Privacy-Preserving Query Processing on Health Data

by

© Mohammad Hoseyn Sheykholeslam

A thesis submitted to the
School of Graduate Studies
in partial fulfilment of the
requirements for the degree of
Master of Science

Department of Computer Science
Memorial University of Newfoundland

August 2018

St. John's Newfoundland

Abstract

Due to the huge volume of digital data and the underlying complexity of data management, people and companies are motivated to outsource their computational requirements to the cloud. A significant portion of these productions are used in health applications. While popular cloud computing platforms provide flexible and low-priced solutions, unfortunately, they do so with little support for data security and privacy. This shortcoming clearly threatens sensitive data in cloud platforms. This is especially true for health information, which should always be adequately secured via encryption. Providing secure storage and access to health information that is generated by systems or used in applications, is the main challenge in today's health care systems. As a result, owners of sensitive information may hesitate in purchasing such services, given the risks associated with the unauthorized access to their data. Considering this problem, researchers have recommended applying encryption algorithms. Data owners never disclose encryption keys in order to keep their encrypted data secure. Because cloud platforms can not search in data which is encrypted with regular encryption algorithms, it is supposed that data owners conceal their secrets with searchable encryption algorithms. Searchable encryption is a family of cryptographic protocols that facilitate private keyword searches directly on encrypted data. These protocols allow data owners to upload their encrypted data to the cloud, while retaining the ability to query over uploaded data. In this project, we focus on symmetric searchable encryption schemes, as well as apply an efficient

searchable encryption scheme which supports multi-keyword searches to provide a privacy preserving keyword search framework for health data. Our framework applies a recent secure searchable encryption scheme and employs an inverted indexing structure in order to process queries in a privacy-preserving manner.

Acknowledgements

First and foremost, I would like to praise Allah, for granting me the strength to follow my way successfully.

Next, I would like to express my gratitude to my supervisors, Dr. Saeed Samet and Dr. Antonina Kolokolova for their infinite support through various issues and for guiding me to always be a creative researcher during my Masters.

I want to extend my special thanks to all former and current colleagues at the e-Health Research Unit who inspired my research and studies with their constructive ideas and discussions.

My acknowledgments extend to all tutors and assistants of the Writing Centre at Memorial University, specifically Kelly Greenfield who spent a lot of time helping to improve my skills to write and publish my research.

I am so thankful to my Mom, Dad, and sisters for their never ending support, love, and prayers. I also have been so fortunate to have such a supportive brother through my life.

Finally and most importantly, I would like to dedicate this thesis to my darling wife, Narjes, whose presence in my life motivated me and ensured my success.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Nowadays, the exponential growth of data challenges all data storage and infrastructures. In 2014, sources predicted that the total size of digital information globally would almost double every two years [3, 5]. This global data is expected to hit 44 zettabytes or 44 trillion gigabytes by 2020. In this era, local storage is not the efficient nor economic solution to save and protect our data. The daily generated information makes all companies, businesses, end-users, and individuals to outsource their data and apply cloud servers and applications.

1.2 Outsourcing Data

Outsourcing data and operations in companies is a significantly critical decision that managers need to consider. Outsourcing has many advantages which can cause growth and save money for companies.

All outsourced data centers try to guarantee the quality of service and minimize the downtime to avoid financial penalties. In a model in which data is outsourced, the maintaining and upgrading of local infrastructures are carried out by the service providers. This advantage gives technology officers and managers more time to spend on their companies' goals and business expansion. Moreover, the competitive low price of services presented by famous companies like Amazon, Google, and Microsoft assist businesses to boost their bottom line. Consequently, outsourcing reduces expenses for equipment maintenance, physical space, and upgrade overheads.

Despite several benefits of outsourcing data, there are critical issues that people should consider. These drawbacks sometimes change the business owners' minds. While outsourcing is cost-effective in most cases, there are some hidden costs which may arise. The data centers, which usually are far away from headquarters, may cost companies. In the case of outsourcing, natural disasters are other possible challenges. The most important disadvantage that threatens outsourced data is exposing confidential data.

Data owners assume that the cloud storage service is secure because it keeps the data safe and follows the rules and algorithms, but it may try to disclose the plaintext

and the access patterns of searched keywords. Consequently, a storing server should be considered a semi-trusted one.

1.2.1 Outsourcing the Health-care Data

Health information is an essential type of data which is mostly considered as confidential data. Previously, medical institutions were responsible for providing required information systems and preserving the confidentiality of their data. The other crucial role of technology officers in health organizations is the setting up a secure mechanism to share private data with all providers and consumers in the format of a trusted network. In such networks, some consumers receive data and services from multiple providers while participating in service offerings in different programs across agencies.

1.2.1.1 Secure Outsourcing

The privacy of health-care data is a critical component of every health department and agency. In order to share health-care information more comfortably, secure frameworks and protocols for accessing health information should be provided. A secure framework that protects outsourced sensitive data can accelerate the strategic plan to outsource clinical information systems while ensuring privacy and effectiveness.

1.2.2 Big Data

The advancement of big data and cloud computing have made the privacy concerns even more pressing. Big data significantly increases the challenges in developing systems to protect privacy. At the same time, being able to process big data to extract meaningful information and knowledge from raw data is another concern that threatens the privacy.

1.3 Searchable Encryption

The most natural solution to guarantee the privacy and security of outsourced data is encryption. A symmetric-key algorithm is a scheme that uses a single cryptographic key for both encryption and decryption operations. It means that people can learn nothing about the encrypted data without the right key. However, encryption key owners want to store their encrypted data on a cloud server while being able to ask the cloud to search on the encoded dataset and send the related encrypted results back without decrypting the dataset and the query. In 2000, the earliest scheme for searching over encrypted data was proposed by Song et al. [61]. That scheme is known as searchable symmetric encryption (SSE).

Since 2000, many studies were conducted on searchable encryption (SE) schemes. The studies tried to solve issues of searchable encryption. Early studies provided schemes for single predefined keywords and combinations of keywords searches, while

the later studies where about multi-client, public-key based, and substring searchable encryption schemes. Recently, researchers have been investigating more efficient and secure schemes.

1.4 Contributions

In this thesis, I focus on efficient multi-client schemes to provide a framework for health-care data. For example, the laboratory in a hospital stores encrypted records of patients' blood tests in a cloud storage server. The data users like doctors or insurers may be interested in searching specific parts of records. As the data owner, the hospital can authorize data users to send their encrypted queries to the cloud server. The framework protects the privacy of original data and queries. Our framework searches over the encrypted data and does not understand anything about the queried keywords and the original data.

The multi-client search framework for the encrypted health-care data makes the applications of search methods more usable and improves the operational efficiency.

Chapter 2

Preliminaries and Definitions

As a prelude to further technical discussions, we need to present some basic concepts and definitions. In this chapter, we review big data concept as well as definitions of cloud-computing and health informatics.

2.1 Big Data

This refers to growing tremendously large and complex data sets. The size of big data, which have not been constant, was measured at most in petabytes (10^{15}) up until 2012 but now is measured in units ranging from exabytes (10^{18}) to yottabytes (10^{24}) [35]. Big data has its specific challenges in simple data operations, which are data collection, storing, and sharing, as well as more complicated operations like data processing and analysis. Protecting privacy, querying, and extracting patterns

are highly important functions in big data sciences and analyses[10].

2.2 Cloud Computing

Cloud computing refers to providing ubiquitous processing resources and data over the Internet to consumers CPU and other devices on demand. These resources, which are networks, virtual machines, databases, applications and other services, can be easily obtained by the clients, released with minimal management effort and always supported [53, 49].

2.3 Cryptography

In 1996, Menezes et al. defined cryptography as the study of mathematical techniques related to confidentiality, data integrity, and authentication which are aspects of information security [50]. Cryptography is about constructing and analyzing protocols that prevent third parties or the public from reading private messages [11]. Various aspects of information security, such as data confidentiality, data integrity, authentication, and non-repudiation are central to modern cryptography [50].

2.4 Edit Distance Metrics

Edit distance metrics are algorithms which determine the least number of editing operations to transform from one string (e.g. word) to another one. These operations are insertion, deletion, and substitution [46]. Some metrics like Damerau-Levenshtein distance [32] considered the transposition of two adjacent letters as another editing operation.

2.5 Electronic Medical Record (EMR)

All paper charts and treatment information in hospitals, pharmacies, clinics, and health offices should be digitalized. These digital records in the health sector referred to as an electronic medical records or EMR. In other words, EMRs are important information which can be used many times and are tracked for monitoring health status and treatment process. EMRs have certain strengths over paper version of medical records because they can easily be saved, reviewed, and shared in health institutions.

2.6 Electronic Health Record (EHR)

The electronic health record or EHR refers to all health information of a patient's treatment and care which is collected by clinicians in hospitals and clinics. A patient's EHR can be shared with all authorized doctors, pharmacists, and specialists who

participate in the patient's treatment. This term also refers to the systematized collection of patient and population health information saved in cloud servers [40].

2.7 Personal Health Record (PHR)

Rather than just specialists in health institutions, personal health records (PHR) can be collected by health-care home devices or by the patient at his/her home. The main difference between EHRs and PHRs is that PHRs should be managed and organized by the patient. The patient determines which persons can have the access to the PHRs. Medical histories, medications, and diagnoses results can be categorized as PHRs. Tang et al. believe that PHR refers to both paper-based and computerized health data and information related to the care of a patient that is maintained by the patient [63].

2.8 Pseudo-Random Function (PRF)

A random function composes a lookup table filled with random entries which are distributed uniformly. The random function maps entries to unique numbers in a specific range. A practical PRF is acceptable if it behaves like a random function and no efficient method can distinguish that a output of a determined entry is produced by a random function or a PRF. A PRF always outputs a specific value given a determined input from its domain and a determined random seed number even if it

operates multiple times [30, 31].

Chapter 3

Background and Challenges

3.1 Overview

In this chapter, we review topics related to our study. We focus on searchable encryption and related domains which are currently important for researchers. While reviewing these areas, we take the most important challenges and research questions into account.

Firstly, we review the searchable encryption definition. Then, we refer to the new primitive technique which is called Blind Storage and discuss its advantages. We then review Oblivious RAM (ORAM), the premier technology for securing cloud-based storages. We refer later to a new generation of ORAM which is more efficient and useful for multi-client frameworks. In the next chapter, we further discuss cryptographic preliminaries.

3.2 Notations

In all following discussed schemes and models, we supposed that the data owner has a collection that contains n documents ($\mathcal{DB}$).

$$\mathcal{DB} = (d_1, ..., d_n) \tag{3.1}$$

W in this study, is considered to be a list of m unique keywords which are extracted from the owner's collection and may be queried later.

$$W = (w_1, ..., w_m) \tag{3.2}$$

$$W_{id_i} = \{w | w \in d_i \text{ and } w \in W \text{ and identifier of the document}(d_i) \text{ is } id_i\} \tag{3.3}$$

$$\therefore W = \{\bigcup_{i=1}^{n} W_{id_i}\}, n = |\mathcal{DB}| \tag{3.4}$$

Ind is a set of documents' identifiers.

$$Ind = \{ind_{w_1}, ..., ind_{w_m}\} \tag{3.5}$$

Each $\mathcal{DB}[w_i]$ or ind_{w_i} in Ind refers to a set of documents, each of which contains the keyword w_i.

$$ind_{w_i} = \mathcal{DB}[w_i] = (id|w_i \in W_{id}) \tag{3.6}$$

3.3 Searchable Encryption (SE)

An encryption algorithm that supports lookup functionality over encrypted data without decryption with the least cloud data leakage is called Searchable Encryption (SE). Searchable encryption schemes allow a client to query from an untrusted server to search within encrypted data without having the encryption key and without learning information about the original plaintext data. Searchable encryption has two main subsets of schemes which are symmetric searchable encryption (SSE) schemes and public key encryption with keyword search (PKES) schemes [19].

3.3.1 General Model

A searchable encryption generally operates three primary phases in order to accomplish keyword lookup: setup, token, and search.

The duty of the setup phase is to create encrypted data and the related encryption key K' using plaintext data and a secret parameter (1^K) which is a K-bit vector. In this phase, the data owner creates inverted indexes of the keyword list Ind and

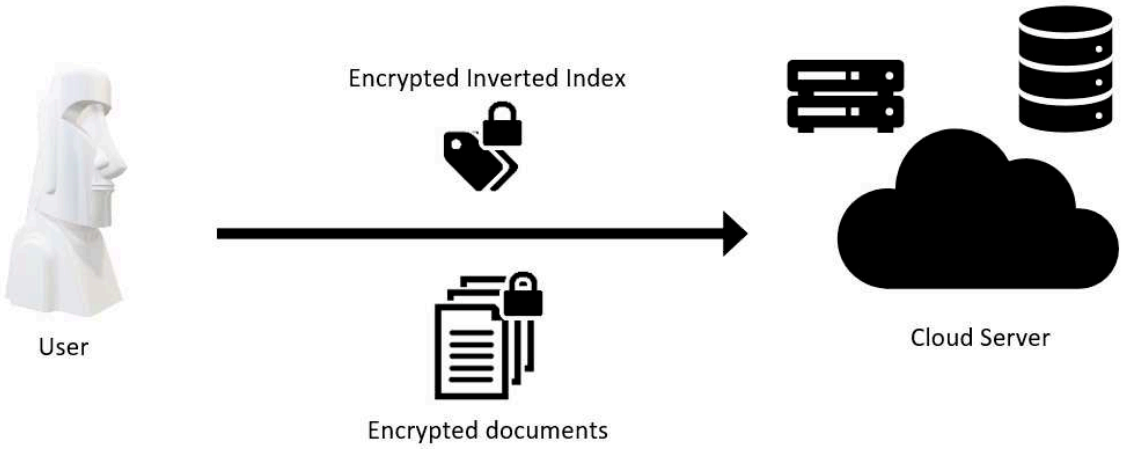

Figure 3.1: Searchable Encryption Setup

outsource them (Figure 3.1). -

$$Ind = Create\_Index_{K'}(\mathcal{DB} = (d_1, ..., d_n), W = (w_1, ..., w_m)). \tag{3.7}$$

The inverted index array is a set of indexes, each of which belongs to a word from word list W and determines documents which contain that specific word. The size of Ind in Equation 3.7 is the same as the size of the word list $|W| = |Ind| = m$.

