overhead in Figure 4.5b. Although the time of building TF-IDF table Φ is not a negligible overhead for the data owner, this is a one-time operation before sending the encrypted documents to the cloud server.

Search Overhead. To evaluate the search overhead, we leverage two types of search inputs in our prototype. The first search query (SQ-1) describes five popular acronyms in the RFC dataset, while the second search query (SQ-2) is a set of general keywords describing

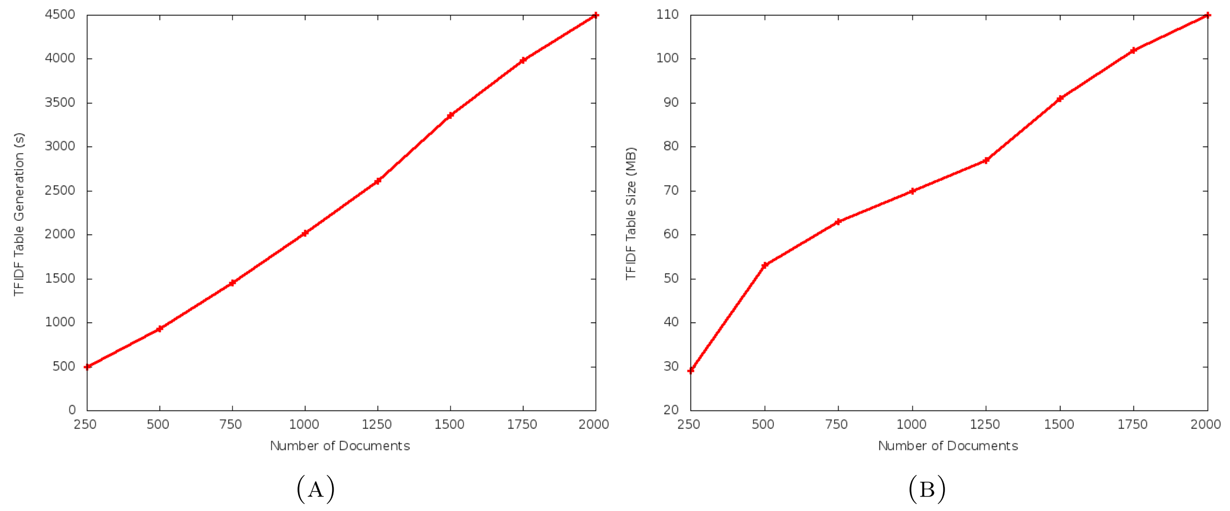


FIGURE 4.5. (**a**) TF-IDF Table Φ Generation. (**b**) TF-IDF Table Φ Size.

technical security standards for the Internet. Table 4.2 shows the keywords used in our experiments.

First, we measure the performance of the *MakeQuery* algorithm. This overhead of *MakeQuery* consists of calculating a pseudo-random permutation applied on each queried keyword and calculating Brakerski's homomorphic encryption on each weighted keyword. Figure 4.6a shows the time cost for generating trapdoors for both SQ-1 and SQ-2 using keyword dictionaries of different sizes. Our results demonstrate that time cost to construct both SQ-1 and SQ-2 search queries stays mostly constant.

Second, we test the overhead of the *Evaluate* algorithm. The *Evaluate* execution at the cloud server consists of calculating similarity scores (i.e., homomorphic dot product) between a given search query and stored searchable index I. Table 4.3 shows the experimental results for RFC documents of different sizes. For each search input we measure the number of documents that match the lookup filter T and the time cost of calculating similarity scores for a set of selected documents. For example, for SQ-1 search input and RFC collection of 1500 documents, there are 472 documents that match the lookup filter T and it takes a total of

TABLE 4.3. Search Query 1 and Search Query 2 Execution Results.

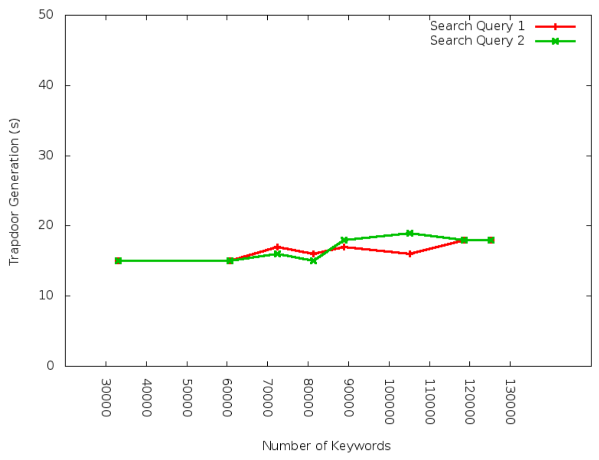
	Search Query 1		Search Query 2	
Number of Documents	Number of Matching Documents in T	Execution Time in Φ (s)	Number of Matching Documents in T	Execution Time in Φ (s)
250	85	217	152	341
500	173	454	307	738
750	255	680	462	1169
1000	324	873	608	1618
1250	400	1107	753	2068
1500	472	1427	919	2738
1750	562	1807	1071	3429
2000	626	2089	1220	3989

TABLE 4.4. Different Size Query Execution Results.

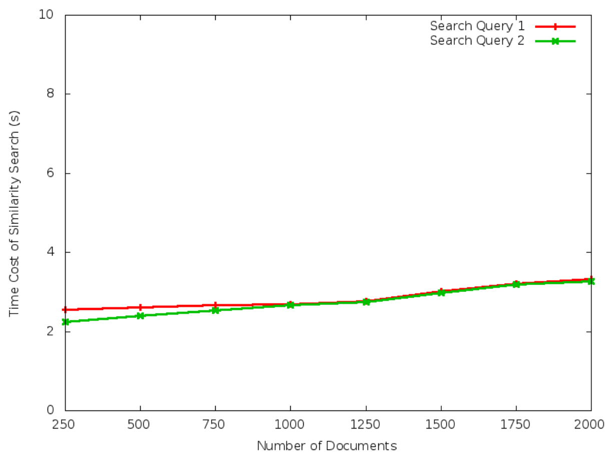
	1000 Documents		2000 Documents	
Search Query Size	Number of Matching Documents in T	Execution Time in Φ (s)	Number of Matching Documents in T	Execution Time in Φ (s)
5	11	57	123	518
10	222	643	1069	3634
25	286	807	1080	3694
50	340	924	1131	3867
100	571	1537	1172	3998
250	916	2424	1418	4880
500	959	2543	1892	6530
1000	988	2605	1950	6609

1427 seconds to measure similarity scores of selected 472 documents. Our results demonstrate that lookup filter T allows the cloud server to efficiently select the set of documents that contain the queried set of keywords without the need of calculating similarity scores for all documents in the dataset.

Third, using the results from Table 4.3 we measure the time cost to calculate the similarity score for each selected document. We define the Similarity Search Performance (SSP) metric as follows: $SSP(Q) = \frac{t_\Phi}{n_{\bar{K}}}$, where t_Φ corresponds to total time cost to calculate similarity scores for selected documents in Φ and $n_{\bar{K}}$ corresponds to the number of documents that match the search query in T. Figure 4.6b shows the SSP metric for both SQ-1 and SQ-2



(A)



(B)

FIGURE 4.6. (**a**) Time Cost of Generating Trapdoor. (**b**) Cloud Similarity Search Time.

search queries. Our results demonstrate that the cloud server has an average overhead of 2.8 seconds on each similarity score calculation. Also, Table 4.4 shows the experimental results for search queries of different sizes. We use two collections of sizes 1000 and 2000 documents and varied the size of search query. The results show that our search is proportional to the number of documents that contain a set of keywords of interest. In short, we notice that the MKSim protocol, although uses homomorphic encryption, adds minimal overhead on computation and storage. We believe that proposed solution can be easily deployed in a real-world cloud environment.

4.5. Conclusion

Searchable encryption is a technique that enables secure searches over encrypted data stored on remote servers. We define and solve the problem of multi-keyword similarity search over encrypted cloud data. In this chapter, we present an efficient similarity searchable encryption scheme that supports multi-keyword semantics. Our solution is based two building blocks: Term Frequency - Inverse Document Frequency (TF-IDF) measurement and ring-LWE-based variant of homomorphic cryptosystem. We use the dot product to quantitatively evaluate similarity measure and rank the outsourced documents with their importance to the search query. We show that our scheme is adaptive semantically secure against adversaries and able to achieve optimal sublinear search time.

CHAPTER 5

Group Multi-keyword Similarity Searchable Encryption (GMKSim)

In order to make an important step toward widespread use of searchable encryption, existing solutions need to include mechanisms to efficiently support hundreds even thousands cloud users in the system[61]. In this chapter we consider a multi-user scenario that involves a data owner that wishes to share a documents and multiple data users that want to query encrypted data using the cloud server.

Curtmola *et al.* [5] were the first to extend their single-user scheme with broadcast encryption[62], where the data owner is able to outsource an encrypted document collection to the cloud server and an arbitrary group of users is allowed to query the data. Broadcast encryption allows the data owner to distribute a shared secret key to a group of data users. However, this solution might not work in a real-world cloud deployment that involves potentially large number of on-demand data users since only one key is shared among all users and each user revocation requires a new key to be distributed to the remaining users. It is desirable that each data user could keep its own secret key, which makes key management easier and more efficient.

We propose a new Group Multi-keyword Similarity Searchable Encryption (GMKSim) scheme that solves the problem of managing access privileges and searching multiple keywords over encrypted cloud data. Our solution is based on distributed broadcast encryption scheme[63] that supports a distributed setup where the data owner sets up the system, but each user generates their own key when joining the system. In fact, we remove the burden of key management from the data owner and let group establishment run by participating data users (i.e., data users are allowed to pick desired participants and establish a shared key to search remote encrypted data). We first give the set of definitions and security notions of GMKSim scheme. Later we present an efficient construction of GMKSim that combines the idea of a single-user MKSim scheme (presented in Chapter 4) with distributed broadcast encryption scheme.

5.1. Scheme Construction

5.1.1. Preliminaries. We begin with the definition of *witness pseudo-random function* (WPRF). Informally, a witness PRF for an NP language L is a PRF F such that anyone with a valid witness that $x \in L$ can compute $F(x)$ without the secret key, but for all $x \notin L$, $F(x)$ is computationally hidden without knowledge of the secret key. Formally, a witness PRF is defined as follows:

Definition 33. ***(Witness Pseudo-Random Function (WPRF)[63]).*** *A witness PRF is a triple of algorithms* (Gen, F, $Eval$) *such that*

- *Gen: a probabilistic algorithm that inputs a security parameter* λ *and a circuit* $R : \mathbb{X} \times \mathbb{W} \to \{0, 1\}$*, and outputs a secret function key* fk *and a public evaluation key* ek.
- *F: a deterministic algorithm that inputs the function key* fk *and an input* $x \in \mathbb{X}$*, and outputs some output* $y \in \mathbb{Y}$ *for some set* $\mathbb{Y}$.
- *Eval: a deterministic algorithm that inputs the evaluation key* ek*, an input* $x \in \mathbb{X}$ *and a witness* $w \in \mathbb{W}$*, and produces an output* $y \in \mathbb{Y}$ *or* $\perp$.

The following correctness property is required to hold:

$$Eval(ek, x, w) = \begin{cases} F(fk, x) & \text{if } R(x, w) = 1 \\ \\ \perp & \text{if } R(x, w) = 0 \end{cases} \quad \text{for all } x \in \mathbb{X}, w \in \mathbb{W}.$$

A multiparty key exchange protocol allows a group of g users to simultaneously post a message to a public bulletin board, retaining some user-independent secret. After reading off the contents of the bulletin board, all users establish the same shared secret key. The multiparty key exchange protocol consists of the following algorithms:

Definition 34. ***(Non-Interactive Multiparty Key Exchange protocol (NIKE-WPRF)[63]).*** *Let $G : \mathbb{S} \to \mathbb{Z}$ be a pseudo-random generator with $|S|/|Z| \leq negl$. Let $WPRF = (Gen,\ F,\ Eval)$ be a witness PRF. Let $R_g : \mathbb{Z}^g \times (\mathbb{S} \times [g]) \to \{0, 1\}$ be a relation that outputs 1 on input $((z_1, \ldots, z_g), (s, i))$ if and only if $z_i = G(s)$. The non-interactive key exchange protocol consists of:*

- *$Publish(\lambda, g)$: a probabilistic algorithm that takes as input the security parameter λ and the group order g. It computes $(fk, ek) \xleftarrow{R} Gen(\lambda, R_g)$. Next, it picks a random seed $sk \xleftarrow{R} \mathbb{S}$ and compute $z \leftarrow G(sk)$. It outputs a secret key sk and public values (z, ek), where sk is kept secret and (z, ek) are published to the bulletin board.*
- *$KeyGen(\{z_i, ek_i\}_{i \in [g]}, sk)$: a deterministic algorithm that inputs group g and user's secret sk. It outputs a group key $k = Eval(ek_i, (z_1, \ldots, z_g), (sk, i))$.*

Broadcast encryption[62] allows an encryptor (data owner) broadcast a message to a subset of recipients (data users). The system is said to be collusion resistant if non-data users can learn information about the plaintext. Boneh *et al.*[64] recently proposed a new distributed broadcast encryption that is based on NIKE where data users generate secret keys on their own and simply append their corresponding public values to the broadcast public key.

Unlike existing broadcast encryption schemes[65–67] where participants are assigned their secret key by a trusted authority (data owner), distributed broadcast scheme has no trusted authority and each user generates a secret key for itself. However, scheme in [64] is based on the indistinguishable obfuscation, for which none practical construction is known[63]. Most recent work by Zhandry *et al.* [63] proposes a distributed broadcast encryption scheme that replaces indistinguishable obfuscators with a witness PRFs. The scheme is simpler and much more efficient that current obfuscation candidates. We use the following definition of distributed broadcast encryption scheme:

DEFINITION 35. ***(Distributed Broadcast Encryption over NIKE (BE-NIKE-WPRF) [63]).*** *Distributed broadcast encryption scheme over multi-party non-interactive key exchange protocol consists of four following algorithms:*

- *Setup: a probabilistic algorithm to setup BE-NIKE-WPRF scheme. The algorithm outputs a secret parameter λ and group order g.*
- *$Join(\lambda, g)$: a probabilistic algorithm to join the scheme that is executed by each participant. The algorithm inputs a secret parameter λ and group order g. The algorithm invokes NIKE-WPRF.$Publish(\lambda, g)$ to output secret sk and public values (z, ek). The user makes (z, ek) publicly available to other participants.*
- *$Enc(\{z_i, ek_i\}_{i\in[g]}, sk, m)$: a probabilistic algorithm to encrypt message m under shared key. The algorithm inputs the set of public values $\{z_i, ek_i\}_{i\in[g]}$, secret key sk and plaintext message m. The algorithm runs NIKE-WPRF.$KeyGen(\{z_i, ek_i\}_{i\in[g]}, sk)$ to derive the shared key k. The algorithm outputs a ciphertext c which is the encryption of message m using the shared key k.*

- *Dec*$(\{z_i, ek_i\}_{i\in[g]}, sk, c_m)$*: a deterministic algorithm to decrypt* c_m*. The algorithm invokes NIKE-WPRF.KeyGen*$(\{z_i, ek_i\}_{i\in[g]}, sk)$ *to derive* k*. If* $k \neq \perp$*, then algorithm decrypts* c_m *using* k *and outputs the original message* m*.*

This completes cryptographic preliminaries used in our solution. We are now ready to present the GMKSim scheme.

