PRIVACY-PRESERVING QUERY PROCESSING ON OUTSOURCED DATABASES IN CLOUD COMPUTING

Samira Barouti

A thesis
in
The Department
of
Concordia Institute for Information System Engineering (CIISE)

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Applied Science (Information System Security)at
Concordia University
Montréal, Québec, Canada

October 2013

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By:	**Samira Barouti**
Entitled:	**Privacy-Preserving Query Processing on Outsourced Databases in Cloud Computing**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Information System Security)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

______________________________ Chair
Dr. Chun Wang

______________________________ Examiner
Dr. Lingyu Wang

______________________________ Examiner
Dr. Otmane Ait Mohamed

______________________________ Supervisor
Dr. Mourad Debbabi

______________________________ Co-supervisor
Dr. Amr M. Youssef

Approved ______________________________
Chair of Department or Graduate Program Director

__________ 20 ______ ______________________________

Robin Drew, Dean

Faculty of Engineering and Computer Science

Abstract

Privacy-Preserving Query Processing on Outsourced Databases in Cloud Computing

Samira Barouti

Database-as-a-Service (DBaaS) is a category of cloud computing services that enables IT providers to deliver database functionality as a service. In this model, a third party service provider known as a cloud server hosts a database and provides the associated software and hardware supports. Database outsourcing reduces the workload of the data owner in answering queries by delegating the tasks to powerful third-party servers with large computational and network resources. Despite the economic and technical benefits, privacy is the primary challenge posed by this category of services. By using these services, the data owners will lose the control of their databases. Moreover, the privacy of clients may be compromised since a curious cloud operator can follow the queries of a client and infer what the client is after. The challenge is to fulfill the main privacy goals of both the data owner and the clients without undermining the ability of the cloud server to return the correct query results.

This thesis considers the design of protocols that protect the privacy of the clients and the data owners in the DBaaS model. Such protocols must protect the privacy of the clients so that the data owner and the cloud server cannot infer the constants contained in the query predicate as well as the query result. Moreover, the data owner privacy should be preserved by ensuring that the sensitive information in the database is not leaked to the cloud server and nothing beyond the query result is revealed to the clients. The results of the complexity and performance analysis indicates that the proposed protocols incur reasonable communication and computation overhead on the client and the data owner, considering the added advantage of being able to perform the symmetrically-private database search.

Acknowledgments

This thesis is a product of three years of work under the supervision of Professor Mourad Debbabi and Professor Amr M. Youssef. They graciously accepted me as Master student even when I knew nothing about security and privacy. They exposed me to relevant literatures in the area of security, privacy and applied cryptography. I consider it as a rare opportunity to work with these brilliant, highly motivated, kind and skillful supervisors. They provided research assistantship support and gave many opportunities for professional development. For all these and many more, I am grateful.

I have also enjoyed collaborations from some of the brightest people within of Concordia University. In particular, I am grateful to Dima Alhadidi, Feras Aljumah and Noman Mohammed for fruitful collaborations. I have enjoyed their encouragement and great feedbacks on original works and draft papers that has made this thesis much better.

I am blessed to have an outstanding husband, Farid, who has supported me through my entire life and my years of graduate studies. Thank you for your support, care, patience and faith in me.

I thank my dad and my mom, Ali Barouti and Masoumeh Azim, for always encouraging me to move forward in my academic pursuits. I am grateful for their prayers and support. Words cannot express my appreciation to my sisters, Sara and Maryam, for their supports and goodwill.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Preamble

The new paradigm of cloud computing has matured from a buzzword to a concrete concept adopted by many leading providers, such as Amazon and Google. By moving both data and computing away from personal computers into large data centers, cloud computing will allow computation to be provided as a public utility. Cloud computing has gained interests in the commercial arena due to its desirable features of scalability, elasticity, fault-tolerance, self-management and pay-per-use. The opportunity to offer a database management system (DBMS) as an outsourced service has recently received considerable attention [1,2]. Database-as-a-Service (DBaaS) [3–5] is a category of cloud computing services that enables IT providers to deliver database functionality as a service. In this model, the information that belongs to the data owner, are organized as relational databases and then outsourced to a third-party service provider, namely cloud server. The cloud server would be given the ability to store the relational databases, and the capability to answer certain types of queries. Database outsourcing reduces the workload of the data owner in answering queries by delegating the tasks to powerful third-party servers with large computational and network resources.

Outsourced databases present invaluable resources for retrieving up-to-date information and can be used for various purposes, ranging from scientific research as in the case of medical data, to demographic trend analysis and marketing purposes. Since outsourced databases are not any longer under the control of the data owner, database outsourcing poses additional privacy risks to the sensitive information of individuals. The cloud servers may not be fully trusted by the data owners or may be susceptible to attacks, launched by malicious parties (both internal and external). Thus, the privacy of individuals whose records exist in the database may be compromised by the cloud server. Protecting the database records from the cloud server is known as the *database confidentiality*.

DBaaS also poses a significant risk while the client is searching the cloud-hosted database. The privacy of the client may be compromised since a curious cloud operator can follow the client's queries and infer what the client is after. Clients' queries have been leaked intentionally in some of Google and Facebook applications [6] and unintentionally in Twitter [7] in real-life incidents. Leaked queries may contain sensitive information such as personal addresses, medical conditions, incomes, credit card numbers and social security numbers [8, 9]. Therefore, clients are increasingly aware of the need to protect privacy in their online activities. Privacy-conscious clients will likely be willing to trade off the query performance for the query privacy and possibly even pay to subscribe to such a service. The main challenge is how to keep the sensitive information contained in the queries confidential without undermining the ability of the cloud server to return correct results. In other words, we are interested in preserving the client's *access privacy* [10] – keeping both the queries and the results confidential from the cloud server and the data owner. In addition, the protocols for the database search should protect the data owner privacy ensure in the sense that the client learns only the result of her query. The restriction is crucial in situations where the database privacy is equally of concern such as financial and healthcare industries.

The following scenario further illustrates the need for having privacy-preserving query processing (PPQP) on outsourced databases in cloud computing. Consider a hospital that outsources the medical diagnoses records of its patients using the DBaaS model. Suppose that a client issues the following queries:

– How many patients are recovered from cancer?

– How many patients are diagnosed with cancer at age 35?

– How many patients are diagnosed with ovarian cancer or womb cancer?

From the constants contained in the queries, the cloud server can infer the following information about the client, respectively:

– The client suffers from cancer.

– The client is 35 years old and suffers from cancer.

– The client is a female who suffers from the cancer.

Therefore, there is a need to develop protocols for query processing on outsourced databases that protect the privacy of both the data owners and the clients such that protects (1) the database confidentiality against the untrusted cloud server, (2) the access privacy of the clients such that the cloud server and the data owners cannot infer the constants in the query predicates as well as the query result and (3) the database privacy such that the cloud server does not provide any other information beyond the query result to the clients.

Previous research in the context of secure data outsourcing has focused on these areas independently. In the case of ensuring database confidentiality of outsourced data, most of the existing works only focus on ensuring that the database records should not be disclosed to the cloud server [11–19]. In the case of protecting the access privacy, also called private information retrieval, the problem has been studied as the theoretical formulation where the client must be able to retrieve the i-th element from N data elements without disclosing to the cloud server discovering what the client is after [20, 21]. Recently, some research works have been done to enable a transition from index-based PIR to SQL-enabled PIR, however they ignore the privacy of the database owner [10, 22]. The existing solutions that target both the privacy of clients and data owners are limited to retrieving a single bit or a block of bits [20, 21] in a specific physical address. In this thesis, we propose protocols that enables to perform search on the encrypted databases while protecting the mutual privacy of the clients and the data owners, assuming that all parties are semi-honest adversaries [23] and the cloud server is untrusted.

Intuitively, the proposed protocols in this thesis organize the records of datasets as binary search trees, encrypt the records using semantically-secure encryption schemes and outsource the encrypted datasets that represent the binary search trees to the cloud server. To execute queries, the search is performed on encrypted datasets by traversing the tree. To achieve this, the protocols depend on homomorphic properties of two semantically-secure encryption schemes. Using homomorphic schemes, specific operations can be performed on the encrypted records directly without the need for decryption. More specifically, query predicates are evaluated using the Goldwasser-Micali (GM) cryptosystem [24] and the Fischlin's protocol [25] for private comparison whereas query aggregate functions are computed using the Paillier cryptosystem [26]. Using semantically-secure encryption schemes, it must be infeasible for a computationally-bounded adversary to derive significant information about a message when given only the ciphertext and the corresponding public key. The engagement of the data owner in the protocol execution should be minimal. We achieve this by using threshold cryptosystems such that the decryption process is performed by a specific number of clients, namely the threshold k. Furthermore, threshold cryptosystems provide increased security in such a way that any collaboration between fewer than k clients does not result in a complete decryption. To search datasets efficiently in logarithmic time, we use left-balancing binary search trees. Left-balancing binary search trees enable a transition from index-based SPIR to SQL-enabled SPIR in such a way that the nodes of a tree can be stored in an array and the indices of children are computed arithmetically.

This thesis consists of two parts. The first part considers a system model for DBaaS where the individuals trust an organization (namely, the data owner) to store their record while preserving their privacy. In this case, the data owner collects the information in the unencrypted form and organizes them as databases. The data owner then encrypts the records with her public key and outsources it to the cloud server and the clients sends the queries to the cloud server for execution. We investigate two types of queries over outsourced databases: SQL queries and keyword search (KS) queries. SQL queries allow the clients to retrieve data from one or more tables in a database that satisfies the comparison predicates in the WHERE clause. We assume that the shape of the query is not private but the constants contained in the query predicate are private and must be protected. This is a reasonable assumption that has been made by similar research proposals in this domain [10]. Keyword search involves two parties, a server holding a set of payloads (records) and their associated keywords, and a client who may send queries consisting of keywords and receive the payloads associated with these keywords [27]. In this case, we aim to hide the search word of the client from the database owner and the cloud server. We focus on Symmetric Private Information Retrieval (SPIR) [28] techniques to achieve the mentioned privacy goals. SPIR provides means for retrieving data from a database in a manner that preserves the mutual privacy of the client and the server; the server has assurance that the client does not learn any information beyond what she is entitled to, and the client has assurance that the server is oblivious of her choice. We can refer to SPIR as generalizations of PIR, but it requires extra computational costs.

In the second part, we consider the scenario where the individuals are not willing to trust an organizations to collect and manage their record. Instead, they prefer to manage their own record. This model is in particular suitable for the Personal Health Records (PHRs) system in which the patients are responsible for managing and sharing their medical records. In this case, the individuals encrypt their records with a public key and outsource them to the cloud server. Similar to the first part, our goal is to protect the privacy of individuals against the curious cloud server and the clients. Moreover, the query privacy of the clients must be protected such that the cloud server and the individuals do not learn anything about the constants in the SQL query. To secure the decryption key of the individuals, we employ the threshold cryptosystems such that an adversary must compromise at least a specific number of individuals to obtain the decryption keys or recover the encrypted records. We will also propose some secure distributed tools that enable the cloud server to operate on the ciphertexts and calculate the maximum, the minimum and the sum. It is noteworthy that in this model, KS queries can also be answered since a KS query can be converted to a SQL query [29].

1.2 Applications

Privacy-preserving query processing has applications in several problem domains including:

Healthcare System. Cloud computing is considered as an appropriate model for future healthcare systems [30–32] that enables the healthcare providers to shift their electronic medical record (EMR) systems to clouds. Medical database outsourcing reduces time-consuming efforts and costly operations to obtain a patient's complete medical record and uniformly integrates this heterogeneous collection of medical data to deliver it to the healthcare professionals. Public healthcare and epidemiological studies rely on summary tables of incident counts, collected directly from hospitals or indirectly via registries. Summary tables provide important statistical information for public health. In addition, the data inside these tables could be used by various parties for research purposes.

In this model, the healthcare providers outsource both the querying services and medical databases to cloud servers whereas clients such as research institutes and insurance companies issue queries to the cloud servers. Typically, the outsourced medical databases contain a number of patients with a particular set of discrete attributes. While, this data often covers a large number of patients, it can still be disclosive for rare permutations. For example, a rare cancer combined with the patient's residential address could be used to discover the patient's name.

Query privacy is also another concern for clients while querying healthcare systems since the healthcare queries reveal information about the disease or the medical treatment history. For instance, if a client issues a query on HIV, the cloud server can infer that she may suffer from HIV.

Financial Systems. Financial institutions require to cooperate with other parties such as governmental offices, credit card companies, and other financial organizations to extract useful information from financial databases. Unlike other industries in which intellectual properties are protected by patents, the financial industry that consists of business processes that has been deemed unpatentable by US Patent Office, at least until recently [33]. Hence, trade secrecy has become the preferred method by which financial institutions protect their intellectual properties and limit the disclosure of their business process, methods and data. The financial databases usually contain sensitive information about customers such as credit card number, payment methods, and history of transactions. Releasing these information into the public domain would clearly disadvantage certain companies and their costumers and benefit their competitors. On the other hand without this information, regulators and investors cannot react to financial stability threats in a timely manner. Therefore, such cooperation requires that the privacy of all involving parties (financial organizations, investors and customers) to be protected.

Census Bureau. Census is an official process through which governments systematically collect information about their population. Many governmental statistical agencies distribute data through a third party to the public to be used for example in demographic research. The published data is classified into two classes [34]:

- *Aggregate count data (contigency tables)* which contain frequency count information stored in tables with one or more attributes. For example, a contingency table may contain a population count based on zip codes, age range, and smoking status; i.e., in each zip code and each age range, how many people smoke?
- *Microdata* which are non-aggregate data and each row refers to a person in the population.

DNA Databases. Consider a pharmaceutical organization interested in purchasing information about particular genome sequences from a public DNA database.

They may need the information to complete the manufacture of a new medication that is a secret. Moreover, DNA databases owners have to keep the individual identities and genome sequences confidential.

In all of the above application scenarios, the concern is that the clients are not willing to disclose the sensitive information in the queries they send to the servers; they demand confidentiality for their queries, both from the data owners and the cloud server holding the data of interest. Additionally, the sensitive information of individuals including income, customer's credit card number, treatment history and DNA sequence are stored by the cloud server and used for responding to the queries issued by the clients. Protecting these information from adversaries is an important concern which needs to be addressed.

1.3 Objectives and Contributions

This thesis deals with the problem of designing protocols for privacy-preserving query processing (PPQP) on outsourced databases in cloud computing. The proposed protocols aim to protect:

– the access privacy of the clients against the data owner and the cloud server, i.e., protecting the sensitive constants contained in client's queries as well as the query result,

– the privacy of the data owner such that the only piece of information disclosed to the clients is the query result.

Moreover, the communication overhead must be less than the naive approach in the average case and be sublinear to the number of records in the database. The main contributions of this thesis can be summarized as follows:

- We propose a protocol that enables a client to perform oblivious walk on a left-balancing binary search tree held by a server and find all possible occurrences of her query in the tree while providing mutual privacy for both parties. This protocol protects the client data from being leaked to the server and at the same time, tree nodes are not revealed to the client.

- We extend the symmetrically-private tree search to the database outsourcing scenario in cloud computing in order to support symmetrically-private database search on the database records. We assume that the database owner is fully trusted in the sense that the individuals provide their database records as clear-text to her. The data owner represent the database records as a binary search tree and the clients execute the queries by walking on this tree in an oblivious manner. The proposed protocols enable to retrieve data from a relational database while keeping the sensitive information in SQL/keyword search queries and the query result from being leaked to the cloud server and the data owner. Moreover, the data privacy of the data owner is protected such that the sensitive information of the individuals are not revealed to the cloud server and the clients. Our complexity analysis shows that the proposed protocols impose sublinear communication and computation cost on the client.

- We then present a SQL query processing protocol in cloud computing in the case that the individuals are not willing to trust the data owner for collecting and managing their record. Instead, they prefer to manage their own record. This model is in particular suitable for the Personal Health Records (PHRs) systems. Cloud-based PHR is a recent trend for patients to store their electronic health records on the cloud and manage them. We propose a solution that allows the patients to protect their sensitive records from the cloud server using public-key encryption schemes. In addition, our solution allows health organizations to produce statistical information about encrypted personal health records stored on the cloud while preventing the patients to infer about what health organizations are concerned about; not to create panic about epidemics in the community. To do so, we develop some techniques that enable the cloud server to securely execute the aggregate SQL queries of the health organization over encrypted databases while satisfying the privacy goals of all parties.

1.4 Thesis Organization

Chapter 2 overviews the security model adopted in this thesis as well as the cryptographic primitives, utilized in the proposed approaches. Chapter 3 covers some previous research works related to the research described in this thesis and discuss our contributions. In Chapter 4, we proposed two protocols for Keyword Search (KS) and SQL queries assuming that the data owner is fully trusted. The proposed protocol in Chapter 5 describes our approach if the data owner is not trusted. Finally, concluding remarks as well as a discussion of the future works are represented in Chapter 6.

Chapter 2

Preliminaries

In this chapter, we present an overview of the security model along with some cryptographic primitives that are required for the proposed protocols.

2.1 Secure Multiparty Computation

Consider the scenario where n connected parties hold input x_i, $1 \leq i \leq n$. The parties wish to compute a function f on inputs $x_1, x_2, \ldots, x_n$. The aim of a Secure Multiparty Computation (SMC) protocol [35] is to enable parties to carry out such distributed computation task in a secure manner. Secure multi party computation is concerned with the possibility of deliberately malicious behavior by some adversarial entities. The aim of these adversaries may be to learn private information or cause the result of the computation to be incorrect. Two important requirements on any secure computation protocol are privacy and correctness [23]. The privacy requirement states that parties should learn their output and nothing additional. The correctness requirement states that each party should receive its correct output. Therefore, the adversary must not be able to cause the result of the computation to deviate from the function that the parties intend to compute.

In general, there are two main types of adversaries:

(a) *Semi-honest adversaries*: Adversaries correctly follow the protocol specification. However, the adversary obtains the internal state of all the corrupted parties and attempts to use this to learn private information. This is a rather weak adversarial model. However, there are some settings where it can realistically model the threats to the system. A semi-honest adversary is also called *honest-but-curious* or *passive*.

(b) *Malicious adversaries*: Adversaries can deviate from the protocol specification, according to the adversary's instructions. In general, providing security in the presence of malicious adversaries is preferred, as it ensures that no adversarial attack can succeed. Malicious adversaries are also called *active* adversaries.

It is of course easier to design a solution that is secure against semi-honest adversaries, than it is to design a solution for malicious adversaries. A common approach is to first design a secure protocol for the semi-honest case, and then transform it into a protocol that is secure against malicious adversaries [36]. This transformation can be done by requiring each party to use zero-knowledge proofs [37] to prove that each step it is taking, follows the specification of the protocol. More efficient transformations are often required, since this generic approach might be rather inefficient and add considerable overhead to each step of the protocol.

In this thesis, we assume that the adversary model is semi-honest. Therefore, the clients, the database owner and the cloud server follow the protocol steps, but they try to infer additional information beyond their prescribed output. In order to formally claim and prove that a protocol is secure, a precise definition of security for multiparty computation is required. According to Goldreich [38], a protocol is private if no party learns anything more than its prescribed output. In particular, the only information that should be learned about other parties' inputs is what can be derived from the output itself.

An important theorem in SMC is the composition theorem [38]. The basic idea behind the composition theorem is that it is possible to design a protocol that uses an ideal functionality as a subprotocol, then analyze the security of the protocol when a trusted party computes this functionality. If, we prove the security of the larger protocol that uses the functionality as a subprotocol in a model where the parties have access to a trusted party computing the functionality. The composition theorem then states that when the *ideal calls* to the trusted party for the functionality are replaced by real executions of a secure subprotocol computing this functionality, the protocol remains secure.

Many of the protocols based on SMC, as it is the case with the proposed protocols in this thesis, involve the composition of privacy-preserving sub-protocols in which all intermediate outputs from one sub-protocol are inputs to the next sub-protocol. These intermediate outputs are either simulated by the final output and the local input for each party or as random shares. Using the Composition Theorem [38], it can be proven that if each sub-protocol is privacy-preserving, then the resulting composition is also privacy-preserving.

2.2 Cryptographic Primitives

2.2.1 Private Information Retrieval

Given a server which holds x; an n-bit string. The client's objective is to retrieve the i-th bit without revealing to the server which item has been queried. We stress that in this model the database is public. Private Information Retrieval (PIR) protocols allow the client to retrieve data from a public database with communication strictly smaller than n. PIR is a weaker version of 1-out-of-n oblivious transfer in which it is also required that the client should not get information about other database items. Chor *et al.* [20] have introduced PIR in the setting of multiple servers where identical copies of the string x are stored by $k \geq 2$ servers. Chor *et al.* have also shown that single-database PIR does not exist in the information-theoretic sense. Nevertheless, Kushilevitz and Ostrovsky [21] have proposed a method for constructing single-database PIR assuming a certain secure public-key encryption. Most of PIR protocols are limited to retrieving a single bit, a block of bits [20, 21], or a textual keyword [39].