The documents should be encrypted with the owner's key (K') using an encryption algorithm (Enc). The owner uploads the encrypted collection and the inverted index to the server.

$$\mathcal{EDB} = Enc_{K'}(\mathcal{DB} = (d_1, ..., d_n)). \tag{3.8}$$

The $Token$ function takes as input the produced secret key, a query, and returns the related token tk. The query can be a boolean function of multiple keywords.

$$tk = Token_{K'}(query) \tag{3.9}$$

The last main operator is search which uses token tk and the inverted indexes in order to retrieve the set of matching encrypted documents and sends them to the user.

3.3.2 Symmetric Searchable Encryption (SSE)

In SSE schemes, data owners create encrypted data and related indexes using a symmetric secret key. The indexes will be used later to search over encrypted data. In general, a SSE consists of two main function: Setup and Search.

Setup. This function generates a secret key for the received database and outputs the associated encrypted database. The data owner receives the secret key while the encrypted database is stored in the cloud storage server.

Search. Having the secret key of the encrypted data enables the client to send a secure query to the storage server. The storage operates the search function and receives the identifiers of encrypted documents or records. The storage server sends the encrypted items to the querier afterwards.

3.3.2.1 Dynamic Symmetric Searchable Encryption (DSSE)

A dynamic searchable symmetric encryption scheme (DSSE) includes all functions of a SSE scheme. A DSSE also has an extra function which is *Update*. The *Update* function generates a new secret key and reproduce the encrypted database. This function will be invoked after updating documents or keyword list of the database. *Update* also have to be called after revoking a user. That being dealt with, some SSE schemes are introduced in Chapter 4 which do not require updating the database after revoking a user.

3.3.3 Security Definitions

Song et al. [61] as the first proposers of a SE scheme proved that their scheme is secure. Later, the scientists in this area proposed some formal security definitions which are discussed as follows.

3.3.3.1 Indistinguishable Against Chosen Plaintext Attack (IND-CPA)

A scheme is IND-CPA if the user cannot distinguish the ciphertexts of two chosen plaintexts. In other words, the encrypted data does not leak anything about plaintext data. The IND-CPA is not proper notion of security for SE schemes because it does not consider query or trapdoor as the matter of leakage.

3.3.3.2 Semantic Security Against Adaptive Chosen Keyword Attack (IND-CKA)

In 2003, Goh [37] defined a secure index and two security models for indexes which was semantic security against adaptive chosen keyword attack (IND-CKA). A IND-CKA secure scheme doesn't let the adversary deduce the content of a document using its index. In other words, a server, which contains an index and two encrypted document containing same number of words, cannot distinguish the document that the index is related to. Chang and Mitzenmacher [27] in their study as well as Goh in his second definition [37] introduced new versions of IND-CKA which an adversary cannot distinguish indexes of documents with containing different number of words. Chang and Mitzenmacher tried to guarantee the privacy of users' queries while Goh did not guarantee trapdoors' leaking in either of his both definitions of IND-CKA. This means that the trapdoors may reveal the queried keywords which is not secure enough definition for SE schemes.

3.3.3.3 Non-adaptive and Adaptive Indistinguishability Security for SSE

In 2006, and later in 2011 Curtmola et al [30, 31], considering the shortcomings of previous security notions for SSE, presented non-adaptive and adaptive indistinguishability security which avoided previous notions' problems. In non-adaptive definition (IND-CKA1), a scheme preserves the privacy of keywords and trapdoors against an adversary who runs the queries in a batch. This definition is not quite

practical in real world and is improved in second definition of Curtmola et al [30, 31]. In the adaptive indistinguishability notion of security for SSE (IND-CKA2), which is practical and considered as a solid definition for SSE, an adversary client can choose a query considering previous obtained trapdoors and search results.

3.4 Bloom Filter

A Bloom filter refers to a hashed data structure which securely indicates the membership of an element in a set. A Bloom filter encompasses an array with b bits and k independent hash functions which maps each input to a position in the b bits array.

$$Bloom\ Array : [\underbrace{0, 0, ..., 0}_{b}] \tag{3.10}$$

$$h_t : \{0,1\}^* \rightarrow [1, b] \text{ for } t \in [1, k] \tag{3.11}$$

In order to apply a Bloom filter to determine the set's membership, the Bloom array initially is set to zero for all bits. Then, for each element e_i in the set $\{e_1, ..., e_n\}$, all $h_t(e_i)$ positions in Bloom array are set to one. In order to check membership of an element e_x by a server which don't have access the the original set, it's sufficient to only check that the all $h_t(e_x)$ positions in Bloom array were set to one.

Figure 3.2 illustrates the elements of the set is mapped to array cells with hash

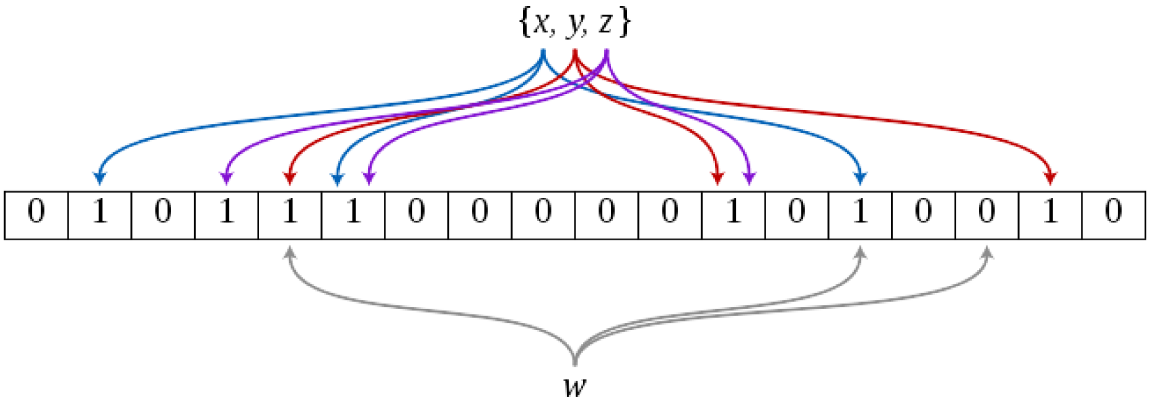

Figure 3.2: Bloom filter's membership check. Adapted from Streaming Algorithms for Straggler Detection, by D. Eppstein, WADS 2007

functions. This also indicates that w is not member of the set because one of the hashes points to a cell which is set as 0. In the Figure 3.2, there are 18 bits in Bloom array and three hash functions.

3.5 Blind Storage

A blind storage is a new primitive technique which has been first defined by Naveed [52] in 2016. This scheme is capable of storing user's documents on a remote storage server while the storage can not detect the number of stored files nor the size of every single file. Although the server will notice the existence of the specific file while it is being accessed by the user, that file's name and content will not be disclosed.

The blind storage supports 'Create', 'Read', 'Update', and 'Delete' (CRUD) op-

erations while they are hidden from the server. The blind storage does not need any specific function or tool on remote servers and can be used in conventional cloud storage servers like Google Drive and Dropbox.

3.6 Oblivious Random Access Memory (Oblivious RAM, ORAM)

Although encryption algorithms hide the contents of data from the remote server, none of those algorithms can conceal the users' access patterns of reading and writing data. The access pattern can leak the number of operations as well as operands and operator of each operation. The information leaked from access patterns can seriously endanger the privacy of encrypted data [65]. The oblivious RAM primitive principally ensures privacy-preserving for users' data on an untrusted remote storage server.

The main goal of oblivious RAM schemes is addressing the revealing memory access issue of CPUs from untrusted memories while reading or storing processes. This scheme keeps search and the memory access independent [54]. While the basic scheme is extremely secure and recommended by Goldreich and Ostrovsky [38], many studies were carried out to improve its performance and make it scheme more practical [21, 29, 33, 38, 39, 45, 56, 57, 62, 67].

Oblivious RAM (ORAM) [70] is a cryptographic primitive that hides the access

pattern of a trusted CPU to an untrusted memory. From an untrusted memory perspective, any two accesses to the memory are indistinguishable, even if the trusted CPU is accessing the same data.

As was mentioned previously, ORAM hides the access patterns of CPU from untrusted memory. In an outsourcing context, we can consider that the CPU is a client who queries and that untrusted memory is a cloud server which stores our data. Running the queries by the server does not leak any information about the location and type of retrieved encrypted data.

3.7 Basic Cross-Tags (BXT) Protocol

In 2013, Cash et al. [26] proposed the Basic Cross-Tags or BXT protocol. The BXT, as an extension to single-keyword SSE (SKS), is a new scheme for conjunctive keyword search. The BXT addressed its previous schemes' issues for conjunctive keyword search. Some of those problems were related to time and space complexities. Sometimes the time and space complexities, as the major criteria of SSE algorithms, cause those schemes to become infeasible for big data. The main challenge, which was discussed in the earlier studies, was the linear relationship of the cloud server's workload and the number of encoded documents for searching each keyword set. Cash et al. [26] proposed a sub-linear conjunctive search scheme for the first time.

This protocol applies some main functions to operate a conjunctive keyword search on encrypted data. These modules are discussed as follows:

EDBSetup($\mathcal{DB}$). This function outputs $TSet$ as the "secure inverted index" and $XSet$ as the related set data structure. These two items are considered as the encrypted database $\mathcal{EDB}$ and will be sent to the cloud server. $TSet$ contains the encrypted indexes of keywords from W. $XSet$ is a specific data structure that contains a set of $xtag$s for every keyword in W and all related indexes. Each $xtag$ in $XSet$, which is calculated using two unpredictable functions, is the result of keywords in W and their related indexes. The client, who owns the database and later wants to query on that, generates encryption keys for encrypting the database.

Search. This protocol contains several functions which should be operated by the server and clients. We can suppose that the client wants to query a conjunction of keywords like:

$$\bar{w} = \bigwedge_{1}^{k} w_i \tag{3.12}$$

The client chooses the keyword w_f from that conjunction which is used least often in the outsourced data and f is the index of that keyword in the conjunction. Later, the client generates the encryption key (K_e) related to w_f, finds the inverted index($stag$) for that keyword, and calculates inverted indexes ($xtrap$) for the rest of the keywords in that conjunction. The client sends K_e, $stag$ and all $xtrap$s to the server. The cloud server, which holds ciphertexts and runs queries on the encrypted data, uses the $stag$ to retrieve the least frequent keyword's index set $T(w_f)$ from $TSet$. After retrieving the encrypted set, the

cloud server decrypts all indexes in $T(w_f)$ and sends each decrypted index ind to the server if the following formula is true for all $xtrap$s which f is selected from a pseudo-random function family(PRF).

$$\forall i \in (1, ..., k), f(xtrap_i, ind) \in XSet \tag{3.13}$$

3.8 Oblivious Cross-Tags (OXT) Protocol

In this section, we review the Oblivious Cross-Tags (OXT) protocol which has been proposed by Cash et al. [26] in 2013 and is theoretically based on the BXT scheme. The main issue of the BXT is its serious leakage. $Xtrap$s are created using PRF and a secret key K_X. As was indicated about the BXT in the previous section, the server should have the decryption key K_e in order to retrieve the indexes of documents which contain w_i as well as $xtrap$s of other keywords in the conjunction to calculate $xtag$ and to check if that belongs to $XSet$. In other words, the server needs decryption key K_e and $xtrap$s to calculate $xtag$s and find out which indexes of $stag$ contain other queried keywords in that conjunction. Providing such information discloses some sensitive facts, like statistical correlation of queries and documents, to curious and leaky cloud servers. The presenters of the OXT protocol addressed this problem in their scheme by using an oblivious shared computation of $xtag$ between server and client during the search process. They engaged a blinded Diffie-Hellman

(DH) exponentiation over a group G of prime order p. In order to reduce overhead of interactions between the client and remote server, Cash et al. [26] considered that the blinding parts of the formula must be the *EDBSetup* phase and stored in $TSet$.

The OXT protocol, like the BXT protocol, has two phases of EDBSetup and search which are discussed as follows:

EDBSetup($\mathcal{DB}$). As it is shown in Algorithm 1, EDBSetup generates more secure $TSet$ and $XSet$. In the first step of this phase, the algorithm creates $strap_i$ for every keyword w_i in keyword list W using PRF and a secret key K_s. $Straps$ are required to generate the encryption keys K_e for encrypting both indexes of documents which match w_i and related record-decrypting keys (rdk) of matching documents. The indexes of documents and $xtraps$ of queried keywords were previously required in the search phase of the BXT to calculate $xtags$. In the OXT, in order to compute $xtags$ in a manner that the cloud server does not learn anything about either $xtraps$ or $inds$, a blinded DH computation should be applied and pre-computed blinding factors should be stored in $TSet$ as a part of tuples. Each tuple of $T[w_i]$ is related to a keyword w_i and a matching document index ind_j and ind_j belongs to I_{w_i} set. A blinding factor in each tuple is y_c, which is calculated by multiplication of $xind_j$ and z_c^{-1}. The $xind_j$ is output of a PRF (F_p) that gets ind_j and z_c is generated by the same F_p that takes a counter c as its input. The applied F_p is a PRF with the range of Z_p^*. $XSet$ in the OXT is updated regarding its previous definition in the BXT.

The new $XSet$ stores a more secure formula of $xtag$s for every keyword w_i and every matched document index ind_j is calculated using the following formula:

$$xtag_{i,j} = xtrap(w_i)^{xind_j} \tag{3.14}$$

In previous $xtag$s, $xtrap(w_i)$s are calculated as follows:

$$xtrap(w_i) = g^{F_2(K_X, w_i)} \tag{3.15}$$

After generating $xtag$s they will be added to $XSet$.