5.1.2. Algorithm Definitions.

Definition 36. ***(Group Multi-keyword Similarity Searchable Encryption (GMKSim)).*** *An index-based GMKSim scheme over a set of documents* D *is a tuple of eight polynomial-time algorithms* $GMKSim$ = *(Gen, BuildIndex, Join, SetupGroup, Revoke,* $MakeQuery$*, Evaluate, Decrypt), as follows:*

- *(*S_1*,* S_2*,* PK*,* SK*,* λ*,* g*)* $\leftarrow$ *Gen*(1^s)*: a probabilistic algorithm run by the data owner to setup the GMKSim scheme. The algorithm invokes* $MKSim.Gen$ *with an input of a secret parameter* s*, and outputs a set of keys* S_1*,* S_2*,* PK*,* SK*. The algorithm runs BE-NIKE-WPRF.Setup to output secret parameter* λ *and group order* g*.*
- $(I, C) \leftarrow BuildIndex(S_1, S_2, PK, D, K)$*: a probabilistic algorithm run by the data owner to encrypt a document collection* D*. The algorithm invokes* $MKSim.BuildIndex$ *with an input of keys* S_1*,* S_2 *and* PK*, a document collection* D *and a keyword dictionary* K*. It outputs a searchable index* I *and a set of encrypted documents* C*.*
- $(sk, (z, ek)) \leftarrow Join(\lambda, g)$*: a probabilistic algorithm run by each data user to participate in the scheme. The algorithm invokes BE-NIKE-WPRF.Join with an input of secret parameter* λ *and group order* g*. It outputs a pair* $(sk, (z, ek))$*.*
- $c_r \leftarrow$ *SetupGroup*$(\{z_i, ek_i\}_{i\in[h]}, sk)$*: a probabilistic algorithm run by the group owner to establish the group* $h \subseteq g$ *of authorized data users. The algorithm runs BE-NIKE-WPRF.Enc*

with an input of public values $\{z_i, ek_i\}_{i\in[h]}$*, group owner's secret key* sk *and a sampled secret* r*. The output is encrypted ciphertext* c_r*.*

- $c_r \leftarrow Revoke(\{z_i, ek_i\}_{i\in[h\backslash o],}, sk)$*: a probabilistic algorithm run by the group owner to remove a user* o *from the set of authorized users. The algorithm invokes BE-NIKE-WPRF.Enc that inputs the set of public values* $\{z_i, ek_i\}_{i\in[h\backslash o]}$*, group owner's secret key* sk *and a new secret* r*. The output is encrypted ciphertext* c_r*.*
- $\Omega \leftarrow MakeQuery(S_2, PK, K, \bar{K}, c_r)$*: a probabilistic algorithm run by a data user to construct a search query. The algorithm invokes BE-NIKE-WPRF.Dec with an input of public values* $\{z_i, ek_i\}_{i\in[h]}$*, secret key* sk *and ciphertext* c_r*. It outputs a secret* r*. If* $r \neq \perp$*, the algorithm invokes* $MKSim$*.MakeQuery with keys* S_2*,* PK*, keyword dictionary* K*, and set of keywords of interest* $\bar{K}$*. The output is a search query* Ω *encrypted under secret* r*.*
- $L \leftarrow Evaluate(PK, I, \Omega, c_r)$*: a deterministic algorithm run by the cloud server. The algorithm invokes BE-NIKE-WPRF.Dec an input of public values* $\{z_i, ek_i\}_{i\in[h]}$*, secret key* sk *and ciphertext* c_r*, and it outputs secret* r *to decrypt search query* Ω*. The algorithm runs* $MKSim$*.Evaluate with an input of public key* PK*, searchable index* I *and search query* Ω*. The algorithm outputs a sequence of identifiers* $L \subseteq C$*.*
- $D_i \leftarrow Decrypt(S_1, SK, C_i)$*: a deterministic algorithm that inputs a set of secret keys* S_1*,* SK *and an encrypted document* C_i*. The algorithm outputs a document* D_i*.*

We now formalize the security of proposed scheme.

- We require that the cloud server should not learn anything about the documents: an adversary with an access to searchable index I and encrypted document collection

$C = (C_1, C_2, \ldots, C_n)$ should learn nothing about original documents $D = (D_1, D_2, \ldots, D_n)$

- We require that the cloud server should not learn anything from the search queries beyond the access and search patterns: an adversary with an access to a search queries $(Q_1, Q_2, \ldots, Q_m)$ generated by the data user learns nothing about the content of each search query Q_i or the content of resulted documents.
- We require the revocation for the data users: once a data user removed from the set of authorized data users, he/she is no longer able to execute a search over encrypted documents.

In GMKSim we use the adaptive semantic security notion of a single-user MKSim scheme. It provides the security against an adaptive adversary: cloud server does not learn anything about the documents and search queries beyond the access and search patterns. However, with an addition of access privilege property, we need to expand our security definitions towards the *Revoke* algorithm. We define the probabilistic experiment *Rev* as follows:

Definition 37. ***(Revocation).*** *Let $GMKSim$ = (Gen, BuildIndex, Join, SetupGroup, Revoke, MakeQuery, Evaluate, Decrypt) be a group MKSim scheme, s be a security parameter, and $A = (A_1, A_2, A_3)$ be an adversary. We use the following probabilistic experiment $Rev_{GMKSim,A}(s)$:*

```
$Rev_{GMKSim,A}(s)$:
$S_1,\ S_2,\ PK,\ SK,\ \lambda,\ g \leftarrow Gen(1^s)$
$(st_A, D, K) \leftarrow A_1(1^s)$
$(sk_A, (z_A, ek_A)) \leftarrow Join(\lambda, g)$
$c_r \leftarrow SetupGroup((z_A, ek_A), sk)$
$(I, C) \leftarrow BuildIndex(S_1, S_2, PK, D, K)$
$st_A \leftarrow A_2^{O(I,C,st_{\mathbb{S}},\cdot)}(st_A, sk_A, (z_A, ek_A), c_r)$
$c_r^{'} \leftarrow Revoke((z_A, ek_A), sk)$
$\Omega \leftarrow A_3(st_A)$
$L \leftarrow Evaluate(st_S, PK, I, \Omega, c_r^{'})$
if $L \neq \perp$, output 1, otherwise output 0,
```

where $O(I, C, st_{\mathbb{S}}, \cdot)$ is an oracle that inputs a search query Ω and outputs ciphertexts C indexed by $L \leftarrow Evaluate(PK, I, \Omega, c_r^{'})$ if $L \neq \perp$ and $\perp$ otherwise. We claim that Revoke algorithm achieves user revocation if for all polynomial-size adversaries $A = (A_1,\ A_2,\ A_3)$ the following is correct:

$$Pr[Rev_{GMKSim,A}(s) = 1] \leq negl(s), \tag{19}$$

where the probability is over the coins of Gen, Join, SetupGroup, Revoke and BuildIndex.

5.1.3. GMKSim Construction. Figure 5.1 shows the details of $GMKSim$ = (*Gen*, *BuildIndex*, *Join*, *SetupGroup*, *Revoke*, *MakeQuery*, *Evaluate*, *Decrypt*). In our construction we combine the ideas of a single-user $MKSim$ = (*Gen*, *BuildIndex*, *MakeQuery*,

```
$\underline{Gen(1^s):}$
    (1) generate $S_1$, $S_2$, $PK$, $SK \leftarrow MKSim.Gen(1^s)$.
    (2) generate $\lambda$, $g \leftarrow BE\text{-}NIKE\text{-}WPRF.Setup(1^s)$.
Output the key set $S_1$, $S_2$, $PK$, $SK$, secret parameter $\lambda$ and group order $g$.
$\underline{BuildIndex(S_1, S_2, PK, D, K):}$
    (1) set $(I, C) \leftarrow MKSim.BuildIndex(S_1, S_2, PK, D, K)$.
Store $(I, C)$ to the cloud server.
$\underline{Join(\lambda, g):}$
    (1) generate $(sk, (z, ek)) \leftarrow BE\text{-}NIKE\text{-}WPRF.Join(\lambda, g)$.
Keep $sk$ private, output $(z, ek)$ to the cloud server.
$\underline{SetupGroup(\{z_i, ek_i\}_{i\in[h]}, sk}$
    (1) pick $h \subseteq g$ and get public values $\{z_i, ek_i\}_{i\in[h]}$ from the cloud server.
    (2) sample $r \leftarrow \{0,1\}^s$ and compute $c_r \leftarrow BE\text{-}NIKE\text{-}WPRF.Enc(\{z_i, ek_i\}_{i\in[h]}, sk, r)$.
Output $c_r$ to the cloud server.
$\underline{Revoke(\{z_i, ek_i\}_{i\in[h\backslash o]}, sk)}$
    (1) set $(h\backslash o) \subseteq g$ and retrieve public values $\{z_i, ek_i\}_{i\in[h\backslash o]}$ from the cloud server.
    (2) generate new $r \leftarrow \{0,1\}^s$
    (3) compute $c_r \leftarrow BE\text{-}NIKE\text{-}WPRF.Enc(\{z_i, ek_i\}_{i\in[h\backslash o]}, sk, r)$.
Output new $c_r$ to the cloud server.
$\underline{MakeQuery(S_2, PK, K, \bar{K}, c_r):}$
    (1) retrieve $c_r$ and from the cloud server.
    (2) compute $r \leftarrow BE\text{-}NIKE\text{-}WPRF.Dec(\{z_i, ek_i\}_{i\in[h]}, sk, c_r)$. If $r = \perp$, output $\perp$.
    (3) calculate $\Omega' \leftarrow MKSim.MakeQuery(S_2,\ PK,\ K,\ \bar{K})$.
    (4) set $\Omega \leftarrow \rho(r, \Omega')$.
Output $\Omega$.
$\underline{Evaluate(PK, I, \Omega, c_r):}$
    (1) compute $r \leftarrow BE\text{-}NIKE\text{-}WPRF.Dec(\{z_i, ek_i\}_{i\in[h]}, sk, c_r)$.
    (2) compute $\Omega' \leftarrow \rho^{-1}(r, \Omega)$.
    (3) output $L \leftarrow MKSim.Evaluate(PK, I, \Omega')$, where $L \in C$.
Output $L$.
$\underline{Decrypt(S_1, SK, C_i):}$
Output $D_i \leftarrow MKSim.Decrypt(S_1, SK, C_i)$.
```

FIGURE 5.1. GMKSim Scheme Construction.

Evaluate, *Decrypt*) and a distributed broadcast encryption scheme *BE-NIKE-WPRF* = (*Setup*, *Join*, *Enc*, *Dec*). We require standard security notions for broadcast encryption. Specifically, in addition of providing PCPA-security, it provides revocation-scheme security against a group of all revoked users.

Our construction relies on a pseudo-random permutation $\rho : \{0,1\}^r \times \{0,1\}^t \to \{0,1\}^t$, where r is the secret parameter and t is the size of search query Ω in the $MKSim$ scheme. We assume the honest-but-curious adversarial model for the cloud server. We also assume that the cloud server does not collude with revoked users, otherwise our construction cannot prevent a revoked user from executing a search.

To describe $GMKSim$ in details we use following hospital example illustrated in Figure 5.2. Consider a doctor (data owner) that performed a blood test on a patient and wishes to share resulted documents with group of nurses (data users) in the hospital. The doctor considers building a searchable index I from the resulted documents D, and sending both index I and encrypted results C to the hospital blackboard running on the cloud server. To remove the burden of key management, the doctor enables distributed setup, where each nurse generates its own secret key (when joining the system) and establishes a group of authorized participants (e.g., a head nurse includes her subordinate nurses). The doctor begins with sampling a secret parameter s, which is used as input to generate the set of secret keys. The key generation algorithm outputs a set of keys S_1, S_2, PK, SK for a single-user scheme, secret key λ and group order g for a distributed broadcast encryption scheme. The doctor next invokes the $BuildIndex$ algorithm of a single-user scheme $MKSim$ outlined in Figure 4.1. The outputs are searchable index I and set of encrypted documents C. Note, the searchable index I consists of a lookup filter T, based on SSE-2 construction, and a TF-IDF table Φ constructed using a term weight importance. Next, each nurse launches the $Join$ algorithm with secret λ and group g (both distributed by the doctor) to generate an output of $(sk, (z, ek)$. Secret key sk is kept private, while (z, ek) are published to the cloud server.

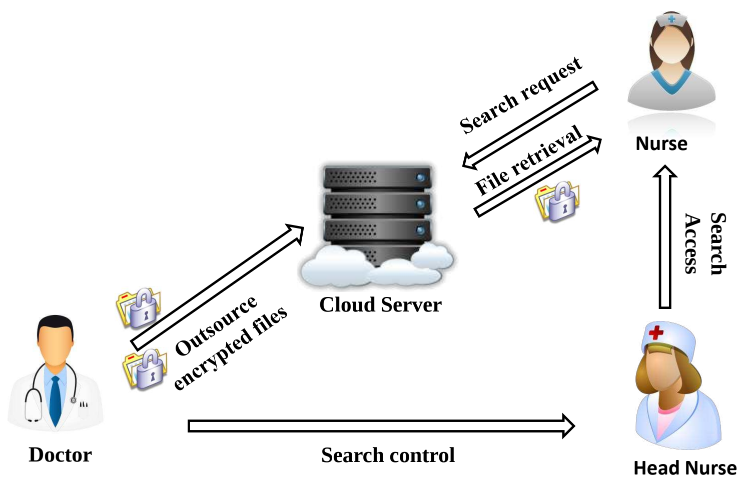


Figure 5.2. Group Hospital Example.

Now, the head nurse (group owner) creates a group of authorized users (other participants) that are allowed to execute search over encrypted cloud data. The head nurse launches the *SetupGroup* algorithm where she picks public values $\{z_i, ek_i\}_{i \in h}$ of authorized participants $h \subseteq g$, samples random secret parameter r, and uses the distributed broadcast encryption to output the ciphertext c_r. Finally, head nurse sends c_r to the cloud server. Now the document collection is available for selective retrieval.

To search for a keywords of interest $\bar{K}$, a nurse (data user) executes $MakeQuery$ algorithm that includes four steps. First, she contacts the cloud server and retrieves the ciphertext c_r. Next, she invokes the distributed broadcast encryption algorithm with her own secret sk, public values $\{z_i, ek_i\}_{i \in h}$ to recover the secret r. If r successfully recovered, she inputs keywords $\bar{K}$ to a single-user *MKSim*.$MakeQuery$ algorithm that outputs search query Ω'. Next, the nurse forms a permuted search query by applying PRP ρ with secret key r, i.e. $\Omega = \rho(r, \Omega)$ which is sent to the cloud server. The cloud server, upon receiving

Ω, recovers the search query by executing $\rho^{-1}(r, \Omega)$. Here, the key r used in ρ is known only by the head nurse, the cloud server and the set of authorized data users h. Once Ω' is decrypted, the cloud server executes a single-user *MKSim.Evaluate* algorithm to find a set of encrypted documents that match $\bar{K}$ keywords of interest.