Symmetric private information retrieval [40] is an extension to PIR which addresses database privacy by preventing the client from learning information about any record except the retrieved records. Oblivious Transfer (OT) schemes generally have no requirement for sublinear communication complexity, which renders them useless. It is noteworthy that all existing communication-efficient 1-out-of-n OT schemes are essentially SPIR.

2.2.2 Homomorphic Encryption

Homomorphic encryption is a form of encryption where a specific algebraic operation performed on the plaintext is equivalent to another (possibly different) algebraic operation performed on the ciphertext. The idea of performing simple computations on encrypted messages was first suggested by Rivest *et al.* [41], who referred to such computations as *privacy homomorphisms*. The original motivation for privacy homomorphisms was to allow simple computations on encrypted data without decrypting them. The ability to perform simple deterministic computations on encrypted data makes homomorphic cryptosystems ideal for creating privacy-preserving protocols.

Recently, these cryptosystems received attention in cloud computing community. To understand the importance of homomorphic encryption, consider the following scenario: a user stores her data on some third party's servers. If she does not trust the third party, she may wish to store her data encrypted under a public key encryption scheme and keep her private key local. However, in order to use this data, in a traditional public key encryption scheme, the user would need to download the encrypted data to her local machine, decrypt it, perform her desired computations, and if she wishes to store the result, the user has to encrypt it and send it back to the server. With homomorphic encryption, a user could instead operate directly on the encrypted data.

In this thesis, we utilize Paillier cryptosystem [26] which is an additive homomorphic public key encryption. Using Paillier's scheme, given two ciphertexts $E(x)$ and $E(y)$ of two plaintexts x and y respectively, an encryption of their sum $E(x+y)$ can be efficiently computed by multiplying the ciphertexts modulo a public key n^2, i.e., $E(x+y) = E(x).E(y) \bmod n^2$. The core steps of Paillier encryption/decryption are represented in Fig. 1. If using p and q of equivalent length, a simpler variant of the above key generation steps would be to set $g = n+1$, $\lambda = \phi(n)$, and $\mu = \phi(n)^{-1} \bmod n$ where $\phi(n) = (p-1)(q-1)$.

In addition to the Paillier cryptosystem, we utilize the Goldwasser-Micali (GM) cryptosystem [42] as well. GM cryptosystem is a semantically-secure scheme based on the quadratic residuosity assumption. GM-scheme allows computing the exclusive-or of two encrypted bits, i.e., $E(b_0)\ .\ E(b_1) = E(b_0 \oplus b_1)$. The GM scheme is as follows: the public key is a composite $N = pq$, where p and q are primes and $p = q = 3 \bmod 4$. The private key consists of the factorization of N. To encrypt bit $m_i \in \{0, 1\}$, choose a random element $y \in Z_N$ and send $C = x^{m_i}\ .\ y^2 \bmod N$. Decryption of ciphertext C proceeds by determining whether C is a quadratic residue or not. First,

Figure 1: *Paillier Cryptosystem [26]*

Key Generation.

1. Alice generates two distinct large prime numbers, namely p and q such that $GCD\Big(pq, (p-1)(q-1)\Big) = 1$ and computes the modulus $n = p.q$ and $\lambda = \text{lcm}(p-1, q-1)$.
2. Alice selects an integer g such that $g \in Z^*_{n^2}$.
3. She computes $\mu = (L(g^\lambda \bmod n^2)) \bmod n$, where $L(u) = \frac{u-1}{n}$

   The public key consists of (n, g). The secret key is (λ, μ).

Message Encryption. Suppose that Bob wishes to send message m to Alice:

1. Bob selects a random number r and computes $c = g^m.\ r^n \bmod n^2$.
2. Bob sends the ciphertext c.

Message Decryption. Alice computes the message $m = L(c^\lambda \bmod n^2)\ .\ \mu \bmod n$.

calculate the Jacobi symbol $J = \left(\frac{C}{N}\right)$. If $J \neq 1$, then the ciphertext is ill-formed (i.e., the encryption algorithm was not run honestly, or else the message was corrupted in the transmission); therefore, simply output $\perp$. If $J = 1$, we may decide whether C is a quadratic residue by computing $b' = C^{(N-p-q+1)}/4 \bmod N$. Note that C is a quadratic residue iff $b' = 1$. At this point, the original plaintext can be recovered by computing $b = (1 - b')/2$. GM cryptosystem is semantically secure under the quadratic residuosity assumption.

Sander, Young, and Yung [43] described an AND-homomorphic extension of the GM-encryption scheme henceforth referred to as AND-GM encryption. In this scheme, each bit of the message is encrypted using a vector of basic GM encryptions with size λ. λ is chosen to be a sufficiently large number such that $2^{-\lambda}$ is small enough. In AND-GM cryptosystem, we encrypt $b = 1$ as a sequence of λ random

Figure 2: *Goldwasser-Micali (GM) Cryptosystem [42]*

Key Generation. The modulus used in GM encryption is generated randomly in the same manner as in the RSA cryptosystem.

1. Alice generates two distinct large prime numbers, namely p and q and computes the modulus $N = pq$.
2. Alice finds a random non-residue number x such that the Legendre symbols satisfy $\left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) = -1$ and hence the Jacobi symbol $\left(\frac{x}{n}\right)$ is $+1$.

   The public key consists of (x, N). The secret key is the factorization (p, q).

Message Encryption. Suppose that Bob wishes to send message m to Alice:

1. Bob encodes m as a sequence of bits $(m_1, ..., m_n)$.
2. For each bit m_i, Bob generates a random value y such that $GCD(y, N) = 1$ and outputs $c_i = y^2 x^{m_i} \bmod N$
3. Bob sends the ciphertext $(c_1, ..., c_n)$.

Message Decryption. Alice receives ciphertext $(c_1, ..., c_n)$ from Bob.

1. For each i, using the prime factorization (p, q), Alice determines whether the value c_i is a quadratic residue: if so, $m_i = 0$, otherwise $m_i = 1$.
2. Alice outputs the message $m = (m_1, ..., m_n)$.

Figure 3: *GM to AND-GM ciphertext conversion [43]*

```
convert(c)
//converts GM to GM-AND ciphertext
given $c = E_{\mathrm{GM}}(m, y) = x^m\ y^2$
for $i = 0$ to $\lambda - 1$
    choose random $r_i$ such that $GCD(r_i, N) = 1$
    choose random $s_i \in \{0, 1\}$
    if $s_i = 0$ then    $C_i \leftarrow c\ .\ x\ .\ E_{\mathrm{GM}}(0, r_i) \bmod N$
    else                 $C_i \leftarrow E_{\mathrm{GM}}(0, r_i) \bmod N$
output $C_0, \ldots, C_{\lambda-1}$
```

quadratic residues (i.e., as λ GM-encryptions of 0), and $b = 0$ as a sequence of λ random elements (i.e., as λ GM-encryptions $Enc(a_i)$ for random bits $a_1, \ldots, a_\lambda$). The decryption scheme takes a sequence of λ elements of GM ciphertext and returns 1 if all elements are quadratic residues, and 0 otherwise (i.e., if there is a quadratic non-residue among the elements). There is a small probability of $2^{-\lambda}$ that a 0-bit is encrypted as a sequence of λ quadratic residues. In this case, the decryption does not give the desired result. If λ is chosen sufficiently large (in practice 86), it never happens. Figure 3 represents the algorithm for converting a basic GM ciphertext into an AND-GM ciphertext without knowing either the secret key or the plaintext. It can be easily shown that given two ciphertexts b and b' encrypted using AND-GM scheme, the component-wise product of b and b' is an AND-GM encryption of $b \wedge b'$.

2.2.3 Threshold Cryptosystems

The traditional cryptosystems consider the case where there is one sender and one receiver in which the sender encrypts a message with the public key of the receiver and sends the resulted ciphertext to the receiver. The receiver then decrypts the ciphertext to obtain the plaintext. However, in some cases, the receiver is not an individual but instead an organization, e.g. a company or a governmental agency [44]. Therefore, there is a need to a distributed scheme that has one public key but many secret keys so that the decryption power is distributed between multiple receivers. From the security viewpoint, the threshold cryptography aims to protect the secret key by sharing it amongst n parties in such a way that a predetermined number of parties, namely the threshold k, should collaborate to fully decrypt a message. Any collaboration between fewer than k parties does not result in a complete decryption. The benefit of a threshold scheme is increased security, because an adversary who corrupts at most $k-1$ parties gains no advantage in determining the secret key of the system or in breaking the underlying cryptographic protocol. Some threshold cryptosystem employs a trusted dealer to set up the public key and the secret key and distribute the shares of the secret key between the parties (e.g. [45, 46]). The dealer must be minimally trusted not to reveal the secret key and therefore represents a single point of attack. Thus, it is often desirable to distribute the key-generation phase among the parties which removes the trusted dealer and ensures no entity ever knows the secret information [24, 47–50].

The idea of utilizing threshold cryptosystems in the cloud computing was first noticed by Gentry [51]. The problem is to design a computation- and communication-efficient multi-party protocol that allows n parties contribute inputs $x_1, \ldots, x_n$ respectively and jointly evaluate $f(x_1, \ldots, x_n)$ securely. Gentry suggested using a Fully-Homomorphic Encryption (FHE) scheme and a powerful cloud that carry out the computational-intensive operations. The parties first run a multiparty protocol to generate a joint public key for (an arbitrary) FHE scheme, together with a secret sharing of the corresponding secret key (the i-th party gets the i-th share of the common secret key). Once this is done, the parties encrypt their inputs using the common public key and have the cloud compute an encryption of the result. They then run yet another MPC protocol to perform threshold decryption and recover the result. Later, Asharov *et. al* [52] and López-Alt *et. al* [53] used a similar idea to develop efficient FHE schemes for multi-party computations.

In this thesis, we employ two threshold cryptosystems without depending on the trusted dealer: threshold Paillier [47] and threshold GM [24] cryptosystems. Normally, the threshold cryptosystems have two core algorithms: a distributed key generation, and a distributed decryption algorithm.

For the threshold GM cryptosystem [24], the distributed key generation algorithm can be performed by executing the distributed RSA key-generation protocols of [49, 50]. Following execution of the key-generation protocol, each party $i \in \{1, 2, \ldots, n\}$ obtains the GM modulus $N = pq$ and the additive shares (p_i, q_i). The public key will be calculated as $N = (\Sigma_{i=1}^{n} p_i) \,.\, (\Sigma_{i=1}^{n} q_i)$ while none of the parties know the factorization of N (i.e., p and q). Decryption of a ciphertext C proceeds as follows: the party i outputs $b_i = C^{(-p_i - q_i)}/4 \bmod N$. Parties publicly compute $b_0 = C^{(N-5)/4} \bmod N$. Deciding whether C is a quadratic residue or not may be done by computing $b' = \prod_{i=0}^{k} b_i \bmod N$. The decrypted bit is $b = (1 - b')/2$.

For the threshold Paillier cryptosystem [47], the distributed RSA key generation protocol is executed so that each party obtains $N = pq$ and the additive share (p_i, q_i). All parties then generate the share of $\phi(N) = (p-1)(q-1)$ by setting

$$x_i = \begin{cases} N - (p1 + q1) + 1 & i = 1 \\ -(p_i + q_i) & i > 1 \end{cases}$$

such that $\sum_{i=1}^{n} x_i = \phi(N)$. The parties then commit to the value x_i by publishing $h_i = g^{x_i} \bmod N^2$ where $g = N + 1$. They then set publicly

$$h = \prod_{i=1}^{n} h_i - 1 \bmod N^2 = g^{\phi(n)} - 1 \bmod N^2$$

To decrypt the ciphertext c, each party P_i computes $m_i = c^{x_i} \bmod N^2$. The plaintext can be recovered by computing

$$m = \frac{1}{h}\left(\prod_{i=1}^{n} m_i \ - \ 1 \bmod N^2\right) \bmod N$$

2.2.4 Private Comparison

Yao's classical millionaires' problem [35] (or private integer comparison) involves two parties who want to compare their wealth: they wish to know who is richer but do not want to disclose any other information about their wealth to each other. More formally, given two input values x and y, which are held as private inputs by two parties, respectively. The problem is to securely evaluate the condition $x > y$ without exposing inputs.

Yao [35] first proposed such a protocol for the private comparison problem, which is an instantiation of secure multiparty computation. Nevertheless, the cost of the protocol is exponential in both time and space. Later, Yao [54] and Goldreich [55] used the technique of scrambled circuits to solve the general multiparty computation problem. By applying this technique to the greater than (GT) problem, the cost of the resulting protocol in computation and communication is linear. On the other hand, protocols for solving the GT problem directly are more efficient (e.g. [25, 56–59]). These protocols usually require a constant number of rounds.

In this thesis, we utilize Fischlin protocol [25] for private comparison. In Fischlin protocol one of the parties acts as a server. In this setting, say, Alice knows the private keys to open encryptions and Bob works over his input bits and Alice's encrypted input bits to produce some information that allows Alice to know the output of the comparison being evaluated. Fischlin protocol uses the homomorphic properties of GM-encryptions scheme to compute an expression logically equivalent to

$$\begin{aligned} x > y &\iff \bigvee_{i=1}^{n} \Big(x_i \wedge \neg y_i \wedge \bigwedge_{j=i+1}^{n} (x_j = y_j) \Big) \\ &\iff \bigoplus_{i=1}^{n} \Big(x_i \wedge \neg y_i \wedge \bigwedge_{j=i+1}^{n} \neg (x_i \oplus y_i) \Big) \end{aligned}$$

and

$$\begin{aligned} x = y &\iff (\bigwedge_{i=1}^{n} (x_i = y_i)) \\ &\iff (\bigwedge_{i=1}^{n} \neg (x_i \oplus y_i)) \end{aligned}$$

where $|x| = |y| = n$. The protocol steps are depicted in Fig. 4.

2.3 Left-Balancing Binary Search Trees

A binary search tree is a binary tree where each node has two pointers to other nodes (namely, *left* child and *right* child) and a *key* and satisfies the restriction that the key in any node is larger than the keys in all nodes in its left subtree and smaller than the keys in all nodes in its right subtree. A left-balancing binary search tree (i.e. Complete binary trees) [60, 61] is a binary search tree with two key properties: (i) all levels except the bottommost are filled, and (ii) in the bottommost level, the nodes are inserted from left to right. These trees provide an easy way to store a tree in an array. The idea is that instead of explicitly storing pointers which point from a given node to its child nodes, we use arithmetic to compute the index of child nodes. In this scheme, the root node is stored in the position 1 in the array, and when a node is stored in the position i in the array, the child nodes are stored in the positions $2i$ and $2i + 1$. This method has the advantage that all nodes on a given level lie in a consecutive block. In addition, left and right children are adjacent which should make the scheme more cache friendly. The benefit of left-balancing binary search tree

Figure 4: *Fischlin Private Comparison Protocol [25]*

```
Protocol GTE-F($x$,$y$)                                    $\lambda$: error parameter

Alice:

  – generates public key $pk = (N, z)$ for security parameter $k$.
  – encrypts $y$ bit-wise: $Y_i \leftarrow E_{pk}(y_i)$ for $i = 1, ..., n$.
  – sends $N$, $z$, $Y_1, \ldots, Y_n$ to Bob.

Bob:

  – encrypts input $x$ bit-wise: $X_i \leftarrow E_{PK}(x_i)$ for $i = 1, ..., n$
  – computes clauses $[x_i = y_i] = \neg(x_i \oplus y_i)$:
    for $i = 1$ to $n$ compute $c_i = X_i \,.\, Y_i \,.\, z \bmod N$
  – embeds $c_i$ into extended encryptions $c_i^{AND}$ of $\lambda$ elements:
    for all $i = 1, \ldots, n$ sets $c_i^{AND} := (c_{i,1}^{AND}, \ldots, c_{i,\lambda}^{AND})$
  – embeds encryptions $X_i$ and $\bar{Y}_i$ of $x_i$ and $\neg y_i$ into encryptions $X_i^{AND}$ and $\bar{Y}_i^{AND}$
    of $\lambda$ elements:
    for all $i = 1, \ldots, n$ sets $X_i^{AND} := (X_{i,1}^{AND}, \ldots, X_{i,\lambda}^{AND})$
    for all $i = 1, \ldots, n$ sets $\bar{Y}_i^{AND} := (\bar{Y}_{i,1}^{AND}, \ldots, \bar{Y}_{i,\lambda}^{AND})$
  – computes terms $[x_i \wedge \neg y_i \bigwedge_{j=i+1}^{n} x_j = y_j]$:
    for $i = 1$ to $n$ computes $\Delta_i \leftarrow X_i^{AND} . \bar{Y}_i^{AND} . \prod_{j=i+1}^{n} c_j^{AND} \bmod N$
  – computes terms $[\bigwedge_{i=1}^{n} x_i = y_i]$:
    $c := \prod_{i=1}^{n} c_i^{AND} \bmod N$
  – sends $\Delta_1 \ldots \Delta_n$ in randomly permuted order to Alice.
  – randomly permutes the $\lambda$ ciphertexts of $c$ and sends to Alice.

Alice:

  – receives $n$ sequences $\Delta_1 \ldots \Delta_n$ of $\lambda$ GM-ciphertext and $c$ from Bob.
  – if there exists a sequence of $\lambda$ quadratic residues in $\Delta$ then output $x > y$
    else if $c$ is a sequence of $\lambda$ quadratic residues then output $x = y$
    else output $x < y$
```

is that the average cost of looking up an item is $O(\log N)$ where N in the number of nodes in the tree.

In [61] constructing left-balancing binary search search trees is explained for single dimensional as well as multidimensional data. To construct such a tree, we calculate the largest number M in the form of $M = 2^n$ such that $M \leq N$ where N is the number of nodes we wish to insert. The tree will hold $M-1$ elements on all levels excluding the bottommost. The bottommost level holds M elements divided between $M/2$ in the left subtree and $M/2$ in the right subtree. We compute the remainder $R = N - (M-1)$ and the data set is partitioned into two subsets based on the value of R. If $R \leq M/2$

$$LT = (M-2)/2 + R \tag{1}$$

$$RT = (M-2)/2 \tag{2}$$

Otherwise, if $R > M/2$

$$LT = (M-2)/2 + M/2 \tag{3}$$

$$RT = (M-2)/2 + R - M/2 \tag{4}$$

where LT and RT denote the number of nodes on the left and the right subtrees, respectively. This procedure is recursively executed on the resulted subsets to output the tree.

Search on 1-Dimensional Binary Search Trees. Binary search trees are able to support point search as well as range search. The point search aims to search for all nodes with a given key k and return pointers to them if they exist and NULL, otherwise. The algorithm of point search is represented in Figure 5 that takes the current root r and the key k as arguments. It uses a subroutine `report`(v), which traverses the subtree rooted at node v and reports all the stored nodes. Note that $r.key$ denotes the *key* attribute of node r. The time complexity of `Point-Search` is $O(\log N)$ on average and $O(N)$ in the worst case [60], where N is the number of nodes.

Figure 5: *Point Search on Binary Search Trees [62]*

```
Algorithm Point-Search $(r, k)$

if $r == NULL$
    then return NULL
If $k == r.key$
    then return Point-Search $(r.left, k)$
    report $(r)$
    return Point-Search $(r.right, k)$
If $k < r.key$
    then return Point-Search $(r.left, k)$
else return Point-Search $(r.right, k)$
```

Similarly, the goal of a range search query is to find pointers to all nodes with the key in the range $[k_1, k_2]$. The range search algorithm is shown in Figure 6 that takes as arguments the current root r, the upper bound k_1 and higher bound k_2. The time complexity of `Range-Search` is $O(m + \log n)$ where m is the number of nodes in range $[k_1, k_2]$ and $0 \leq m \leq n$ [62].

2.4 Multidimensional Indexing

A database index is a supplementary data structure used to access data from the database efficiently. Data is indexed either directly by the values of one or more

Figure 6: *Range Search on Binary Search Trees [62]*

```
Algorithm Range-Search ($r, k_1, k_2$)

if $r == NULL$
    then return NULL
If $r.key < k_1$
    then return Range-Search ($r.right, k_1, k_2$)
If $r.key > k_2$
    then return Range-Search ($r.left, k_1, k_2$)
else
    Range-Search($r.left, k_1, k_2$)
    report r
    Range-Search($r.right, k_1, k_2$)
```

attributes or by hashes (generally not cryptographic hashes) of those values. The attributes used to define an index constitute the key. Indices are typically organized into tree structures, such as k-Dimensional trees (kD-trees) where the database records are stored in internal nodes as well as leaf nodes. kD-trees are multidimensional binary search trees where the key used differs between levels. At each root, the keys on the left subtrees are less than or equal to the key of the root. Similarly, nodes on the right subtree have the key that is greater than or equal to the root. kD-trees are used extensively as indices for multidimensional data (geographical, multimedia and database). In databases, three basic types of multi-dimensional queries are considered [63]: *exact matching queries* (a.k.a. point search), *partial matching queries* and *range queries*. An exact matching query searches a database to find objects with specific attribute values. A partial matching query specifies only values for a subset of the the attributes and searches for all records with the specified attribute values. Let N and d be the number of records and the number of attributes in the database, respectively. The time complexity of the exact matching queries and the partial matching queries is respectively $O(d \log N)$ and $O(N^{1-s/d} + k)$ where s is the number of attributes, specified in the query ($s < d$) and k is the number of reported records [62].