Algorithm 1 Prepare Encrypted Data

procedure EDBSetup(Data D, Array of Encryption Keys $RDK[]$)

$XSet \leftarrow$ empty set for xtags

$T \leftarrow$ empty associative array for allowed keywords W

for each $w \in W$ **do**

$t \leftarrow$ empty list for encrypted indexes

$strap \leftarrow F_1(K_s, w)$ ▷ F_1 is a pseudo random function

$K_e \leftarrow F_1(strap, 2)$ and $K_z \leftarrow F_1(strap, 1)$

$c \leftarrow 1$

for each ind_i of D's documents which contains w **do**

$rdk_i \leftarrow \mathrm{RDK}(ind_i)$ ▷ rdk_i is the encryption key for document i

$xind \leftarrow F_2(K_I, ind_i)$ ▷ F_2 is a pseudo random function

$z_c \leftarrow F_2(K_z, c)$ and $c \leftarrow c + 1$

$y \leftarrow xind \cdot z_c^{-1}$

$e \leftarrow \mathrm{Enc}(K_e, rdk_i)$

$t \leftarrow t \cup (e, y)$

$xtag \leftarrow g^{F_2(K_X, w) \cdot xind}$

$XSet \leftarrow XSet \cup xtag$

end for

$T[w] \leftarrow t$

end for

$TSet \leftarrow$ empty tuple set

$(TSet, K_T) \leftarrow TSetSetup(T)$

$ED = (TSet, XSet)$

$SendEDToServer(ED)$

end procedure

Search. The general idea in the search phase is to calculate the $xtag$ of each $xind$ in tuples of $T[w_i]$ and test if it is a member of $XSet$ or not. The search phase can be separated into two main functions. *TokenGeneration*, which is illustrated in Algorithm 2, should be operated by the client. The server operates the second part, which is the actual search process. *TokenGeneration* generates $stag$ for the least frequent keyword (w_f) of the conjunction of keywords which has to be queried. *TokenGeneration* also generates the $xtoken[c]$ array of c^{th} tuple of $T[w_f]$. The client sends the $stag$ and $xtoken[c]$ to the server.

$$xoken[c] = \{xtoken[c, i] | 1 < i < |\bar{w}| \text{ and } i \neq f\} \tag{3.16}$$

Algorithm 2 Client Side of the Search Protocol

1: **procedure** TokenGeneration
2: $stag \leftarrow TSetGetTag(K_T, w_1)$
3: $SendStagToServer(stag)$
4: $strap \leftarrow F_1(K_S, w_1)$
5: $K_e \leftarrow F_1(strap, 2)$
6: $K_z \leftarrow F_1(strap, 1)$ ▷ F_1 is a pseudo random function
7: $c \leftarrow 1$
8: **while** $ReceivedStopFromServer()$ **do**
9: $z_c \leftarrow F_1(K_z, c)$ ▷ F_1 is a pseudo random function
10: $xtoken[c] \leftarrow$ empty array ▷ xtoken is a two dimensions array
11: **for** $i = 2, \ldots, n$ **do**
12: $xtoken[c][i] \leftarrow g^{F_1(K_X, w_i) \cdot z_c}$
13: **end for**
14: $SendXtokenToServer(xtoken[c])$
15: $c \leftarrow c + 1$
16: **end while**
17: **end procedure**

Algorithm 3 shows the second main function of the search phase which should be operated by the server. At first, the server receives the *stag* and achieves the related t or $T[w_f]$. The server later receives the *xtokens* from the client. For each $xtoken[c]$, the server accesses the c^{th} tuple of $T[w_f]$ which is (e_c, y_c). As previously stated, $T[w_f]$ is an array of tuples whose length is the same as

the number of documents which contain w_f. After catching the $xtoken[c]$ and (e_c, y_c), the server calculates $xtag$ for all indexes of $xtoken[c]$ and sends e_c to the client if all indexes of $xtoken[c]$ were members of $XSet$. Later, the client can decrypt e_c and find out which indexes contain the whole queried conjunction.

Algorithm 3 Server Side of the Search Protocol

```
1: procedure SEARCH(Encrypted Date ED)        ▷ ED contains TSet and XSet
2:     stag ← AskStagFromPatient()
3:     t ← TSetRetrive(TSet, stag)
4:     for c = 1, ..., sizeof(t) do
5:         xtoken[c] ← AskCthTokenArrayFromPatient()
6:         (e_c, y_c) ← c^th tuple of t
7:         for i = 2, ..., n do          ▷ n is the number of keywords in the query
8:             if xtoken[c][i]^y is member of XSet then
9:                 SendEToPatient(e_c)
10:            end if
11:        end for
12:    end for
13:    SendStopToPatient()
14: end procedure
```

As was discussed in this section, the issue of the BXT was addressed and the semi-trusted server checked the membership of $xtag$s without having $xtrap$s, indexes (ind), and decryption key of indexes (K_e).

3.9 Multi-Client OXT Protocol (MC-OXT)

In their study, Jarecki et al. [42] extended the OXT protocol and provided the Multi-Client OXT (MC-OXT) which supported multi-client SSE (MC-SSE) while preserving the OXT features and functionalities. In the MC-SSE, the data owner encrypts its data and outsources it to the cloud server which securely operates keyword searches. The client, who differs from the data owner in this scheme, sends the query to the data owner and receives the related tokens to perform the keyword search in the cloud server. The server applies tokens, searches for the results, and returns the encrypted indexes to the data owner.

In this protocol, Jarecki et al. [42] defines three phases for their multi-client solution which are described as follows:

EDBSetup(DB, RDK). In the MC-OXT protocol, the EDBSetup phase has almost the same steps as the EDBSetup in the OXT. The only difference is that the MC-OXT protcol shares an additional key between the data owner and cloud server.

GenToken(K,$\bar{w}$). GenToken is the specific phase designed for the MC-OXT. The purpose of this phase is to authorize a client and to enable him/her to search a specific conjunction of keywords over the encrypted data stored in the cloud server. In this regard, the client sends the conjunction of keywords to the data owner. The data owner calculates the *stag* and *strap* for the least frequent

keyword (w_f). The data owner also generates *bxtrap* terms for the rest of the keywords in that conjunction. The *bxtrap* terms are being calculated as follows:

$$bxtrap_i = g^{F_p(K_X, w_i)\cdot\rho_i}, \rho_i \in Z_p^* \tag{3.17}$$

The data owner sends *bxtrap* terms as well as the encrypted *stag* and ρ_i to the client for the search process. $\rho_1, ..., \rho_n$, which are generated for all keywords in the conjunction except the least frequent keyword, are one-time blinding factors which hide real *xtrap* from the client. The data owner encrypts the *stag* and blinding factors ρ_i, saves them in a single message *env*, and provides the *env* to the client. During the search process, having the *env* file shows that the client is authenticated.

Search. At the first step of the search, the client computes required keys using *strap* previously provided by the data owner. Instead of *xtoken* terms of the OXT protocol, the client computes *bxtoken* terms as follows:

$$bxtoken[c] = \{bxtoken[c, i] | 1 < i < |\bar{w}| \text{ and } i \neq f\} \tag{3.18}$$

$$bxtoken[c, i] = bxtrap_i^{Z_c} \tag{3.19}$$

The client sends *bxtoken* terms and the *env* to the server. The server decrypts the *env* and achieves *stag* and blinding factors. After obtaining the *stag*, like the OXT protocol, the server retrieves the $TSet(w_f)$ and looks for tuples in

$TSet(w_f)$ which are valid for the following statement:

$$\forall i, 1 \leq i \leq |\bar{w}| \text{ and } i \neq f, bxtoken[c,i]^{y_c/\rho_i} \in XSet \tag{3.20}$$

In this formula, which is a replacement for the one in the OXT protocol, the server obtains the *xtoken*s by raising the the *bxtoken* to the power of y/ρ. The ρ values are blinding factors achieved by decrypting the *env*.

One of the issues of the MC-OXT protocol is leaking the size of $T(w_f)$ to the client. Indeed, the negotiation between the client and server for transferring *bxtoken* terms and results may leak the $|T(w_f)|$. This issue of the original scheme should be considered and resolved in complementary schemes.

3.10 Research Challenges

The keyword search and query processing on encrypted data schemes have actively been updated and improved for more than a decade, although significant challenges and open issues still exist.

In general, private-key solutions are faster but suffer from a key management problem. On the other hand, public-key solutions provide more flexibility but their running times are much higher than private-key protocols. Furthermore, parties may sometimes be forced to share data in order to comply with regulations or agreements. The open issues includes lots of important topics like improving Paillier scheme, adding update function to searchable encryption schemes for large databases [25],

designing dynamic search [44], and preserving cloud security.

While some of the challenges are introduced by new methods and developments in cryptography, the other challenges are more established. The most important aspect of a scheme is to be applicable in real world systems. Every system that has been proposed represents a trade-off between functionality, security, and performance.

In this project, we propose a framework for multi-client keyword search on outsourced encrypted electronic health records (EHR) using searchable encryption schemes.

3.10.1 Searchable Encryption Weaknesses

In general, a searchable encryption protocol may leak information in various ways. The main types of leakage in searchable encryption schemes are: information of indexes, search patterns, and access patterns.

Index information. This type of leakage basically points out the information about preserved documents and related contained keywords in those documents. The number of documents in the database, keywords in each document, and size of keyword list can be placed in this category. Some studies indicate that similarities between the documents belong to this class as well.

Search pattern. All information that may be leaked during the search process is categorized as the search pattern leakage group. The group may refer to two different searches which have the same results. The statistical analysis that may determine the actual searched keywords can be categorized in this class.

Access pattern. The access pattern group mainly covers the leaked information from query results. The comparison of results of two queries can express some ideas about restrictiveness of queries.

3.10.2 Ideal Solution

The ideal solution for SE schemes reveals nothing about the remotely stored files and indexes, results of queries, or the pattern of search. Most of the schemes, previously applied on health records, usually leak at least the search and access patterns. We introduce an efficient framework and illustrate its complexity, performance, and feasibility over EHR. That being said, the minimum information leakage in some steps of the scheme still persist. In Chapter 4, various related SE schemes, their performance, and their security are discussed.

Chapter 4

Related Works

All SE models can be reviewed according to different features. In each SE model, there are three kinds of parties. The first one is the data owner or data writer. The second one is the server or data storer. The third one is the reader or client who sends the queries to the server.

4.1 Symmetric vs. Asymmetric primitives

Some searchable encryption schemes, which are based on symmetric algorithms, allow only one user to create searchable ciphertexts from plaintexts and create trapdoors for those ciphertexts using a symmetric encryption key. In the traditional definition, it is also the same while reading and searching keywords on those ciphertexts. In other words, only the person who holds the symmetric key can read and search over

ciphertexts. These schemes are called single-user or single writer/single reader (S/S). In 2000, Song et al. [61] proposed the first single-user scheme. Afterwards, some symmetric schemes presented which changed the definition of symmetric searchable encryptions. Those schemes allowed more readers to query the database.

Asymmetric searchable schemes, which are based on asymmetric encryption and use private and public keys in their structure, can be searched and queried by more than one user. These schemes can be considered as multi-user schemes. In 2004, Boneh et al. [16] proposed the first SE scheme using public keys. That scheme was entitled "Public Key Encryption with Keyword Search" (PEKS).

4.2 SE's General Model

In 2000, Song et al. [61] achieved searchable encryption by crafting a two-layered structure. In this scheme, the owner outsourced the encrypted data and hashed keywords to the cloud server. This server can indicate that a keyword exists in ciphertexts by extracting the hash value of the keyword and comparing it with the embedded hashed keywords. While Song et al. proved that their system was IND-CPA secure, it was less trusted than an encryption algorithm without the searching advantage. There were some disadvantages for Song's scheme. The first one was that before encrypting a file, its text should be split into same sized strings. This method does not follow common file encryption standards. The other disadvantage was about their specific two-layer algorithm. This algorithm was only applicable for

text based files.

Brinkman et al. [23] applied Song and her colleagues' model to XML based files. Popa et al. [59] also illustrated Song et al.'s model can be used for encrypting databases.

4.3 Single-user vs. Multi-user

As discussed earlier, SE schemes are built on the client/server model, where the server stores encrypted data on behalf of one or more clients (i.e., the writers). To request content from the server, one or more clients (i.e., readers) are able to generate trapdoors for the server, which then searches on behalf of the client. In these models, the reader and writer can be either the same client or different clients. In the case that the writer and reader are different, the model is also called date sharing [19].

4.3.1 User Revocation

One of the important requirements of multi-reader schemes is the user revocation. The data owner cancels the access permission of a user to the owner's data by revoking a user [48]. There are two main ways for user revocation: direct [7, 47, 55] and indirect [6, 41, 58, 68, 71] revocation methods. In the direct method, the data owner re-encrypts data and specifies the revocation list. In the indirect method, the data owner, who periodically updates users' access keys, only updates non-revoked users' keys. In 2006, Curtmola et al. [30] applied the broadcast encryption (BE)

[36] in order to implement their multi-user scheme. The user revocation in the implementation of Curtmola et al. was challenging. This scheme had only one shared key for all readers, so the user revocation required distributing a new key between all current authorized users.

4.4 Searchable Encryption Architectures

As discussed in Section 4.3, SE models can be explored regarding the number of users participating in models. The SE models can be categorized into single-user and multi-user schemes.

4.4.1 Single-user Schemes

The general idea in a single-user scheme is that the user, who owns the symmetric encryption key, encrypts the plaintext and outsources the searchable ciphertext. The same person also creates trapdoors and provides those to the data holder or the cloud in order to search over the encrypted data. Asymmetric key systems are also used for single-user schemes.