If a nurse o is no longer an authorized user in group, the head nurse invokes the *Revoke* algorithm on o. Specifically, the head nurse samples a new secret r' and generate a new cloud server ciphertext $c_r^{'}$ using distributed broadcast encryption algorithm that exclude the public values (z, ek) of nurse o. The new cloud ciphertext $c_r^{'}$ is distributed to the cloud server to replace the old c_r. Since revoked data user o is not able to recover the new secret r' in the $MakeQuery$ algorithm, permuted Ω will not yield a valid search query after applying $\rho^{-1}(r^{'}, \Omega)$ permutation at the cloud server. This simple extra layer given by the pseudo-random permutation ρ prevents data users from performing successful search once they are removed from the system. Proposed solution is very efficient since the cloud server only needs to evaluate a pseudo-random permutation to decide whether a data user is authorized to search an encrypted document collection.

The following theorem shows that the $GMKSim$ scheme satisfies revocability.

Theorem 3. The $GMKSim = (Gen, BuildIndex, Join, SetupGroup, Revoke, MakeQuery,$ *Evaluate*, *Decrypt*) achieves revocability according to Definition 37.

Proof. The proof is straightforward, and we only state the intuition behind the proof. The revocation of proposed scheme relies on the following assumption. The PCPA-security of *BE-NIKE-WPRF* scheme guarantees that the simulated search query $\Omega^{\star}$ is indistinguishable from the real search query Ω. Indeed, this is true since decryption of the new message $c_r^{'}$ (generated by the group owner that excludes the public values of revoked user) will never output a valid r' that is used to generate the real Ω. $\square$

5.2. Conclusion

In this chapter we presented a Group Multi-keyword Similarity Searchable Encryption (GMKSim) scheme that solves the problem of managing access privileges and searching multiple keywords over encrypted cloud data. In our solution an arbitrary group of data users can submit queries to the remote server to search an encrypted document collection. We design a scheme supports distributed setup, where participants choose their own secret key rather than receive the key from a trusted authority. Finally, we argued the revocability correctness and efficiency of our construction.

CHAPTER 6

Substring Position Searchable Symmetric Encryption (SSP-SSE)

As we have seen in Chapter 3, many Searchable Encryption (SE) schemes have been proposed in recent years [1–6, 68, 9, 69, 48, 57, 70, 12, 11, 48, 71, 17, 50]. (Note, we use the term searchable encryption somewhat loosely to include schemes such as private information retrieval also.) Generally, SE solutions involve building an encrypted searchable index that hides the sensitive information from the remote server, yet it allows a search on the encrypted data. SE solutions differ in the level of efficiency and security guarantees they offer; however, most of them support only *exact* keyword search. As a result, there is no tolerance of format inconsistencies which are part of typical cloud user behavior; and they happen frequently. It is quite common that the search queries do not exactly match the pre-set keywords due to lack of exact knowledge about the data. For example, a financial company stores its employees income tax documents in encrypted form in the cloud. A tax accountant may issue a search query of “mcd”, which describes multiple keywords suchs as “mcdaniel”, “mcdavid”, “mcdonald”, “mcdunn”, and she wants to find a position of the first occurrence of the query in each encrypted document that contain the string of characters. The significant drawback of existing schemes underlines an important need for new techniques that support search flexibility over encrypted documents. In this chapter, we consider the problem of efficient substring position search over encrypted data. The users can query the remote untrusted server for a set of encrypted documents that contain a substring of characters. The cloud server retrieves the set of matching documents together with positions where the queried string begins.

An important application of this chapter is in the area of searching a genome sequence against a genomic databases. Such search can be used in the analysis of genetic diseases, genetic fingerprinting or genetic genealogy, and requires a set of results that not simply match the genome, but rather the position of the genome sequence within the genome database. The major contribution of our work is to initiate the study of a very important problem, namely, substring position search over encryption data. Our solution should not be considered as a complete approach of the subject, which has very strong future directions of research. Nonetheless, our solution provides the preliminary foundation for the study of the subject, including formal definitions, building blocks, basic construction as well as security proofs. In this work, we continue exploring the line of recent searchable encryption solutions, but from the slightly different standpoint.

We now give an overview of our contributions:

- We present a Substring Position Searchable Symmetric Encryption (SSP-SSE) scheme that allows a substring search over encrypted document collection. The scheme is based on a position heap tree data structure recently proposed by Ehrenfeucht *et al.*[72].
- We formally define two leakage functions and security against adaptive chosen-query attack of a tree-based SSP-SSE scheme. Apart from traditional *access* and *search* patterns we include the definition of *path* pattern in the leakage functions of a tree-based searchable encryption. We show that SSP-SSE enjoys the strong notion of semantic security[5].
- We present a construction that is very efficient and does not require large ciphertext space. Our encryption takes $O(kn)$ time and the ciphertext is of size $O(kn)$, where

k is the security parameter and n is the size of stored data. The search protocol takes $O(m^2 + occ)$ time and three rounds of communication, where m is the length of the queried substring and occ is the number of occurrences of substring in the document collection.

We organize the rest of the chapter as follows: In Section 6.1 we briefly review preliminaries and cryptographic notations used in our solution. In Section 6.2 we present algorithms and data structures that allow a substring search on the plaintext data. We give a brief overview of each data structure and later present a discussion on choosing the right data structure to enable substring search in untrusted cloud environment. In Section 6.3 we provide the details of SSP-SSE scheme and define the security definitions and requirements. Section 6.4 is devoted to security and performance analysis. Lastly, we conclude in Section 6.5.

6.1. Notations

6.1.1. Preliminaries.

Let $D = \{D_1, D_2, \ldots, D_l\}$ be an original set of documents and let $C = \{C_1, C_2, \ldots, C_l\}$ be an encrypted collection of documents from D. If D_i and D_j are two documents, we denote text t as their concatenation by $D_i||D_j$. If A is an algorithm, then $a \leftarrow A(\ldots)$ represents the result of applying the algorithm A.

6.1.2. Cryptographic Notations.

Definition 38. **(Symmetric Key Encryption (SKE)).** *A symmetric key encryption scheme consists of the following PPT algorithms:*

- $Gen(1^k)$ *: a key generation algorithm that inputs a security parameter k and outputs a secret key K.*

- *$Enc(K, m)$: a probabilistic algorithm that inputs a secret key K and message m, and outputs a ciphertext c.*
- *$Dec(K, c)$: a deterministic algorithm that inputs a secret key K and ciphertext c, and outputs a message m or special symbol $\perp$ (if decryption failed).*

DEFINITION 39. **(SKE Correctness).** *Given the symmetric encryption scheme SKE that consists of three algorithms (Gen, Enc, Dec), for all k and all m such that $K \leftarrow Gen(1^k)$, we require:*

$$Dec(K, Enc(K, m)) = m. \tag{20}$$

We also require SKE to be secure against pseudorandom chosen-plaintext attacks (PCPA). We now give the definition of PCPA-security of SKE scheme.

DEFINITION 40. **(PCPA-security).** *Let SKE = (Gen, Enc, Dec) be a symmetric encryption scheme, A be an adversary and there is a probabilistic experiment $PCPA_{SKE,A}(k)$ that is run as follows:*

- *Use secret parameter k to output the secret key $K \rightarrow Gen(1^k)$.*
- *The adversary A is given oracle access to $Enc_K()$.*
- *The adversary A outputs a message m.*
- *Let $c_0 \leftarrow Enc_K(m)$ and $c_1 \overset{R}{\leftarrow} C$. C denotes the set of all possible ciphertexts. A bit b is chosen at random and c_b is given to the adversary A.*
- *The adversary A is again given to the oracle access to $Enc_K()$, and A runs number of polynomial queries to output a bit b'.*
- *The experiment outputs 1 if $b = b'$, otherwise 0.*

Symmetric encryption scheme SKE is PCPA-secure if for all polynomial-size adversaries A,

$$Pr[PCPA_{SKE,A}(k) = 1] \leq \frac{1}{2} + negl(k), \tag{21}$$

where the probability is over the choice of bit b and the coins of Gen and Enc.

We also make use of pseudo-random function (Definition 8) and pseudo-random permutation (Definition 9), which are polynomial-time computable functions that cannot be distinguished from random functions by any probabilistic polynomial-time adversary.

6.2. Substring Search Algorithms

In this section, we present the most popular algorithms and data structures that allow a substring search on the plaintext data. Specifically, we focus on mature data structures like suffix tree [73] and suffix array [74] that have been widely used in many substring search applications. We also, present the details of recently proposed position heap tree [72]. For each data structure we give a short overview with examples and then we present the computation and storage efficiency. Lastly, we present a discussion on choosing the right data structure to enable substring position search in untrusted cloud environment.

6.2.1. Suffix Tree.

Suffix tree [73, 75, 76] is a trie-like representation of text supporting a wide range of applications on strings. The suffix tree is pre-processed data structure that enables a substring search on the stored string. We now give the definition of the suffix tree:

Definition 41. **(Suffix Tree).** *A suffix tree for string $t = t_1 \ldots t_n$ is a rooted, directed tree with following properties:*

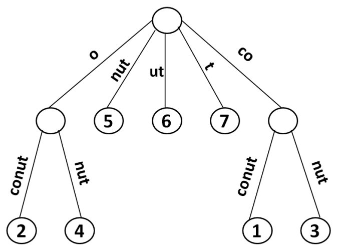
- *Each edge is labeled with a non-empty substring of t, named as edge label.*
- *Every internal node as at least two children.*
- *No two edges out of a node have edge labels starting with the same character.*
- *The tree has n leaves, labeled from 1 to n.*

Definition 42. **(Path Label).** *The path label of a node is the concatenation of the edge labels on the path from the root to that node.*

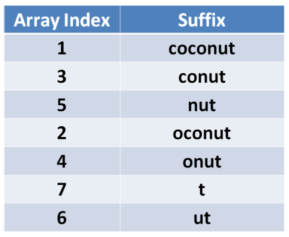
Definition 43. **(Suffix Tree Search).** *A string χ is a substring of t if and only if it is a prefix of some suffix of t.*

Figure 6.1a shows an example of suffix tree constructed from the text "coconut". To check if a string χ is a substring, the algorithm searches for a path from the root whose labels match χ. For instance, searching for a string "coconut", we begin at the root node and start checking the neighbor edge labels, down to the matching node, i.e. node 1 is the matching and it corresponds to the occurrence in the text. Similarly, the search of "co" leads us to the intermediate node whose leaf nodes $(1, 3)$ are the positions in text.

The suffix tree can be constructed in $O(n)$ time for a string of length n [76]. Also, it can be shown that a suffix tree has at most $2n$ nodes and storing edge label for all edges would require $O(n^2)$ in the worst case. (Consider a suffix tree for the strings $t_1t_2 \ldots t_n$, where t_i is unique. The suffix tree would contain distinct edge for each of the n suffixes $t_1 \ldots t_n$, $t_2 \ldots t_n$, $\ldots, t_n$. These suffixes have a total length of $O(n^2)$). Searching for a substring χ of length m takes $O(m + occ)$ time to find all occurrences of χ, where occ is the number of occurrences.

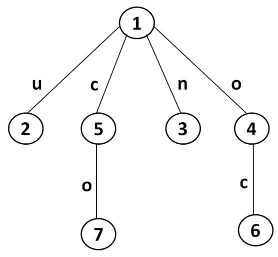


(A)



Array Index	Suffix
1	coconut
3	conut
5	nut
2	oconut
4	onut
7	t
6	ut

(B)



(C)

FIGURE 6.1. An example of data structure constructed from the text "coconut". (**a**) A suffix tree. (**b**) A suffix array. (**c**) A position heap tree.

6.2.2. SUFFIX ARRAY. Suffix array[74] is a sorted index array of all suffixes of a string. Suffix array data structure is used in full text indices, within the field of computational biology and others.

DEFINITION 44. **(Suffix Array).** *Given a text t of length n, the suffix array for t is an array of integers of range 1 to n specifying the lexicographic ordering of the suffixes of the string t.*

A suffix tree can be built in $O(n)$ time for a string of length n[74]. To search for a substring χ of length m, the search can be executed as simple binary search over the suffix array, i.e. for each element in the suffix array we then compare the suffix of t at element position with a substring χ. Thus, the search for any substring can be performed in $O(m \times log(n))$ time. This complexity can be improved by adding the longest common prefix information, so the search can be executed in $O(m + log(n))$ (see [74] for details).

Consider an example of suffix array in Figure 6.1b constructed from the text "coconut". Searching for "coconut" gives the occurrence of (1), while searching for "co" results in $(1, 3)$ occurrences in the text.

6.2.3. Position Heap Tree.

We now give an overview of a position heap tree data structure[72].

Definition 45. **(Position Heap).** *The position heap Λ of text t is a tree constructed by iteratively inserting the suffixes $(t_1, t_2, \ldots, t_n)$ of t in ascending order into Λ. That is, t_i is inserted by creating a new node in Λ that is the shortest prefix of t_i that is not already a node of the tree, and labeling it with position i.*

Figure 6.1c shows an example of position heap tree Λ constructed from the text "coconut". The first suffix "t" of text creates the root node in Λ with position label of "1". Next, second suffix "ut" of text creates the new node with position 2 and connecting edge with label "u"; third suffix "nut" creates the new node with position 3 and connecting edge with label "n". Similarly, seventh suffix "coconut" creates a new node with position 7 and connecting edge with label "o" since there is already a node in Λ with edge label "c" created by fifth suffix "conut". Following Definition 45, the position heap Λ is constructed. The construction can be executed for any text t, and since it is deterministic, the position heap Λ for a text is unique.

We now present the definition of the search in the position heap tree.

Definition 46. **(Position Heap Search).** *Position heap search of all occurrences of a substring χ of text t in Λ consist of the following steps:*

- *Index into the position heap Λ to find the longest prefix p of χ that is a node of Λ. For each ancestor p' of p, lookup the position i stored in p'. Here, position i is an occurrence of p'. Determine if this occurrence is followed by $\chi - p'$. If yes, report i as an occurrence of χ.*
- *If $p = \chi$, also report all positions of descendants of p.*

Using the example tree in Figure 6.1c, the search for a substring "co" leads to the node (7). The set of traversed ancestor nodes $(5, 1)$ needs an inspection with text t. Indeed, only position 5 matches to the substring "co". So, the positions of substring query "co" are $(5, 7)$. In case of substring query "coconut", the search algorithm falls of the tree, thus the search algorithm returns a set of traversed nodes $(7, 5, 1)$ for an inspection, where 7 is the only matching occurrence of "coconut" in the text.

Position heap tree for a text of length n can be constructed in $O(n)$ time[72]. All positions of substring χ of length m can be found in $O(m^2+occ)$, where occ is the number of occurrences reported. We refer the reader [72] for detailed discussion on position heap properties.