Range queries are the most general type of queries that specify a region with which the records to be retrieved must intersect. Figure 7 shows the range search algorithm on kD-trees [62]. It takes as input the root of kD-tree v and the query range R. It uses a subroutine ReportSubtree(v), which traverses the subtree rooted at node v and reports all the stored nodes. A query in a d dimensional tree storing N records, can be performed in $O(N^{1-1/d} + k)$ time, where k is the number of reported records [62].

Figure 7: *Range Search on kD-Trees [62]*

```
Algorithm kD-Tree Search($v, R$)

if $v$ is a leaf
    then report $v$ if it lies in $R$
else if $v.left$ is fully contained in $R$
    then ReportSubtree($v.left, R$)
else if $v.left$ intersects $R$
    then kD-Tree Search($v.left, R$)
else if $v.right$ is fully contained in $R$
    then ReportSubtree($v.right$)
else if $v.right$ intersects $R$
    then kD-Tree Search($v.right, R$)
```

Chapter 3

Related Work

This chapter covers previous works related to this thesis. We begin by briefly reviewing the notion of Private Information Retrieval (PIR). Afterwards, we give an overview of prior works aimed to make data access more expressive while protecting privacy. We briefly highlight Basic Homomorphic Encryption (BHE) and Hybrid Homomorphic Encryption (HHE), Transparent PIR (TransPIR) and SQL-enabled PIR (SQL-PIR) and we will use these works as the basis of comparison with our proposed protocols.

3.1 Private Information Retrieval

The notion of private information retrieval (PIR) was introduced by Chor, Goldreich, Kushilevitz and Sudan [20] and has already received a lot of attention. The study of PIR is motivated by the growing concerns about the user's privacy when querying a large commercial database.

The PIR problem consists of devising a communication protocol involving just two parties, the server and the user, each having a secret input. The server's secret input is called the data string, an n-bit string $B = b_1 b_2 \ldots b_n$. The user's secret input is an integer i, $1 \leq i \leq n$. The protocol should enable the user to learn b_i in a communication-efficient way and at the same time hide i from the database. A natural extension of PIR is private block retrieval in which the n-bit data string is considered to be composed of $\frac{n}{b}$ blocks, each of size b bits. The user's objective is to retrieve the i-th block from the database. A trivial solution to the PIR problem is simply to ask the server for the whole database and look up the desired bit or block on the user side. However this approach incurs excessive communication cost to the server which is linear with respect to the number of bits to represent the database records.

There are two main types of PIR: *information-theoretic* and *computational*. In information-theoretic PIR [20], the server is unable to determine any information about the client's query even with unbounded computing power. In computational PIR (cPIR) [21], the privacy of the query only needs to be guaranteed against servers restricted to polynomial-time computations. A number of applications have been proposed for PIR, including patent and pharmaceutical databases [64], online census information [65], and real-time stock quotes [65].

A common assumption for PIR schemes is that the user knows the index or address of the item to be retrieved. However, Chor et al. [39] proposed a way to access data with PIR using keyword searches over three data structures: binary search tree, Trie [66] and perfect hashing. They showed how the client can privately traverse certain server-held data structures. Their solutions for binary search trees, upon which we build our framework in Chapter 4, protect only the user privacy. They also provide a solution for tries that is also server-private. The trie is a data structure that stores the data of each node in a path from the root to the node, rather than the node itself. A trie has a number of advantages over binary search trees such as faster lookup and easy updating [67]. However, the trie data structure is not well suited in the database context because of the difficulty in handling range queries. Song *et. al.* [68] also proposed a solution for the secure outsourcing of the single-dimensional data (such as documents and emails) that protects the access privacy of the client and the database confidentiality; however their approach ignores the data owner privacy.

The above mentioned techniques provide basic data access models, limited to either retrieving a single bit or a block of bits using indices or textual keywords. This property limits the deployment of PIR in complex systems storing structured data sources such as relational databases. Therefore there is a need for a more expressive data access model. Reardon *et al.* [69], Olumofin and Goldberg [10] and Wong *et al.* [22] have considered this issue by proposing techniques for privacy-preserving SQL-query processing but their approaches leak extra information about the database beyond the query result.

3.1.1 TransPIR

Transparent PIR (TransPIR) [69] is among the first attempt to extend PIR to SQL. TransPIR evaluates private relational queries with the goal of minimizing the communication complexity. It uses PIR for data block retrieval from the database server, whose function has been reduced to private block retrieval from the server.

In this system, relational algebra queries are translated into query plan by a query processor. The query plan comprises of function calls executable by a virtual database which is a component that maps relation information into PIR blocks. The virtual database responds to this call by fetching the appropriate block from the server based on the layout of the database. To reduce communication complexity, the indices are constructed over the tables of database. The client then performs traditional database functions (such as parsing and optimization) locally on her machine.

The benefit of TransPIR is that the database server does not learn any information even about the textual content of the user's query. The drawbacks are poor query performance because the database is unable to perform any optimization, and the lack of interoperability with any existing relational database system.

3.1.2 BHE/HHE

Wang et al. [22] propose an algorithm which protects the privacy of client's query considering range and join queries. The query privacy is achieved through query obfuscation. Homomorphic encryption and bucketization are the primitives utilized in these approaches. To help a client to construct a private query, the server bucketizes the public database and sends the bucket summary to the client. The client subsequently determines which bucket(s) contains the data of interest. To retrieve the matching database records, Basic Homomorphic Encryption (BHE) and Hybrid Homomorphic Encryption (HHE) algorithms have been proposed. BHE reveals nothing about the private data and queries and requires a single round of communication between the client and the server. However, it requires sending the entire database records to the client for filtration that is linear in the number of records. To further reduce the costs of BHE, HHE has been introduced. HHE trades off privacy by efficiency; the client selects a subset of buckets that include the buckets of interest and computation is performed only on these subsets.

Note that BHE and HHE require frequent public-key homomorphic encryption/decryption which incurs computation cost to the both sides. This is specially a major problem on the client side because it is typically a machine with limited computation and communication capabilities. BHE provides perfect privacy, however the communication overhead is roughly linear in the number of records. On the other hand, HHE achieves better efficiency (in terms of both communication and computation) by revealing a superset of buckets of interest. But it is also vulnerable to certain type of attacks against curious database server; since the client discloses the matching buckets, the server is able to tighten the extent of buckets. In this case, the server can see the matching buckets and guess the client's query even though the buckets are hidden among others.

3.1.3 SQL-PIR

SQL-PIR [10] integrates PIR with SQL to protect the query privacy and enable data retrieval from databases. The protection mechanism is based on the observation that the shape of an SQL query is not private, but the constants supplied by the user is private and must be protected. SQL-PIR considers some attributes to be sensitive and it aims to hide the constants associated with these attributes.

The user sends a desensitized version of the original SQL query by removing private constants. The database executes this public SQL query and generates appropriate B^+ indices [70] for each sensitive attribute to support further rounds of interaction with the client. The B+ tree is a generalization of a binary search tree in that a node can have more than two children.

The user subsequently performs a number of keyword-based PIR operations [39] using the value of the sensitive attributes against the indices to obtain the result for the query. Capabilities for dealing with complex queries is built on the user side. In the case of existence of two or more sensitive attributes in the query predicate, the user requests separate indices. The user will subsequently perform PIR keyword search on each of indices and combine the partial result by set operations (union, intersection) to obtain the final query result.

In this thesis, a similar notion of query privacy protection is utilized: the goal is to protect private constants incorporated in query predicate. SQL-PIR considers the query privacy of client however it fails to protect data owner privacy. In particular SQL-PIR reveals to the client, database records which are not relevant to the query. Moreover, SQL-PIR reveals the entire textual shape of the query to the server in particular the operands that combine the logical condition on attributes.

3.2 Privacy-Preserving Set Operations

Freedman *et al.* [71] provide a solution to solve the problem of data set intersection for both the semi-honest and the malicious environments. The proposed protocol enables two parties each holding a set of input to jointly compute the intersection of their input sets without leaking any additional information. This protocol is based on the representation of data sets as the roots of a polynomial. In this approach, the client constructs a polynomial whose roots are her inputs and sends the homomorphic encryptions of the polynomial's coefficients to the server. The server uses the homomorphic properties of the encryption system to evaluate the polynomial at each of the database tuple and outputs the evaluation results to the client. The client decrypts the results and checks for possible matches.

Kissner and Song [72] extended this framework to design a protocol for privacy-preserving multiset operations. A privacy-preserving multiset operation considers the problem of computing the union, intersection, and element reduction on n private multisets from n users, such that each user only learns the resulted set. Similar to [71], the multiset of each user is represented by a polynomial and operations on polynomials lead to privacy-preserving multisets operations.

Private query processing on outsourced databases can be achieved by obtaining the intersection of two private datasets where query constants are the client's input and the database records are the server's input. Our proposed protocol is significantly different from private intersection schemes because SQL queries are richer and more complex compared to a simple set intersection. The private matching protocol is able to answer exact matching queries on a single attribute whereas the proposed protocol in this thesis answers exact-matching and interval-matching queries on multiple attributes.

3.3 Oblivious Keyword Search

Keyword search (KS) is a fundamental database operation. It assumes that each database tuple is tagged with an appropriate keyword. It involves two parties: a server, holding a database comprised of a set of records and their associated keywords, and a client, who sends queries including keywords and receives the records associated with these keywords. Oblivious keyword search protocols provide privacy for both parties. It enable the client to search for the records associated with queried keyword while hiding the queries from the database. Moreover, it provides server privacy by preventing the clients from learning anything but the results of the queries.

In [73], Ogata and Kurosawa introduced the idea of oblivious keyword search and proposed two protocols satisfying their definition. However, the communication complexity of these protocols is linear in the database size. Freedman *et al.* [27] provide more communication-efficient single-round oblivious keyword protocols secure against malicious users and semi-honest servers.

In [39] Chor et al. showed how the user can privately traverse data structures held by a server. Their solutions for binary search trees provide only user privacy. They also give a solution for tries that is server-private, as well. These techniques yield useful PIR by keyword solutions for situations where the user wants to retrieve a record stored under a known keyword. However, trie is not appropriate to handle range queries.

Keyword search protocols can be used to answer range queries by running a keyword search on every value in the query range. However, these schemes are extremely inefficient in particular when the data is fairly sparse in the queried range.

3.4 Private Database Outsourcing

Private database outsourcing model was first introduced by Hacigumus et al. [11]. They considered protecting the database records of a client from an untrusted database service providers. Most existing approaches in this category resort to data encryption to protect the database confidentiality [74]. Although various techniques have been proposed to protect the database hosted on the cloud server [12, 16], they cannot be adopted in this problem for several reasons. First, to evaluate the query on the encrypted data, the client must encrypt the query by the same scheme and the same key that are used by the data owner and send it to the cloud server. The cloud server may then forward the encrypted query to the data owner, where the query can be decrypted by her encryption key.

Second, a common approach in the existing research proposals is to send a set of encrypted records to the client for filteration and further processing [11–19]. Therefore, the cloud server may reveal extra information beyond the query result to the client. Thus, the proposed techniques for secure database outsourcing will not protect the query privacy and the database privacy.

Recently, CryptDB [75] has been proposed that is built on the top of the existing relational database management systems (RDBMS) and protects the privacy of database records in the cloud computing. The proposal employs various encryption schemes to support all types of SQL queries over encrypted databases. It depends on a fully trusted component, namely CryptDB, that maintains all the secret and public keys and transforms the users' SQL queries to a query that can be executed

Table 1: *Related Work - Summary*

Algorithm	**Client Privacy**	**Data Owner Privacy**	**SQL/KS Queries**	**Database Confidentiality**
Symmetric Private Information Retrieval	✓	✓		
Privacy-Preserving Set Intersection [71,72]	✓	✓		
Private Database Outsourcing [11,18]			✓	✓
Reardon *et al.* [69]	✓		✓	
Olumofin and Goldberg [10]	✓		✓	
Wang *et al.* [22]	✓		✓	
Our proposal	✓	✓	✓	✓

over encrypted records. CryptDB has low overhead on query execution time, however it requires a fully trusted component which is the single point of attack; if the attacker can compromise CryptDB, she can decrypt the database records as well as retrieving the query. The proposed protocol in this thesis makes no assumption about existence of a trusted party; we consider all parties to be semi-honest adversary and our objective is to protect the query privacy and database privacy in the presence of non-trusted parties.

Instead of using encryption, Aggarwal *et al.* [76] proposed secret sharing to hide database records from the adversaries. This approach requires multiple non-colluding servers to keep the share of each database record. However, the assumption of having servers, unaware from each other is strong in the real world. In addition, this approach has significant communication and computation overhead since for executing each query, the the shares of the database must be retrieved to reconstruct the database.

3.5 Privacy-Preserving Data Mining

Data mining and knowledge discovery in databases are new research areas that investigate the extraction of previously unknown patterns from large amounts of data. Privacy preserving data mining (PPDM) [23, 77–79] is a novel research direction in data mining and cryptography, where two or more parties owning confidential databases wish to run a data mining algorithm on the union of their databases without revealing sensitive information of individuals. In particular, although the parties realize that combining their data has some mutual benefit, none of them is willing to reveal its database to any other party. In this case, we need to consider a distributed computing scenario, rather than a scenario where all data is gathered in a central server, which then runs the algorithm on all data.

The term privacy-preserving data mining was first introduced by Agrawal and Srikant [77] and Lindell and Pinkas [23]. These papers considered two fundamental problems of PPDM: privacy-preserving data collection and mining a data set partitioned across several private enterprises. Agrawal and Srikant devised an algorithm that allows multiple parties to contribute their private records for efficient centralized data mining while limiting the disclosure of their values. On the other hand, Lindell and Pinkas invented a cryptographic protocol based on secure multiparty computation for decision tree construction over a data set horizontally partitioned between two parties. The main disadvantage of cryptographic PPDM is that these approaches are difficult to scale when more than a few parties are involved. Moreover, it does not address the question of whether the disclosure of the final data mining result may breach the privacy of individual records.

Other research works in the literature that influence the development of PPDM include privacy preserving classification [80–82], privacy-preserving association rule mining [83–85], privacy-preserving clustering [86–88], privacy-preserving Bayes classifier/Bayesian network [89, 90], and privacy-preserving multivariate statistical analysis [91, 92].

Private database search is considered as an integral part of PPDM. In particular, the result of private aggregate queries, executed by symmetric-private database search protocols, are used as the input to data mining algorithm to extract information and/or patterns. In other words, the protocols proposed in this thesis can be used as subprotocols in PPDM to preserve the data miner's query privacy as well as database records privacy. Therefore, combination of PPDM and symmetric-private database search enables the data miner (i.e., client) to mine the published sanitized data by posing private SQL aggregate queries that provides privacy for data miner and database owner (by symmetric-private database search) as well as individual privacy (by PPDM). In other words, PPDM is a complement to symmetric-private database search that provides individuals privacy.

3.6 Privacy-Preserving Data Publishing

Privacy-preserving data publishing (PPDP) provides methods and tools for publishing useful information while preserving data privacy. In particular identity of individuals, whose records exist in database, must not be identified from published data. Recently, PPDP has received considerable attention in research communities, and many approaches have been proposed for different data publishing scenarios.

A typical scenario for PPDP consists of three phases: in the *data collection* phase, the data publisher collects data from record owners (e.g., patients). In the *data sanitization* the collected data are sanitized by the data publisher. In the *data publishing* phase, the data publisher releases the sanitized data to data miners or to the public, who will then mine the published data to extract useful information. In order to limit the possibility that an individual could be identified from the released data, a combination of sanitization techniques are used including generalization [93–95], suppression [95, 96], swapping [97, 98], and randomization [77].

PPDP differs from our work in several major ways: PPDP focuses on techniques for publishing the data while protecting the privacy of the record owners. In contrast, symmetric-private database search aims to execute queries on the database while protecting the privacy of the client's query and the database records. In fact, it is expected that symmetric-private SQL/keyword search queries are executed on the published data, produced by the PPDP techniques so that the privacy of all parties (the database owner, the record owner and the client) is protected.

It should be noted that there is a significant body of works on distributed privacy preserving data integration, aggregation, and mining (e.g. constructing decision trees [23], computing association rules, classification and clustering [13, 85, 99], and differential privacy [100–102]). These works provide a rich and useful set of privacy preserving tools for the purpose of protecting the privacy of records in databases. More precisely, these works aim to protect the record owner from being identified through query result [103]. They allow a trusted server to release obfuscated answers to aggregate queries to avoid leaking information about any specific record in the database. Such works can be considered to have a different goal and model and can be added as a front end in the proposed protocol to provide privacy-preserving answers to the client's queries.

Chapter 4

Symmetrically-Private Database Search in Cloud Computing

In this chapter, we present SQL query processing protocols that preserve both the data privacy and the query privacy among the data owner, the clients, and the cloud server. These protocols help the clients to keep their sensitive information, contained in the SQL query, from being leaked. Moreover, the sensitive information of individuals in database and database records are protected against the clients that can be modeled as semi-honest adversaries.

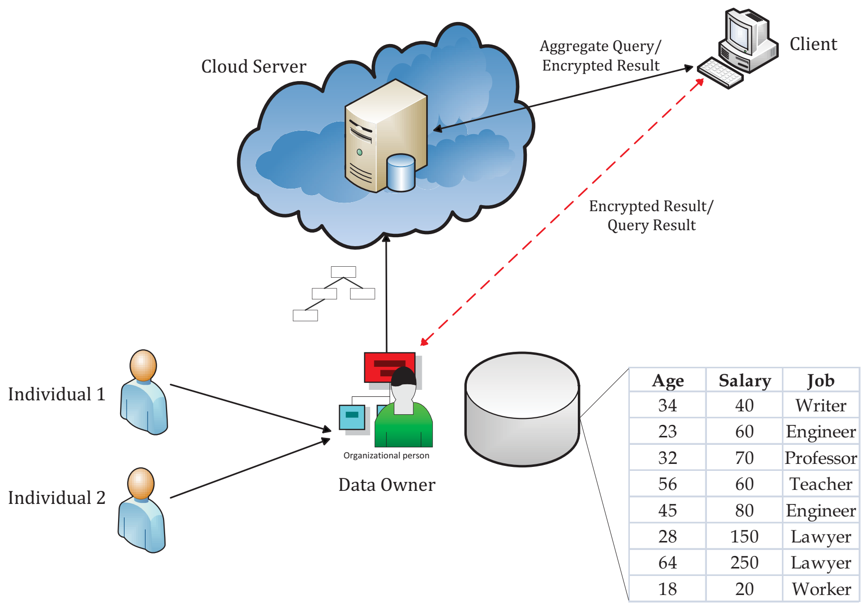
4.1 Introduction

Most software systems request sensitive information from clients and construct a query from the filled form, but privacy concerns can make the client unwilling to provide such information. Thus, development of practical schemes, that protect the query privacy of the clients, is crucial in important application domains like patent databases, pharmaceutical databases, online censuses, real-time stock quotes, location-based services, and Internet domain registration. For instance, the current process for Internet domain name registration requires a client to first disclose the name for the new domain to an Internet domain registrar. Subsequently, the registrar could then register the new domain and thereby resell the domain at the higher price to the client. Therefore, many clients find it unacceptable to disclose the sensitive information contained in their queries to the server [10].

Basic techniques in privacy-preserving SQL-query processing place no restriction on the information leaked about other items in the database, which are not of interest to the client [10, 22, 69]. However, an extension of these techniques adds that restriction by insisting that a client learns only the result of her query. The restriction is crucial in situations where the database privacy is equally of concern.

The problem addressed by Symmetric Private Information Retrieval (SPIR) [28] is to provide a client with the means to retrieve data from a database without the database (or the database administrator) learning any information about the particular item that was retrieved. The rudimentary data access model of SPIR is one of the hindering factors in deploying SPIR-based query processing model. These models are limited to retrieving a single bit or a block of bits in a specific index. There is therefore a need for an extension of SPIR to a more expressive model that is suitable for retrieval from relational databases. In this chapter, we explore the query processing model on relational database that provides data privacy and query privacy among the clients, the data owner and the cloud server.

We consider a cloud computing environment that stores an outsourced database. The main entities in Database-as-a-Service (DBaaS) are individuals, a database owner, clients and a cloud server. In this thesis, individuals refer to someone who owns the information e.g., a patient who has medical records and wants the data owner to store her data while preserving her privacy. The database owner refers to someone who collects the information from individuals and outsources her database to a third-party service provider namely a cloud server, e.g., a hospital manager who collects information about the patients. The cloud server would be given the ability to store the database and the capability to answer certain types of queries issued by the clients. The client is generally someone who can perform search over the databases stored on the cloud server. An illustrative example for the system model is represented in Figure 8.