4.4.1.1 The Practical Technique of Song et al.

As previously discussed in Section 4.2, Song et al. [61] proposed the first applicable SE scheme which was based on encrypting fixed-size words and embedding the related hash value for those words. The generated searchable ciphertext is outsourced to a

cloud server. For the search phase, the user sends the encrypted keyword and the key used for generating the related hash value of the keyword to the cloud server. The server later checks all embedded hash values with the received encrypted keyword and the key to verify if they match or not. The server sends the related ciphertext if they match up.

The scheme of Song et al. leaks the position of matched hash values and consequently the position of matched keyword and ciphertext in each query. Also, statistical analysis gives some information about the queries to the semi-trusted cloud server and allows for the possibility of the plaintext or keywords to be figured out.

4.4.1.2 Goh's Secure Indexes

In 2003, Goh [37] tackled some weaknesses and constraints of Song et al.'s searchable encryption model. Goh applied indexes for encrypted documents. They used a Bloom filter (BF) [14] for each document as the document's index. The Bloom filter, which is discussed in Section 3.4, provides linear search time in the number of encrypted documents. The structural issue of hash functions and Bloom filters is their false-positive possibility which means two different keywords have same Bloom array positions. Goh applied two level hash functions and unique document identifiers in order to mitigate false positive risks. A proper identifier should be selected to produce unique Bloom filters even for two documents with the same keyword sets.

4.4.1.3 Chang and Mitzenmacher's Prebuilt Dictionary

Chang and Mitzenmacher [27] proposed two solutions which were similar to what Goh presented. They used a prebuilt dictionary to generate an array of bits (index) for each document. Each bit in the array of a document indicated the presence of a keyword in that document. The prebuilt dictionary could be held in the client side (first solution) or the server side (second solution). In the second solution, the prebuilt dictionary should be encrypted for the sake of security. Both solutions supported the document collection's updating. Their security definition has been broken by Curtmola et al. [30].

4.4.1.4 Curtmola et al.'s Inverted Indexes

In addition to presenting a couple of standard security definitions, Curtmola et al. in 2006 [30] and later in 2011 [31] proposed two new constructions which satisfied their strong security definitions for SSE. Their schemes were based on associating indexes to distinct keywords instead of documents, which is called inverted indexes. Curtmola et al. illustrated that their schemes are optimal and more efficient than other previous SSE schemes. They also presented an extension of their schemes which was a multi-user searchable symmetric encryption (MSSE).

4.4.1.5 Liesdonk et al.'s SSE Schemes with Efficient Update

In 2010, Van Liesdonk et al. [64] proposed two new schemes which used an inverted index for keywords. Liesdonk et al. proved that their schemes are IND-CKA2 secure and satisfy Curtmola's security definitions. Their schemes also had logarithmic time complexity in the number of unique keywords for their secure search and their documents were updatable. Their first scheme was faster in searching computation while the second one had less communication overhead. The second scheme also efficiently provided updating stored documents as well as undoing the updates.

4.4.1.6 Effective Fuzzy Keyword Search Scheme

In 2010, Li et al. proposed a scheme to provide private fuzzy keyword search. This scheme acts like other SE schemes if the searched keyword matches with the predefined keyword list of outsourced encrypted data. However, the scheme was more powerful than regular SE schemes because the scheme returns possible matching files for the keywords which are similar to original searched keyword based on edit distance[1] metrics The idea behind their solution was to generate trapdoors for similar strings to the each main keyword in keyword list. They proved that their scheme is secure in their updated definition of IND-CKA1 which allowed encrypted indexes leak Edit distances.

[1] `Edit distance` is defined in Section 2.4

4.4.1.7 Chase and Kamara's Scheme for Structured Data

Chase and Kamara [28] targeted the issues of privately querying structured data. In order to achieve their purposes, they presented a novel searchable encryption model which was called structured encryption as well as the related security definition for the structured data. In general, their scheme generalizes the index generation of SSE for structured and labeled data. They illustrated the performance of their scheme for matrix-structured and labeled data. In summary, their scheme setup contains the following steps: padding data items, permuting the location of data items and matrix cells, and encrypting the permuted matrix cells. The cloud storage server can operate the query on matrices by receiving the permuted location of cells and encrypted contents. The server also runs search for labeled data as soon as it receives the permuted keyword. The server then returns permuted indexes of data items. While Chase and Kamara's scheme is IND-CKA2 secure and hides its structure from honest-but-curious server, the scheme leaks the search and access patterns.

4.4.1.8 Kamara et al.'s Dynamic SSE scheme

In 2012, Kamara et al. presented a new scheme in order to make SSE schemes more practical for real-world cloud storage servers. This IND-CKA2 secure scheme mainly provided the efficient updating of encrypted documents as well as operating the keyword search in a sub-linear time. Kamara et al. extended Curtmola et al.'s inverted index scheme [30] and applied homomorphic encryption to modify the

encrypted documents' pointers without decrypting them.

4.4.1.9 Bösch et al.'s Selective Document Retrival (SDR) Scheme

Bösch et al. [20] proposed a new cryptographic primitive called selective document retrieval (SDR) which is comparable with SSE. This primitive was used for outsourcing searchable encrypted data. They also provided a scheme based on SDR which guaranteed the privacy of indexes, trapdoors, and query results. Their searchable encrypted data construction was based on Chang and Mizenmacher's index generation [27] and homomorphic encryption. Their search algorithm also was implemented based on Brakerski and Vaikuntanathan's lattice-based symmetric scheme [22].

4.4.1.10 Cash et al.'s Conjunctive Keyword Search

In 2013, Cash et al. presented a new SSE scheme for conjunctive keyword search over outsourced encrypted data. The scheme implements its idea using communication with the cloud server. The idea of Cash et al.'s scheme, which is based on Curtmola et al.'s inverted index [30], is that the user creates an index for the least frequent keyword in the conjunction and sends specific index for the rest of keywords and sends them to the outsourced server. The server retrieves the encrypted index set for the least frequent keyword and checks if the other keywords belong to the index set and returns the encrypted IDs of documents which contains all keywords. Cash et al. generalized the definition of IND-CKA2 for conjunctive keyword search schemes and proved that their scheme is IND-CKA2 secure.

4.4.2 Multi-user Schemes

4.4.2.1 The First SE Scheme Using Public Key Encryption

In 2004, Boneh et al. [16] were the first group of researchers who proposed an asymmetric searchable encryption in 2004. This scheme is called public encryption keyword search (PEKS). PEKS enables a third-party to test weather a ciphertext, which is encrypted with a public key, contains a specific keyword. The third-party is required to receive a particular key from public key holder in order to check weather the ciphertext contains that specific keyword. PEKS implies Boneh and Franklin's Identity Based Encryption (IBE)[17]. While the security of PEKS can be proved based on the security of IBE constructions, PEKS scheme is vulnerable to the off-line keyword-guessing attack [24, 70]. This type of attack enables the third-party to store particular keys (trapdoor) and apply them to understand plaintext data.

Later in 2007, Boneh et al. [18] introduced a new scheme to provide private information retrieval (PIR) over encrypted data in order to preserve users' access patterns hidden. Their scheme provided the non-interactive communication between any user with the data owner. An extenssion to this scheme allowed Boneh el al.'s scheme to tolerate malicious users [18].

4.4.2.2 Using Group Ciphers for Bloom Filters by Bellovin and Cheswick

Using Pohlig-Hellman Encryption as a group cipher for Bloom filters, Bellovin and Cheswick [12], presented a scheme for secure and protected searches among mutu-

ally semi-trusted parties in 2004. Their model enabled a semi-trusted server, which holds the database, to perform queries of multiple clients in such a way that neither database owner nor the server understands the original queries of clients. The main issue of their scheme is allowing false-positives due to the use of Bloom filters.

4.4.2.3 Curtmola et al.'s Multi-User Setting

In 2006, Curtmola et al. [30] presented two algorithms for mutli-user SE. They updated their scheme in 2011 [31]. They defined the multi-user setting for searchable symmetric encryption. In this scheme, an arbitrary group of users are authorized to query the encrypted data saved on a semi-trusted server. This model, using the efficient structure, enabled the data owner to revoke query privileges from current users and authorize new users. The presenters of the multi-user setting considered that the cloud server was honest-but-curious. The idea behind the multi-user setting is to combine a single-user SSE scheme with a broadcast encryption algorithm. Nevertheless, they stated that their model can even be robust against malicious servers if they applied memory checking [15] and universal arguments techniques [9].

4.4.2.4 Baek et al.'s Revisited PEKS Scheme

Baek et al. [8] proposed an updated PEKS in order to address some issues in Boneh et al.'s original scheme [16] which were unsolved. Baek et al. indicated that their scheme had removed secure channel. Their scheme also supported refreshing keywords and processing multiple keywords. The main idea behind their scheme was adding key

pair for the storage server. They also proved the security of their model.

4.4.2.5 Applying Query Rerouter

Raykova et al. [60] in their proposed definition and implementation, which was called Secure Anonymous Database Search, applied two trusted parties the query router (QR) and index server (IS). QR and IS are trusted and receive a limited information which enables them to operate. IS holds secure indexes which produced by the data owner and operates actual query without learning about queries and data. QR route queries and results between queriers and IS without revealing the identity of queriers. Raykova et al. introduced Re-routable Encryption which translates the encrypted queries for IS in such a way that keeps queries untraceable. The proposed scheme by Rayka et al. allows false-positives because of using Bloom filter.

4.4.2.6 Yang et al.'s Bilinear Map

In 2011, Yang et al. [69] presented a scheme for multi-user private queries on an encrypted database with user revocation ability. In this scheme, allocating a distinct key to each user prevented the re-encryption of the database and re-generation of query keys after revoking a user. They applied bilinear maps in their scheme so all users generate a same index for a specific key. Yang et al. presented an extended definition of IND-CKA2 for the multi-user keyword search and proved that their scheme fulfills the definition's conditions.

4.4.2.7 Jarecki et al.'s Multi-Client OXT and Symmetric Private Information Retrieval (PIR) schemes

Jarecki et al. [42] proposed an extension of the OXT scheme for multi-user SSE which is called mulit-client SSE (MC-SSE). They also proposed another scheme which preserves the privacy of outsourced database third-parties or clients from the data owner. This feature is an extra to what basic SSE schemes provide. The SSE schemes allow the authorized clients only access to the results of what they queried while the cloud server does not learn anything about the queries and plaintext results. In this scheme Jarecki et al. extended the Cash et al.'s OXT scheme [26]. For the security definition of the scheme, Jarecki et al. considered the data owner as an adversarial entity in addition to the cloud server which was previously defined as an adversary by SSE security definitions. The designers of this scheme proved that their scheme is secure regarding their definition.

4.4.2.8 Orencik et al.'s Multi-Keyword Search

In 2016, Orencik et al. proposed a multi-keyword search over encrypted data while preserving the privacy of the search operation. In this scheme, Orencik et al. addressed the leaking of access patterns as well as the correlation of queries and their results issues. They also presented a new compression method to mitigate the communication between the scheme's parties as well as a novel raking and scoring algorithm for their multi-keyword search scheme. In addition to their schemes, they

proposed security definition which preserves privacy of queries and distinguishability. Orencik et al. proved that their scheme meets the definition's criteria.

4.4.2.9 Dai et al.'s SSE Schemes Against Memory Leakage

In 2016, Dai et al. proposed two schemes which addressed the memory leakage problem of SSE schemes. In their first IND-CKA2 secure scheme, which called memory leakage-resilient searchable symmetric encryption (MLR-SSE), Dai et al. extended Curtmola's scheme [30] and applied physical unclonable functions (PUF)[2] in order to strengthen the privacy of private-keys. Dai et al. also presented dynamic MLR-SSE (DMLR-SSE) which was more efficient and non-adaptive indistinguishability secure. Dai et al. proved that the DMLR-SSE is as efficient as Cash et al.'s SSE scheme [25] while it is more secure than Cash et al.'s scheme because Dai et al. applied PUFs in their scheme.

[2]A `physical unclonable function` is a hardware which provides a unique identity for semiconductors like processors

Chapter 5

Proposed System

In this chapter, a framework is proposed for the privacy preserving multi-user keyword search over outsourced encrypted health data. The recently proposed SSE scheme by Jarecki et al. [42], which is discussed in Sections 3.7, 3.8, and 3.9, is applied. This searchable symmetric encryption (SSE) scheme provides search ability over encrypted information for authorized users. The data owner, who outsources his/her encrypted data to cloud storage servers, authorizes the users to access the encrypted data.

5.1 Involved Models and Parties

This section introduces the parties and their duties in the proposed privacy-preserving keyword search over encrypted health data system. In this section, use case dia-

grams are applied to illustrate all parties responsibilities in our system. Parties in our framework are introduced as follows:

Patient The patient or the data owner processes and outsources the data to the cloud server. In a single-user architecture, the patient, who owns and keeps the encryption key, runs the query at the cloud server and is able to decrypt the retrieved encrypted results. However, in a multi-user architecture, the patient delegates the search ability to the authorized users. In other words, the patient provides search tokens to the authorized users.

Cloud Server The cloud storage server receives the encrypted information as well as encrypted indexes or meta-data and protects them against loss or theft. However, in our scheme the cloud server is considered as a semi-trusted or honest-but-curious one which executes the search operation and returns the results honestly. This means, while the server cannot learn the plaintext information or searched keyword, it may act as an adversary and leak access patterns or other information about outsourced encrypted data.

User The user in our environment can be physicians, pharmacists, or insurers. A user wants to search a keyword in the outsourced data by the patient or data owner.