6.2.4. Discussion. Recall our goal is to construct a scheme that allows a substring search over encrypted data outsourced to the cloud. In our system model the data owner has a set of documents with sensitive information that he wants to upload in encrypted form to the cloud provider. To enable the substring search, the data owner constructs a searchable index I from the set of plaintext documents D, and then he places I at the cloud provider to allow the substring search. Our goal is to choose an *optimal* data structure that has low storage requirements and fast search execution. Later we use selected data structure to construct the searchable index I in our scheme.

As we have noted previously, many substring matching algorithms have been proposed and they differ in terms of storage requirements and search execution. We outline the comparison of substring search data structures in Table 6.1. We use several comparison parameters: the construction time, the search execution time and the storage requirements. The construction time corresponds to the time it takes to create a data structure with input of the text t of size n. The search execution time is the time it takes to find all occurrences of

Table 6.1. Comparison of plaintext substring search data structures. n is the length of the text t, m is the length of the substring χ, occ is the number of occurrences of χ in t.

Data Structure	Construction	Search	Cloud Storage
Suffix Tree	$O(n)$	$O(m + occ)$	$O(n^2)$
Suffx Array	$O(n)$	$O(m + log(n))$	$O(n^2)$ [4]
Position Heap Tree	$O(n)$	$O(m^2 + occ)$	$O(n)$

substring χ of length m in the text t. The cloud storage describes a storage of all combined textual labels in each data structure.

From Table 6.1 we can see that the suffix tree, suffix array and position heap tree have $O(n)$ preprocessing time of text t of size n. In search, the suffix tree has the best $O(m+occ)$ execution time for a substring χ of length m. However, the suffix tree has at most $2n$ nodes and it would take $O(n^2)$ space to store the text at the cloud provider. On the other side, the suffix array can be constructed with n elements, but it has $O(n^2)$ storage. Only position heap tree allows us to have the low storage $O(n)$ with n nodes; however the substring search execution takes $O(m^2 + occ)$ time. In the rest of the chapter, we use the position heap tree data structure as the main construction block for our scheme. In our choice between the data structures we believe that the $O(n)$ storage requirement is the predominant criteria since expanding any large dataset (e.g., human genome with 3 billion letters) to a $O(n^2)$ storage would cause a substantial waste of cloud computing resources.

[4]Note that suffix array data structure stores only the array of integers (no need to store the suffixes of text) and the array can be accessed by running a binary search algorithm in $log(n)$ time, i.e. each time we access the element in the suffix array, we execute a lexicographical comparison of strings of the suffix at element position and the the given substring query. This can be executed locally (by the data owner), however, in our system model defined in Section 3.1, the data owner sends the data and constructed searchable index to the malicious cloud provider. Both the data and the searchable index are encrypted so no plaintext (and no lexicographical order) is leaked to the cloud provider. If we were to encrypt the suffix array by encrypting each element of suffix array, then the cloud provider would not be able to execute the search in $log(n)$ (in fact, it would observe the ciphertext at each element in the array which gives no order in binary search execution). However, to keep the binary search $log(n)$ time, one solution is to store encrypted suffixes in each node of the binary search tree and use an expensive homomorphic encryption[77] that allows search on the encrypted binary search tree. However, this would take $O(n^2)$ worst case storage for all suffixes in the tree.

6.3. Basic Construction

6.3.1. Algorithm Definitions.

We now present the definition of our scheme.

Definition 47. **(Substring Position Searchable Symmetric Encryption (SSP-SSE).** *A tree-based SSP-SSE scheme over a set of documents D is tuple of six polynomial-time algorithms (KeyGen, BuildTree, Encrypt, ConstructQuery, Search, Decrypt), as follows:*

- $K \leftarrow KeyGen(1^k)$*: a probabilistic key generation algorithm to setup the SSP-SSE scheme. The algorithm takes a secret parameter k and outputs a set of secret keys K.*
- $(\Lambda) \leftarrow BuildTree(D)$*: a deterministic algorithm to build a position heap tree Λ. The algorithm takes a document collection $D = (D_1, \ldots, D_l)$ and outputs a position heap tree Λ.*
- $(I, C) \leftarrow Encrypt(K, \Lambda, D)$*: a probabilistic algorithm to encrypt a position heap tree Λ and document corpus D. The algorithm inputs a set of secret keys K, a position heap tree Λ and a documents corpus D. The output of algorithm is a searchable index I and encrypted collection $C = (C_1, \ldots, C_l)$.*
- $[(Q) \leftarrow ConstructQuery(K, \chi)] \leftrightarrow [(L) \leftarrow Search(I, Q)]$ *: two deterministic algorithms that are executed interactively between the cloud user and the cloud provider. ConstructQuery algorithm inputs a set of secret keys K, a substring χ, and it outputs a search query Q. Search(I, Q) is an algorithm that inputs a searchable index I and a search query Q. The algorithm finds the set of matching encrypted document identifiers $L \in C$.*

- $(D_i, pos_{D_i}) \leftarrow Decrypt(K, C_i)$: *a deterministic algorithm that takes a set of secret keys K and a ciphertext C_i as input, and outputs an original document D_i, $\forall i \in [1;n]$, and a set of χ's positions pos_{D_i} in D_i.*

Definition 48. **(SSP-SSE Correctness).** *We say that the tree-based SSP-SSE scheme is correct if $\forall k \in \mathbb{N}$, $\forall$ K produced by $KeyGen(1^k)$, $\forall$ D, $\forall$ Λ output by $BuildTree(D)$, $\forall$ χ, $\forall$ $i \in [1;n]$:*

$$\begin{aligned} &Search(Encrypt(K, \Lambda, D), ConstructQuery(K, \chi)) = \\ &= C(\chi) \bigwedge Decrypt(K, C_i) = (D_i, pos_{D_i}) \end{aligned} \tag{22}$$

The SSP-SSE correctness ensures proper output if all SSP-SSE algorithms are executed honestly by the cloud provider.

6.3.2. Security Model. Security goal of any searchable encryption scheme is to reveal as little information as possible to the adversary. Intuitively, in SSP-SSE scheme we want to provide the following security guarantees: given a searchable index I and a set of encrypted documents $C = \{C_1, \ldots, C_l\}$ to the adversary, no valuable information about the original documents $D = \{D_1, \ldots, D_l\}$ is leaked to the adversary; given a set of incoming search queries $Q = \{Q_1, \ldots, Q_t\}$, the adversary cannot learn any practical information about the content of the search query Q_i or the original document collection D. However, these security guarantees are difficult to achieve and most known searchable encryption schemes [3, 57, 5, 2, 48] reveal some information, namely the *access pattern* and the *search pattern*. In

SSP-SSE we follow the similar approach of [5] to weaken the security guarantees and allow some limited information to the adversary.

DEFINITION 49. **(Access pattern).** *Given the n encrypted documents C, where $C = \{C_1, \ldots, C_l\}$, the search query vector Q, where $Q = \{Q_1, \ldots, Q_t\}$ of size t, the access pattern $\kappa(C, Q)$ includes the set of document identifiers induced by a search query vector Q.*

DEFINITION 50. **(Search pattern).** *Given the n encrypted documents C, where $C = \{C_1, \ldots, C_l\}$ the search query vector Q, where $Q = \{Q_1, \ldots, Q_t\}$ of size t the search pattern $\gamma(C, Q)$ is a $n \times t$ binary matrix such that $\forall i \in [1; n]$ and $\forall j \in [1; t]$, the cell element of i^{th} row and j^{th} column is 1, if a document identifier id_i is returned by a search query Q_j. The search pattern reveals whether the same search was executed in the past or not.*

Since our solution is based on the position heap tree data structure, we would like to capture the path pattern security notion. The path pattern of the position heap tree reveals the path traversed from the root node to the matching node for a given search query.

DEFINITION 51. **(Path pattern).** *Given the n encrypted documents C, where $C = \{C_1, \ldots, C_l\}$, and the searchable index I built from the document collection, the path pattern of (C, I) induced by the search query vector Q, where $Q = \{Q_1, \ldots, Q_t\}$ of size t, is a tuple $\delta(C, I, Q)$ that reveals the set of identifiers of nodes in the index I that are reached by query $Q_{i \in [1;t]}$.*

Now we define the leakage functions to capture all the information leakage we have in this work:

- Leakage $\mathbb{L}_1(I, C)$. Given the encrypted collection $C = \{C_1, \ldots, C_l\}$ and the searchable index I, the leakage consists of the following information: the number of encrypted documents, the size of encrypted documents, the identifier of each encrypted document.
- Leakage $\mathbb{L}_2(Q, I, C)$. Given the encrypted collection $C = \{C_1, \ldots, C_l\}$, the searchable index I and the search query Q, the leakage function outputs the access pattern $\kappa(C, Q)$, search pattern $\gamma(C, Q)$ and path pattern $\delta(C, I, Q)$.

DEFINITION 52. **(Security against adaptive chosen-query attack CQA2).** *Let SSP-SSE be tree-based SSE scheme that consists of six algorithms as described in Definition 47. Let $\mathbb{A}$ be a stateful adversary, $\mathbb{S}$ be a stateful simulator. We consider two probabilistic experiments $Real_{\mathbb{A}}$ and $Ideal_{\mathbb{A},\mathbb{S}}$ that involve $\mathbb{A}$ as well as $\mathbb{S}$ with two stateful leakage algorithms $\mathbb{L}_1$ and $\mathbb{L}_2$ and security parameter k:*

$\underline{Real_{\mathbb{A}}(k)}$: *The challenger runs the $KeyGen(1^k)$ to output the key set K. The adversary $\mathbb{A}$ sends constructed plaintext position heap tree Λ and collection D to the challenger, and receives a tuple $(I, C) \leftarrow Encrypt(K, \Lambda, D)$ from the challenger. The adversary $\mathbb{A}$ makes a polynomial number of adaptive string searches $\chi = \chi_1, \ldots, \chi_t$ and sends them to the challenger. $\mathbb{A}$ then receives the search queries generated by the challenger such that $Q_i \leftarrow ConstructQuery(K, \chi_i)$. The adversary returns 1 if his queries return the expected result, otherwise 0.*

$\underline{Ideal_{\mathbb{A},\mathbb{S}}(k)}$: *The adversary $\mathbb{A}$ outputs the tuple (D, Λ), where $\Lambda \leftarrow BuildTree(D)$, and sends it to the simulator. Given the leakage $\mathbb{L}_1$, simulator $\mathbb{S}$ generates the tuple (I, C) and sends them to the adversary. $\mathbb{A}$ makes a polynomial number of adaptive string searches $\chi = \chi_1, \ldots, \chi_t$ and sends them to the simulator. Given the leakage $\mathbb{L}_2$ the simulator $\mathbb{S}$ sends the* *appropriate search queries to the adversary. Finally,* $\mathbb{A}$ *returns* 1 *in the case of successful experiment, otherwise* 0.

We say that SSP-SSE is adaptively secure against chosen query attack if for all probabilistic polynomial time adversaries $\mathbb{A}$*, there exist a non-uniform probabilistic polynomial time simulator* $\mathbb{S}$ *such that:*

$$|Pr[Real_{\mathbb{A}}(k)] = 1 - Pr[Ideal_{\mathbb{A},\mathbb{S}}(k) = 1]| \leq negl(k) \tag{23}$$

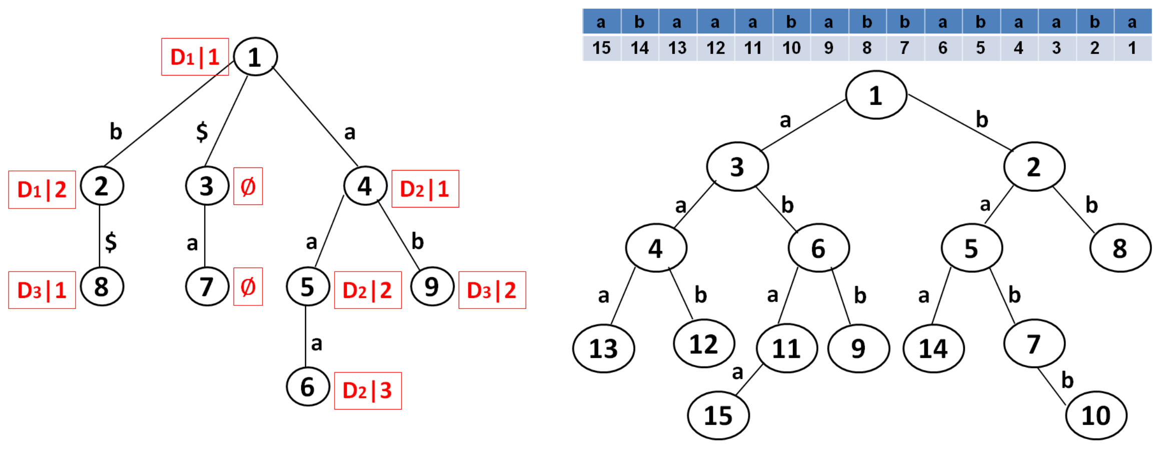
6.3.3. SSP-SSE Construction. We now present the details of proposed SSP-SSE scheme.

The scheme consists of two phases, namely *setup* phase and *search* phase. The setup phase is done once by the data owner to upload the set of encrypted documents and searchable index to the cloud provider. In this phase, the data owner uses the $KeyGen$, $BuildTree$ and *Encrypt* algorithms to encrypt the document collection as well as construct searchable index. The search phase is performed every time by the cloud user when a query is submitted. In this phase, cloud user invokes the *ConstructQuery* algorithm to generate the search query. The cloud provider executes the $Search$ algorithm to output matching results. Finally, the cloud user invokes the *Decrypt* algorithm to decrypt document collection to original view. Our scheme is based on a set of important notations shown in Figure 6.2. We outline the details of setup phase in Figure 6.5. We later show the search phase in Figure 6.6.

NOTATIONS.

- $t = (t_1, t_2, \dots, t_n)$ - the text constructed from document collection D. t_i is the letter in text t at position i.
- $\nu[i]$ - the node in Λ at index i ($i \in [1; n]$).
- $V(\nu[i])$ - the position value of node $\nu[i]$ in Λ.
- $pid(D_j) = id(D_j)||pos_{D_j}$ - concatenation of document identifier D_j ($j \in [1; l]$) with position i of character t_i in the document D_j.
- $L(\nu[i])$ - the path label of node $\nu[i]$ in position heap tree Λ.
- $L_{parent}(\nu[i])$ - the path label of $\nu[i]$'s parent node.
- $depth(\nu[i])$ - the depth of node $\nu[i]$ in Λ.
- $\overline{\nu[i]}$ - the encrypted node in $\overline{\Lambda}$ at index i.
- $\overline{V}(\overline{\nu[i]})$ - the encrypted value of node $\overline{\nu[i]}$ in $\overline{\Lambda}$.
- $\overline{L}(\overline{\nu[i]})$ - the encrypted path label of node $\overline{\nu[i]}$.
- $\overline{L_{parent}}(\overline{\nu[i]})$ - the encrypted path label $\overline{\nu[i]}$'s parent node.
- $descendants(\overline{\nu[i]})$ - the set of descendant (child) nodes in the subtree rooted at node $\overline{\nu[i]}$. If $\overline{\nu[i]}$ is the leaf node, then $descendants(\overline{\nu[i]}) = 0$.
- $ancestors(\overline{\nu[i]})$ - the set of ancestor (parent) nodes at node $\overline{\nu[i]}$. If $\overline{\nu[i]}$ is root node, then $ancestors(\overline{\nu[i]}) = 0$.