Age	Salary	Job
34	40	Writer
23	60	Engineer
32	70	Professor
56	60	Teacher
45	80	Engineer
28	150	Lawyer
64	250	Lawyer
18	20	Worker

Figure 8: *System Model in DBaaS*

This chapter deals with constructing protocols for privately answering aggregate queries in such a way that the privacy of the clients and the database owner is preserved. Roughly speaking, a protocol is client-private if the cloud server and the database owner learn nothing about the constants in the client's query and query results. Similar to [10], we presume that the shape or textual content of SQL queries is not private, but the constants the client supplies in the WHERE clause are private, and must be protected. Our approach to preserve query privacy over a relational database is based on hiding all the constants included in queries.

Similarly, a protocol is server-private if the client learns no additional information about the server's database beyond the correct answer to her query. A protocol which is both client-private and server-private is called symmetrically-private. The symmetrically-private protocols are important in the applications where the privacy of database records is an important concern such as in the healthcare systems.

4.2 Threat Model and Adversary Capabilities

We consider the clients, the cloud server, the database owner and any other attackers who can view the data retrieved from the server and monitor activities on the client and the server as adversaries. We assume that the adversaries are semi-honest: they follow the protocol's steps correctly (i.e. they are not malicious), but they are free to infer clients' queries and database owners' private data. The proposed protocol is secure at the presence of semi-honest adversaries.

4.3 Symmetrically-Private Keyword Search

Consider a database owner $\mathcal{D}$ who owns a raw data set D. To reduce the workload of the data owner in answering queries, the data owner delegates the tasks of database storage and query processing to a cloud server $\mathcal{S}$. The cloud server may not be fully trusted by the data owner or may be susceptible to attacks by malicious parties (both internal and external). Therefore, the data owner needs to encrypt the database records before outsourcing them to the cloud server. The client $\mathcal{C}$ wants to execute aggregate queries on the encrypted database stored on the server without disclosing the constant in her queries as well as the query result. The server $\mathcal{S}$ wants to answer the client's query without disclosing any records that are not part of the result.

In a naive approach, the client performs a linear search on the server's database by following the millionaire protocol [35, 104] to evaluate the SQL query at each database entry. However, this approach incurs excessive communication overhead which is linear in the number of records in the database due to the linear search. To reduce the communication overhead, the data owner organizes the database as a binary search tree, creates an encrypted database from this tree using the Goldwasser-Micali (GM) encryption [42] and outsources the encrypted database to the cloud server. The client's objective is to traverse the resulted tree represented by the encrypted database to obtain the query result. This can be achieved by exploiting the homomorphic properties of the GM encryption using Fischlin's protocol for private comparison and by utilizing SPIR for private information retrieval. Tree search can considerably reduce communication and computation overhead.

In this section, we propose a protocol that executes keyword search queries on a database on the cloud server side. It consists of three steps: (i) tree construction and database encryption, (ii) oblivious tree search and payload retrieval (iii) query result decryption. The Symmetrically-Private Keyword Search (SP-KS) protocol takes as inputs the following:

– The database which consists of N pairs $\{(k_i, p_i)\}_{i\in[N]}$ where k_i denotes the keyword and p_i represents the payload (database record) owned by $\mathcal{S}$. We assume without loss of generality, that all k_i's are distinct.

– A search word w owned by $\mathcal{C}$. $\mathcal{C}$ obtains p_i if there is i such that $k_i = w$ or a special symbol $\perp$, otherwise.

In the following, we elaborate the basic steps of SP-KS protocol that preserves the privacy of both the client and the data owner. The proposed protocol improves over previous research proposals in this domain with less communication and computation on the client but incurs more computation cost on the server.

4.3.1 Tree Construction and Database Encryption

The data owner defines $L = \sqrt{N}$ bins, where N denotes the number of records in the database. She maps N pairs into the L bins using a random, publicly-known hash function H with a range of size L. Therefore, the record (k_i, p_i) is mapped to the bin $H(k_i)$. For each bin j, the data owner sorts the assigned pairs according to the keyword. Afterwards, the data owner employs the algorithm explained in Section 2.3 to construct a left-balancing binary search tree T_j for each bin j depending on keyword. The resulted T_j's are traversed to generate the database A_j. Later, A_j's are concatenated to form a single database A that contains the entire records. To store the roots of the trees, an index array I with length L is created such that $I[j]$ contains the root index of the tree T_j in the database A.

Note that index-based database A contains database records in cleartext. To outsource A to the cloud server, the data owner must encrypt the database pairs. For this purpose, the data owner generates two public keys PK_{PL} and PK_{GM} for the Paillier cryptosystem [26] and the threshold Goldwasser-Micali (GM) [42] cryptosystem, respectively. The database owner makes use of Paillier encryption to encrypt the payloads while threshold GM encryption is used to encrypt keywords bit by bit. The database owner computes the secret key SK_{GM} of GM cryptosystem. The secret key SK_{GM} (factors of the public key PK_{GM}) are then distributed among the existing clients as detailed by Katz and Yung [24]. At the end of this step, the database owner outsources the encrypted index-based dataset A to the cloud server and publishes the public keys to all clients.

Example 1. *The data owner wants to outsource the dataset shown in Table 2. Suppose that the first column is the keyword and the second column indicates the associated payload. The original table has $N = 8$ records and consequently the number of bins is $\sqrt{N} \approx 3$. We define the public hash function as $H(x) = x \bmod 3 + 1$. The* *assigned records to each bin, sorted according to the Age attribute is represented in Figure 9. For each bin, the left-balancing binary search tree is generated as explained in Section 2.3. The resulted left-balancing binary search trees are shown in Figure 10. afterwards, each tree is traversed to generate a database. The resulted datasets are concatenated to construct the index-based database shown in Table 3. To outsource the index-based database to the cloud server, the payloads (values of the Job attribute) is encrypted by Paillier cryptosystem whereas the keywords are encrypted by the threshold GM cryptosystem. The outsourced database is presented in Table 4. In addition, the data owner constructs an array I that contains the root indexes of the generated left-balancing binary search trees as demonstrated in Table 5.*

Age	**Job**
50	Writer
57	Engineer
22	Dancer
55	Lawyer
60	Writer
29	Engineer
28	Dancer
49	Lawyer

Table 2: *Table D*

Table 3: *Index-Based database*

Age	**Job**
60	Writer
57	Engineer
49	Lawyer
28	Dancer
55	Lawyer
22	Dancer
50	Writer
29	Engineer

Table 4: *Outsourced database*

Age	**Job**
$E_{GM}(60)$	E_{PL}(Writer)
$E_{GM}(57)$	E_{PL}(Engineer)
$E_{GM}(49)$	E_{PL}(Lawyer)
$E_{GM}(28)$	E_{PL}(Dancer)
$E_{GM}(55)$	E_{PL}(Lawyer)
$E_{GM}(22)$	E_{PL}(Dancer)
$E_{GM}(50)$	E_{PL}(Writer)
$E_{GM}(29)$	E_{PL}(Engineer)

Age	**Job**
57	Engineer
60	Writer

(a) *Bin 1*

Age	**Job**
22	Dancer
28	Dancer
49	Lawyer
55	Lawyer

(b) *Bin 2*

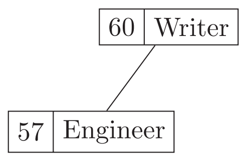
Age	**Job**
29	Engineer
50	Writer

(c) *Bin 3*

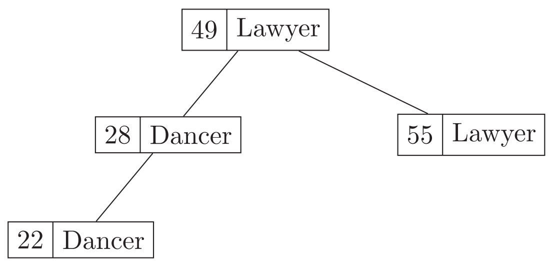
Figure 9: *Generated Bins*

0	2	6

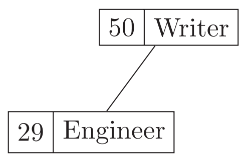
Table 5: *Index Array I*



(a) *Bin 1*



(b) *Bin 2*



(c) *Bin 3*

Figure 10: *Left-Balancing Binary Search Trees*

4.3.2 Oblivious Tree Search and Payload Retrieval

In order to execute keyword search (KS) queries, the client needs to traverse the tree stored on the cloud server side. Traversing a tree is performed by retrieving the root's keyword-payload pair, comparing the search word with the root's keyword, and terminating the search or determining a new root. Since, the database on the cloud server is encrypted by the GM cryptosystem, the comparison must be done on ciphertexts instead of plaintexts. To compare the ciphertexts, we rely on the Fischlin protocol. The Fischlin protocol takes as the inputs the bit representation of two integers, encrypted by the GM cryptosystem, and outputs an encrypted ciphertext sequence. The decryption of the output ciphertext sequence determines which input is greater.

To utilize the Fischlin protocol, the client encrypts her search word w by the data owner's public key using the GM cryptosystem and sends it to the server. For each pair $\{(E_{GM}(k_i), E_{PL}(p_i))\}_{i\in[A]}$ in the database, the cloud server executes one instantiation of Fischlin protocol $\mathsf{GTE\text{-}F}$ detailed in Chapter 2.2.4 using the encrypted search word $E_{GM}(w)$ and the encrypted keyword $E_{GM}(k_i)$ as the inputs. The output of this protocol is the ciphertext sequences Δ_i and c_i that will be stored in a new database T together with the encrypted payloads $E_{PL}(p_i)$ using the Paillier scheme. More precisely, the content of the i-th tuple of the database T ($T[i]$) is the result of comparing the client's search word w with the i-th keyword of the index-based database A in the encrypted form together with the encrypted payloads $E_{PL}(p_i)$ using the Paillier scheme. Afterwards, the database T is made public to the client for the query execution.

Example 2. *(Continued from Example 1) Suppose that the client's search word w equals 28. The client encrypts w bit-by-bit using the data owner's public key and sends the ciphertext sequence to the server. The server executes Fischlin's protocol $\mathsf{GTE\text{-}F}$ to* *calculate ciphertext sequences Δ and c for each encrypted keyword in the outsourced database (Table 4) stored in the cloud server. The resulted ciphertext sequences along with the corresponding Paillier encrypted payloads are stored in the table T (Figure 6). Note that the first two columns are the output of the Fischlin protocol.*

Δ_{60}	c_{60}	E_{PL}(Writer)
Δ_{57}	c_{57}	E_{PL}(Engineer)
Δ_{49}	c_{49}	E_{PL}(Lawyer)
Δ_{28}	c_{28}	E_{PL}(Dancer)
Δ_{55}	c_{55}	E_{PL}(Lawyer)
Δ_{22}	c_{22}	E_{PL}(Dancer)
Δ_{50}	c_{50}	E_{PL}(Writer)
Δ_{29}	c_{29}	E_{PL}(Engineer)

Table 6: *Table T for SP-KS*

To execute keyword search queries efficiently, the client needs to traverse the generated binary search trees. Note that the client needs only to traverse the tree of the bin in which w could exist. To obtain the tree root of this bin, the client calculates $H(w)$ which determines the bin as well as the index of the tree root in the array I. Both parties run SPIR such that the server's input is the array I and the clients's input is $H(w)$. SPIR guarantees that the client learns $I[H(w)]$ and the server learns nothing. After yielding the tree root for the bin that the client is interested in, the client initiates the search by retrieving the content of tree root in array T. The client extracts Δ and c from the retrieved item. As explained before, Δ and c indicate the result of comparing the search word w with the tree root's keyword in encrypted form. Therefore, by decrypting these ciphertext sequences, the client can evaluate the comparison result. Based on the comparison result, the search is continued on the left or on the right subtree by updating the root index. Additionally, the client retrieves the index of the tree root of the next bin to obtain the current bin's boundaries. To fully understand the scheme, we need to detail the following points.

– *Retrieving the root data by SPIR.* The database T contains the result of comparing the client's keyword with each record in the database. To traverse the tree of the bin that the client is interested in, the client needs to know the result of comparing the search word w with the root's keyword. To protect mutual privacy of the client and the server, both parties run SPIR to provide the client with Δ and c generated for the bin root.

– *Ciphertext decryption.* The ciphertext sequences Δ and c are generated by Fischlin's protocol for private comparison. These ciphertext sequences are encrypted by the data owner's public key under the threshold GM cryptosystem. Therefore, decryption of Δ and c is performed by the participation of a specific number of active clients. Each active client, that has been contacted by the querier client, partially decrypts the ciphertexts Δ and/or c by her share of secret key and sends the share of plaintext to the querier client. The querier client then aggregates the partial decryption results to obtain the actual plaintext.

– *Updating the root index.* Initially, the relative index of current root in the tree is $i = 1$ and the offset of all records in the bin $H(w)$ (with respect to the first array element in the database T) is $(I[H(w)] - 1)$. Therefore, the sum of the index i and the offset $(I[H(w)] - 1)$ determines the actual (global) index (i') of the tree root in the array T $(i' = i + I[H(w)] - 1)$. To search the tree, the client retrieves the data stored in the index $(i' = i + I[H(w)] - 1)$ in the array T. If the search word is less than the root's keyword, the search must be performed on the left subtree. The left subtree will be rooted at position $(2i + I[H(w)] - 1)$. Similarly, if the search word is greater than root's keyword, the search is continued on the right subtree by updating index i' to $(2i + 1 + I[H(w)] - 1)$. If the search word is equal to the root's keyword, the encrypted payload is stored on the client side as the query result for decryption and the tree search is terminated. Otherwise if $i' > I[(H(w)+1) \bmod L]$ or $i' > N$, the client concludes that the search word does not exist in the database.

Example 3. *(Continued from Example 2) Client's search word is 28 (w=28). The client calculates* $H(28) = 28 \bmod 3 + 1 = 2$. *Therefore, the client needs to search bin 2. To traverse the tree associated with bin 2, the client and the server engage in SPIR to retrieve the numbers at index 2 and at index 3 of the array* I *(i.e.,* $I(2)$ *and* $I(3)$*). SPIR outputs* $I[2] = 3$ *and* $I[3] = 7$ *to the client which are the indexes of trees' roots in* T. *Therefore, the offset is* $I[2] = 2$ *and the global index* i' *can take the values 3, 4, 5 and 6. Initially,* $i = 1$ *and* $i' = 1 + 2 = 3$. *At each step if* $i' > 6$ *or* $i' > 8$ *the search is terminated. Otherwise, The client retrieves* $T[i'] = T[3] = \{\Delta_{49}, c_{49}, E_{PL}(\text{Writer})\}$. *The client then forwards* $\{\Delta_{49}, c_{49}\}$ *to a specific number of online participants (i.e., clients) for decryption. The participants calculate their share of plaintext and send it to the client. The client then computes the plaintexts* $\mathtt{D}(c)$ *and* $\mathtt{D}(\Delta)$ *and concludes that the root's keyword is greater than* w. *Therefore, the client continues searching on the left subtree. Afterwards, the client updates* $i = 2 * 1 = 2$ *and* $i' = i + I[2] - 1 = 4$. *Similarly, the client retrieves* $T[4] = \{\Delta_{28}, c_{28}, E_{PL}(\text{Dancer})\}$ *from the cloud server and forwards it to the participants for decryption. The resulted plaintexts* $\mathtt{D}(c)$ *and* $\mathtt{D}(\Delta)$ *indicate that the root's keyword is equal to the search word and accordingly the search is terminated.*

4.3.3 Query Result Decryption

After the client obtains the query result (payload), she communicates with the data owner to decrypt it. To prevent the data owner from learning the query result, the client obfuscates the query result. Payload obfuscation is performed by generating a random number R, encrypting it by the data owner's public key and multiplying it by the encrypted payload to obtain $E(p_i + R)$. The obfuscated encrypted payload is sent to the data owner for decryption. The data owner returns the obfuscated decrypted payload to the client and the client subtracts R to obtain the actual payload.

4.4 Symmetrically-Private SQL Search

In Section 4.3, we presented a protocol that allows a client to securely execute her keyword search queries over a database, stored on the cloud server side. In this section, we extend the idea to the SQL queries so that the client will be able to securely evaluate her SQL query and obtain the query result. The client's input is an aggregate SQL query that consists of exact-matching and interval-matching predicates on multiple attributes combined with logical operators (AND/OR/NOT). The server's input is a relational encrypted database. In the following, the basic steps of the Symmetrically-Private SQL Search (SP-SQL) protocol are described. Our approach to preserve the query privacy is based on hiding constants contained in the query predicates. The basic assumption is that the shape of SQL queries is not private, but the constants provided by clients must be protected from other parties. Moreover, the only piece of information that is revealed to the client is the query result. Handling SQL queries are generally more complex than KS queries in the sense that they usually contain more than one attribute. In addition, answering SQL queries requires supporting interval matching and exact matching in the predicate.

4.4.1 Tree Construction

The data owner organizes the database D as a left-balancing k-Dimensional tree (kD-tree), encrypts and stores it on the server. The client sends the sanitized version of SQL queries by replacing constants in the predicate by their corresponding encryption. The server generates the required data for executing the client's query and the client executes the query by traversing the kD-tree in an oblivious manner. To construct left-balancing kD-tree, we use a similar approach that is described in Section 2.3. Assume that the database has n attributes and the corresponding attributes are sorted according to a pre-specified order. At each tree level d, the attribute A_a ($a = d \bmod n + 1$) is chosen and the data owner sorts the records based on A_a. Note that if there exist multiple records with the same value for attribute A_a, the data owner sorts them with respect to the other attributes in the sorted sequence, namely $A_{a+1}, \ldots, A_n$. Let LT and RT denote the number of records in the left subtree and the right subtree, respectively. Then, LT and RT are computed and the records are partitioned into two subsets through the $(LT + 1)$-th record. All records on the left of the $(LT + 1)$-th record constitute the left subtree and the remaining records will construct the right subtree. The procedure is executed recursively until there exists only one record. The resulted tree is then traversed and stored in an index-based database named A. In order to employ Fischlin protocol, the outsourced database is encrypted by the GM encryption, using the data owner's public key. In addition, the client wants to execute aggregate queries (*sum*, *avg*, *count*) on the outsourced database. To support these types of queries, all columns are also encrypted by the Paillier cryptosystem using the public key of the data owner.

Example 4. *Suppose that the data owner wants to outsource Table 7 to the server. The attributes are sorted as Job, Age and Salary. At each level, LT and RT are computed and the records is partitioned according to the chosen attribute. For instance, in the first level, attribute Job will be chosen;* $N = 10$ *and* $M = 8$ *which resulted in* $R = 3$, $LT = 6$ *and* $RT = 3$. *To discover the record of the root, the records are sorted with respect to the selected attribute (i.e., Job). Therefore, the record (Lawyer,49,44) will be chosen as the root in the first level. The resulted left-balancing kD-tree is represented in Figure 11. The index-based database named A is also shown in Table*

Table 7: *Original Database*

Job	Age	Salary
Writer	50	30
Dancer	40	60
Writer	60	35
Dancer	30	37
Engineer	29	60
Engineer	31	35
Engineer	50	70
Dancer	28	44
Lawyer	49	44
Lawyer	55	80

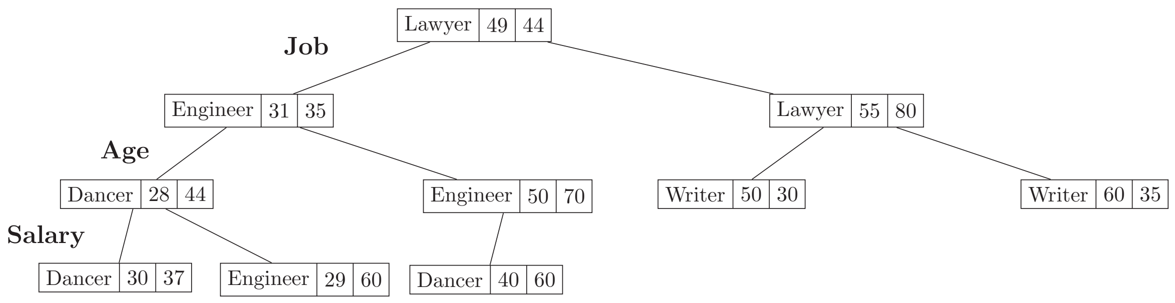


Figure 11: *Left-Balancing kD-tree*

8. The outsourced database, stored by the cloud server, has 6 attributes which are $E_{GM}(Job)$, $E_{GM}(Age)$, $E_{GM}(Salary)$, $E_{PL}(Job)$, $E_{PL}(Age)$, and $E_{PL}(Salary)$.