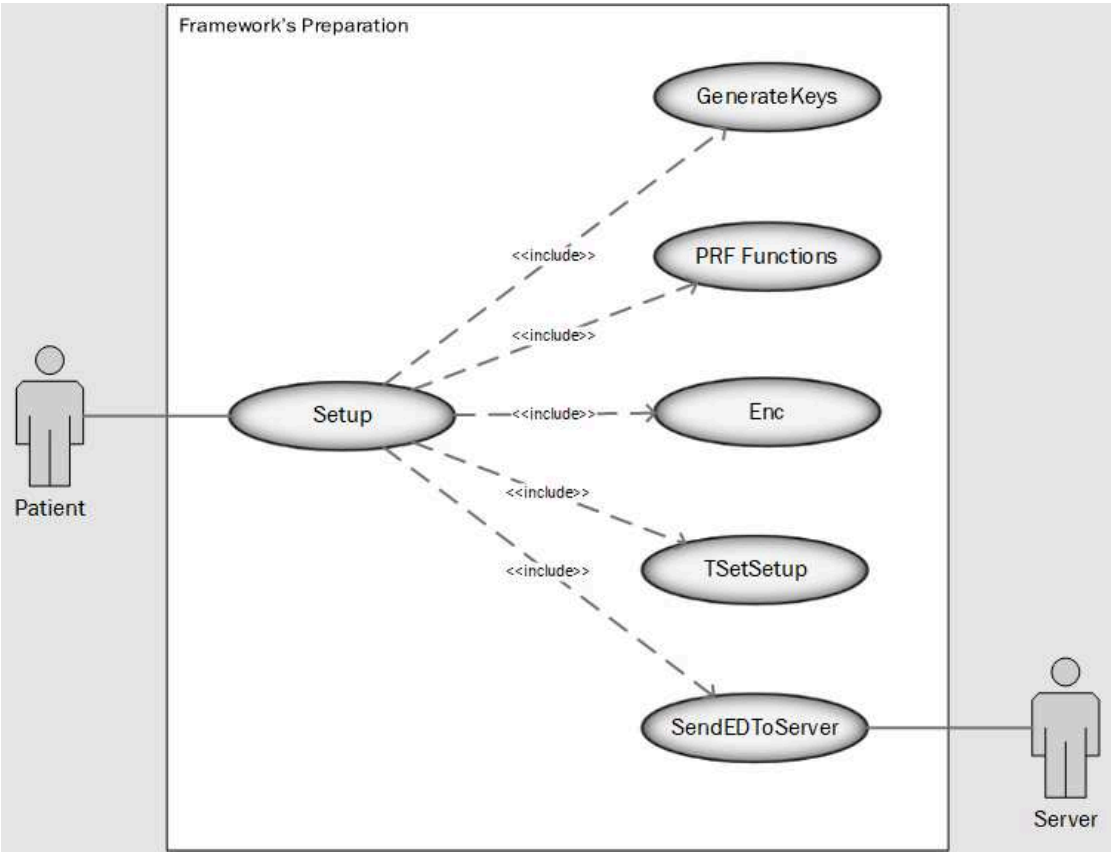

Figure 5.1: Use case diagram of the framework's preparation subsystem

5.2 The framework's UML Diagrams

Various UML diagrams were applied in order to illustrate the structure of our system along with all the constitutive classes as well as users and actors who perform actions and roles in interaction with our system.

Figure 5.1 and Figure 5.2 display the use case diagrams of our framework and involved actors. As shown in Figure 5.1, *Patient* uses the *Setup* use case. This use case prepares the encrypted data and all requirements for securely running queries over the encrypted database. *Setup* includes *GenerateKeys*, *PRFFunctions*, *Enc*, *TSetSetup*, and *SendEDToServer* use cases which are briefly discussed as follows:

1. *GenerateKeys*: This use case is responsible for generating unique keys for PRF functions as well as paired keys for encryption and decryption methods.

2. *PRFFunctions*: The *PRFFunctions* use case provides PRFs. A pseudo random function (PRF) efficiently maps the domain and range of the function. A PRF is considered as a good one if it is not distinguishable from a real random function.

3. *Enc*: *Enc* encrypts the indexes to be stored in tuples of *TSet*.

4. *TSetSetup*: The duty of this use case is the creation of *TSet* out of the set of encrypted indexes.

5. *SendEDToServer*: *SendEDToServer*, included by the *Setup* use case, is responsible for sending the encrypted data to the *Server*. In the preparation section of

the framework, *Server* also deals with *SendEDToServer* use case and receives the encrypted data from the data owner.

The other provided use case diagram, Figure 5.2, exhibits actors and use cases participating in the search protocol. The *User* actor presents *Physician*, *Insurer*, or *Pharmacist* actors. The main use case in Figure 5.2, *Search*, includes the following use cases:

1. *TokenGeneration*: The *Patient* actor creates the token for the received query using *TokenGeneration* and includes three use cases.

   (a) *FindLeastFrequentKeywordQuery*: The data owner applies this use case to select the least frequent keyword in the dataset among all keywords of the query.

   (b) *TSetGetTag*: This use case retrieves the least frequent keyword's *stag*.

   (c) *AuthEnc*: The duty of this use case is encrypting and signing *stag* and blinding factors.

   (d) *PRFFunctions*: The *Patient* applies *PRFFunctions* for generating the least frequent keyword's *strap*.

   (e) *UserSideSearch*: *User* applies the *UserSideSearch* use case to receive the token of his/her query as well as to send *bxtoken*s to *Server*.

      i. *AskTokenFromPatientGivenQuery*: In this use case, *User* receives token for the query if he/she got authenticated.

ii. *Dec*: This *use case* decrypts those received indexes from *Server*.

iii. *PRFFunctions*: The *User* actor applies this use case to generate encryption keys.

2. *ServerSideSearch*: The *Server* actor authenticates *User* and both retrieves and forwards encrypted indexes to *User* if the criteria are met.

   (a) *VerifyAuthDec*: This use case decrypts the *env* and verifies *stag* which consequently authenticates *User*.

   (b) *AskCthTokenArrayFormUser*: This use case requests a specific *bxtoken*.

   (c) *TSetRetrieve*: This use case retrieves the specific tuple from $TSet$.

User applies *AskTokenFromPatientGivenQuery*, gives query to *Patient*, and requests the related encrypted token. *User* sends blinded tokens to *Server* and receives the encrypted indexes if those blinded traps match any indexes in $XSet$. *ServerSideSearch*, *UserSideSearch*, and all other included use cases, shown in Figure 5.2, contribute to the search protocol.

Figure 5.3, which is related to the class diagram of the proposed framework, reveals implemented classes and their relationships. In the following section, I briefly discuss properties and methods of each entity in Figure 5.3.

Query: This class only has a set of strings. Both *Patient* and *User* classes own an object of this class and use that object to keep the boolean query which should be queried on the cloud server.

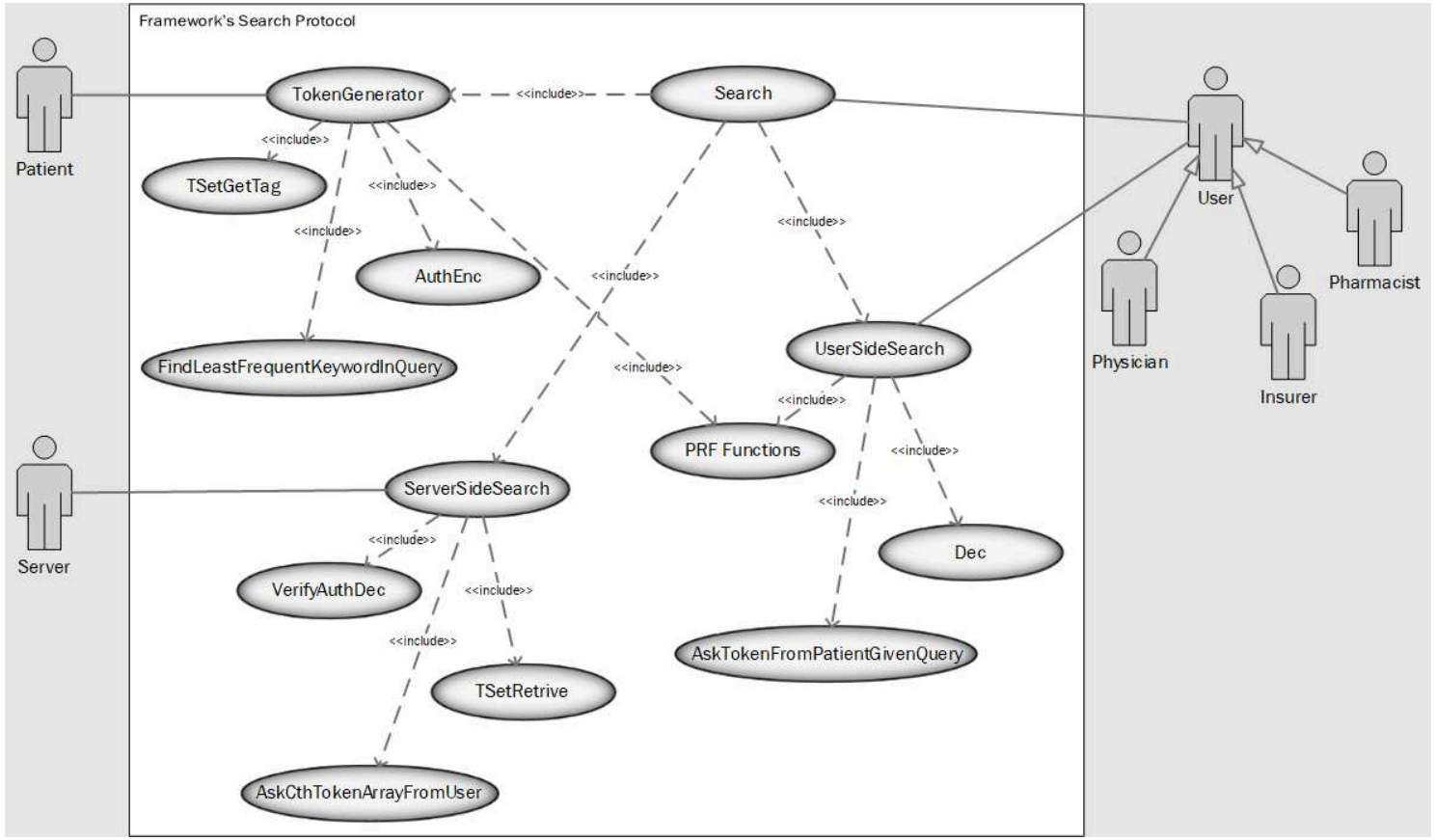

Figure 5.2: Use case diagram of the framework's search protocol

Token: *Token* includes the encrypted envelope which is prepared by *Patient* and should be sent to *Server* by *User*. The *User* class has an object of the *Token* class. *Token* also contains an array of bytes for *strap* which is token for least frequent keyword of the query as well as a two-dimensional array of bytes for *bxtrap*s, which are trapdoors and their usages are discussed in detail in Section 5.3.

PlainEnv: This class has values for *stag* and all blinding factors(ρ_i). Both Patient and Server classes have an object of PlainEnv class as their property.

Patient: The *Patient* class is the most important class in the framework. The *Patient* class holds a set of *xtag*s as the $XSet$. A tuple set, known as $TSet$, is added to the *Patient* class. $TSet$ preserves a list of encrypted inverted indexes which are associated to each keyword. The $TSet$ datastructure has three specific algorithms (*TSetSetup*, *TSetGetTag*, and *TSetRetrieve*) for its setup and access. *Patient* also has an object of *Token* to save the information of *env*, *strap*s and *bxstrap*s which should be sent to *User*. *bxtrap*s are the blinded *xtraps* which are generated by the *Patient* for received *Query*. The *Patient* class uses the $GenerateKeys$ method to prepare a set of keys for encryption methods and pseudo random functions. *AuthEnc* method, which takes the encryption key (K_M), *stag*, and random blinding factors, encrypts its inputs and outputs *env* which determines whether the querier is authenticated.

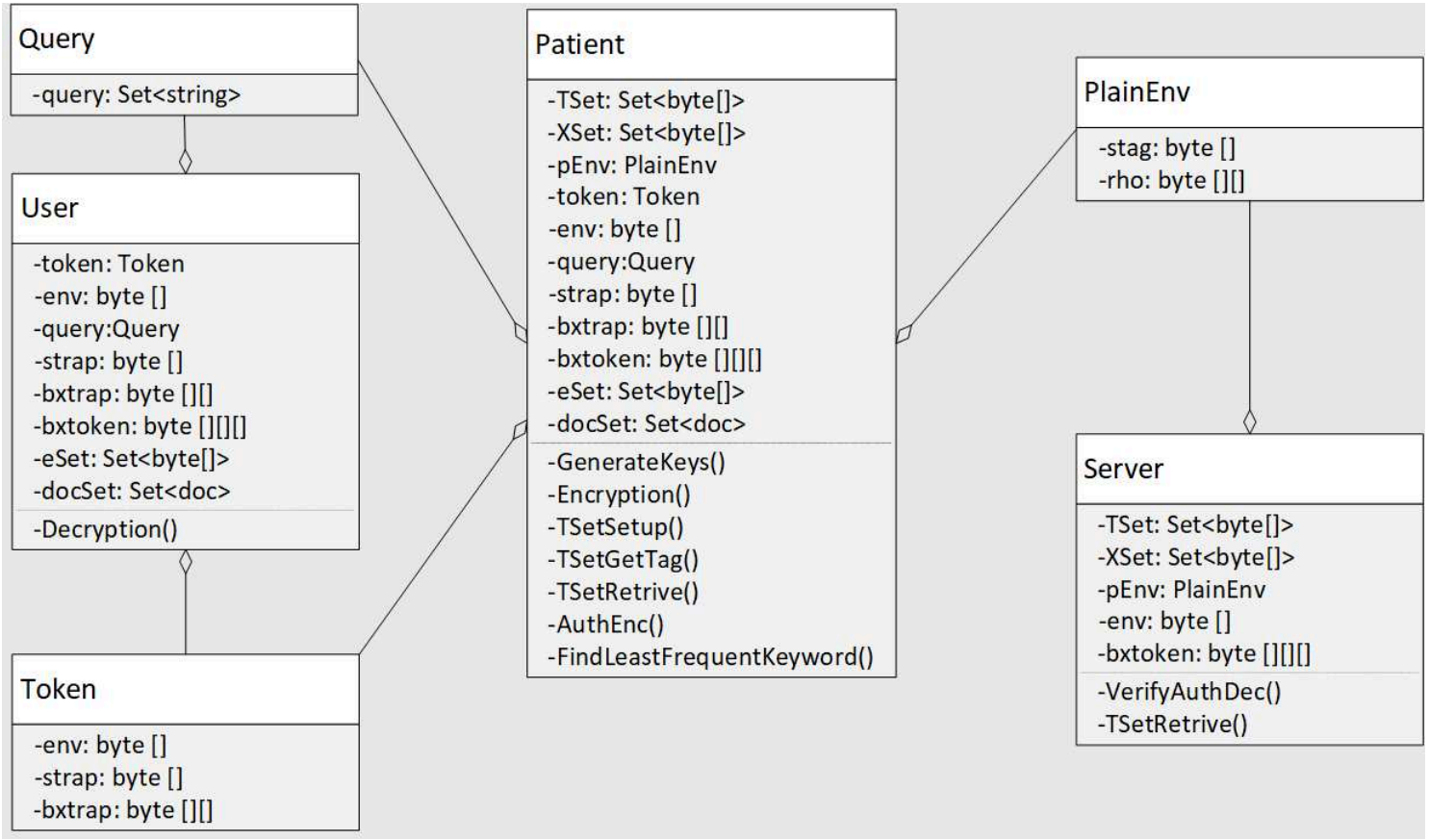

Figure 5.3: Class diagram of the framework

User: The *User* class, which is designed for the querier in the framework, saves its query in an object of the *Query* class. By receiving the *env* from *Patient*, *User* saves it in token in addition to *strap* and generated *bxtrap*s. The user also saves encrypted indexes e, which are received from server, in $eSet$. The *Decryption* method of the *User* class is invoked after all inverted indexes are saved in $eSet$ and *Server*, sends stop which means that the search process is done.