FIGURE 6.2. SSP-SSE Scheme Notations.



(A) (B)

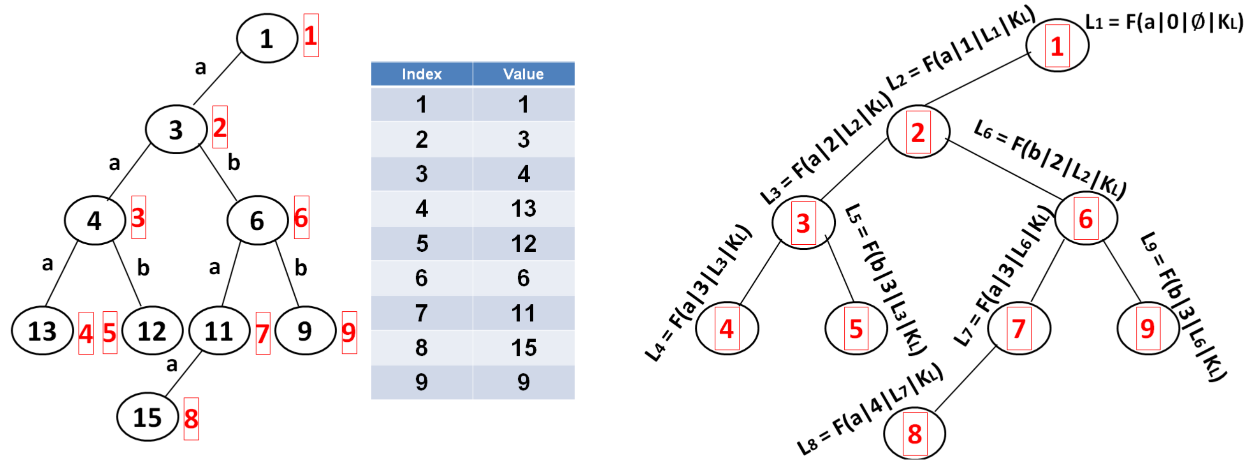
FIGURE 6.3. An example of position heap tree. (**a**) Constructed from the text "ab\$aaa\$bb"' exatracted from documents (D_1, D_2, D_3). (**b**) Constructed from the text "abaaababbabaaba".

6.3.3.1. *Setup Phase.* Let k be a security parameter and let $SKE = (Gen, Enc, Dec)$ be a PCPA-secure symmetric-key encryption scheme. The data owner begins with the $KeyGen$ algorithm that inputs a secret parameter k and outputs a set of keys K_1, K_X, K_Y, K_V

and set of random keys K_Q, K_L, K_2,$K_3 \overset{R}{\leftarrow} \{0,1\}^k$. He will use these keys to encrypt the document collection $D = (D_1, \ldots, D_l)$ and construct searchable index I.

Handling multiple documents. First, the data owner constructs a position heap tree Λ using the *BuildTree* algorithm outlined in Definition 45. The *BuildTree* algorithm inputs the text t, where t is constructed from the document collection $D = D_1||\$ \ldots \$ \ ||D_l$ padded with the unique terminator string \$, and outputs the single position heap tree Λ. In order to handle multiple documents in the collection, the data owner adds an auxiliary information to each node that contains the document identifier D_i and the position of the letter in D_i. For example, if the character "a" appears in the document D_1 at position 1, the node in Λ will have an extra information of $pid(D_1) = id(D_1)||1$. Formally, we concatenate identifier of D_j ($j \in [1;l]$) with position i of character t_i in the document D_j, i.e. $pid(D_j) = id(D_j)||pos_{D_j}$, and add this information in each node in the position heap tree. Figure 6.3a shows an example of position heap tree Λ of the text "ab\$aaa\$bb" constructed from three concatenated documents (D_1, D_2, D_3), where D_1 has a text "bb", D_2 has a text "aaa" and D_3 has a text "ab". Note, a search of "ab" in the position heap tree returns a set of nodes $(9, 4, 1)$ where only 9 is the matching node and it describes the document position of $D_3||2$. Thus, the search query "ab" appears only in the document D_3 at position 2.

Constructing searchable index. The data owner constructs the searchable index that is based on the position heap tree data structure. To present the details, we use an example of the position heap tree Λ showed in Figure 6.3*b*. The Figure depicts constructed position heap tree Λ from text $t =$ "abaaababbabaaba" and text array X (shown at the top of Figure), where each array element has a single character of text t indexed from right-to-left.



Index	Value
1	1
2	3
3	4
4	13
5	12
6	6
7	11
8	15
9	9

FIGURE 6.4. Construction of searchable index. (**a**) An example of position array Y. (**b**) An example of path label encryption of position heap tree.

The data owner begins by extracting position information from Λ as follows: index each node in tree Λ and create a position array Y such that each index in Y corresponds to the node value of Λ. Figure 6.4a shows an example of left-side branch of position heap tree Λ and constructed position array Y. In this example, nodes in Λ are marked with red-color index and their corresponding values (positions) are stored as elements in Y. (In Figure 6.4a we show an example of the position array Y for 9 nodes of Λ for demonstration purpose only. The actual algorithm is executed on all nodes in Λ.) With this, the data owner is ready to encrypt the position heap tree Λ, text array X and position array Y data structures.

First, to encrypt the position heap tree Λ, the data owner uses a pseudorandom function $F : \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^k$ and PCPA-secure symmetric-key encryption scheme SKE = (Gen, Enc, Dec). For each node i in Λ, the data owner applies PRF F with key K_Q on concatenation of the path label of node i, depth of the node i, the encrypted path label of i'th parent node and the secret key K_L. Figure 6.4b shows an example of path label encryption. For instance, the label of node 4 is $L_4 = F_{K_Q}(a||3||L_3||K_L)$, where the $L_3 = F_{K_Q}(a||2||L_2||K_L)$. The root path label is special case and its label $L_1 = K_{K_Q}(a||0||\emptyset||K_L)$.

Let $SKE = (Gen,\ Enc,\ Dec)$ be a PCPA-secure symmetric-key encryption scheme, let $F : \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^k$ be a PRF, and let $P : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a PRP.

SETUP PHASE.

$\underline{KeyGen(1^k)}$: given the security parameter k, generate $K_1,\ K_X,\ K_Y,\ K_V \leftarrow SKE.Gen(1^k)$ and $K_Q,\ K_L,\ K_2, K_3 \xleftarrow{\text{R}} \{0,1\}^k$. Output the key set $K = (K_1,\ K_X,\ K_Y,\ K_V,\ K_Q,\ K_L,\ K_2, K_3)$.

$\underline{BuildTree(D)}$: given the document collection $D = (D_1,\ \ldots,\ D_l)$:

(1) construct text $t = t_1 t_2 \ldots t_n$ from document collection D and and input t of size n to build the position heap tree Λ.

(2) index into Λ, for each node $\nu[i]$ $(i \in [1,\ n])$:

  (a) set $V(\nu[i]) = pid(D_j)||V(\nu[i])$, where D_j $(j \in [1, l])$ is the document in collection D.

(3) output the position heap tree Λ

$\underline{Encrypt(K, \Lambda, D)}$: given the secret key set K, position heap tree Λ and the set of documents $D = (D_1,\ \ldots,\ D_l)$.

Build Encrypted Tree:

(1) index into Λ, traverse from the root node:

(2) for each node $\nu[i]$ $(i \in [1,\ n])$:

  (a) set $\overline{L}(\overline{\nu[i]}) = F_{K_Q}(L(\nu[i])||depth(\nu[i])||\overline{L_{parent}}(\overline{\nu[i]})||K_L)$ (i.e. apply PRF F with key K_Q on concatenation of the path label L of $\nu[i]$, depth of the node $\nu[i]$, encrypted parent label $\overline{L_{parent}}$ of $\nu[i]$ and the secret key K_L.

  (b) set $\overline{V}(\overline{\nu[i]}) = SKE.Enc_{K_V}(i)$.

(3) output encrypted $\bar{\Lambda}$.

Build Encrypted Arrays:

(1) for each character t_i of t indexed from right-to-left (i.e, $t_n t_{n-1} \ldots t_1$), set an array $X[P_{K_2}(i)] = SKE.Enc_{K_X}(t_i)$.

(2) for each $i = [1,\ n]$: set an array $Y[P_{K_3}(i)] = SKE.Enc_{K_Y}(V(\nu[i]))$.

Encrypt Document Collection:

(1) for each document D_i where $i \in [1, l]$, let $C_i \leftarrow SKE.Enc_{K_1}(D_i)$.

(2) output $C = (C_1,\ C_2,\ \ldots,\ C_l)$.

Output: index $I = (\bar{\Lambda},\ X,\ Y)$ and encrypted document collection $C = (C_1,\ C_2,\ \ldots,\ C_l)$.

FIGURE 6.5. SSP-SSE Setup Phase.

In this way, the data owner encrypts all path labels in the tree. This hides the plaintext path labels of same character at different levels of the tree Λ. Moreover, this makes the ciphertext unique for all path labels in the tree. To hide the index information of each node in Λ, the data owner uses SKE encryption with key K_V on the index of node, i.e $V_i =$

$SKE.Enc_{K_V}(i)$, where $i \in [1, n]$. For instance, the value of node 8 is $V_8 = SKE.Enc_{K_V}(8)$. With no plaintext left in Λ, the data owner outputs an encrypted position heap tree $\bar{\Lambda}$.

Second, the data owner utilizes a pseudorandom permutation $P : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ and PCPA-secure symmetric-key encryption SKE to hide plaintext elements of text array X and position array Y. For each i ($i \in [1, n]$) in X, the data owner applies PRP P with secret key K_2 on each i, i.e. $P_{K_2}(i)$. For each corresponding character t_i at index i in X, he applies SKE with secret key K_X on character t_i, i.e. $SKE.Enc_{K_X}(t_i)$. The data owner sets the encrypted array X as $X[P_{K_2}(i)] = SKE.Enc_{K_X}(t_i)$. Next, for each i ($i \in [1, n]$) in Y, he utilizes PRP P with secret key K_3 and SKE with secret key K_Y as follows: $Y[P_{K_3}(i)]$ $= SKE.Enc_{K_Y}(V_i)$, where V_i is i'th element in Y.

Finally, the data owner encrypts each document D_i in the collection D using PCPA-secure symmetric-key encryption scheme SKE with secret key K_1 to produce the encrypted document $C_i \leftarrow SKE.Enc_{K_1}(D_i)$. After all, the data owner uploads the encrypted collection C along the searchable index $I = (\bar{\Lambda}, X, Y)$ to the cloud provider. Now the collection is available for selective cloud retrieval.

6.3.3.2. *Search Phase.* The search phase includes both the *ConstructQuery* and the *Search* interactive algorithms that are executed between the cloud user and the cloud provider. The cloud user keeps the set of secret keys $K = (K_1,\ K_X,\ K_Y,\ K_V,\ K_Q,\ K_L,$ $K_2,K_3)$ received from the data owner.

In order to search a substring χ of length m the cloud user begins with creating a search query Q: for each character χ_i in χ, he applies PRF F with secret key K_Q on concatenation of χ_i, i, output of previous query Q_{i-1} and the secret parameter K_L. The cloud user forms a query $Q = (Q_1, \ldots, Q_l)$ and sends Q to the cloud provider. For instance, for a substring

SEARCH PHASE.

$\underline{[(Q) \leftarrow ConstructQuery(K, \chi)] \leftrightarrow [(L) \leftarrow Search(I, Q)]}$ is an interactive protocol between the cloud user and the cloud provider. The cloud user keeps the key set $K = (K_1, K_X, K_Y, K_V, K_Q, K_L, K_2, K_3)$ and query cloud provider for a substring χ. The cloud provider executes search on searchable index $I = (\bar{\Lambda}, X, Y)$ and returns results back to the cloud user.

(1) cloud user: given the secret key K_Q and the string of interest χ, output the search query Q as follows:
  (a) for each character χ_i, $i \in [1; m]$, where $m = |\chi|$
    (i) set $Q_i = F_{K_Q}(\chi_i || i || Q_{i-1} || K_L)$ (i.e., apply PRF F with key K_Q on concatenation of character χ_i of χ, i, output of query Q_{i-1} and secret key K_L.)
    (ii) set $Q = (Q_1, Q_2, \ldots, Q_m)$.
  (b) send search query Q to the cloud provider.
(2) cloud provider: index into $\bar{\Lambda}$, start at root node
  (a) for each Q_i and each node $\overline{\nu}$ in $\bar{\Lambda}$, match the encrypted label $\overline{L}(\overline{\nu})$ to Q_i. Continue until longest node $\overline{\nu[match]}$ found.
  (b) If $\overline{\nu[match]} \neq \perp$, return $(descendants(\overline{\nu[match]}, ancestors(\overline{\nu[match]}))$, otherwise return $\perp$.
(3) cloud user: let TMP-AN and TMP-DE be two arrays, let TMP-RES = TMP-AN + TMP-DE is an array that combines elements from TMP-AN and TMP-DE.
  (a) for each node $\overline{\nu}$ in $ancestors(\overline{\nu[match]})$:
    (i) if $SKE.Dec_{K_V}(\overline{V}(\overline{\nu})) = \perp$, abort. Otherwise output idx, add to TMP-AN.
  (b) for each node $\overline{\nu}$ in $descendants(\overline{\nu[match]})$:
    (i) if $SKE.Dec_{K_V}(\overline{V}(\overline{\nu})) = \perp$, abort. Otherwise output idx, add to TMP-DE.
  (c) set TMP-RES = TMP-AN + TMP-DE, for each idx in TMP-RES, set $y_{idx} = P_{K_3}(idx)$. Send $(y_1, \ldots, y_{num})$ to the cloud provider.

FIGURE 6.6. SSP-SSE Search Phase Part 1.

"aba", the cloud user creates $Q_1 = F_{K_Q}(a|1|L_{root}|K_L)$ (L_{root} is shared by the data owner to the cloud user), $Q_2 = F_{K_Q}(b|2|Q_1|K_L)$, $Q_3 = F_{K_Q}(a|3|Q_2|K_L)$ and sends $Q = (Q_1, Q_2, Q_3)$ to the cloud provider. The cloud server indexes into the encrypted position heap tree $\bar{\Lambda}$, and for each given Q_i it matches encrypted label $\overline{L}$ of each node in $\bar{\Lambda}$ to Q_i, and continues until the longest matching node $\overline{\nu_{match}}$ in $\bar{\Lambda}$ found. The cloud server returns the set of ancestor and descendant nodes of $\overline{\nu_{match}}$ to the cloud user. Using the example in Figure 6.4b and

SEARCH PHASE.