4.4.2 Query Sanitization

To execute a SQL query on the outsourced database, the client sanitizes the SQL query by replacing the constants contained in the predicate by their GM encryption using the public key of the data owner. For example, if the client's SQL query is

$$\texttt{SELECT SUM(}\text{Salary}\texttt{) FROM } D \texttt{ WHERE } \text{Age} < 50 \texttt{ AND } \text{Job} = '\text{Dancer}'$$

The sanitized query will be

$$\texttt{SELECT SUM(}\text{Salary}\texttt{) FROM } D \texttt{ WHERE } \text{Age} < \mathrm{E}_{\mathrm{GM}}(50) \texttt{ AND } \text{Job} = \mathrm{E}_{\mathrm{GM}}(\text{“Dancer”})$$

Table 8: *Index-based Database*

Job	**Age**	**Salary**
Lawyer	49	44
Engineer	31	35
Lawyer	55	80
Dancer	28	44
Engineer	50	70
Writer	50	30
Writer	60	35
Dancer	30	37
Engineer	29	60
Dancer	40	60

4.4.3 Oblivious Tree Traversal

Sanitized queries are sent to the server to generate the ciphertext sequences Δ and c, that are the output of the Fischlin protocol and are required to compare two ciphertexts. Similar to SP-KS protocol, we need to generate the columns of the database T that are required to evaluate the query predicate. The generated columns depend on the type of the query, as follow:

– Interval matching or exact matching. If an attribute appears in either an interval matching or an exact matching predicate, a column is added to the database T that contains the ciphertext sequence Δ generated by Fischlin's protocol for private comparison. In the case of exact matching, another column that contains the ciphertext sequence c is added. These two columns are generated from the encrypted constants in the predicate and from the corresponding encrypted column in the relational database.

– *sum* and *avg* queries. The Paillier encryption of the attribute, targeted by the aggregate function, is added to the database T.

– *max* and *min* queries. The GM encryption of the attribute, targeted by the aggregate function, is added to the database T.

– *count* queries. No additional column is required.

Example 5. *(Continued from Example 4) Consider the following sanitized SQL query*

$$\texttt{SELECT SUM(}Salary\texttt{) FROM } D \texttt{ WHERE } \mathrm{Age} < \mathrm{E_{GM}}(50) \texttt{ AND } \mathrm{Job} = \mathrm{E_{GM}}(\text{“Dancer''})$$

The generated database T is represented in Table 9.

Table 9: *Table T for SP-SQL*

Salary	Age	Job-CMP	Job-EQU
$E_{\mathrm{HOM}}(49)$	Δ_{44}	Δ_{Lawyer}	c_{Lawyer}
$E_{\mathrm{HOM}}(35)$	Δ_{31}	$\Delta_{\mathrm{Engineer}}$	c_{Engineer}
$E_{\mathrm{HOM}}(80)$	Δ_{55}	Δ_{Lawyer}	c_{Lawyer}
$E_{\mathrm{HOM}}(44)$	Δ_{28}	Δ_{Dancer}	c_{Dancer}
$E_{\mathrm{HOM}}(70)$	Δ_{50}	$\Delta_{\mathrm{Engineer}}$	c_{Engineer}
$E_{\mathrm{HOM}}(30)$	Δ_{50}	Δ_{Writer}	c_{Writer}
$E_{\mathrm{HOM}}(35)$	Δ_{60}	Δ_{Writer}	c_{Writer}
$E_{\mathrm{HOM}}(37)$	Δ_{30}	Δ_{Dancer}	c_{Dancer}
$E_{\mathrm{HOM}}(60)$	Δ_{29}	$\Delta_{\mathrm{Engineer}}$	c_{Engineer}
$E_{\mathrm{HOM}}(60)$	Δ_{40}	Δ_{Dancer}	c_{Dancer}

Initially, the search begins with the tree root at index $i = 1$ of the database T. At each iteration, the client retrieves the corresponding record of the current root by SPIR. The retrieved record contains the ciphertext sequences Δ and c (to support comparison) and the Paillier-encrypted column (to prepare the query result). If the current root is a leaf node (i.e., its left child does not exist $(2i > N)$) and it satisfies the conditions, it is reported as part of the query result and stored by the client for further processing. Exact matching predicates are evaluated by testing the ciphertext sequence c; if all ciphertexts in the sequence c are quadratic residue, the root is reported as the answer and the search is continued on both the left and the right subtrees. Otherwise, the ciphertext sequence Δ is examined to determine the search path: if there exist at least one λ sequence of quadratic residue in Δ, the search will be continued on the right subtree since the queried records have greater value than the root. Otherwise, the left subtree is searched and the index i is updated accordingly in both cases.

Range search over kD-tree is performed in a similar way; if the boundaries of the left (resp. right) subtree is fully contained in the query region, the entire records in the left (resp. right) subtree are reported as the query result. Otherwise, if the boundaries of the left (resp. right) subtree intersects with the query space, search is continued on the left (resp. right) subtree by updating index to $2i$ (resp. $2i+1$). All the comparisons are performed using Fischlin protocol and examining the output.

Example 6. *(Continued from Example 5) The query region is* $\text{Age} \times \text{Job} \ : \ [0-50] \times$ "*Dancer*". *The client starts from the root by setting* $i = 1$. *The output of SPIR is the tuple* $(E_{\text{HOM}}(49)\ , \Delta_{44}\ , c_{\text{Lawyer}})$. *The client decrypts* Δ_{44} *and* c_{Lawyer} *and finds out that the tuple does not satisfy query predicates and the root value for the Job attribute is greater than Dancer. Since at the first level, the records are sorted according to Job attribute, the client concludes that the right subtree does not have any intersection with the query region, but the left subtree does. Therefore, index i is updated to* 2 *and the tuple* $(E_{\text{HOM}}(35), \Delta_{31}\ , c_{\text{Engineer}})$ *is retrieved. The retrieved tuple does not satisfy the condition, but its Age's attribute value is less than 50. Therefore, the client needs to perform search on both subtrees, recursively. At next iteration,* $i = 4$. *The retrieved tuple indicates matching. Since the records are sorted according to Salary and there is no condition in the WHERE clause associated with it, the client needs to perform search on both subtrees recursively. By performing range search over the kD-tree, the client obtains* $\{\Big(E_{\text{HOM}}(60), \Delta_{40}, \text{c}_{\text{Dancer}}\Big)\ , \Big(E_{\text{HOM}}(37),\ \Delta_{30}\ , c_{\text{Dancer}}\Big), \Big(E_{\text{HOM}}(44), \Delta_{28}, \text{c}_{\text{Dancer}}\Big)\}$

4.4.4 Query Result Decryption

The last step is to decrypt the query result while hiding it from the cloud server, the data owner and the other clients. Note that, the query result is encrypted by the data owner public key. Therefore, the client has to communicate with the data owner to decrypt the result. Sending the query result in cleartext jeopardizes the client privacy. Therefore the client must obfuscate the result to make it invisible to data owner. The obfuscation depends on the type of the aggregate function as follow:

– *count*: the cardinality of query result reflects the exact value the client expects.

– *sum*: the client projects the records (that satisfy the query conditions) over the attribute targeted by the *sum* function. She then multiplies the resulted attribute values to capture the encrypted sum. She then generates a random number R and encrypts it by the data owner's public key ($E_{\texttt{HOM}}(R)$). Then the client multiplies the encrypted random number by the encrypted sum to obtain the obfuscated sum, i.e., $E_{\texttt{HOM}}(Sum) \times E_{\texttt{HOM}}(R) = E_{\texttt{HOM}}(Sum + R)$. The client sends obfuscated sum to the data owner. The data owner decrypts the noisy sum to obtain $R + Sum$ and sends it back to client. The client, in its turn, subtracts the noise R that leads to the query result.

– *max,min*: the client projects the records (that satisfy the query conditions) over the attribute targeted by the *max/min* function. She then executes the maximum algorithm on the ciphertexts using Fischlin protocol: the client picks up the first value as the current maximum (resp. minimum). She then compares the current encrypted maximum (resp. minimum) with the other values; if the output of Fischlin indicates that the current maximum (resp. minimum) is less than (resp. greater than) the examined value, the current maximum (resp. minimum) is updated. Afterwards, the client generates a random number R

and encrypt it bit by bit using GM-cryptosystem and the data owner's public key ($E_{\texttt{GM}}(Max)$). She then find the component-wise modular multiplication of the current encrypted maximum/minimum with the encrypted random number to obtain the obfuscated maximum (resp. minimum) (i.e., $E_{\texttt{GM}}(Max).E_{\texttt{GM}}(R) = E_{\texttt{HOM}}(Max \oplus R)$). The clients sends the obfuscated maximum (resp. minimum) to k other clients for decryption and receives $Max \oplus R$ (resp. $Min \oplus R$). To retrieve the query result, the client XORes the obfuscated query result with R to derive the actual maximum (resp. minimum).

Note that *avg* queries can be answered by executing *count* and *sum* queries and dividing the result of the sum by the count.

Example 7. *(Continued from Example 6) The client projects the records (tuples)* $\{\big(E_{\text{HOM}}(60), \Delta_{40},\ c_{\text{Dancer}}\big), \big(E_{\text{HOM}}(37), \Delta_{30}\ , c_{\text{Dancer}}\big), \big(E_{\text{HOM}}(44), \Delta_{28}, \mathrm{c}_{\text{Dancer}}\big)\}$ *over the* Salary *attribute and multiplies them to obtain* $E_{\text{HOM}}(60+37+44) = E_{\text{HOM}}(151)$. *The client then generates a random number R and encrypts it by the data owner public key PK_{PL} to produces $E(R)$. She then multiplies the encrypted sum by $E(R)$ and sends the result to the data owner for decryption. The data owner decrypts the ciphertext and returns* $151 + R$ *to the client. The client simply subtracts R to obtain the query result.*

4.4.5 Improved Protocol

The presented protocol is the natural extension of the SPKS protocol to multidimensional data in order to perform search over relational databases. The proposed SP-SQL protocol provides absolute privacy for all parties in the environment; however PIR protocols have been widely used which are deemed to be impractical for real-world applications [105]. In this section, we step back from absolute privacy in favor of efficiency and propose a protocol which does not make use of SPIR protocols. The improved approach relies on the cloud server to perform the search on the encrypted datasets, instead of the clients. In this case, the client sanitizes the query and sends it to the cloud server. The cloud server conducts the search on the encrypted records by traversing the tree; the server privately evaluates the record in the tree root in the same manner of the basic SP-SQL. To do so, the cloud server communicates with the clients to perform decryption. Based on the result of comparison, the search is continued on the left and/or the right subtree.

The improved SP-SQL does not require to use SPIR. It also shifts the burden of searching from the querying client to the cloud server. The drawback is that the cloud server finds out how many records satisfy the conditions in the SQL query, but the constants in the query and the database records would be kept confidential from the cloud server. Nonetheless, there are some application settings in which the cloud server may be at least partially trusted in the sense that leaking the number of records would not jeopardize the query privacy of the clients.

4.5 Security and Complexity Analysis

In this section, we will provide the security analysis of SP-KS and SP-SQL protocols. The security of the protocols relies on the security of the implementation of the underlying multiparty computation (MPC) primitives including symmetric private information retrieval and private integer comparison. Moreover, the output of each secure MPC primitive is the input to another secure MPC primitive. For instance, the output of SPIR is the ciphertext sequences Δ and c which will be the input to the private integer comparison protocol. Therefore, according to the Composition Theorem [38], the tree traversal protocol is secure; the the cloud server does not learn anything about the constants in the query because the client encrypts them. Moreover, SPIR guarantees that the cloud server does not know which node is currently visited by the client; the client also does not obtain any information about the other items in the database and the retrieved item (which is the ciphertext sequences Δ and c) do not reveal any information about the database records [25].

Query result decryption step is also secure in the sense that the client obfuscates the query result by a random number. Therefore, when the database owner (in the case of *sum,avg*) or the clients (in the case of *max,min*) decrypt the query result, they cannot learn any information about the query result from the obfuscated plaintext.

4.5.1 Possible Attacks and Mitigations

The mentioned security analysis is valid while the parties do not collude. In this section, we will study the possible attacks resulted from the collusion between different parties.

The threshold decryption will be compromised if the number of colluding clients under the control of an adversary exceeds the threshold k. The threshold decryption is required when the querier client wants to decrypt ciphertext sequences Δ and c. In this case, the adversary does not learn any information about the plaintexts that are compared. Instead, she can only understand if the database record at the root satisfies the query condition(s) or not. However, the colluding clients are oblivious about the root of the subtrees, visited by the client. In addition, the client needs to engage in the threshold GM decryption to decrypt obfuscated minimum and maximum. Note that the query result has been obfuscated to prevent the colluding clients from recovering the query result.

The collusion can also occurs between the cloud server and the client adversaries. In this case, the database stored on the cloud server side can be recovered by combining the secret keys of the client adversaries and decrypting the records. However in practice, we can lower the risk to an acceptable level by implementing other mechanisms. One possible solution is to store the client keys in smart cards (or other tamper resistant devices) as proposed in [106]. Another possible solution is to increase the threshold k such that the attackers are not able to compromise too many patients. Despite simplicity, this mechanism has two disadvantages: First, it will decrease the system's availability: as the number of required online data owners increases, it is more unlikely that they are online to perform decryption. Second, it will increase the communication cost on the party who is searching the database because she needs to communicate with more parties for decryption. Therefore, there should be a trade-off between availability-security and efficiency by choosing a proper value for k.

A possible threat to the query privacy of the client, is that the cloud server may forward the sanitized query to the data owner. As mentioned before, the constants in the sanitized query have been encrypted with the data owner's public key. Therefore, if the data owner and the cloud server collude, the data owner can decrypt the constants and find out what the client is looking for. This attack can be easily mitigated by enforcing the clients to generate the secret key without depending on the data owner to act as the trusted dealer. The threshold GM cryptosystem without the trusted dealer has been proposed by Katz and Yung [24]. In this case, all clients execute distributed key generation algorithm that has been described before in Section 2.2.3 and obtain their share of secret key as well as the public key. The public key is then sent to the data owner for database encryption.

4.5.2 Complexity Analysis

This section explores the complexity of the proposed SP-KS and SP-SQL protocols in terms of storage, communication and computation. Moreover, the efficiency of each protocol is compared with the existing research proposals.

Let N be the number of the existing records, n the number of the reported nodes that satisfy the query predicates in the case of partial matching and range matching queries and k is the threshold of the GM scheme. Furthermore, let d denotes the total number of attributes and s indicates the number of attributes in the query predicates. Notice that the computation complexity of the tree traversal for exact matching, partial matching and interval matching is $O(\log N)$, $O(n + N^{1-s/d})$ and $O(n + N^{1-1/d})$, respectively [62]. Furthermore, the communication complexity of SPIR is $O(K \log N)$ where K is the security parameter whereas the computation cost on the client and the server is $O(\log N)$ and $O(N \log N)$, respectively.

Storage Complexity

SP-KS Protocol. For each keyword, the server needs to generate Δ and c to enable comparison. Therefore, total storage overhead of SP-KS protocol is $O(N)$.

SP-SQL Protocols. The server needs to store a table contains the ciphertext together with the generated Δ and c for each column whose corresponding attribute appears in the query predicate. Therefore, the total storage overhead of SP-SQL protocol is $O(Ns)$ on the server.

Communication Complexity

SP-KS Protocol. In SP-KS protocol, the client-server communication is required to retrieve the index of tree root associated with the bin –that contains the query result– and traversing the tree. The length of array I is $\sqrt{N}$. Thus, the communication cost of retrieving the root index by SPIR is $K \log \sqrt{N} = O(K\lambda \log N)$ where K is the security parameter of SPIR and λ is the error parameter of the Fischlin protocol. Traversing the tree with $\sqrt{N}$ nodes requires at most $\log \sqrt{N} = O(\log N)$ steps. At each step, the root is retrieved from the set of N nodes by SPIR. The communication cost of

SPIR is $K \log N$, Therefore, traversing the tree imposes communication complexity of $O(K\lambda \log^2 N)$.

In addition, the client needs to communicate with k active clients to decrypt the ciphertext sequences c and Δ. This step requires totally exchanging $O(ks\lambda \log N)$ bits. Finally, the client communicates with the data owner to decrypt the payload. Since the query result contains at most one keyword-payload pair, we can ignore the communication cost of the query result decryption. Therefore the total communication complexity of SP-KS protocol is $O(K \log^2 N)$

SP-SQL Protocol. The communication cost of SP-SQL is proportional to the number of visited nodes and depends on the type of the query. Each iteration requires the execution of one instance of SPIR and the ciphertext decryption. The execution of an instance of SPIR incurs $O(K \log N)$ on both sides for communication. The communication cost of each ciphertext decryption is $O(kd)$ per visited node.

Therefore, the communication cost of SP-SQL for different type of queries is as follow:

- Exact matching: the number of visited nodes is $O(\log N)$. Therefore, the communication cost is $O\Big(\log N(K \log N + kd)\Big)$ which is $O(K \log^2 N)$.
- Partial matching: the number of visited nodes is $O(n + N^{1-s/d})$. Therefore, the communication cost is $O\Big(n + N^{1-s/d}(K \log N + kd)\Big)$ which is $O\Big(K \log N(n + N^{1-s/d})\Big)$.
- Range matching: the number of visited nodes is $O(n + N^{1-1/d})$. Therefore, the communication cost is $O\Big(n + N^{1-1/d}(K \log N + kd)\Big)$ which is $O\Big(K \log N(n + N^{1-1/d})\Big)$.

Computation Complexity

SP-KS Protocol. Computation complexity on the active participants, the client, the server and the data owner can be computed as follow:

Active Participants. In every iteration, each active participant requires to compute its own share of plaintext to decrypt Δ and c. Therefore, the total computation overhead on each participant is $O(\log N)$.

Client. In the beginning of the SP-KS protocol, the client needs to encrypt the search word in $O(m)$ time, obtain the bin number in $O(1)$ and retrieves the tree root from array I by SPIR in $O(K \log N)$. Searching the tree is done by at most $\log \sqrt{N} = O(\log N)$ iteration and each iteration consists of retrieving root by SPIR, performing private comparison and updating index. The root is retrieved from the bin with size $\sqrt{N}$ that incurs $K \log \sqrt{N} = O(K \log N)$ computational complexity. The computation cost of ciphertext decryption (from the received shares of plaintext) is $2km$. Updating the index is performed by simple arithmetic in constant time. Therefore, the cost of tree traversal on client is $O(K \log^2 N)$.

Server. In each query session, the server needs to execute private comparison query on all array elements to generate c and Δ. The computation complexity of this step is $6Nm\lambda = O(N)$ on the server. Moreover, engaging in the SPIR on the array I imposes $\sqrt{N} \log \sqrt{N} = O(\sqrt{N} \log N)$. Tree traversal requires $\log \sqrt{N}$ iteration of SPIR on database T that imposes total computation cost of $O(N \log^2 N)$ to the server. Therefore, the total computation complexity on the server is $O(6Nl\lambda) + O(N \log^2 N) = O(N \log^2 N)$.

Data Owner. In the preprocessing step, the data owner must construct the left-balancing binary search tree and encrypt the records. Computation complexity of tree construction and encryption is $O(N \log N)$ and $O(N)$, respectively. Moreover, the time complexity of the payload decryption is $O(1)$.

SP-SQL Protocol. According to a similar complexity analysis for the range queries, the computation complexity on the active participants and the client is $O(dm(n + N^{1-1/d}))$ and $O((n + N^{1-1/d}) \log N)$, respectively. The time complexity of SP-SQL on the server is $O((nN + N^{2-1/d}) \log N)$. For the data owner, the preprocessing step imposes $O(N \log N) + O(Nd)$ for tree construction and the database encryption.

For the exact matching queries, the computation complexity on the active participants and the client is $O(dm(n + d \log N))$ and $O(K(n + d \log N) \log N)$, respectively. The time complexity of SP-SQL on the server is $O(K(nN + dN \log N) \log N)$. For the data owner, the preprocessing step imposes $O(N \log N) + O(Nd)$ for tree construction and the database encryption. In addition, the data owner must decrypt at most one ciphertext (encrypted sum). Therefore, SP-SQL incurs $O(1)$ computation cost on the data owner in each query session.

A Comparison with Previous Works

In this section, we compare the complexity of SP-KS and SP-SQL protocols with the existing works in the literature. To be consistent with the existing works, we consider the two-party SP-KS and SP-SQL protocols where the database belongs to the server and records are stored as cleartext. In this case, the payload is encrypted by Paillier encryption using the server's public key while the keywords are encrypted by GM encryption using client's public key. Therefore, the cost of database encryption and threshold decryption is not considered. The modified two-party SP-KS protocol is secure in the presence of colluding clients. The result of comparing SP-KS with existing works is represented in Table 10. Our result indicates that SP-KS has sublinear communication overhead and it is efficient in terms of computation complexity on the client.

The comparison of SP-SQL with the naive approach is presented in Table 11. To the best of our knowledge, there is no approach for symmetrically-private SQL search over relational databases that protects the privacy of both client's query and database records. Therefore, we compare SP-SQL protocol with the naive approach explained in Section 4.3. As the comparison results show, SP-SQL achieves better communication complexity and computation complexity on the client, however the computation complexity on the server is more than the naive approach. This is reasonable, because the server is a powerful machine and is responsible for the majority of work.