Server: The *Server* class preserves $TSet$ and $XSet$ as searchable encrypted data. The *VerifyAuthDec* method of *Server* decrypts *env* and obtains blinding factors. Obtaining valid factors proves that the user is authenticated. *Server* applies *TSetRetrieve* to achieve the related tuple of the received *stag* from $TSet$.

5.3 Process Flows

The major goal of this section is to demonstrate how processes and algorithms combine to drive the privacy-preserving query processing system.

5.3.1 Prepare Encrypted Data

As illustrated in Algorithm 4, before outsourcing data to a cloud server, the data owner, which is a patient in our system, needs to create the searchable encrypted

Algorithm 4 Prepare Encrypted Data

1: **procedure** SETUP(Data D, Array of Encryption Keys $RDK[]$)
2: $XSet \leftarrow$ empty set for xtags
3: $T \leftarrow$ empty associative array for all allowed keywords (W)
4: $KeySet \leftarrow GenerateKeys()$ ▷ $KeySet$ contains K_S, K_X, K_T, and K_M
5: **for each** $w \in W$ **do**
6: $strap \leftarrow F_1(K_S, w)$ ▷ F_1 is a pseudo random function
7: $K_e \leftarrow F_1(strap, 2)$ and $K_z \leftarrow F_1(strap, 1)$
8: $c \leftarrow 1$
9: **for each** ind_i of D's documents which contains w **do**
10: $rdk_i \leftarrow \text{RDK}(ind_i)$ ▷ rdk_i is the encryption key for document i
11: $xind \leftarrow F_2(K_I, ind_i)$ ▷ F_2 is a pseudo random function
12: $z_c \leftarrow F_2(K_z, c)$
13: $y \leftarrow xind \cdot z_c^{-1}$
14: $e \leftarrow Enc(K_e, ind_i)$
15: $t \leftarrow t \cup (e, y)$
16: $xtag \leftarrow g^{F_2(K_X, w) \cdot xind}$
17: $XSet \leftarrow XSet \cup xtag$
18: $c \leftarrow c + 1$
19: **end for**
20: $T[w] \leftarrow t$
21: **end for**
22: $(TSet, K_T) \leftarrow TSetSetup(T)$
23: $ED = (TSet, XSet, K_M)$ ▷ K_M is a shared key between *Patient* and *Server*
24: $SendEDToServer(ED)$
25: **end procedure**

data. In this regard, the patient should operate the *SETUP* procedure. In the first step of the *SETUP* procedure, the required keys for encryption and *PRFFunction*s are generated. Afterwards, for each keyword in the provided keyword list of data, all encrypted indexes are stored in $TSet$ and generated *xtag*s are accumulated in $XSet$. The accumulated *xtag*s help the server to indicate that an encrypted keyword exists in an encrypted document without revealing any information about the plain keyword or document. Finally, the patient has to outsource $TSet$ and $XSet$ as the encrypted data to the server. In order to delegate the search ability to clients, the patient and server are required to share an encryption key (K_M) for encrypting the *stag* and blinding factors and authenticate the clients.

5.3.2 Query Execution

In this section, I discuss the query process in our framework. The query or search in a multi-client framework consists of three procedures which should be operated by all three parties of our framework. These three procedures are *TokenGenerator*, *UserSideSearch*, and *ServerSideSearch* which are illustrated in Algorithms 5, 6, and 7.

At the initial step, the client sends his/her query to the patient. Algorithm 5 shows that after receiving the plain query, the patient generates the *s-terms* and *bxtrap*s for the query. As discussed in Sections 3.7, 3.8, and 3.9, *s-terms* include *stag* and *strap*. *stag* is a secure inverted index of a keyword of the query which

Algorithm 5 Generates authorized query (token) for User

procedure TokenGenerator($KeySet$, Query $\bar{w}$)
  $w_1 \leftarrow FindLeastFrequentKeywordInQuery(\bar{w})$
  $stag \leftarrow TSetGetTag(K_T, w_1)$
  $strap \leftarrow F_1(K_S, w_1)$ ▷ F_1 is a pseudo random function
  $n \leftarrow |\bar{w}|$ ▷ n is the number of keywords in Query
  for each $w_i \in \bar{w}$ where $i = 2, \ldots, n$ **do**
    $\rho_i \leftarrow$ random blinding factor from Z_p^* domain
    $bxtrap_i \leftarrow g^{F_p(K_X, w_i) \cdot \rho_i}$
  end for
  $env \leftarrow AuthEnc(K_M, (stag, \rho_2, \ldots, \rho_n))$ ▷ terms should be signed by *Patient*
  $token \leftarrow \{env, strap, bxtrap_2, \ldots, bxtrap_n\}$
  $SendTokenToUser(token)$
end procedure

is sent to the server and allows server to access the related encrypted records and documents which contain this specific keyword. This keyword of the query has the least frequency in the outsourced data in comparison to other keywords contained in the query. *Xtrap*s are trapdoors of the other keywords in the query which helps the server to understand if a document has those keywords or not. However, in order to keep the *xtrap*s secure from the user, the patient applies blinding factors on *xtrap*s and produce *bxtraps*. Finally, the server encrypts the blinding factors and the *stag* and encapsulates those as the *env* and sends that along produced *bxtraps* and *strap* as the query token to the user who initially sends the plain query. Later, the user

should send *env* to the server which proves that the user is authenticated by the patient.

The next couple of procedures for the keyword search, which are depicted in Algorithm 6 and Algorithm 7, should be operated simultaneously by the querier (*User*) and the server.

In the user side, which is explained in Algorithm 6, the querier(*User*) sends *env* and *stag* to the server as soon as getting *token* from the data owner (*Patient*). Afterwards, the querier starts to generate *bxtoken* continuously and sends them to the server, until it receives the *stop* command from the server. Each *bxtoken* contains a modified $bxtrap_i^{z_c}$ for each keyword in the query which helps server to check if that keyword of the query belongs to the document ind_c or not. The user receives an encrypted document if all keywords belong to that document. The user later can decrypt that document using K_e which is generated by a PRF using *strap*.

As displayed in Algorithm 7, in the server side of the keyword search, the server checks if the querier is authenticated by decrypting the received *env* using the shared key. As explained earlier, the server receives a shared encryption key from the data owner (*Patient*) in *Setup* phase. Afterwards, the server accesses the tuple list and encrypted records which contains the least frequent keyword by applying *stag* in *TSetRetrieve* as the input. Subsequently, the server receives *bxtoken*s from the user and checks if all $bxtrap[i]^{y/\rho_i}$s in each *bxtoken* are members of $XSet$. If any *bxtoken* satisfies mentioned condition, the server sends the related encrypted record to the

Algorithm 6 User Side of the Search Protocol

1: **procedure** UserSideSearch
2: $token \leftarrow AskTokenFromPatientGivenQuery(\bar{w})$
3: $SendStagToServer(stag)$ and
4: $SendEnvToServer(env)$ ▷ env is encrypted and signed by $Patient$
5: $K_e \leftarrow F_1(strap, 2)$ and $K_z \leftarrow F_1(strap, 1)$
6: $eSet \leftarrow$ empty set of encrypted documents
7: $docSet \leftarrow$ empty set of decrypted documents
8: **while** $ReceiveStopFromServer()$ **do**
9: $z_c \leftarrow F_1(K_z, c)$ ▷ F_1 is a pseudo random function
10: $bxtoken[c] \leftarrow$ empty array ▷ bxtoken is a two-dimensional array
11: **for** $i = 2, \ldots, n$ **do**
12: $bxtoken[c][i] \leftarrow bxtrap_i^{z_c}$
13: **end for**
14: $SendBXtokenToServer(bxtoken[c])$
15: **if** $e \leftarrow ReceiveEFromServer()$ **then**
16: $eSet \leftarrow eSet \cup e$
17: **end if**
18: $c \leftarrow c + 1$
19: **end while**
20: **for each** e_j in $eSet$ **do**
21: $ind_j \leftarrow Dec(K_e, e_j)$
22: $docSet \leftarrow docSet \cup ind_j$
23: **end for**
24: **end procedure**

Algorithm 7 Server Side of the Search Protocol

1: **procedure** ServerSideSearch(Encrypted Date ED) ▷ ED contains $TSet$ and $XSet$
2: $env \leftarrow AskEnvFromUser()$
3: $(stag, \rho_2, \ldots, \rho_n) \leftarrow VerifyAuthDec(K_M, env)$
4: **if** $stag$ is NULL **then**
5: $retrun$
6: **end if**
7: **for** $c = 1, \ldots, sizeof(t)$ **do**
8: $bxtoken[c] \leftarrow AskCthTokenArrayFromUser()$
9: $(e_c, y_c) \leftarrow c^{th}$ tuple of t
10: $checkTerms \leftarrow true$
11: **for** $i = 2, \ldots, n$ **do** ▷ n is the number of keywords in the query
12: **if** $bxtoken[c][i]^{y/\rho_i}$ is not member of $XSet$ **then**
13: $checkTerms \leftarrow false$
14: **end if**
15: **end for**
16: **if** $checkTerms$ is true **then**
17: $SendEToUser(e_c)$
18: **end if**
19: **end for**
20: $SendStopToPatient()$
21: **end procedure**

user. In other words, this part of the algorithm checks if encrypted documents, which contain the least frequent keyword, contain the rest of keywords in the query. The algorithm then sends those documents which meet the requirements to the querier.

In the following chapter, the implementation and experimental results of the proposed system.

Chapter 6

Implementation and Experimental Results

This chapter discusses the implementation of the framework for Privacy-Preserving Query Processing on Health Data. In other words, the implemented framework provides searchable symmetric encryption for multiple users in the health area. This framework is implemented in Java language using the IntelliJ IDEA framework. The implemented framework uses Maven, a build automation tool, to build the project and its libraries. This project engages various libraries in order to enhance the performance and to provide more features. The most important applied libraries are the Clusion library and the Bouncy Castle library. Clusion is a handy library which provides searchable symmetric encryption algorithms, as well as other functionalities, such as Indexing, which are essential for this project. The Bouncy Castle

library prepares the cryptographic primitives. In the following sections, I give a brief introduction to the applied programming language, programming framework, and libraries.

6.1 Programming Language

6.1.1 Java

Java is known as one of the most popular languages in desktop, mobile, and client-server web development. In 2013, Beneke and Wieldt announced that 9 million developers use Java as their programming language around the world. Java is a concurrent, object-oriented, class-based programming language. The creators of Java intended that this programming language have few implementation dependencies. The Java slogan, "write once, run anywhere", reflects the fewer dependencies concept. This phrase means that a Java application, which is compiled on a platform, can run on all other platforms and operating systems without recompilation. The Java compiler converts the Java applications' source codes to bytecodes (Java binary code) which can run on every platform that operates Java virtual machine (JVM). JVM is a virtual computing machine that interprets Java bytecodes. In other words, JVM is an interpreter between Java binary codes and operating systems. This feature in Java language supports the project to easily run on all of the various current platforms.

6.2 Integrated Development Environment (IDE)

6.2.1 IntelliJ IDEA

IntelliJ IDEA is a favourite integrated development environment (IDE) for developing programs in Java language. The JetBrains company developed this IDE in the community and proprietary editions, but both can be used for commercial development. In the Infoworld report, IntelliJ achieved the first rank in overall score between the most favourite Java programming IDEs: Eclipse, NetBeans, JDeveloper, and JetBrains IntelliJ IDEA. In this benchmark, the documentation, ease of use, plug-in ecosystem, and Java features were assessed [13]. In 2014, Google developed its first Android IDE, Android Studio 1.0, based on the free edition of IntelliJ IDEA. The latest version of IntelliJ IDEA supports Java 9, provides UI designer for Android, and Play 2.0 for Scala [66].

6.3 Apache Maven

Apache Maven is mainly used as an automation build tool for Java language which describes how the target program or application has to be built as well as the dependencies. An XML file for each project determines the steps of building the application, dependencies, modules, structures, folders and required libraries, and plug-ins. Maven handles dependencies of the project and automatically downloads, stores, and uses them to build the target application [1].