$\underline{[(Q) \leftarrow ConstructQuery(K, \chi)] \leftrightarrow [(L) \leftarrow Search(I, Q)]}$ is an interactive protocol between the cloud user and the cloud provider. The cloud user keeps the key set $K = (K_1, K_X, K_Y, K_V, K_Q, K_L, K_2, K_3)$ and query cloud provider for a substring χ. The cloud provider executes search on searchable index $I = (\bar{\Lambda}, X, Y)$ and returns results back to the cloud user.

(4) cloud provider: get $Y_i = Y[y_i]$ $(i \in [1, num])$, output $(Y_1, \ldots, Y_{num})$.

(5) cloud user: let AN and DE be two arrays.

  (a) for $i = [1, m]$, if $SKE.Dec_{K_Y}(Y_i) = \perp$, abort, otherwise add output to AN.

  (b) for $i = [m+1, num]$, if $SKE.Dec_{K_Y}(Y_i) = \perp$, abort, otherwise add output to DE.

  (c) parse each element from AN as $pid(D)||pos$.

  (d) for each pos in AN, for $j = pos$, $j > (j - m)$ (where $(j - m) > 0$), $j--$, let $x_j = P_{K_2}(j)$, send $(x_1, \ldots, x_h)$ to the cloud provider.

(6) cloud provider: get $X_i = X[x_i]$ $(i \in [1, h])$, output $(X_1, \ldots, X_h)$.

(7) cloud user: let REAL-AN be an array.

  (a) for $i = [1; h]$, if $SKE.Dec_{K_X}(X_i) = \perp$, abort. Otherwise parse the output as t_j.

  (b) for each pos in AN, compare characters $\chi_u = t_j$, where $u = 0$, $u < m$, $u++$ and $j = pos$, $j > (j - l)$ (where $(j - l) \not< 0$), $j--$. If all $\chi_u = t_j$ match at given pos, add pos to REAL-AN, otherwise ignore pos.

  (c) let RES = REAL-AN + DE. Parse each element of array RES as $id(D_h)||pos_{D_h}$, where pos_{D_h} is the position of substring χ in document D_h $(h \in [1, l])$.

$\underline{Decrypt(K_1, K_2, C_i):}$

(1) retrieve set $C = (C_1, \ldots, C_k)$ from the cloud provider.

(2) $D_i \leftarrow SKE.Dec_{K_1}(C_i)$, where $i \in [1; k]$.

(3) output $((D_1, pos_{D_1}), \ldots, (D_k, pos_{D_k}))$.

Figure 6.7. SSP-SSE Search Phase Part 2.

search query $Q = (Q_1, Q_2, Q_3)$, the cloud provider returns the set of encrypted ancestor nodes $(SKE.Enc_{K_V}(1), SKE.Enc_{K_V}(2), SKE.Enc_{K_V}(6))$ and set of encrypted descendant nodes $(SKE.Enc_{K_V}(7), SKE.Enc_{K_V}(8))$.

Now, the cloud user applies SKE scheme with secret key K_V to decrypt the ancestor and descendant nodes, i.e. $(1, 2, 6)$ ancestor nodes and $(7, 8)$ descendant nodes. Next, he uses PRP P with secret key K_3 on each decrypted node, i.e. $y_{idx} = P_{K_3}(idx)$, where idx is $(1, 2, 6, 7, 8)$ and sends the resulted query y to the cloud provider. The cloud provider uses array Y to fetch the elements at index y_i $(i \in [1; 5])$ as $Y[y_i]$ and sends back results.

Once received, the cloud user applies SKE with secret key K_Y to decrypt the positions in ancestor and descendant nodes, i.e, $(1, 3, 6)$ positions in ancestor nodes and $(11, 15)$ positions in descendant nodes. According to Definition 46 descendant nodes $(11, 15)$ are the positions of query “aba” in the text and ancestor nodes $(1, 3, 6)$ require an inspection since some of them can point at “aba” in the text. Note, since the substring “aba” has length of 3, the substring may exist at positions $(6, 5, 4)$ and $(3, 2, 1)$ in the text. So, to launch the inspection, the cloud user applies PRP P with secret key K_X at on each position $(6, 5, 4, 3, 2, 1)$ as $x_{idx} = P_{K_2}(idx)$ and sends query x to the cloud provider.

Now, the cloud provider uses array X and sends back the elements of the array at index $X[x_i]$ $(i \in [1; 6])$. The cloud user uses $SKE.Enc$ with secret key K_X to decrypt the characters t_j at positions $(6, 5, 4, 3, 2, 1)$ (i.e. received characters are (a, b, a, a, b, a)). Using this information, the cloud user verifies if substring characters χ_i match received characters t_j at each ancestor position. The inspection of ancestors shows that only $(6, 3)$ are the positions. Thus, the cloud user concludes that substring query “aba” is at position $(3, 6, 11, 15)$ in the text.

Note, if multiple documents involved in the original text construction, ancestor and descendant nodes contain the document identifiers, which can be later used by the cloud user to download the matching encrypted documents and decrypt them locally using PCPA-secure symmetric-key encryption SKE with secret key K_1.

6.4. Evaluation

6.4.1. Security Analysis. In this section we focus on the the security of SSP-SSE scheme. First, we show that the SSP-SSE scheme is correct according to the Definition

48. Second, we prove that SSP-SSE scheme is secure against chosen-query attack (CQA-2) executed by adaptive adversary according to the Definition 52.

THEOREM 4. ***(Correctness).*** Substring Positions Searchable Symmetric Encryption (SSP-SSE) scheme consisting of six polynomial-time algorithms ($KeyGen$, $BuildTree$, $Encrypt$, $ConstructQuery$, $Search$, $Decrypt$) is correct according to Definition 48.

PROOF. The $Search$ algorithm inputs the searchable index I and the search query Q. The index I consists of the encrypted position heap tree $\bar{\Lambda}$ and two arrays X, Y (both encrypted). Since the path labels in $\bar{\Lambda}$ and the search query Q are both encrypted with the same instance of pseudorandom function F with same secret key K_Q, the correctness of SSP-SSE scheme relies on the correctness of pseudorandom function.

When the cloud provider receives the search query Q, it traverses the path labels in the encrypted position heap tree $\bar{\Lambda}$ according to Definition 46. Search query Q is constructed using the pseudorandom function F applied on the substring χ with key K_Q. Each encrypted path label in $\bar{\Lambda}$ is constructed using the pseudorandom function F with key K_Q on set of characters extracted from the plaintext document collection $D = \{D_1, \ldots, D_l\}$. Therefore, the search algorithm outputs true if the document D_i contains the string of characters χ. Thus, the cloud provider outputs a set documents that match the search query Q. $\square$

THEOREM 5. ***(Security).*** Let SKE be a symmetric PCPA-secure encryption scheme, F be a pseudorandom function and P be a pseudorandom permutation. Substring Position Searchable Symmetric Encryption (SSP-SSE) presented above is ($\mathbb{L}_1$, $\mathbb{L}_2$)-adaptively secure against chosen-query attacks defined in Definition 52 (CQA-2 security), where $\mathbb{L}_1$ and $\mathbb{L}_2$ are the possible leakages.

In a nutshell, the proof of security of SSP-SSE scheme works as follows. The simulator $\mathbb{S}$ generates a simulated searchable index $\widetilde{I}$ that consist of simulated encrypted position heap tree $\widetilde{\Lambda}$, simulated position array $\widetilde{Y}$ and simulated text array $\widetilde{X}$, i.e. $\widetilde{I} = (\widetilde{\Lambda}, \widetilde{Y}, \widetilde{X})$; and simulated set of ciphertexts $\widetilde{C} = \{\widetilde{C_1}, \ldots, \widetilde{C_l}\}$. Both $\widetilde{I}$ and $\widetilde{C}$ are constructed using the leakage $\mathbb{L}_1$ that disclose number of encrypted documents, size of encrypted documents and identifier of each encrypted document. The simulated encrypted position heap tree $\widetilde{\Lambda}$ is constructed using the pseudorandom function F and symmetric-key encryption SKE with random values $\{0, 1\}$. Both simulated $\widetilde{Y}$ and $\widetilde{X}$ are constructed using the pseudorandom permutation P and symmetric-key encryption SKE on random values $\{0, 1\}$. The security of proposed scheme relies on the following assumptions. The pseudorandomness of F guarantees that the simulated encrypted position heap tree $\widetilde{\Lambda}$ is indistinguishable from the real encrypted position heap tree $\bar{\Lambda}$. The pseudorandomness of P will guarantee that simulated $\widetilde{Y}$ and $\widetilde{X}$ are indistinguishable from the real Y and X. Moreover, simulated set of ciphertext $\widetilde{C}$ is indistinguishable from the real encrypted document collection C.

The search algorithm is simulated in similar way that requires to keep track of different dependencies between the result output and the search query. However, since the real search query is constructed with pseudorandom function F and pseudorandom permutation P, the simulator is not able to distinguish it from the simulated query. Similarly, simulated outcome of search is indistinguishable from the real set of nodes. We outline the formal proof as follows.

Proof. Polynomial-size simulator $\mathbb{S}$ can be defined such that for any challenger and any polynomial-time adversary $\mathbb{A}$, the outputs of two experiments $Ideal_{\mathbb{A},\mathbb{S}}(k)$ and $Real_{\mathbb{A}}(k)$ with secret parameter k are computationally indistinguishable according to the Definition 52. We now describe the details of experiment $Ideal_{\mathbb{A},\mathbb{S}}(k)$ that presents the simulator $\mathbb{S}$.

- $\mathbb{S}(1^k, \mathbb{L}_1)$: The simulator $\mathbb{S}$ has a leakage $\mathbb{L}_1$ which gives the simulator information about the number and size of documents as well as identifier of each encrypted document. The simulator $\mathbb{S}$ randomly generates a set of simulated ciphertexts $\widetilde{C}$ and simulated searchable index $\widetilde{I}$ as follows:
  - Simulator $\mathbb{S}$ outputs the set of ciphertexts $\widetilde{C} = \{\widetilde{C_1}, \ldots, \widetilde{C_l}\}$, where $\widetilde{C}_i \overset{R}{\leftarrow} \{0,1\}^{|D_i|}$.
  - Simulator $\mathbb{S}$ sets the simulated encrypted position heap tree $\widetilde{\Lambda}$ where each node is set as $\widetilde{V}(\nu[i]) \overset{R}{\leftarrow} \{0,1\}^k$ and each path label of node $\nu[i]$ is set as $\widetilde{L}(\nu[i]) \overset{R}{\leftarrow} \{0,1\}^k$, where $i \in [1;n]$. The simulator outputs the encrypted position heap tree $\widetilde{\Lambda}$.
  - Simulator $\mathbb{S}$ now constructs simulated arrays $\widetilde{X}$ and $\widetilde{Y}$: $\widetilde{X}[i] = \{0,1\}^k$ and $\widetilde{Y}[i] = \{0,1\}^k$, where $i \in [1;n]$.
  - Simulator $\mathbb{S}$ outputs simulated searchable index $\widetilde{I} = (\widetilde{\Lambda}, \widetilde{Y}, \widetilde{X})$ and set of simulated ciphertexts $\widetilde{C}$.

At this point, the simulator $\mathbb{S}$ generated the set of simulated encrypted documents $\widetilde{C}$ and simulated index $\widetilde{I}$. Next, the adversary $\mathbb{A}$ adaptively queries the polynomial-size simulator $\mathbb{S}$ as follows.

- $\mathbb{S}(1^k, \mathbb{L}_1, \mathbb{L}_2)$: The adversary $\mathbb{A}$ sends a new query Q to the simulator $\mathbb{S}$. The simulator now starts collecting various dependencies between incoming search query and resulted output.
  - With given search query Q, simulator $\mathbb{S}$ traverses the simulated encrypted position heap tree $\widetilde{\Lambda}$ starting from the root node, following the simulated path

labels to find the set of matching encrypted nodes in $\widetilde{\Lambda}$. The simulator outputs the set of simulated matching nodes: $\widetilde{ancestors}$ and $\widetilde{descendants}$.

– With given search requests $(\widetilde{y}_1, \ldots, \widetilde{y}_{num})$, the simulator performs a search in simulated array $\widetilde{Y}$ and returns matching elements $(\widetilde{Y}_1, \ldots, \widetilde{Y}_{num})$.

– With given search requests $(\widetilde{x}_1, \ldots, \widetilde{x}_h)$, the simulator performs a search in simulated array $\widetilde{X}$ and returns matching elements $(\widetilde{X}_1, \ldots, \widetilde{X}_h)$.

We now need to show that the outputs of two experiments $Ideal_{\mathbb{A},\mathbb{S}}(k)$ and $Real_{\mathbb{A}}(k)$ are indistinguishable. Since the simulator generates randomly set of ciphertexts $\widetilde{C}$, the output of the simulator is truly indistinguishable from the real ciphertexts that are generated with PCPA-secure symmetric encryption scheme SKE scheme using secret key K_1. Otherwise, this would mean that simulator could distinguish between the output of PCPA-secure symmetric encryption scheme SKE and the random value. Next, the simulated encrypted position heap tree $\widetilde{\Lambda}$ is truly indistinguishable from the real encrypted position heap tree. Otherwise, this would mean that simulator could distinguish between the output of pseudorandom function F with secret key K_Q, and the random values. Similarly, the simulated arrays $\widetilde{Y}$ and $\widetilde{X}$ are truly indistinguishable from the real arrays Y and X. Otherwise, this would mean the simulator can distinguish between the output of pseudorandom permutation P with keys K_2, K_3, SKE scheme with keys K_Y, K_X and the random values. Thus, this concludes that the outputs of two experiments are indistinguishable. □

6.4.2. Performance Analysis. In this section we outline the performance of proposed solution. We assume encryption and decryption using SKE scheme take $O(k)$ time, where k is the security parameter. We also assume element selection from array takes $O(1)$ time.

Encryption Efficiency. We first focus on the encryption efficiency of SSP-SSE scheme. Given plaintext position heap tree Λ with n nodes, we compute encrypted position heap tree Λ using SKE in $O(kn)$ time. The arrays X and Y each have n elements and can be computed in $O(kn)$ time. Therefore, encryption takes $O(kn)$ time and the total ciphertext is $O(kn)$ size.

Search Protocol Efficiency. We now analyze the efficiency of proposed search algorithm. The cloud user inputs a substring χ of length m and outputs a search query in $O(m)$ time. The cloud provider uses $\bar{\Lambda}$ and performs m matches in the tree and retrieves *occ* descendant nodes, in $O(m + occ)$ time. The cloud user then computes $y_1, \ldots, y_{m+occ}$ elements and the cloud provider retrieves $Y[y_1], \ldots, Y[y_{m+occ}]$ in $O(m+occ)$ time. The cloud user then computes $x_1, \ldots, x_{m^2}$ elements (the cloud user wants to inspect m ancestor positions and the substring χ of m length may appear at each ancestor position) and the cloud provider retrieves $X[x_1], \ldots, X[x_{l^2}]$ in $O(m^2)$ time. Now the cloud user performs an inspection of m ancestors m times, making execution in $O(m^2)$ time. Thus, both the cloud user and the cloud provider take computation time $O(m^2 + occ)$ in the query protocol and three rounds of communication to complete the execution of protocol.