Algorithm	**Computation on The Client**	**Computation on The Server**	**Communication**
SP-KS	$O(K \log^2 N)$	$O(N \log^2 N)$	$O(K \log^2 N)$
Freedman *et al.* [27]	$O(\sqrt{N})$	$O(N)$	$O(\sqrt{N})$
Ogata and Kurosawa [73]	$O(N)$	$O(N)$	$O(N)$

Table 10: *Comparison of SP-KS Protocol with Existing Works*

Algorithm	**Query Type**	**Computation on the client**	**Computation on the server**	**Communication on the client**
Basic SP-SQL	Exact matching	$O(k \log^2 N)$	$O(N \log^2 N)$	$O(kK \log^2 N)$
	Partial matching	$O\Big(k \log N(n + N^{1-s/d})\Big)$	$O(N \log N(n + N^{1-s/d})\Big)$	$O\Big(kK \log N(n + N^{1-s/d})\Big)$
	Range matching	$O\Big(k \ \log N(n + N^{1-1/d})\Big)$	$O(N \log N(n + N^{1-1/d})\Big)$	$O\Big(kK \log N(n + N^{1-1/d})\Big)$
Improved SP-SQL	Exact matching	$O(ks \log N)$	$O(k \log^2 N)$	$O(s)$
	Partial matching	$O\Big(ks(n + N^{1-s/d})\Big)$	$O\Big(k \log N(n + N^{1-s/d})\Big)$	$O(s)$
	Range matching	$O\Big(ks(n + N^{1-1/d})\Big)$	$O\Big(k \ \log N(n + N^{1-1/d})\Big)$	$O(s)$
Naive approach	All types of queries	$O(sN)$	$O(sN)$	$O(sN)$

Table 11: *Comparison of SP-SQL With The Naive Approach*

4.6 Performance Analysis

We concentrate on SP-SQL in the conducted experiments because SP-KS can be considered as a special case of SP-SQL. To evaluate the performance of SP-SQL, we implemented a prototype in Java 1.6 using the BigInteger class provided by the Java standard API. The secret shares of the GM cryptosystem are 256-bit long. Moreover, we employed the publicly available Bank Marketing dataset [107]. This dataset has 45,211 records with 17 attributes. The client's and the server's side experiments were conducted on an Intel dual Core i5 2.3GHz Notebook with 4GB RAM.

In the first experiment, we derive the total execution time of the basic SP-SQL for different types of queries as shown in Figure 12. We also compare the execution time of SP-SQL with the naive approach. The experimental result indicates that SP-SQL reduces the computation overhead on the client by 61% for the range queries execution and 44% for the execution of the partial matching queries, compared with the naive approach. We also observe that as the number of attributes in the query increases, the execution time decreases since the number of comparisons reduces.

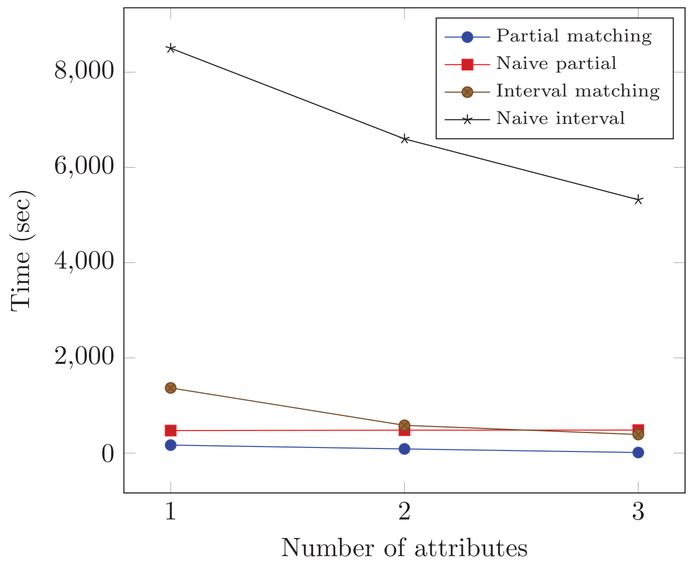


Figure 12: *The Effect of The Query Predicate Types*

For SQL queries, we vary the number of attributes in the query in the second experiment. Figure 13 plots the client CPU and the cloud server CPU. The server CPU decreases as the number of attributes increases. Similarly, the client CPU initially decreases as the number of attributes increases, because the number of comparisons and the number of visited nodes decrease. However, as the number of attributes increases, the cost of aggregating the shares of the plaintext increases. Therefore, the client CPU gradually increases.

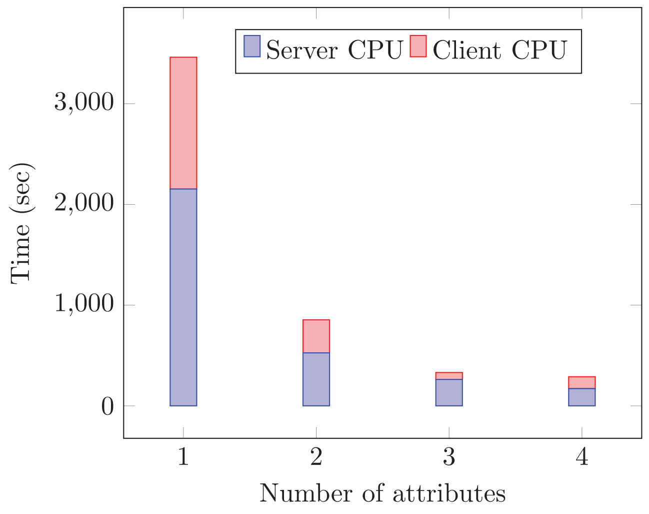


Figure 13: *Query Response Time*

The third experiment measures the effect of the database size on the query execution time on the client. We use datasets containing 400 records, 4000 records, and 40,000 records. Our experimental result, as shown in Figure 14, indicates that the basic SP-SQL would work well with small to medium size datasets (with the total number of 1,000-20,000 records) and the queries that contain multiple attributes in the predicates. Notice that financial datasets contain high-dimensional data with multiple columns. Therefore, we believe that the proposed protocols are appropriate candidates for this type of applications.

In the final experiment, shown in Figure 15, we compare the total execution time of the two approaches of SP-SQL for a 45,211-record dataset. The experiment indicates that for queries with small number of attributes, the improved approach outperforms the basic one significantly. This can be justified because the cost of SPIR, that is proportional to the number of the visited nodes, is totally removed in the second approach. However, as the number of attributes in the query predicates

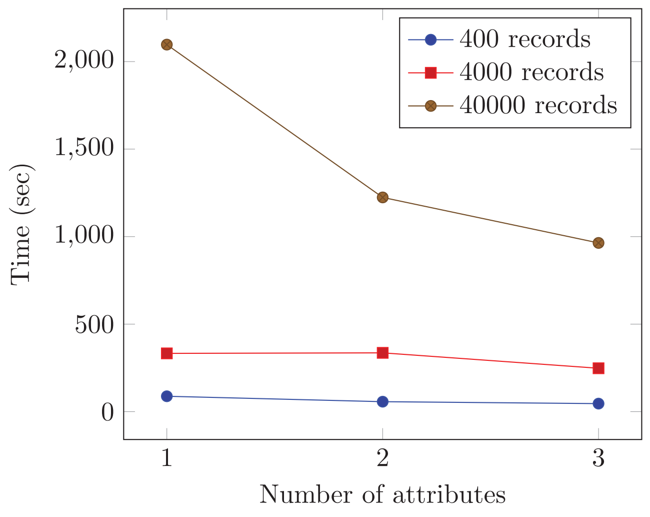


Figure 14: *The Effect of The Database Size*

increases, the number of the visited nodes decreases and the cost of SPIR would be relatively small compared to the cost of the private comparison.

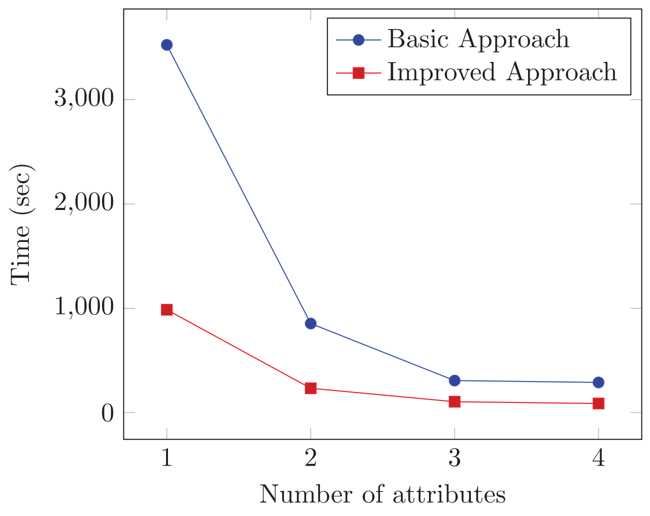


Figure 15: *Basic vs. Improved SP-SQL Approaches*

4.7 Discussion and Conclusions

In this chapter, we have provided two protocols for symmetrically-private database search that leverage symmetric private information retrieval and private integer comparison to protect the query privacy of the clients and the data privacy of the database owner. In this section, we will answer some frequently raised questions.

PIR vs. SPIR. In this chapter, we utilize SPIR protocol to retrieve the data of the root in order to protect the index of the root as well as the data of other records. Note that the database that is queried by the client using SPIR, only contains the ciphertext sequences Δ and c. These ciphertext sequences do not leak any information about the input of the database (i.e., the database record) to the client. Therefore, instead of SPIR we can use PIR which is more cache friendly; using PIR, the client can recover additional items that may be utilized later without the need of PIR for retrieval.

Semi-Honest Adversary Model. In this thesis, we assume that the adversaries are semi-honest. This is the common security definition used in the SMC literature [108]. It is realistic to assume that the client and the server are semi-honest in our problem scenario since the client and the server are collaborating to execute queries for mutual benefits: the company, providing cloud services, seeks to extend its business by building reputation and trust for its own services. On the other hand, the client is searching the database to extract useful information for her own benefit.

Individual Privacy. SP-SQL and SP-KS protocols do not preserve individual privacy. Therefore, an individual whose record exists in the database, may be identified from the query result. Note that the information that can be derived from the query result are considered to be secure. Therefore, secure multiparty computation techniques cannot solve this problem. To overcome this deficiency, the data owner can employ Privacy-Preserving Data Publishing (PPDP) [109] techniques that anonymize the raw data. The anonymized data prevent the adversary to re-identify an individual from the released database and/or the released query result.

Communication-Computation Trade-off. The efficiency of the proposed protocols highly depend on the utilized SPIR protocol. Some SPIR protocols are costly in terms of computation but efficient in terms of communication [110]. Others are costly in terms of communication but efficient in terms of computation [111]. Therefore, the choice of the underlying SPIR protocol depends on the environment. For example, consider the following two scenarios:

– **Scenario 1**. Consider a company that provides DBaaS in the cloud environment. This company hosts databases of different data owners on a single cloud server. Consequently, the cloud server may receive thousands of database queries requesting for information. In this case, if the underlying SPIR is costly in terms of computation on the server side, the cloud server will become the single point of bottleneck.

– **Scenario 2**. Consider a hospital that outsources the medical databases to a private cloud server and provides the clients with cellphone applications for querying. In this case, the computation complexity on the client and the communication overhead are the major concerns because mobile devices are limited in hardware resources and bandwidth.

As the above mentioned scenarios illustrate, an SPIR protocol should be chosen according to the efficiency requirements of the parties.

Chapter 5

Secure Healthcare Query Processing in Cloud Computing

5.1 Introduction

There have been many fatal and highly contagious diseases throughout history. The Black Death was one of the most devastating pandemics in human history, peaking in Europe in the 14th century and killing between 75 million and 200 million people [112]. Centers for Disease Control and prevention (CDC) [113] estimate that between about 8,870 and 18,300 H1N1-related deaths occurred between April 2009 and April 10, 2010. Early detection of such diseases could save millions of lives. With the vast number of people traveling around the world, an outbreak in a busy city such as New York or London could end up spreading around the world within few days. There are many organizations which work on studying epidemiology and preventing them from spreading around the world; World Health Organization (WHO) [114] is one of them.

To better understand what caused a disease, health organizations and researchers need as much data as possible about the infected patients. Therefore health organizations and researchers need to have access to the latest updated information about the patients in order to conduct epidemiological studies. Typically, a patient has many different healthcare providers including primary care physicians, specialists, therapists, hospitals and pharmacies. As a result, a patient's Electronic Health Records (EHRs) [115] is usually scattered throughout the entire healthcare sectors. From the clinical perspective, in order to deliver quality patient care, it is critical to access the integrated information. Therefore, sharing Electronic Health Records (EHRs) is one of the key requirements in healthcare domain for delivering high quality healthcare services [116]. However, the sharing process could be very complex and involved with various entities with different duties and objectives. A shared EHR may consist of sensitive information of the patients such as demographic details, allergy information, medical histories, and laboratory test results. Access control solutions must be in place to guarantee that access to sensitive information is limited only to those entities that have a legitimate privilege, allowed by the patients. For example, a patient may not be willing to share his medical information regarding an HIV/AIDS diagnosis with a dentist unless a specific treatment is required. Therefore, it is important to address security challenges such as data confidentiality and access control [117].

It has become a recent trend for the patients to take these matters into their own hands by managing their records using a Personal Health Record (PHR) system. PHR is a patient-centric model of managing health information that allows a patient to create, manage, and control her personal health data in a centralized place through the web. The patients have the full control of their medical records and can share their health and fitness data with a wide range of users of their choice, including healthcare providers, their family members and insurance companies. In the past few years, many providers have created platforms to manage PHRs with features including flexible access control, mobile access, and complex automated diagnoses that analyze the patients records and alert them when a preventive checkup is needed. These providers include Microsoft HealthVault [118] and Dossia [119].

Recently, architectures of storing PHRs in cloud have been proposed [120]. The main concern about these services is the privacy and the security of patients' personal health data. Since the health records are stored on a third-party provider, the patients will eventually lose the control of their data and the data will be under the control of the servers. Therefore, the PHR data could be leaked if an insider in the cloud provider's organization misbehaves. As a famous incident, a Department of Veterans Affairs database containing personal health information of 26.5 million military veterans, including their social security numbers and health conditions was stolen by an employee who took the data home without authorization [121]. For these reasons, researchers have begun searching for a way to allow patients to store their medical records on the cloud using a Database-as-a-Service (DBaaS) model while preserving their privacy. Li *et al.* [122] have suggested Attribute-Based Encryption (ABE) as a solution to secure the stored medical records. ABE is utilized to encrypt and store the PHR data on semi-trusted servers, so that patients as well as various users from public domains with different professional roles, can have controlled access to PHRs.

To produce statistical information about health records, patients can give access to health organizations. According to a report from the consulting firm PwC [123], health organizations are falling short in protecting the privacy and security of patient information [124]. Additionally, according to the same report [123], more than half of health organizations said they had at least one issue with information security and privacy since 2009 and the most frequently observed issue is the improper use of protected health information by someone who worked in the organization.

In this chapter, we propose a solution that allows the health organization to produce statistical information about encrypted PHRs stored in the cloud. In addition, the proposed solution should not enable the patients to infer about what the health organizations are concerned about in order to not create panic about epidemics in the community.

5.2 Threat Model

In this section, we first identify the involved entities and the privacy objectives. Then the threat model and the assumptions underlying the system design will be presented.

5.2.1 Entities Involved in The Protocol

There are three main entities in the system as illustrated in Fig. 16:

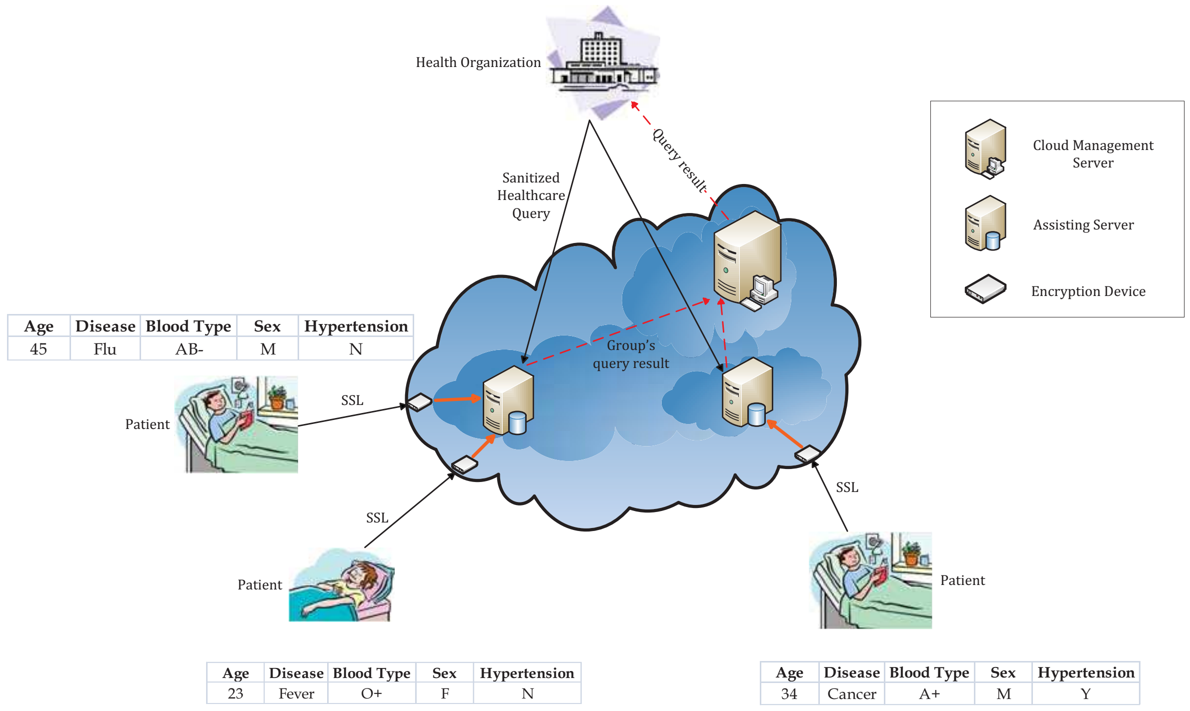
– *Patients* who own medical records and want to store them on the cloud server and protect their confidentiality. Note that the security of these records depends on the security of handling the plaintext data before it arrives to the cloud server and the the security of data-at-rest while storing them on the cloud. For this reason, when patient data is initially uploaded from patients or doctors, it will arrive at the cloud server via a secure channel (e.g. SSL). However, this encryption only protects the data while it is in transit. After the data arrives on the cloud server, it is delivered as plaintext to the cloud server. A common way to protect the medical record of the patients from being leaked to the cloud server is through encryption. For this reason, the necessary keys for encryption will be resident in the encryption device on the cloud when the data is encrypted. These devices are designed in such a way that, after a patient places a key into the key memory of the device, it cannot be read externally [125]. To generate the keys, the patients will be organized in smaller groups and jointly generate one key for the threshold Paillier cryptosystem and one key for the threshold

GM cryptosystem together with the share of the secret keys. The public keys and the share of the secret keys will be stored on the encryption devices. The devices encrypt the received plaintext using these keys and send the resulted ciphertexts to the assisting server (see below).

– *Cloud server* that stores the encrypted health records of the patients and executes the queries of the health organization over the encrypted records. The cloud server should enable the patients to access and update their records if required. The cloud server will assign an assisting server to each group. The assisting server will be responsible for storing the encrypted record of the patients and executing the SQL queries of the health organization.

– *Health organization* that wants to execute queries over the encrypted database of the patients and produce statistical information.

The patients are assumed to behave properly, but they may try to derive information from the queries issued by the health organization. Similar to [10], we assume that the shape or textual content of SQL queries is not private, but the constants provided by the health organization in the WHERE clause are private, and must be protected. Our approach to preserve query privacy over a relational database is based on hiding all these constants. The cloud server is trusted in the sense that it will execute the received requests correctly and it does not temper the patients' medical records (inadvertently or deliberately); however we do not rely on it to maintain the data confidentiality. In other words, the server is modeled as "honest but curious" in our trust model.



Age	Disease	Blood Type	Sex	Hypertension
45	Flu	AB-	M	N

Age	Disease	Blood Type	Sex	Hypertension
23	Fever	O+	F	N

Age	Disease	Blood Type	Sex	Hypertension
34	Cancer	A+	M	Y

Figure 16: *Architecture*

5.2.2 Assumptions and Threat Model

The threat model that we consider is one where an adversarial entity controls some subset of the parties and wishes to attack the protocol execution. The parties under the control of the adversary are called *corrupted*, and follow the adversary's instructions. We assume that there is no trusted entity in the environment and all parties act as semi-honest adversaries.