6.4 Bouncy Castle Library

The Bouncy Castle library is a well-known cryptography library which provides many cryptographic algorithms and APIs in Java and C# languages. Bouncy Castle includes low-level and high-level components which are the light-weight API and the Java cryptography extension (JCE) provider. The first level component, or light-weight API, provides all base cryptographic algorithms. Some developers use the low-level component of Bouncy Castle for applications and devices with limited memory. The JCE provider is using the other component and provides easy-to-use cryptographic methods with a lot of predefined configurations. Many Java projects and applications, which require cryptographic operations, use JCE provider [2].

6.5 The Clusion Library

The Clusion library, which is implemented in Java language, provides multiple searchable symmetric encryption (SSE) schemes as modules. This library supports recent studies and popular operations like disjunctive, conjunctive, and boolean keyword search. All implemented schemes in Clusion generally have a sub-linear time complexity for their search phase. The Clusion library uses Bouncy Castle Library in the back-end which is introduced in Section 6.4. The Clusion library, which is provided under the GNU General Public License v3 (GPLv3), is easily accessible on the Internet [43, 51].

6.5.1 Manipulate Various Record Types

Clusion supports various types of files as the data which can be outsourced. PDF files, Microsoft Word, Microsoft Power Point, HTML, and txt files are all included. Clusion applies *Apache Lucene* in order to obtain the keyword list for all data. In order to manipulate PDF and Microsoft documents, Clusion uses *Apache PDFBox* and *Apache POI* libraries.

6.6 Experimental Setup

I conducted some experiments on a high-end desktop with an Intel Core i7-4790 CPU @ 3.60 GHz (8 CPUs) as processor and a 16GB RAM as memory, running Microsoft Windows 10 64-bit (Build 15063).

My analysis demonstrates the performance of the implemented method and framework regarding computation and execution time, as well as storage and communication overheads. The computation and execution time calculated for the searchable encryption scheme includes the construction of the inverted index, building the searchable encrypted data, determining unique keywords, and their frequencies for all documents, generating the associated dictionary, and running the actual search.

6.6.1 Dataset

I selected the dataset of "EHR Products Used for Meaningful Use Attestation" which was publicly available on the Health IT dashboard website [4]. This dataset was acquired over the Medicare Electronic Health Record Incentive Program (MEIP). This program motivates EHR providers to upgrade and show that they are adopting certified EHR technology. The MEIP dataset [34] in the accessed time had 1932500 records. This data was acquired in seven years of the EHR Incentive Program. I applied an application to create a separate file for each record to test the performance of the developed framework for various sizes of data. Table 6.1 provides the description for all data fields in the dataset.

Table 6.1: Dataset documentation

Data Field	Data Description
NPI	National Provider Identifier
Provider_Type	Type of Health Care Provider
Business_State_Territory	U.S. State or Territorial Location of Provider
ZIP	ZIP Code Where Hospital or Health Care Professional Practice Is Located.
Specialty	Clinical Specialty of Health Care Provider
Hospital_Type	Type of Hospital.
Program_Type	CMS Incentive Program in Which Provider Is Registered
Vendor_Name	Electronic Health Record (EHR) Vendor Name
EHR_Product_Name	Electronic Health Record (EHR) Product Name
Product_Classification	Electronic Health Record (EHR) Product Classification
Attestation_ID	Unique Identification Number for Each Meaningful Use Attestation

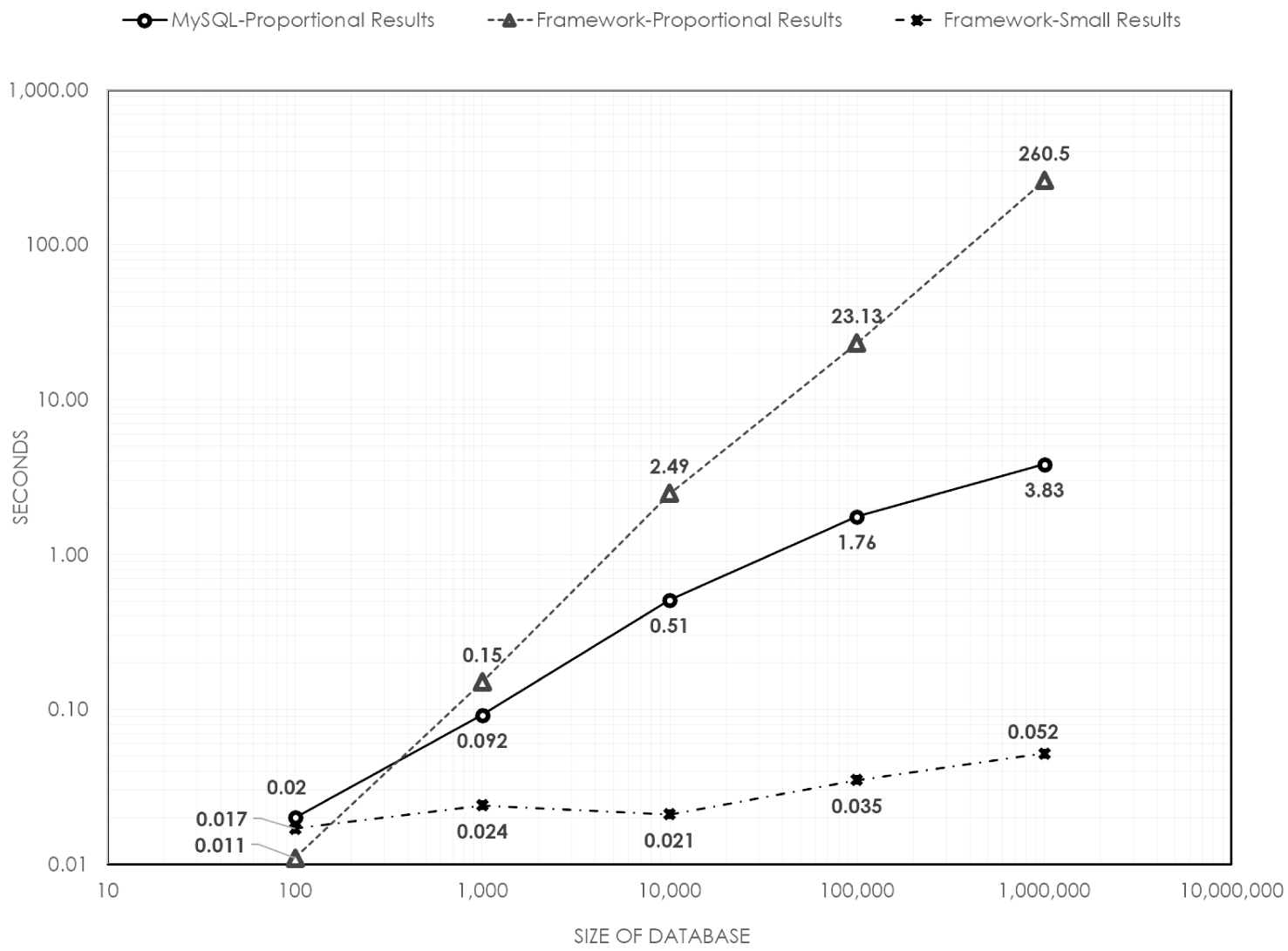

Figure 6.1: Scaling Database Size

6.7 Results

The experiments of this thesis assessed the performance of the implemented framework for privacy preserving query processing. These experiments assessed the performance of the framework in various ways. In these experiments, the corresponding effects of the following paradigms on the performance and execution time were investigated: a database's scale; the conjunctive multi-term query; constant result set. In order to assess the scalability of the framework of the query engine, I generated several subsets of my dataset in various sizes. The original dataset of experiments had about two million records of EHR.

The first experiment investigated the execution time of single-term searches for constant result sets and proportional result sets. I also demonstrated the execution time for proportional result sets by MySQL. Figure 6.1 shows that the execution time of a privacy preserving query in the framework has a linear relation with the database size and the execution time is independent if the size of result set is constant. Figure 6.1 also shows that the framework provides results faster than MySQL only for single-term queries with small result sets. In Figure 6.1, the vertical and horizontal axis are respectively related to the time and size of database and both are shown in the logarithmic scale.

In the second experiment, single-term queries versus two-term queries were assessed. Figure 6.2 shows the results of single-term and two-term queries with various selectivities. The selectivity of a keyword w_i ($|\mathcal{DB}[w_i]|$) is the number of all doc-

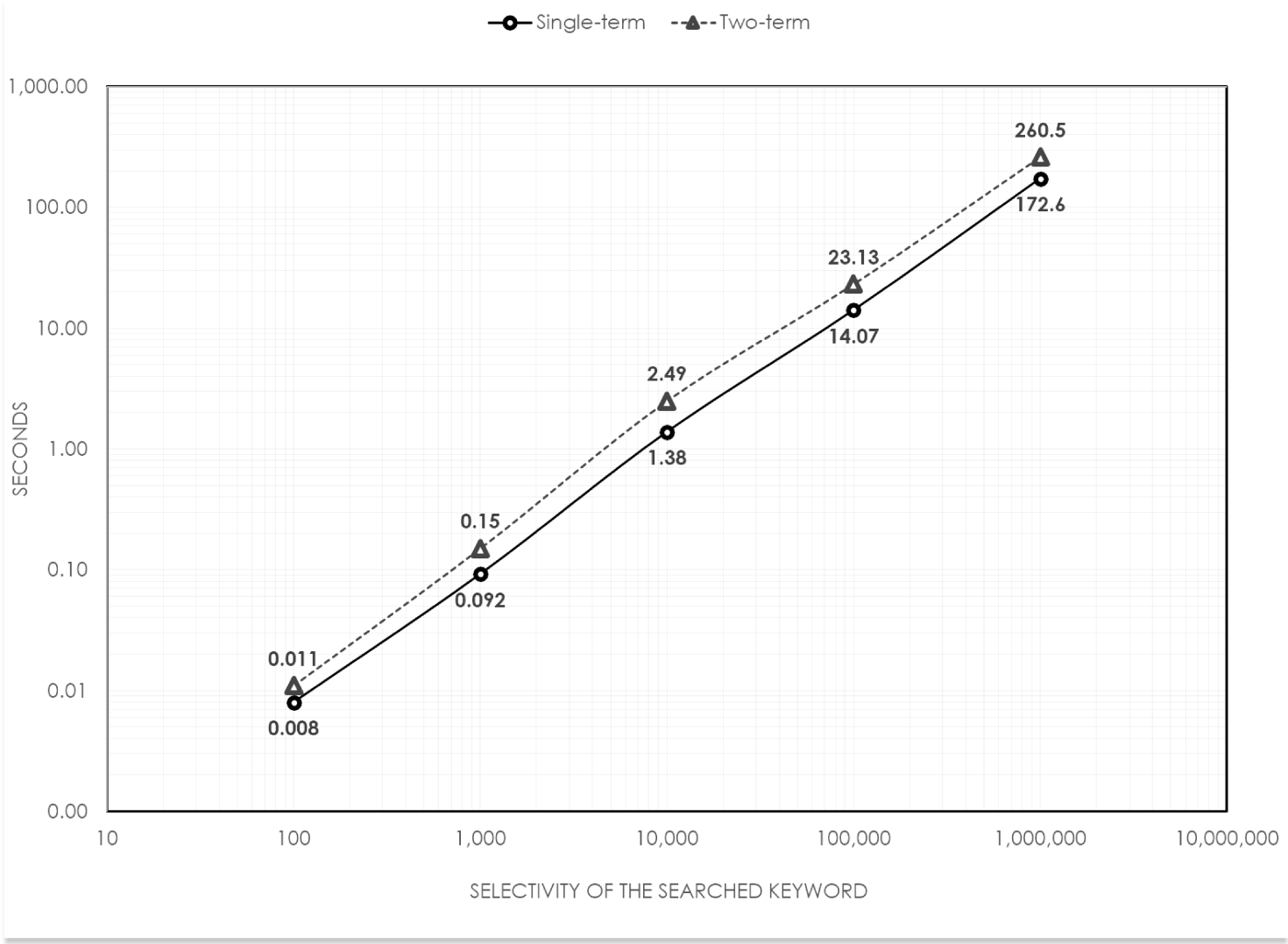

Figure 6.2: Single-Term and Two-Term Queries Against Selectivity

uments which contain that keyword (w_i) in the dataset. Figure 6.2 demonstrates execution times for querying "*Dermatology*" in various databases where the selectivity of that keyword can be 100, 1,000, 10,000, 100,000, and 1,000,000. Figure 6.2 also shows execution times for querying the conjunctive keywords "*Dermatology*" and "*Illinois*" for the mentioned various databases and selectivities. In all queries, the "*Dermatology*" term acted as the *s-term*. For two-term queries, all tuples for the *s-term* retrieved from the $TSet$ are checked against the $XSet$. While two-term queries had to do more duties and check retrieved tuples against $XSet$, their execution times were not too much longer than single-term queries. In other words, the execution time was totally affected by disk input/output and network latencies.

The last experiment of this thesis investigated the effect of the result set's size or constant selectivity on the execution time while querying databases with different sizes. This experiment showed that two-term queries on various databases, which received the same result sets, have almost the same running times. Figure 6.3 illustrates that "*Nephrology*" and "*Dermatology*" keywords, which had constant selectivities of 100 and 10000 in all databases, spent 0.01 second and 2.5 seconds for their query executions in all databases respectively. In other words, the execution time is mostly impressed by tuple retrievals and I/O latencies, which were observed in the previous experiment as well.