Experimental Results. We have developed and implemented a proof-of-concept prototype of *SSP-SSE* scheme using C++ language. Our prototype leverages *libtomcrypt* cryptographic library [5] that is portable C cryptographic library that supports symmetric ciphers, one-way hashes, pseudo-random number generators, and a plethora of support routines. We use *libtomcrypt* to build the searchable index I and encrypt the document collection. We utilize *AES-CTR* encryption for SKE symmetric-key encryption scheme, *HMAC-SHA1* for pseudorandom function F and DES encryption for pseudorandom permutation P.

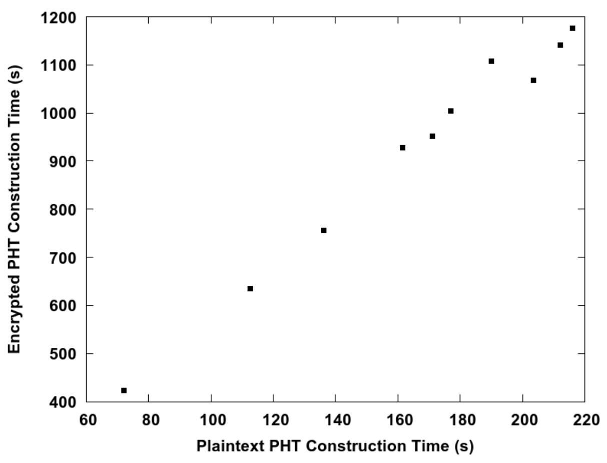
[5]https://github.com/libtom/libtomcrypt

Table 6.2. Experimental Database.

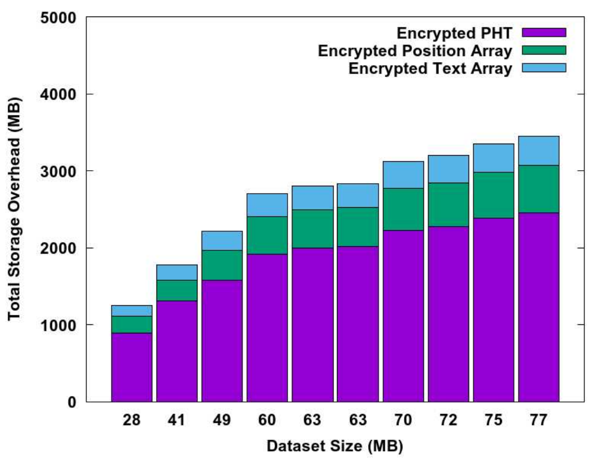
Organism Name	Description	mRNA Size (MB)	Organism Name	Description	mRNA Size (MB)
Dufourea Novaeangliae		28	Papilio Polytes		41
Bactrocera Dorsalis		49	Fopius Arisanus		60
Halyomorpha Halys		63	Tribolium Castaneum		63
Stomoxys Calcitrans		70	Orussus Abietinus		72
Nasonia Vitripennis		75	Linepithema Humile		77

We show a thorough experimental evaluation of the *SSP-SSE* scheme on a real-world dataset: the Genome database[78] (published by National Center for Biotechnology Information, National Institutes of Health) that contains sequence data from the whole genomes of over 1000 species or strains. The database include all three main domains of life (bacteria, archaea, and eukaryota) as well as many viruses, phages, viroids, plasmids, and organelles. All experiments have been performed on a 6 core Intel(R) Xeon(R) E5645 @ 2.40GHz processor and 98 GB memory running 64-bit Fedora 23. The cloud server, data owner and cloud user applications were run on the same machine, as the network communication overhead was assumed to be negligible.

For our experiments we pick large mRNA transcript datasets of various insects. Table 6.2 shows the details of experimental set. Figure 6.8a shows an overhead of constructing encrypted position heap tree Λ. We compare the time of construction of plaintext position heap tree (original algorithm) and the encrypted position heap tree proposed in this work. Figure 6.8b shows a storage overhead of searchable index I that consists of encrypted position heap tree Λ, position array Y and text array X. In short, we notice that proposed scheme



(A)



(B)

Figure 6.8. Experimental Results. (**a**) A construction of position heap tree. (**b**) A searchable index storage.

adds insignificant overhead on computation time, however its storage overhead depends on the block cipher size of underlying encryption schemes. We believe that proposed solution can be easily deployed in a real-world cloud environment.

6.5. Conclusion

In this work, we present a new substring position searchable symmetric encryption scheme (SSP-SSE) that allows efficient substring search on encrypted documents outsourced to the cloud. Specifically, our solution efficiently finds occurrences and positions of a substring over encrypted cloud data. We formally define the leakage functions and security notions of SSP-SSE. We show that our scheme is secure against chosen-query attacks executed by an adaptive adversary.

CHAPTER 7

Multi-User Substring Position Searchable Symmetric Encryption (MSSP-SSE)

The original system model shown in Figure 3.1 (Chapter 3.1) includes only 3 single entities. In this chapter we consider a simple extension to our system model, where a data owner has a document collection and there is a group of data users that want to query encrypted data at the cloud provider. The data owner can control the search access by granting and revoking searching privileges to other users. We propose a new multi-user substring position search symmetric encryption (MSSP-SSE) scheme that solves the problem of managing access privileges and searching a substring over encrypted cloud data. First, we present the definitions of a multi-user substring position search symmetric encryption scheme. Later we give an efficient construction that combines ideas of a single-user SSP-SSE scheme with a broadcast encryption scheme.

7.1. Scheme Construction

7.1.1. Preliminaries.

We begin with definition of Broadcast Encryption scheme.

Definition 53. **(Broadcast Encryption (BE)[62]).** *Let $\mathbb{U}$ denote the user space, i.e. the set of all possible users in the system. A broadcast encryption scheme is a set of four polynomial-time algorithms $BE = (Gen, Enc, Add, Dec)$ defined as follows:*

- *Gen is a probabilistic algorithm that takes as input a security parameter k and outputs a master key mk.*

- *Enc is a probabilistic algorithm that takes as input a master key mk, a set of users $G \subseteq \mathbb{U}$ and a message m. The algorithm outputs a ciphertext c.*
- *Add is a probabilistic algorithm that takes as input a master key mk and a user identifier $U \subseteq \mathbb{U}$, and outputs a user key uk_U.*
- *Dec is a deterministic algorithm that takes as input a user key uk_U and a ciphertext c and outputs either an original message m or the failure $\perp$.*

The broadcast encryption scheme BE is secure if its ciphertext leaks no useful information about the original message to any user not in G.

7.1.2. Algorithm Definitions.

Definition 54. **(Multi-User Substring Position Searchable Symmetric Encryption (MSSP-SSE).** *A tree-based MSSP-SSE scheme over a set of documents D is tuple of eight polynomial-time algorithms ($KeyGen$, $BuildTree$, $Encrypt$, $AddUser$, $RemoveUser$, $ConstructQuery$, $Search$, $Decrypt$), as follows:*

- *$(K, mk) \leftarrow KeyGen(1^k)$: a probabilistic key generation algorithm to setup the SSP-SSE scheme. The algorithm takes a secret parameter k and outputs a set of secret keys K and mk.*
- *$(\Lambda) \leftarrow BuildTree(D)$: a deterministic algorithm to build a position heap tree Λ. The algorithm takes a document collection $D = \{D_1, \ldots, D_l\}$ and constructs a position heap tree Λ.*
- *$(I, C, st_S, st_O) \leftarrow Encrypt(K, \Lambda, D, G)$: a probabilistic algorithm to encrypt a position heap tree and document corpus. The algorithm inputs a set of secret keys K, a position heap tree Λ, a documents corpus D and a set of authorized users $G \subseteq \mathbb{U}$.*

The output of algorithm is a searchable index I, an encrypted collection $C = \{C_1, \ldots, C_l\}$, a set of states st_S, st_O.

- $(uk_U) \leftarrow AddUser(mk, U)$*: is a probabilistic algorithm run by the data owner to add a data user. It takes an input of a key mk and a user identifier U, and it outputs a secret key uk_U used by a data user.*
- $(st_S, st_O) \leftarrow RemoveUser(mk, U)$*: is a probabilistic algorithm run by the owner to remove a data user U from group G. The algorithm takes as input a key mk and a data user identifier U. It outputs a set of updated states st_S and st_O.*
- $[(Q) \leftarrow ConstructQuery(K, \chi, uk_U, st_S)] \leftrightarrow [(L) \leftarrow Search(I, Q, st_S, uk_U)]$*: two deterministic algorithms that are executed interactively between the authorized cloud user and the cloud provider. The algorithm inputs a set of secret keys uk_U, K, a state st_S and a substring χ, and it outputs a search query Q. The algorithm uses a query Q, searchable index I, an input of st_S state and secret uk_U. It outputs a sequence of identifiers $L \in C$.*
- $(D_i, pos_{D_i}) \leftarrow Decrypt(K, C_i)$*: a deterministic algorithm that takes a set of secret keys K and a ciphertext C_i as input, and it outputs an original document D_i, $\forall i \in [1; n]$, and a set of χ's positions pos_{D_i} in D_i.*

The security model of multi-user substring position search symmetric encryption scheme can be defined similarly to a single-user SSP-SSE scheme: given a searchable index I, set of encrypted documents $C = \{C_1, \ldots, C_l\}$ and set of incoming search queries $Q = \{Q_1, \ldots, Q_m\}$ to the adversary, no valuable information is leaked from a tuple of eight polynomial-time algorithms to the adversary beyond what can be inferred from the access, search and path patterns.

```
$KeyGen(1^k)$ :
    (1) generate $K \leftarrow SSP\text{-}SSE.KeyGen(1^k)$.
    (2) generate $mk \leftarrow BE.Gen(1^k)$.
Output the key set $K$ and $mk$.
$BuildTree(D)$ :
Given a document collection $D = \{D_1, \ldots, D_l\}$, output $\Lambda \leftarrow SSP\text{-}SSE.BuildTree(D)$.
$Encrypt(K, \Lambda, D, G)$ :
    (1) set $(I, C) \leftarrow SSP\text{-}SSE.Encrypt(K, \Lambda, D)$.
    (2) set $st_S \leftarrow BE.Enc(mk, G, r)$, where $r \overset{R}{\leftarrow} \{0,1\}^k$ and group $G$ include the cloud
        provider.
    (3) set $st_O = r$.
Output $(I, C)$, $st_S$ and $st_O$.
$AddUser(mk, U)$ :
    (1) calculate $uk_U \leftarrow BE.Add(mk, U)$.
Output $uk_U$.
$RemoveUser(k, mk, U)$ :
    (1) sample $r \overset{R}{\leftarrow} \{0,1\}^k$.
    (2) calculate new $st_S \leftarrow BE.Enc(mk, G \setminus U, r)$ and $st_O = r$.
Output $st_S$ and $st_O$.
$[(Q) \leftarrow ConstructQuery(K, \chi, uk_U, st_S)] \leftrightarrow [(L) \leftarrow Search(I, Q, st_S, uk_U)]$
    (1) cloud user:
        (a) get $st_S$ from the cloud provider.
        (b) if $BE.Dec(uk_U, st_S) = \perp$, output $\perp$, else calculate $r \leftarrow BE.Dec(uk_U, st_S)$.
        (c) calculate $Q' \leftarrow SSP\text{-}SSE.ConstructQuery(K, \chi))$ and $Q \leftarrow \rho_r(Q')$.
    (2) cloud provider:
        (a) compute $r \leftarrow BE.Dec(uk_U, st_S)$.
        (b) calculate $Q' \leftarrow \rho_r^{-1}(Q)$.
        (c) get $L \leftarrow SSP\text{-}SSE.Search(I, Q')$, where $L \in C$.
        (d) output $L$.
$Decrypt(K, C_i)$ :
Output $(D_i, pos_{D_i}) \leftarrow SSP\text{-}SSE.Decrypt(K, C_i)$.
```

FIGURE 7.1. MSSP-SSE Construction.

7.1.3. MSSP-SSE CONSTRUCTION. Figure 7.1 shows the details of our multi-user scheme MSSP-SSE = (*KeyGen*, *BuildTree*, *Encrypt*, *AddUser*, *RemoveUser*, *ConstructQuery*, *Search*, *Decrypt*). Let SSP-SSE = (*KeyGen*, *BuildTree*, *Encrypt*, *ConstructQuery*, *Search*, *Decrypt*) be a single-user substring position searchable symmetric encryption scheme. Let $BE = (Gen, Enc, Add, Dec)$ be a broadcast encryption scheme. Let U denote the set of all users and let $G \in U$ denote the set of users authorized to search. Let ρ be a pseudorandom permutation such that $\rho : \{0,1\}^k \times \{0,1\}^t \rightarrow \{0,1\}^t$ [6], where t is the size of search query Q in SSP-SSE scheme.

The MSSP-SSE construction works as follows. First, the data owner samples the secret parameter k and generates the set of encryption keys K. Also, the data owner samples the secret key r for the pseudorandom permutation ρ and the master key mk for the broadcast encryption BE. Next, the data owner encrypts the document corpus with PCPA-secure symmetric encryption scheme SKE and outputs the searchable index I. It then generates a cloud provider state st_S, which is the output of a broadcast encryption BE that takes an input of secret key r, master key mk and the set of identities G. The data owner stores the searchable index I, the encrypted document corpus C and the cloud provider state st_S at the cloud provider. In order to allow the user U to search the remote collection, the data owner generates a user key uk_U using broadcast encryption scheme with master key mk and user's identity U.

In order to search for a substring χ, the authorized user first contacts the cloud provider to receive the latest state st_S and uses its user key uk_U to output the secret key r. It then constructs a single-user search query Q' and encrypts it with pseudorandom permutation ρ with r, and outsources $\rho_r(Q')$ to the cloud provider. The cloud provider recovers the search query Q' by computing $\rho_r^{-1}(\rho_r(Q'))$. Here, the key r is only known by the data owner and the set of authorized users that includes the cloud provider.

If user U is no longer the authorized user in the system, the data owner samples a new key r', generates new cloud provider state st'_S. The new state st'_S is sent to the cloud provider to replace the old st_S. Newly generated search queries utilize r' in pseudorandom permutation

[6] ρ can be constructed as pseudorandom permutation over domains of arbitrary size [79].

ρ and ρ^{-1}, thus no unauthorized users are able to output a valid search queries to the cloud provider.

MSSP-SSE utilizes the security and performance of a single-user SSP-SSE scheme. Our construction is very efficient since the cloud provider needs only to execute a pseudorandom permutation to evaluate the access privileges, thus eliminating the need of more expensive authentication protocols.

7.2. Conclusion

In this chapter we consider a natural extension of SSP-SSE scheme, where an arbitrary group of data users can submit substring queries to search the encrypted collection. We formally define a Multi-User Substring Position Searchable Symmetric Encryption (MSSP-SSE) and present an efficient construction that does not require authentication, thus achieving better performance than simply using access control mechanisms.