We assume that the number of corrupted patients is less than a specific threshold, denoted by k. In our scheme, if the assisting server and at least k patients, who have the shares of the secret key, collude they can recover the secret key and decrypt the constants in the query predicate. We also assume that there are mechanisms which ensure integrity and availability of the remotely stored data. Our scheme focuses only on confidentiality issues and does not provide protection against attacks such as data tampering and denial of service. Our scheme is a building block that can be integrated into larger more comprehensive frameworks for securing database on untrusted cloud servers.

5.3 Secure Distributed Techniques

In this section, we present distributed techniques that can be used to support secure distributed computation. Assume that there exists n parties and the input of the i-th party is the value v_i. $1 \leq i \leq n$. The protocols presented in this section aim to calculate the sum, the maximum and the minimum of v_i's as input. At the end of the protocol execution, the parties do not learn anything except the result. The proposed protocols assume that the parties are not malicious and they correctly carry out the prescribed functions.

5.3.1 Secure Distributed Sum

Distributed algorithms frequently require the sum of inputs from individual parties. Our approach to securely find the sum of the parties' input is based on the traditional ring-based approach [126] where the parties form a ring and messages are forwarded in a pre-defined direction. A master party P_1 is elected and starts the computation by sending $v = v_1 + r$ (for a random r) to its neighbor, which adds its own input to v and forwards the result along. Once arrived back at the master, the final sum is obtained as $s = -r + v$ and is broadcast to other parties. Traditional ring-based approach is susceptible to the attack from the colluding parties [127], since if the party $i - 1$ and the party $i + 1$ collude, they can recover the input of the i-th party. Numerous improvements have been proposed to defeat against the collusion attack such as permuting the path after each execution [127], sharing v_i's between the parties (the same source [127]) and distributing the shares of the random number r between the parties [128].

Our approach utilizes randomization as well as threshold encryption to prevent collusion attack. In this case, our proposed approach is secure against the colluding parties as long as the master (see below) does not collude. Assuming n parties ($n \geq 3$) and a non-colluding master, the following method securely computes sum.

One party is chosen as master, named P_1. The remaining parties are numbered $P_2, \ldots, P_n$. The master is responsible for initiating the secure distributed sum protocol. The parties execute distributed key generation algorithm [48] to obtain the public key and the shares of secret key for the threshold Paillier cryptosystem. The master then generates a random number R, adds its input to this number and encrypts the resulted sum by the group public key using Paillier encryption. Then, the master forwards the ciphertext to P_2. The i-th party P_i ($2 \leq i \leq n$) receives $E(R + \Sigma_{j=1}^{i-1} v_j)$. Since this ciphertext is encrypted with the group public key, it cannot decrypt it individually and learn anything. Party P_i then encrypts its input v_i by the group public key and multiplies $E(v_i)$ by the encrypted sum to obtain $E(R + \Sigma_{j=1}^{i} v_j)$. Then, it passes the encrypted sum to the $i + 1$-th party. The n-th party performs the above step and passes the result to the master. The master must decrypt the result to obtain the actual sum. To decrypt the query result that is encrypted with the group public key, the master contacts the parties and sends them the ciphertext for the decryption. These parties partially decrypt the encrypted sum using their share of secret key and sends their share of plaintext to the master. The master then recovers the noisy sum by combining the shares of plaintext and extract the noise R to find the actual sum. Then it broadcasts the actual query result to the other parties. The steps of Secure Distributed Sum is presented in Algorithm 1.

The proposed approach is secure while the master is not colluding with the other colluding parties. In this case, even if all the parties that are asked to perform decryption, are corrupted and collude with each other they can only find out the noisy sum. But, if the contacted parties and parties $i-1$ and $i+1$ collude, they can find out the input of the i-th party.

Algorithm 1 Secure Distributed Sum

Require: n parties $P_1, P_2, \ldots, P_n$, each party P_i has a local input v_i. **Output:** $\Sigma_{i=1}^{n} v_i$.

```
 1: $P_1$ is acting as a master.
 2: $P_1, \ldots, P_n$ executes distributed key generation algorithm. Each party $P_i$ obtains
    the group public key $pk$ and the secret key $sk_i$.
 3: $P_1$ generates random number $R$, add it to $v_1$ and encrypts $v_1 + R$ by Paillier
    cryptosystem using $pk$ to obtain $c_1 = E_{PL}(v_1 + R, pk)$.
 4: $P_1$ forwards $c_1$.
 5: for Each party $i$ $(2 \leq i \leq n)$ do
 6:     $P_i$ receives the message $c_{i-1}$ from $P_{i-1}$;
 7:     $P_i$ encrypts $v_i$ by $pk$;
 8:     $P_i$ computes $c_i = c_{i-1}.E_{PL}(v_i, pk)$;
 9:     $P_i$ forwards $c_i$ to $P_{(i+1) \bmod n}$.
10: end for
11: $P_1$ receives the message $c_n = E(R + \Sigma_{i=1}^{n} v_i, pk)$ from $P_n$.
12: $P_1$ sends $c_n$ to $k'$ parties for decryption.
13: $P_1$ aggregates the ciphertexts from the $k'$ parties and obtains $R + \Sigma_{i=1}^{n} v_i$.
14: $P_1$ calculates $-R + R + \Sigma_{i=1}^{n} v_i = \Sigma_{i=1}^{n} v_i$ to obtain actual sum.
15: $P_1$ broadcast the actual sum to the parties $P_2, \ldots, P_n$.
```

5.3.2 Secure Distributed Maximum/Minimum

Consider several parties having their own input. The problem is to securely compute the maximum and the minimum of these local inputs. Formally, given n parties $P_1, \ldots, P_n$, having local inputs $v_1, \ldots, v_n$. We wish to securely compute $\texttt{max}\{v_1, v_2, \ldots, v_n\}$ and $\texttt{min}\{v_1, v_2, \ldots, v_n\}$. To calculate the maximum and the minimum, the private comparison protocol and the threshold GM cryptosystem are used. We explain the technique for calculating the maximum; distributed minimum function can be securely computed in a similar way.

Parties jointly generate a group public key PK for the k-out-of-n threshold GM cryptosystem such that all parties obtain the shares of the secret key [24]. One party is chosen as the master, numbered 1. The master encrypts her local input, bit by bit, using the group public key and forwards it to the party P_2 as the current maximum. Each party P_i, upon receiving the message from P_{i-1}, encrypts her local input v_i by the group public key and executes Fischlin protocol, given the current encrypted maximum in the message and her encrypted local value as the inputs. The output of Fischlin protocol is an encrypted ciphertext sequence Δ that indicates if the current maximum is greater than the v_i or not. To decrypt the sequence Δ, P_i contacts with k parties and sends the generated Δ for decryption. Each party calculates the share of the plaintext and sends it back to P_i. Afterwards, P_i constructs the plaintext from the received shares: if the decrypted Δ contains a sequence of λ 1s, it means that the current maximum is greater than v_i; therefore P_i does not modify the received message and forwards it to P_{i+1}, as it is. Otherwise, v_i is the current maximum and P_i must encrypt it, bit by bit, using the group public key and forwards it to P_{i+1}. At the end of query result forwarding, the master obtains the encrypted maximum and decrypts it bit by bit, by communicating with k randomly-chosen parties. Finally, the master broadcasts the maximum to the other parties. The algorithm for secure maximum is represented in Algorithm 2. Distributed maximum/minimum protocol is secure if the number of semi-honest parties controlled by the adversary is at most equal to $k-1$.

The proposed distributed protocols require that all parties to be online for executing the protocol. However, each party can decide if she is willing to participate in the query execution or not.

Remark 1. The proposed secure maximum/minimum protocol illustrates how a party can perform comparison on two ciphertexts without knowing the secret key. This idea can be extended to enable a party to sort a sequence of ciphertexts without

Algorithm 2 Secure Distributed Maximum

Require: n parties $P_1, P_2, \ldots, P_n$, each party P_i has a local input v_i with the bit length l. **Output:** $\texttt{max}\{v_i\}$.

```
1: $P_1$, $P_2, \ldots,$ $P_n$ executes the distributed key generation to produce the group
   public key $PK$ and shares of secret key $SK_1, SK_2, \ldots, SK_n$.
2: $SK_i$ will be assigned to $P_i$.
3: $P_1$ is acting as a master.
4: $P_1$ encrypts $v_1 = (v_{1,l}, \ldots, v_{1,1})_2$ by GM encryption using $PK$ to obtain $c_1 =$
   $\{E_{GM}(v_{1,l}), \ldots, E_{GM}(v_{1,1})\}$.
5: $P_1$ forwards $c_1$ to $P_2$.
6: for Each party $P_i$ $(2 \leq i \leq n)$ do
7:    receives the message $c_{i-1}$ from $P_{i-1}$;
8:    encrypts $v_i$, bit by bit by $PK$ to obtain the set $\{E_{GM}(v_{i,l}), \ldots, E_{GM}(v_{i,1})\}$;
9:    executes Fischlin protocol where inputs are $c_{i-1}$ and $\{E_{GM}(v_{i,l}), \ldots, E_{GM}(v_{i,1})\}$
      to obtain an encrypted ciphertext sequence $\Delta$;
10:   $P_i$ forwards $\Delta$ to $k$ randomly-selected parties $P'_1, \ldots, P'_k$.
11:   for Each party $j$ $(1 \leq j \leq k)$ do
12:      decrypts $\Delta$ with her share of secret key $SK_j$.
13:      sends decrypted ciphertext sequence to $P_i$.
14:   end for
15:   receives decrypted $\Delta$ from $P'_1, \ldots, P'_k$
16:   extracts the actual ciphertext sequence from the received plaintexts.
17:   if there is a sequence of $\lambda$ 1s in the random position of the decrypted ciphertext
      sequence then
18:      // the current maximum is greater than $v_i$
19:      $c_i = c_{i-1}$
20:   else
21:      // the current maximum is not greater than $v_i$
22:      $c_i = E_{GM}(v_i)$
23:   end if
24:   sends $c_i$ to party $P_{(i+1) \text{ mod n}}$
25: end for
26: $P_1$ receives the message $c_n$ from $P_n$.
27: $P_1$ communicates with $k$ randomly-selected parties and send them $c_n$.
28: The parties $P'_1, \ldots, P'_k$ decrypts $c_n$ to obtain $m_1, \ldots, m_k$.
29: $P'_j$ sends $m_j$ to $P_1$ where $1 \leq j \leq k'$.
30: $P_1$ combines $m_1, \ldots, m_k$ to obtain $\texttt{max}\{v_i\}$ where $1 \leq i \leq n$.
31: $P_1$ broadcasts $\texttt{max}\{v_i\}$ $(1 \leq i \leq n)$ to $P_2, \ldots, P_n$.
```

knowing the secret key. In this case, any comparison-based sorting algorithm can be utilized and the comparison is performed on the encrypted values, using any private integer comparison protocols.

5.4 Secure Healthcare Query Processing in Cloud Computing

In this section, we present a protocol that allows the patients to store their medical record on the cloud server. Patients' information is stored in a database and mined for statistical information by the health organization. The proposed solution should protect the data privacy of the patients in such a way that the cloud server and the health organization do not learn anything about the sensitive information of the patients. Moreover, the patients and the cloud server should not be able to infer anything about the constants in the queries of the health organization.

The inputs to the protocol is as follow: the health organization provides an aggregate SQL query that consists of exact-matching and interval-matching predicates combined with logical operators (AND/OR/NOT). The cloud server's input is the encrypted health records of the patients. The cloud server is responsible for executing the SQL queries such that the privacy goals of the patients and the health organization are reached.

The naive approach to achieve the mentioned privacy objectives is that the health organization communicates with each patient and securely evaluates its queries on the patients' record. This can be achieved by exploiting the Fischlin's protocol for private comparison. However, this approach incurs excessive communication and computation overhead on the health organization side which is linear in the number of patients. To reduce the overhead, the patients are organized into smaller groups.

The patients in each group jointly generate two public keys for Goldwasser-Micali [42] and Paillier [26] encryption schemes. Then they encrypt their records and outsource them to the cloud server for storage. The cloud server assigns an assisting server to each group which is responsible for receiving SQL query of the health organization and securely executing it on the records of the patients to obtain the partial results. The assisting servers then collaborate to obtain the final query result from the partial results and report it to the health organization.

In the following, we elaborate the basic steps of our protocol that protects the data privacy of the patients and the query privacy of the health organization.

5.4.1 Setup and Tree Construction

The cloud server defines $L = \lfloor \sqrt{N} \rfloor$ groups where N is the total number of patients. It then randomly maps each patient into exactly one group. Let $n = \lceil \frac{N}{L} \rceil$ denotes the number of patients in each group. The cloud server assigns an assisting server to each group which is responsible for executing the health organization's queries over the medical database of the patients. The assisting servers also collaborate with each other to obtain the query result from the partial results and send it to the health organization.

Note that if each patient encrypts her record with her own unique public key, the health organization needs to generate one query per patient. In this case, the computation and communication overhead on the health organization will be similar to the naive approach. To encrypt the patients' records with a single key and also protect the records from the other patients in the same group, we utilize the threshold GM [24] and threshold Paillier [48] cryptosystem without the trusted dealer.

In the i-th group, the patients execute the distributed key generation algorithm for the threshold Paillier and the threshold GM cryptosystems to obtain jointly-generated public keys pk'_i and pk_i for Paillier and GM cryptosystems, respectively together with their share of secret keys. Each patient then stores the group public keys and her private keys on a Field-Programmable Logic Array (FPGA) as detailed in [125]. The FPGAs are programmed to form an independent semi-trusted third party platform within the cloud infrastructure. Since these devices run as autonomous compute elements, the cloud administrator does not have low-level access to computations running within them. These FPGAs are designed in such a way that, after a patient places a key into the key memory on the device, the key cannot be read externally. The FPGAs is then delivered to the cloud operator for installation. Note that decrypting a ciphertext by the cloud server is performed by sending a ciphertext to the FPGA of the patients. Since the FPGAs have the share of secret key, they can decrypt the ciphertext partially and sends it back to the sender.

The threshold cryptosystems enables the patients to encrypt their record with a single public key while at least a minimal number of patients are required to decrypt a ciphertext. All the patients in each group uploads their medical records through their FPGA on the cloud server. Note that the medical records are multidimensional data and the FPGA encrypts them using Paillier and GM cryptosystems by the associated group public key. Therefore, the encrypted record of each patient, produced by the FPGA, has two columns for each attribute in the database: one column that contains the encryption of the attribute value using the group public key for threshold Paillier cryptosystem, and another column that stores the GM encryption of the attribute value using the group public key for threshold GM cryptosystem.

Example 8. *Consider the health records with the attributes* Age *and* Surgery*, where the value of the attribute* Surgery *specifies the type of the surgery that the patient undergoes (e.g. 1: Transgender, 2: Plastic, 3:Vascular, 4: Urology). Assume that* *the patients, whose records are represented in Table 12, are willing to outsource their health records to the cloud server. The total number of patients is $N = 10$; therefore, these patients must be organized in $L = \sqrt{10} \approx 3$ groups, namely, G_1, G_2 and G_3. Assume that the patients 1,9 and 10 are assigned to G_1; patients 2, 4, 5 and 8 are assigned to G_2 and patients 3, 6 and 7 are assigned to G_3. The assignments are performed randomly. The patients in the group G_i jointly generate the public key pk_i for the threshold GM cryptosystem. Moreover, the patients in the group G_i jointly generate the public key pk'_i for for the threshold Paillier cryptosystem and publish pk'_i to the health organization. The members of the group G_i encrypt each attribute value of their records with the threshold GM cryptosystem using pk_i and the threshold Paillier cryptosystem using pk'_i as shown in Fig. 17. After that, the patients outsource their encrypted records to the cloud server.*

	Age	**Surgery**
Patient 1	34	1
Patient 2	39	2
Patient 3	20	1
Patient 4	59	3
Patient 5	63	4
Patient 6	27	2
Patient 7	78	4
Patient 8	11	2
Patient 9	83	3
Patient 10	42	3

Table 12: *Health Records*

The assisting server collects the encrypted records and organizes them as a kD-tree using the algorithm explained in Section 2.3. Constructing the tree requires sorting the records at each level and dividing the database into two subsets by calculating the number of nodes in the left subtree (LT) and the right subtree (RT). Since the records are encrypted, the sorting algorithm must be executed on the ciphertexts.

	Age$_{GM}$	**Surgery**$_{GM}$	**Age**$_P$	**Surgery**$_P$
Patient 1	$E_{pk_1}(34)$	$E_{pk_1}(1)$	$E_{pk'_1}(34)$	$E_{pk'_1}(1)$
Patient 9	$E_{pk_1}(83)$	$E_{pk_1}(3)$	$E_{pk'_1}(83)$	$E_{pk'_1}(3)$
Patient 10	$E_{pk_1}(42)$	$E_{pk_1}(3)$	$E_{pk'_1}(42)$	$E_{pk'_1}(3)$

(a) *G_1 Database*

	Age$_{GM}$	**Surgery**$_{GM}$	**Age**$_P$	**Surgery**$_P$
Patient 2	$E_{pk_2}(39)$	$E_{pk_2}(2)$	$E_{pk'_2}(39)$	$E_{pk'_2}(2)$
Patient 4	$E_{pk_2}(59)$	$E_{pk_2}(3)$	$E_{pk'_2}(59)$	$E_{pk'_2}(3)$
Patient 5	$E_{pk_2}(63)$	$E_{pk_2}(4)$	$E_{pk'_2}(63)$	$E_{pk'_2}(4)$
Patient 8	$E_{pk_2}(11)$	$E_{pk_2}(2)$	$E_{pk'_2}(11)$	$E_{pk'_2}(2)$

(b) *G_2 Database*

	Age$_{GM}$	**Surgery**$_{GM}$	**Age**$_P$	**Surgery**$_P$
Patient 3	$E_{pk_3}(20)$	$E_{pk_3}(1)$	$E_{pk'_3}(20)$	$E_{pk'_3}(1)$
Patient 6	$E_{pk_3}(27)$	$E_{pk_3}(2)$	$E_{pk'_3}(27)$	$E_{pk'_3}(2)$
Patient 7	$E_{pk_3}(78)$	$E_{pk_3}(4)$	$E_{pk'_3}(78)$	$E_{pk'_3}(4)$

(c) *G_3 Database*

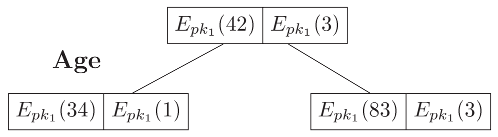
Figure 17: *Outsourced Health Records in Groups*

The sorting algorithm on the ciphertexts has been described before in Remark 1 (Section 5.3) as an extension of secure maximum/minimum. Therefore, constructing the left-balancing kD-tree from the ciphertexts can be performed by the assisting server even though it does not have the decryption key. The kD-tree has the advantage of reducing the number of comparisons, required for the query execution.

Example 9. *(Continued from Example 8) The generated kD-trees for each group are shown in Fig. 18. The partitioning attributes in each group may be different. At the first level,* Age *is used to partition the records of G_1 and G_2 whereas* Surgery *is used to partition the records of G_3.*

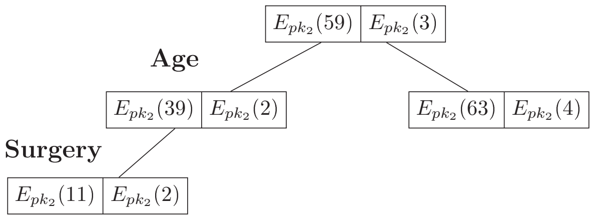
5.4.2 Query Sanitization

The health organization needs to execute a query such that the constants in the query predicate are not revealed to the patients and the cloud server. Accordingly, the health organization sanitizes its SQL query by replacing the constants contained



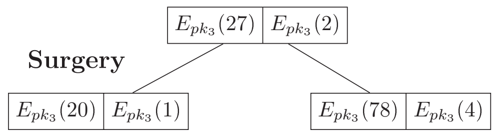
$E_{pk_1}(42)$ $E_{pk_1}(3)$
Age
$E_{pk_1}(34)$ $E_{pk_1}(1)$
$E_{pk_1}(83)$ $E_{pk_1}(3)$

(a) *G_1 kD tree*



$E_{pk_2}(59)$ $E_{pk_2}(3)$
Age
$E_{pk_2}(39)$ $E_{pk_2}(2)$
$E_{pk_2}(63)$ $E_{pk_2}(4)$
Surgery
$E_{pk_2}(11)$ $E_{pk_2}(2)$

(b) *G_2 kD tree*



$E_{pk_3}(27)$ $E_{pk_3}(2)$
Surgery
$E_{pk_3}(20)$ $E_{pk_3}(1)$
$E_{pk_3}(78)$ $E_{pk_3}(4)$

(c) *G_3 kD tree*

Figure 18: *Generated kD-Trees*

in the predicates by their GM encryption using the public key of each group. For instance, if the query of the health organization is

$$\texttt{SELECT MAX(Age) FROM } D \texttt{ WHERE Surgery} = 1$$

The sanitized query that is forwarded to the i-th group $(1 \leq i \leq n)$ will be

$$\texttt{SELECT MAX(Age) FROM } D \texttt{ WHERE Surgery} = E_{pk_i}(1)$$

In addition to the sanitized query, the health organization generates a token for each group that is encrypted by the group public key using the Paillier cryptosystem. The encrypted token is a random number which is manipulated by the assisting servers to produce the noisy query result. Generating the token depends on the type of the aggregate function in the query; for *count* and *sum* functions, the health organization generates a random number R and produces L additive shares of R, namely $R_1, R_2, \ldots, R_L$ such that $R = R_1 + R_2 + \ldots + R_L$. The random share R_i

will be the token that is sent to the assisting server of the group i. For the *max* and *min* functions, the health organization populates the same random numbers R, as the token, for all groups. The health organization then encrypts the token of each group using Paillier cryptosystem by the group public key.