In summary, the experiments showed that the implemented framework is very effective having about two million records of EHR. The framework ran significantly

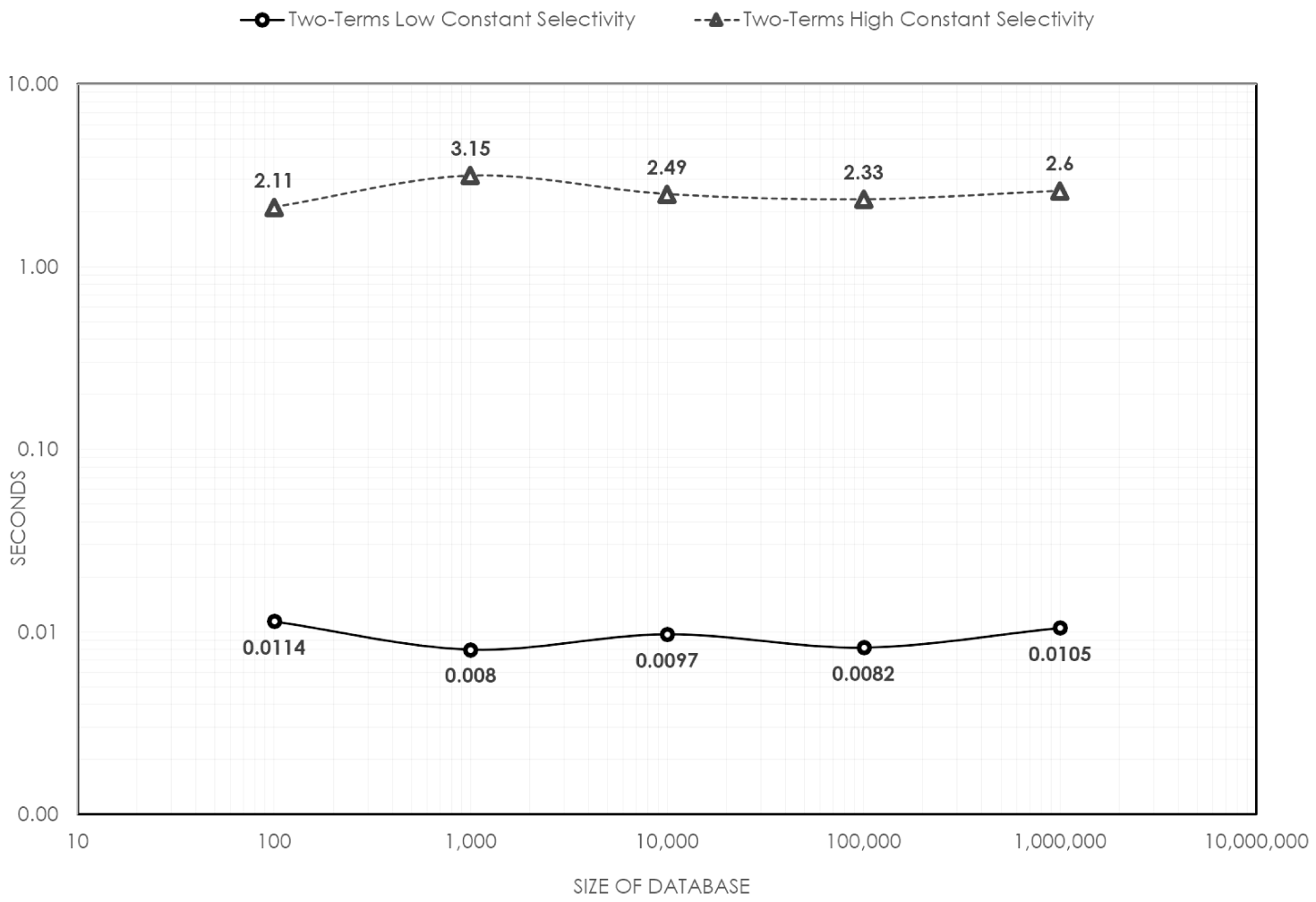

Figure 6.3: Constant Selectivity

quicker for those queries with small retrieved records. For two-term queries, the performance of the framework depends on the selecting of the appropriate *s-term*. In other words, poor selectivity ruins the framework's efficiency. These experiments also showed that the framework can compete with MySQL while *s-term* is properly selected.

Chapter 7

Conclusion and Future Works

7.1 Conclusion

In this thesis, I targeted the data privacy violation which happens for data owners (e.g. patients) in the health domain. In health systems, whole data is usually shared between trusted parties. I introduced an applicable framework for the privacy-preserving query processing and keyword search on health data which addresses the aforementioned issue. The main purpose of this framework is protecting the health information of users while dealing with cloud or online health systems and outsourcing their private health data. This system lets the data owner encrypt his/her private data, outsource that data and authorize some data users to run queries and extract their required data while the query is operated by a cloud server. The system maintains privacy of the query and results against the cloud server. To

clarify, this framework provides secure multi-client query processing and keyword search on health data. In this research I have applied the notable SSE scheme, Multi-Client Oblivious Cross-Tags (MC-OXT), which is proposed by Cash et al. [26]. According to the MC-OXT scheme's security, the limited amount of leakage of the system while running queries is determined and it is proved that the system is *IND-CKA2* secure [19, 26, 42]. *IND-CKA2* is defined in Section 3.3.3.3. The complexity analysis and experimental results prove that the proposed framework is applicable in a practical environment.

7.2 Future Works

The implemented framework can be extended in various domains. The implementing of dynamic query processing, which influences data owners to add, update, or remove data while the authorized users are able to run query could be a next step. A recent study conducted by Kamara and Moataz [43], provided a new method for arbitrary disjunctive and boolean queries in a sub-linear time which is efficient and can be applied for multi-client framework query processing on health data.

Bibliography

[1] Apache Maven — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=ApacheMaven&oldid=837443283, 2018. [Online; accessed 21-May-2018].

[2] Bouncy Castle (cryptography) — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=BouncyCastle(cryptography)&oldid=803039177, 2018. [Online; accessed 22-May-2018].

[3] Data is expected to double every two years for the next decade. https://qz.com/472292/data-is-expected-to-double-every-two-years-for-the-next-decade/, 2018. [Online; accessed 10-July-2018].

[4] Health IT Data. https://dashboard.healthit.gov/datadashboard/documentation/ehr-products-mu-attestation-data-documentation.php, 2018. [Online; accessed 22-May-2018].

[5] The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things. `https://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm`, 2018. [Online; accessed 10-November-2017].

[6] N. Attrapadung and H. Imai. Attribute-based encryption supporting direct/indirect revocation modes. In *Ima International Conference on Cryptography and Coding*, pages 278–300. Springer, 2009.

[7] N. Attrapadung and H. Imai. Conjunctive broadcast and attribute-based encryption. In *International Conference on Pairing-Based Cryptography*, pages 248–265. Springer, 2009.

[8] J. Baek, R. Safavi-Naini, and W. Susilo. Public key encryption with keyword search revisited. In *Computational Science and Its Applications–ICCSA 2008*, pages 1249–1259. Springer, 2008.

[9] B. Barak and O. Goldreich. Universal arguments and their applications. In *Computational Complexity, 2002. Proceedings. 17th IEEE Annual Conference on*, pages 194–203. IEEE, 2002.

[10] T. Becker, E. Curry, A. Jentzsch, and W. Palmetshofer. New horizons for a data-driven economy: Roadmaps and action plans for technology, businesses, policy, and society. In *New Horizons for a Data-Driven Economy*, pages 277–291. Springer, 2016.

[11] M. Bellare and P. Rogaway. Introduction to modern cryptography. *Ucsd Cse*, 207:207, 2005.

[12] S. M. Bellovin and W. R. Cheswick. Privacy-enhanced searches using encrypted bloom filters. *IACR Cryptology EPrint Archive*, 2004:22, 2004.

[13] A. Binstock. Infoworld review: Top java programming tools, 2010. [Online; accessed 6-October-2017].

[14] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[15] M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. *Algorithmica*, 12(2-3):225–244, 1994.

[16] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology-Eurocrypt 2004*, pages 506–522. Springer, 2004.

[17] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal on Computing*, 32(3):586–615, 2003.

[18] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. E. Skeith III. Public key encryption that allows PIR queries. In *Advances in Cryptology-CRYPTO 2007*, pages 50–67. Springer, 2007.

[19] C. Bösch, P. Hartel, W. Jonker, and A. Peter. A survey of provably secure searchable encryption. *ACM Computing Surveys (CSUR)*, 47(2):18, 2015.

[20] C. Bösch, Q. Tang, P. Hartel, and W. Jonker. Selective document retrieval from encrypted database. In *Information Security*, pages 224–241. Springer, 2012.

[21] E. Boyle and M. Naor. Is there an oblivious ram lower bound? In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 357–368. ACM, 2016.

[22] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Annual cryptology conference*, pages 505–524. Springer, 2011.

[23] R. Brinkman, L. Feng, J. Doumen, P. H. Hartel, and W. Jonker. Efficient tree search in encrypted data. 2004.

[24] J. W. Byun, H. S. Rhee, H.-A. Park, and D. H. Lee. Off-line keyword guessing attacks on recent keyword search schemes over encrypted data. In *Secure Data Management*, pages 75–83. Springer, 2006.

[25] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS*, volume 14, pages 23–26. Citeseer, 2014.

[26] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology–CRYPTO 2013*, pages 353–373. Springer, 2013.

[27] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security*, pages 442–455. Springer, 2005.

[28] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 577–594. Springer, 2010.

[29] K.-M. Chung and R. Pass. A simple oram. Technical report, CORNELL UNIV ITHACA NY, 2013.

[30] 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*, pages 79–88. ACM, 2006.

[31] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[32] F. J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

[33] I. Damgård, S. Meldgaard, and J. Nielsen. Perfectly secure oblivious ram without random oracles. *Theory of Cryptography*, pages 144–163, 2011.

[34] M. E. I. P. data. Ehr products used for meaningful use attestation dataset. `https://dashboard.healthit.gov/datadashboard/data/MU_REPORT.csv`.

[35] S. Everts. Information overload. `https://www.sciencehistory.org/distillations/magazine/information-overload`, 2016. [Online; accessed 20-February-2018].

[36] A. Fiat and M. Naor. Broadcast encryption. In *Annual International Cryptology Conference*, pages 480–491. Springer, 1993.

[37] E.-J. Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

[38] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[39] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia. Oblivious ram simulation with efficient worst-case access overhead. In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 95–100. ACM, 2011.

[40] T. D. Gunter and N. P. Terry. The emergence of national electronic health record architectures in the united states and australia: models, costs, and questions. *Journal of medical Internet research*, 7(1):e3, 2005.

[41] J. Hur and D. K. Noh. Attribute-based access control with efficient revocation in data outsourcing systems. *IEEE Transactions on Parallel and Distributed Systems*, 22(7):1214–1221, 2011.

[42] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Outsourced symmetric private information retrieval. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 875–888. ACM, 2013.

[43] S. Kamara and T. Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 94–124. Springer, 2017.

[44] S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *International Conference on Financial Cryptography and Data Security*, pages 258–274. Springer, 2013.

[45] E. Kushilevitz, S. Lu, and R. Ostrovsky. On the (in) security of hash-based oblivious ram and a new balancing scheme. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 143–156. Society for Industrial and Applied Mathematics, 2012.

[46] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[47] X. Liang, R. Lu, X. Lin, and X. S. Shen. Ciphertext policy attribute based encryption with efficient revocation. 2010.

[48] C. Liu, W. Hsien, C. Yang, and M. Hwang. A survey of attribute-based access control with user revocation in cloud data storage. *IJ Network Security*, 18(5):900–916, 2016.

[49] P. Mell and T. Grance. The NIST definition of cloud computing. 2011.

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

[51] T. Moataz and S. Kamara. The clusion library. `https://github.com/encryptedsystems/Clusion`, 2018. [Online; accessed 22-May-2018].

[52] M. Naveed. *Secure and practical computation on encrypted data*. PhD thesis, University of Illinois at Urbana-Champaign, 2016.

[53] P. Neto. Demystifying cloud computing. In *Proceeding of Doctoral Symposium on Informatics Engineering*, 2011.

[54] R. Ostrovsky. Efficient computation on oblivious rams. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 514–523. ACM, 1990.

[55] R. Ostrovsky, A. Sahai, and B. Waters. Attribute-based encryption with non-monotonic access structures. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 195–203. ACM, 2007.

[56] R. Ostrovsky and V. Shoup. Private information storage. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 294–303. ACM, 1997.

[57] B. Pinkas and T. Reinman. Oblivious ram revisited. In *CRYPTO*, volume 6223, pages 502–519. Springer, 2010.

[58] M. Pirretti, P. Traynor, P. McDaniel, and B. Waters. Secure attribute-based systems. *Journal of Computer Security*, 18(5):799–837, 2010.

[59] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100. ACM, 2011.

[60] M. Raykova, B. Vo, S. M. Bellovin, and T. Malkin. Secure anonymous database search. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 115–126. ACM, 2009.

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

[62] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path oram: an extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 299–310. ACM, 2013.

[63] P. C. Tang, J. S. Ash, D. W. Bates, J. M. Overhage, and D. Z. Sands. Personal health records: definitions, benefits, and strategies for overcoming barriers to adoption. *Journal of the American Medical Informatics Association*, 13(2):121–126, 2006.

[64] P. Van Liesdonk, S. Sedghi, J. Doumen, P. Hartel, and W. Jonker. Computationally efficient searchable symmetric encryption. In *Secure Data Management*, pages 87–100. Springer, 2010.

[65] X. S. Wang, K. Nayak, C. Liu, T. Chan, E. Shi, E. Stefanov, and Y. Huang. Oblivious data structures. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 215–226. ACM, 2014.

[66] Wikipedia. IntelliJ IDEA — Wikipedia, the free encyclopedia, 2018. [Online; accessed 21-May-2018].

[67] P. Williams, R. Sion, and B. Carbunar. Building castles out of mud: practical access pattern privacy and correctness on untrusted storage. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 139–148. ACM, 2008.

[68] X. Xie, H. Ma, J. Li, and X. Chen. New ciphertext-policy attribute-based access control with efficient revocation. In *Information and Communication Technology-EurAsia Conference*, pages 373–382. Springer, 2013.

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

[70] W.-C. Yau, S.-H. Heng, and B.-M. Goi. Off-line keyword guessing attacks on recent public key encryption with keyword search schemes. In *Autonomic and Trusted Computing*, pages 100–105. Springer, 2008.

[71] S. Yu, C. Wang, K. Ren, and W. Lou. Attribute based data sharing with attribute revocation. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 261–270. ACM, 2010.