CHAPTER 8

Conclusions

Searchable encryption allows a client to encrypt its data in such a way that this data can still be searched. The most immediate application of SSE is to cloud storage, where it enables a client to securely outsource its data to an untrusted cloud provider without sacrificing the ability to search over it. In this chapter we summarize the results presented in this dissertation and elaborate upon future research directions.

8.1. Results

In this dissertation we solved the following open problems:

- We present a systematic literature review and comparison on searchable encryption techniques (Chapter 3). Through this systematic literature survey we found that challenges faced in searchable encryption techniques fall into main chategories: (1) ability to support search expressiveness, provide different query types and security features, namely ability to handle adaptive adversaries (2) ability to meet the requirements of performance and system usability through search and query execution time, and index storage. We identified ways to potentially overcome these challenges and their limitations.
- We present a secure and efficient searchable encryption scheme that allows multi-keyword query over an encrypted document corpus and retrieves the relevant documents ranked based on a similarity score (Chapter 4). We construct the searchable encryption scheme that is CKA2-secure in the random oracle model[24, 5]. Our scheme achieves semantic security against *adaptive* adversaries that choose their

search queries as a function of previously obtained trapdoors and search outcomes. We present a construction that achieves the *optimal* search time. Unlike many previous schemes that are glued to the linear search complexity, our search is sublinear to the total number of documents that contain the queried set of keywords.

- We present a secure and efficient searchable symmetric sncryption scheme that allows a substring search over encrypted cloud document collection (Chapter 6). The scheme is based on a position heap tree data structure recently proposed by Ehrenfeucht *et al.*[72]. We formally define two leakage functions and security against adaptive chosen-query attack on proposed scheme. Apart from traditional *access* and *search* patterns we include the definition of *path* pattern in the leakage functions of a tree-based searchable encryption. We show that SSP-SSE enjoys the strong notion of semantic security[5]. We present a construction that is very efficient and does not require large ciphertext space.
- We present a natural extension of both schemes, where an arbitrary group of data users can submit substring queries to search the encrypted collection (Chapter 5 and Chapter 7). We formally define a group multi-keyword similarity searchable encryption scheme and a multi-user substring position searchable symmetric encryption scheme. Both extensions include efficient construction that does not require authentication, thus achieving better performance than simply using access control mechanisms.

Work in this dissertation has also led to publication of two journal papers ([80] and one currently under review) and three peer-reviewed conference papers [81–83].

8.2. Future Directions

In this section we will investigate the several directions we can follow to extend the results discussed in this dissertation.

- Dynamic searchable encryption (DSE) enables a data owner to encrypt his document collection in a way that it is still searchable and efficiently updatable. Presented schemes are *static* and if the data owner wants to modify the original collection, he/she will have to regenerate a searchable index and share an updated trapdoor information with cloud users. Obviously, this induces large computation and communication overheads. In a DSE scheme, encrypted keyword searches should be supported even after documents are arbitrarily inserted into the collection or deleted from the collection.
- Our schemes utilizes a *honest-but-curious* threat model of the cloud adversary. In this model, the adversary follows the protocol specification and it provides documents that matches the search query. However, it is desired to support a *malicious* threat model where the cloud server may return incorrect search results to the cloud users. One future direction is to extend our schemes to support *verifiability*, where the users can check whether the returned documents contain the queried keywords.
- An interesting approach for future research is certainly a *problem-driven approach*; identifying the real-world problems, requirements, and needs first and then trying to address them by means of searchable encryption would lead to concrete and useful application scenarios, for example, search in outsourced (personal) databases, secure email routing, search in encrypted emails, and electronic health record (EHR) systems. In order to make an important step toward widespread use of searchable

encryption, multiuser schemes need to become more efficient and scalable for large datasets. It is possible to extend our work by exploring the different key distribution schemes to allow multiple data users to launch the search over encrypted cloud data.

BIBLIOGRAPHY

[1] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, 2000.

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

[3] T. Moataz and A. Shikfa, "Boolean symmetric searchable encryption," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, 2013.

[4] C. Orencik, M. Kantarcioglu, and E. Savas, "A practical and secure multi-keyword search method over encrypted cloud data," in *Proceedings of the 6th IEE International Conference on Cloud Computing*, 2013.

[5] 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*, 2006.

[6] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proceedings of the 4th IACR Theory of Cryptography Conference*, 2007.

[7] G. D. Crescenzo and V. Saraswat, "Public key encryption with searchable keywords based on jacobi symbols," in *Proceedings of the 8th International Conference on Cryptology in India*, 2007.

[8] C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling secure and efficient ranked keyword search over outsourced cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1467–1479, 2012.

[9] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in *Proceedings of the 30th IEEE International*

Conference on Computer Communications, 2011.

[10] M. Blanton, M. J. Atallah, K. B. Frikken, and Q. M. Malluhi, "Secure and efficient outsourcing of sequence comparisons," in *Proceedings of the 17th European Symposium on Research in Computer Security*, 2012.

[11] M. Blanton, "Achieving full security in privacy-preserving data mining," in *Proceedings of the 3rd IEEE International Conference on Privacy, Security, Risk and Trust*, 2011.

[12] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order-preserving encryption for numeric data," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2004.

[13] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *Proceedings of the 27th Annual International Cryptology Conference*, 2007.

[14] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi, "Searchable encryption revisited: consistency properties, relation to anonymous ibe, and extensions," *Journal of Cryptology*, vol. 21, no. 3, pp. 350–391, 2008.

[15] S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Private data indexes for selective access to outsourced data," in *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, 2011.

[16] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *Proceedings of the 28th IEEE International Conference on Data Engineering*, 2012.

[17] C. Wang, K. Ren, S. Yu, and K. Urs, "Achieving usable and privacy-assured similarity search over outsourced cloud data," in *Proceedings of the 31th Conference on Information Communications*, March 2012.

[18] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proceedings of the 29th Conference on Information Communications*, 2010.

[19] Google.com, "Google drive," 2016. `https://drive.google.com/`.

[20] Microsoft.com, "Microsoft onedrive," 2016. `https://onedrive.live.com/`.

[21] Apple.com, "Icloud," 2016. `https://www.icloud.com/`.

[22] Dropbox.com, "Dropbox," 2016. `https://www.dropbox.com/`.

[23] ArsTechnica.com, "Arstechnica," 2016. `http://arstechnica.com/apple/2012/04/apple-holds-the-master-key-when-it-comes-to-icloud-security-privacy/`.

[24] O. Goldreich, *The foundations of cryptography - volume 2, basic applications.* Cambridge University Press, 2004.

[25] O. Goldreich, *Foundations of cryptography: volume 1.* Cambridge University Press, 2006.

[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd ed., 2009.

[27] J. Katz and Y. Lindell, *Introduction to modern cryptography: principles and protocols.* Chapman & Hall/CRC, 2007.

[28] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot, *Handbook of applied cryptography.* CRC Press, Inc., 1st ed., 1996.

[29] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," *SIAM J. Comput.*, vol. 13, pp. 850–864, Nov. 1984.

[30] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, pp. 422–426, July 1970.

[31] I. H. Witten, A. Moffat, and T. C. Bell, *Managing gigabytes : compressing and indexing documents and images.* Morgan Kaufmann, 2 ed., 1999.

[32] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proceedings of the EUROCRYPT 2004*, 2004.

[33] G. Di Crescenzo and V. Saraswat, "Public key encryption with searchable keywords based on jacobi symbols," in *Proceedings of the Cryptology 8th International Conference on Progress in Cryptology*, 2007.

[34] J. Baek, R. Safavi-Naini, and W. Susilo, "On the integration of public key data encryption and public key encryption with keyword search," in *Proceedings of the 9th International Conference on Information Security*, 2006.

[35] C. Gu and Y. Zhu, "New efficient searchable encryption schemes from bilinear pairings," *International Journal of Network Security*, 2010.

[36] A. Swaminathan, Y. Mao, G.-M. Su, H. Gou, A. L. Varna, S. He, M. Wu, and D. W. Oard, "Confidentiality-preserving rank-ordered search," in *Proceedings of the ACM Workshop on Storage Security and Survivability*, 2007.

[37] H. S. Rhee, J. H. Park, and D. H. Lee, "Generic construction of designated tester public-key encryption with keyword search," *Inf. Sci.*, vol. 205, pp. 93–109, Nov. 2012.

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

[39] E. Kushilevitz and R. Ostrovsky, “Replication is not needed: Single database, computationally-private information retrieval,” in *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, 1997.

[40] C. Cachin, S. Micali, and M. Stadler, “Computationally private information retrieval with polylogarithmic communication,” in *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, 1999.

[41] H. Lipmaa, “An oblivious transfer protocol with log-squared communication,” in *Proceedings of the 8th International Conference on Information Security*, 2005.

[42] H. Lipmaa, “First cpir protocol with data-dependent computation,” in *Proceedings of the 12th International Conference on Information Security and Cryptology*, 2010.

[43] O. Goldreich, “Towards a theory of software protection and simulation by oblivious rams,” in *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, 1987.

[44] O. Goldreich and R. Ostrovsky, “Software protection and simulation on oblivious rams,” *Journal of the ACM*, vol. 43, pp. 431–473, May 1996.

[45] B. Pinkas and T. Reinman, “Oblivious ram revisited,” in *Proceedings of the 30th Annual Conference on Advances in Cryptology*, 2010.

[46] M. T. Goodrich and M. Mitzenmacher, “Privacy-preserving access of outsourced data via oblivious ram simulation,” in *Proceedings of the 38th International Conference on Automata, Languages and Programming - Volume Part II*, 2011.

[47] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia, “Practical oblivious storage,” in *Proceedings of the Second ACM Conference on Data and Application Security and Privacy*, 2012.

[48] S. Kamara, C. Papamanthou, and T. Roeder, “Dynamic searchable symmetric encryption,” in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, 2012.

[49] C. Wang, K. Ren, S. Yu, and K. M. R. Urs, “Achieving usable and privacy-assured similarity search over outsourced cloud data,” in *Proceedings of the 31st Annual IEEE International Conference on Computer Communications*, 2012.

[50] A. Boldyreva and N. Chenette, “Efficient fuzzy search on encrypted data,” in *Proceedings of the 21st International Workshop on Fast Software Encryption*, 2015.

[51] D. Grossman and O. Frieder, *Information retrieval: algorithms and heuristics*. Springer, 2004.

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

[53] C. Gentry, *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[54] C. Gentry, “Fully homomorphic encryption using ideal lattices,” in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, 2009.

[55] Z. Brakerski, “Fully homomorphic encryption without modulus switching from classical gapsvp,” in *Proceeding of the 32nd Annual International Cryptology Conference CRYPTO 2012*, 2012.

[56] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “(leveled) fully homomorphic encryption without bootstrapping,” in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, 2012.

[57] Y.-C. Chang and M. Mitzenmacher, “Privacy preserving keyword searches on remote encrypted data,” in *Proceedings of the 3rd International Conference on Applied Cryptography and Network Security*, 2005.

[58] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “Fully homomorphic encryption without bootstrapping.” Cryptology ePrint Archive, Report 2011/277, 2011.

[59] RFC, “Request for comments database,” 2016. `http://www.ietf.org/rfc.html`.

[60] B. Nichols, D. Buttlar, and J. P. Farrell, *Pthreads programming.* O’Reilly & Associates, Inc., 1996.

[61] C. Bösch, P. Hartel, W. Jonker, and A. Peter, “A survey of provably secure searchable encryption,” *ACM Computing Surveys*, vol. 47, pp. 18:1–18:51, Aug. 2014.

[62] A. Fiat and M. Naor, “Broadcast encryption,” in *Proceedings of the 13th Annual International Cryptology Conference CRYPTO ’93*, 1993.

[63] M. Zhandry, “How to avoid obfuscation using witness prfs,” in *Proceedings of the 13th International Conference on Theory of Cryptography, TCC 2016*, 2016.

[64] D. Boneh and M. Zhandry, “Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation.” Cryptology ePrint Archive, Report 2013/642, 2013.

[65] D. Boneh, C. Gentry, and B. Waters, “Collusion resistant broadcast encryption with short ciphertexts and private keys,” in *Proceedings of the 25th Annual International Cryptology Conference CRYPTO 2005*, 2005.

[66] R. Sakai and J. Furukawa, “Identity-based broadcast encryption.” Cryptology ePrint Archive, Report 2007/217, 2007.

[67] C. Delerablee, P. Paillier, and D. Pointcheval, "Fully collusion secure dynamic broadcast encryption with constant-size ciphertexts or decryption keys," in *Proceedings of the First International Conference on Pairing-based Cryptography*, 2007.

[68] J. Lai, X. Zhou, R. H. Deng, Y. Li, and K. Chen, "Expressive search on encrypted data," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, 2013.

[69] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Proceedings of the CRYPTO 2013*, 2013.

[70] E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, 2007.

[71] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proceedings of the 29th Conference on Information Communications*, 2010.

[72] A. Ehrenfeucht, R. M. McConnell, N. Osheim, and S.-W. Woo, "Position heaps: A simple and dynamic text indexing data structure," *J. Discrete Algorithms*, vol. 9, no. 1, pp. 100–121, 2011.

[73] P. Weiner, "Linear pattern matching algorithms," in *Proceedings of the 14th Annual Symposium on Switching and Automata Theory*, 1973.

[74] U. Manber and G. Myers, "Suffix arrays: A new method for on-line string searches," *SIAM Journal on Computing*, vol. 22, no. 5, pp. 935–948, 1993.

[75] D. Gusfield, *Algorithms on strings, trees, and sequences: computer science and computational biology.* Cambridge University Press, 1997.

[76] E. Ukkonen, "On-line construction of suffix trees," *Algorithmica*, vol. 14, no. 3, pp. 249–260, 1995.

[77] C. Gentry, K. Goldman, S. Halevi, C. Julta, M. Raykova, and D. Wichs, "Optimizing oram and using it efficiently for secure computation," in *Proceedings of the 13th Privacy Enhancing Technologies Symposium*, 2013.

[78] NCBI, "Genome database," 2016. `http://www.ncbi.nlm.nih.gov/genome`.

[79] B. Morris, P. Rogaway, and T. Stegers, "How to encipher messages on a small domain," in *Proceedings of the CRYPTO 2009*, 2009.

[80] M. Strizhov and I. Ray, "Secure multi-keyword similarity search over encrypted cloud data supporting efficient multi-user setup," *Trans. Data Privacy*, 2016.

[81] M. Strizhov and I. Ray, "Substring position search over encrypted cloud data using tree-based index," in *Proceedings of the 2015 IEEE International Conference on Cloud Engineering, IC2E 2015*, 2015.

[82] M. Strizhov, "Towards a practical and efficient search over encrypted data in the cloud," in *Proceedings of the 2015 IEEE International Conference on Cloud Engineering, IC2E 2015*, 2015.

[83] M. Strizhov and I. Ray, "Multi-keyword similarity search over encrypted cloud data," in *Proceedings of the 29th International Conference on Systems Security and Privacy Protection, SEC 2014*, 2014.