The health organization forwards the sanitized query together with the encrypted token to assisting servers. Therefore, in this step the health organization should create L sanitized queries and L encrypted tokens.

5.4.3 Tree Traversal and Query Execution

The health organization forwards the sanitized SQL query and the encrypted token to the assisting servers for execution. To execute the query of the health organization, the assisting server must traverse the kD-tree, constructed from the encrypted records of the patients. To do so, the assisting servers follow the tree traversal algorithm explained in Chapter 4 without the need of the symmetrically-private information retrieval (SPIR) for retrieving root's record. The search starts from the root; the assisting server uses Fischlin protocol and the threshold GM decryption to evaluate the query predicate. Based on the result of the query evaluation, the search is continued on the left tree or the right subtree or both. Therefore, at the end of this step, the assisting servers will end up with the records that satisfy the query predicate. The assisting servers then compute the encrypted query result depending on the type of the aggregate function as follows:

– *count*: The assisting server of the group G_i counts the number of records that are reported as the query result and encrypts this value using the Paillier cryptosystem with the group public key.

– *sum*: In the beginning, the assisting servers encrypt 0 (as the current sum) by the Paillier encryption using the group public key. In the tree traversal step, if at each level the conditions in the query predicate are satisfied, the assisting server projects the record over the Paillier-encrypted column targeted by the aggregate function and multiplies it by the the current sum to update the query result. At the end, the assisting server will end up with the sum that is encrypted with the group public key using the threshold Paillier cryptosystem.

– *max (resp. min)*: Initially, the assisting servers pick up a small negative (resp. large positive) number that denotes the current maximum (minimum) and encrypts it by the GM and the Paillier cryptosystems using the group public key. GM-encrypted ciphertext is utilized for the comparison while Paillier-encrypted ciphertext is used for generating noisy query result. During the tree traversal if a record satisfies the query condition(s), the assisting server projects the record over the columns that contain the GM- and the Paillier-encryption of the record. It then executes Fischlin protocol using the encrypted current maximum (resp. minimum) and the GM-encryption of the record, to find out if this record has greater (resp. smaller) value or not. If so, the assisting server initializes the current maximum (resp. minimum) to the GM- and the Paillier-encrypted ciphertexts. Otherwise, the current maximum (resp. minimum) remains unchanged. At the end, the assisting servers end up with the query result encrypted with the Paillier and GM encryption. For the remaining step of the protocol, the assisting servers only need the Paillier-encrypted ciphertext.

At the end of this step, the assisting servers obtain the partial query result (that has been encrypted using Paillier scheme by the public key of the group), derived from the database of the group.

Example 10. *(Continued from Example 8) Consider the kD-trees presented in Fig. 18 and the sanitized query*

$$\texttt{SELECT MAX(Age) FROM } D \texttt{ WHERE } \mathsf{Surgery} = E_{pk_i}(1) \text{ where } 1 \leq i \leq L$$

All assisting servers receive an encrypted token $E_{pk'_i}(R)$ from the health organization. The assisting server of the group G_i extracts $E_{pk_i}(1)$ from the query and performs the point search on the kD-tree constructed by the patients in the group G_i. The assisting servers report the records that satisfy the predicate $\mathsf{Surgery} = E_{pk_i}(1)$ by executing the Fischlin protocol and the threshold GM cryptosystem to decrypt the output ciphertext sequence. The record of the Patient 1 in G_1 and the record of the Patient 3 in G_3 satisfy the predicate. Therefore, the output of the tree traversal for assisting server of groups G_1, G_2 and G_3 will be $\{E_{pk_1}(34), E_{pk'_1}(34)\}$, $\{E_{pk_2}(-1000), E_{pk'_2}(-1000)\}$ and $\{E_{pk_3}(20), E_{pk'_3}(20)\}$, respectively. The resulted outputs will be projected over the column $\mathbf{Age}_P$ to obtain $\{E_{pk'_1}(34)\}$, $\{E_{pk'_2}(-1000)\}$ and $\{E_{pk'_3}(20)\}$ as the encrypted query result.

5.4.4 Query Result Decryption

So far, the assisting servers were able to obtain the encrypted partial query result. Therefore, the assisting servers must collaborate with each other to compute the final query result and submit it to the health organization.

The partial query results of the groups are encrypted with different keys. Therefore, in order to compute the final query result, the partial query result must be in the cleartext. Since, the assisting servers are not willing to reveal the query result to each other, the assisting servers first obfuscate the partial query result. The obfuscation is performed by the mean of multiplying the encrypted query result by the encrypted token, sent by the health organization (both of them are ciphertexts generated by the same key under the Paillier cryptosystem). The obfuscation allows the assisting servers to collaborate with each other to calculate the noisy query result while hiding the actual query result of their group. In addition, since the noise is generated by the health organization, it allows the health organization to recover the actual query result from the noisy query result. To produce the noisy query result, all assisting servers multiply their encrypted token (received from the health organization) by the Paillier-encrypted query result to generate encrypted noisy query result.

Afterwards, all assisting servers decrypt the resulted noisy query result – that is encrypted by the Paillier cryptosystem – by contacting patients in their group. The assisting servers then need to obtain the final noisy query result by aggregating their partial noisy result. To do so, the assisting servers send the noisy query result in plaintext to the cloud server. The cloud server then aggregates the partial noisy query result to obtain the noisy query result; in the case of *count* and *sum*, the cloud server adds up all the partial results and submits it to the health organization. In the case of *max* and *min* aggregate functions, the cloud server executes maximum/minimum algorithm on the plaintext and sends the resulted value to the health organization. Note that the noise generated for obfuscating the maximum/minimum of all groups is the same, therefore it will not affect the algorithm correctness (i.e., if $a < b$ then $a + R < b + R$). Finally, the health organization in its turn subtracts the noise and obtains the query result.

Example 11. *(Continued from Example 10) We have seen that the result of executing the SQL query*

$$\texttt{SELECT MAX(}\mathsf{Age}\texttt{) FROM } D \texttt{ WHERE } \mathsf{Surgery} = 1$$

on the groups G_1, G_2 and G_3 was $\{E_{pk'_1}(34)\}$, $\{E_{pk'_2}(-1000)\}$ and $\{E_{pk'_3}(20)\}$ respectively. Moreover, the token sent by the health organization to the i-th group is $E_{pk'_i}(R)$. The assisting servers multiply the received token $E^R_{pk'_i}$ by all records in the encrypted query result to obtain the encrypted noisy query result, i.e., $\{E_{pk'_1}(34+R)\}$, $\{E_{pk'_2}(-1000 + R)\}$ and $\{E_{pk'_3}(20 + R)\}$. The assisting servers then decrypt these *ciphertexts to obtain* $34 + R$*,* $-1000 + R$ *and* $20 + R$*. They send their noisy plaintexts to the cloud server. The cloud server executes the maximum algorithm on the* $34 + R$*,* $-1000 + R$ *and* $20 + R$ *and eventually ends up with* $34 + R$ *as the maximum. The cloud server forwards* $34 + R$ *to the health organization. The health organization subtracts the noise* R *to obtain* 34 *as the result of executing the SQL query on the medical database.*

5.5 Security and Complexity Analysis

In this section, we discuss the security analysis of the proposed protocol for executing healthcare queries. We will also provide the complexity analysis and a discussion on the limitation of this work.

5.5.1 Possible Attacks and Mitigations

The proposed protocol is secure while the parties, namely the health organization, the cloud server and the patients, do not collude. The main concern with threshold cryptosystems comes from the collusion attack. Any collusion that contains less than k patients in each group cannot learn any information about the ciphertext sequences Δ and c generated for comparison as well as constants in the query of the health organization. The most serious collusion attacks are when (i) the cloud server colludes with more than k patients in each group to recover the encrypted database records, (ii) the cloud server and at least k patients in a group collude to infer constants in the query submitted by the health organization.

However in practice, we can lower the risk to an acceptable level by increasing the threshold k such that the attackers are not able to compromise too many patients. Despite simplicity, this mechanism has two disadvantages: First, it will decrease the system's availability: as the number of required online data owners increases, it is more unlikely that they are online to perform decryption. Second, it will increase the communication cost on the party who is searching the database because she needs to communicate with more parties for decryption. Therefore, there should be a trade-off between availability-security and efficiency by choosing a proper value for k.

5.5.2 Complexity Analysis

Let N denotes the number of patients in the PHR system. These patients are organized in $L \approx \sqrt{N}$ groups and each group contains $n = N/L$ patients. We assume that k denotes the threshold of GM decryption where $1 \leq k \leq n$. Moreover, we assume that the number of bits required to store a value in the attribute domain, is l. For the range query and the point query, the total number of comparisons on a kD-tree with $\sqrt{N}$ records, will be $O(N^{0.5-1/2d})$ and $O(\log N)$, respectively where d is the number of attributes in the table [62]. Recall that for the Fischlin protocol, the communication overhead is $O(\lambda.l)$ and the computation cost on the client and the server is $O(\lambda)$ and $O(\lambda l)$, respectively.

For the sake of brevity, we only consider the cost of the steps that have major effect on the performance of the protocol.

Storage Overhead. For each column in the medical database with d columns, the cloud server must stores two columns: one column that contains the Paillier encryption and one that stores the GM encryption of the attribute value of the record. Thus, the table on the cloud server has $2d$ columns.

Communication Overhead. The communication complexity of the protocol on the health organization and the assisting servers is as follow:

Health organization. The health organization needs to communicate with the assisting servers to send the sanitized query and receive the final result from the cloud server. Therefore, the total communication cost is $O(L) = O(\sqrt{N})$.

Assisting servers. The assisting servers should traverse the tree to execute the SQL query. The communication complexity of the protocol depends on the number of comparisons and consequently on the type of predicates in the query:

– Range queries. For each comparison performed in tree traversal, the assisting server must communicate with the patients' HSMs to decrypt ciphertext sequences Δ and c, each has $\lambda.l$ ciphertexts. Therefore, the communication overhead for a range query will be $O(k\lambda l\ N^{0.5-1/2d})$ on each assisting server. Since $\sqrt{N}$ assisting servers reside on the cloud server, the total communication cost of protocol execution will be $O(k\lambda l\ N^{1-1/2d})$.

– Point queries. According to a similar discussion, the communication overhead of executing an exact-matching query will be $O(kl\lambda \log N)$ on each assisting server. Consequently, the communication overhead on the cloud server is $O(kl\lambda\sqrt{N}\log N)$.

Computation Overhead. We calculate the computation overhead on the involved parties as follow:

Health organization. The health organization needs to sanitizes the query by encrypting the constants with the public key of each group. Therefore, the total number of sanitized queries will be $L = \sqrt{N}$, leading to $O(\sqrt{N})$ time complexity on the health organization.

Assisting server. The computation overhead on the assisting servers side is due to traversing the tree. The computation overhead therefore depends on the type of predicates as well as the type of aggregate function as explained in the follow.

– Range queries. The computational cost of range query over kD-tree is $O(N^{1-1/d} + m)$ where m is the number of records in the range. For each comparison, the assisting server must execute Fischlin protocol (t_{Fis}), aggregate the share of plaintexts – resulted from decryption of ciphertext sequences Δ and c – (t_{Aggr}), and test if the ciphertext sequences are quadratic residue or not (t_{QR}). Therefore, the computation overhead of tree traversal is $(N^{0.5-1/2d} + m).(t_{Fis} + t_{Aggr} + t_{QR})$; the computational cost of Fischlin protocol, the result aggregation and quadratic residue test is $O(\lambda l)$, $O(k)$ and $O(\lambda)$. Therefore, the computational cost of traversing the tree is $O(N^{0.5-1/2d} + m)(k + \lambda l)$.

– Point queries. Similarly, the computation overhead of executing an exact-matching query will be $O((k + \lambda l) \log N)$ on each assisting server.

In addition in the case of max/min aggregate functions, if a record satisfies the query condition(s), the assisting server must execute the Fischlin protocol to find the maximum/minimum of the encrypted records. This step imposes additional computational cost of $m.t_{Fis} = O(ml\lambda)$ to the assisting server.

5.5.3 Implementation and Performance

To evaluate the performance of the proposed protocol, we implemented a prototype of system relying on some existing works [129] in Java 1.6. The secret keys p and q of the GM cryptosystem are 256-bit long. Moreover, we used the publicly available *Breast Cancer* data set [130]. This data set has 286 records with 9 categorical attributes. The patient's and the server's side experiments were conducted on an Intel dual Core i5 2.3GHz Notebook with 4GB RAM. The number of patients in each group is fixed at $L = 286$ leading to approximately 81800 patients in the PHR system. To understand the source of the overhead, we conduct different experiments. In the first one, we measure the query execution time for different types of aggregate SQL queries, but running with only one core enabled. The result is presented in Table 13 and Fig. 19 where the error parameter λ of Fischlin protocol is 45.

Query	**Query time**(ms)
Select by =	41.93
Select range	216.14
Select sum	33.01
Select max/min	217.32

Table 13: *Assisting server latency for different types of SQL queries*

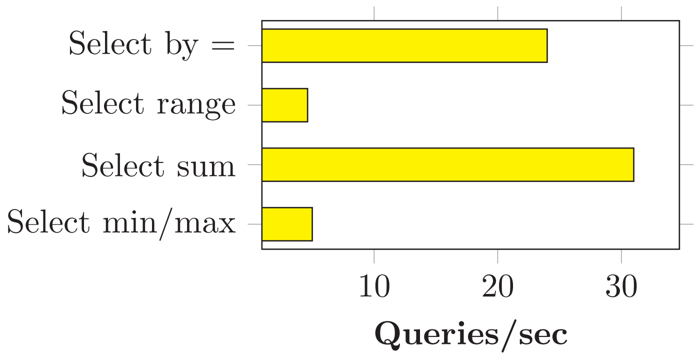


Figure 19: *Query Time*

In the second experiment, we study the effect of the error parameter λ on the execution time given different values for the threshold k. The tested values of the error parameter λ give a comparable level of correctness. As presented in Fig. 20, the time complexity of the threshold decryption is linear with the error parameter λ. When k is small, the query time is dominated by Fischlin's protocol which is independent from the threshold k. Therefore, there is a small difference in the query time when $k = 36$ and $k = 71$. However, as k increases the effect of the threshold decryption becomes more visible and the execution time starts to grow. Fig. 21 presents the effect of the threshold k on the execution time, given different values for the error parameter λ. According to a similar analysis, for small value of k there is a small change in the execution time but as k increases the query time becomes linear with k.

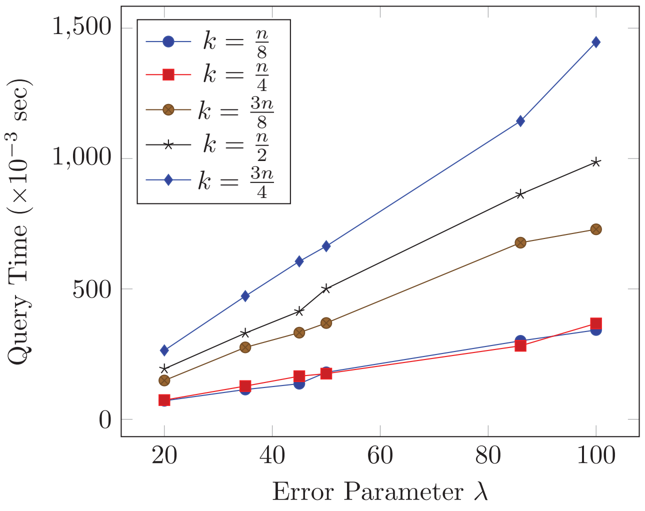


Figure 20: *Effect of λ for various values of k*

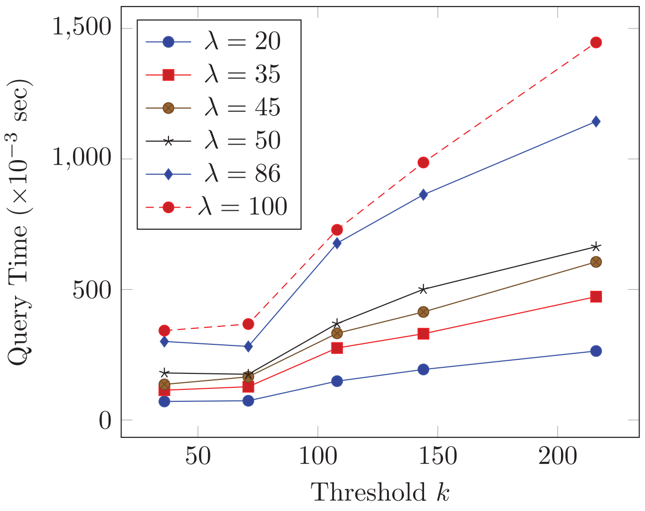


Figure 21: *Effect of k for various values of λ*

Finally, we calculate the execution time of an arbitrary SQL query assuming $\lambda = 45$ and $k = \frac{n}{4} = 71$. In addition to λ and k, the execution time of a query heavily depends on the number of comparisons that are performed to traverse the kD-tree. Therefore, we consider three different scenarios: (1) the worst case scenario is when evaluating predicates targeting a single attribute for interval matching such that all the tree nodes are traversed, (2) the best case scenario is when evaluating predicates targeting all attributes for exact matching, and (3) the real-world scenario is when evaluating predicates targeting multiple attributes for range and exact matching predicates. In the worst case, the time required to evaluate the predicate is 110 seconds (approximately 2 minutes) for each group, whereas in the best case it is 0.3 seconds. In the real-world scenario, we derive the execution time of a SQL query that contains interval matching and exact matching predicates. For each type of predicate, we execute four SQL queries with different number of attributes. The results are presented in Fig. 22. The results indicate that as the number of attributes in the query increases, the execution time decreases due to the limited search space and the reduction of the number of comparisons.

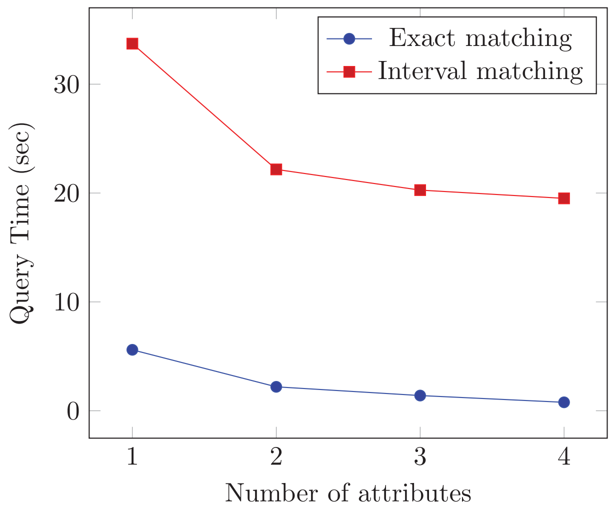


Figure 22: *Query time of a SQL query that contains exact matching (blue) and interval matching (red) predicates*

Our experimental results indicate that the proposed protocol would work well with medium size databases (with a total number of 100,000-400,000 patients) and the queries that contain multiple attributes. These results are from a first implementation of the proposed protocol. Further optimizations may likely lead to a better performance.

5.6 Conclusions and Discussion

In this chapter, we presented a protocol for executing secure queries over the cloud-based personal health records that employs private integer comparison and threshold cryptosystems to protect the query privacy of the health organization and the data privacy of the patients. In this section, we will answer some frequently raised questions.

PROBABILISTIC ENCRYPTION VS. DETERMINISTIC ENCRYPTION. A deterministic encryption scheme always produces the same ciphertext for a given plaintext. Deterministic public-key encryption is not secure in particular when the domain of the message is small since the adversary can simply encrypt each of the possible value in the domain under the recipient's public key, and compare the resulted ciphertext to the target ciphertext. To avoid this, probabilistic encryption utilizes randomness such that when encrypting the same message several times, it will yield different ciphertexts. Probabilistic public-key encryption schemes are slower than the symmetric-key encryption. In this thesis, we utilized two probabilistic encryption schemes (including Paillier [26] and Goldwasser-Micali [42]). Despite being slower, these schemes provide security for the records in the database. It is worth to mention that health organizations do not frequently conduct statistical analysis and studies on the medical databases (every month or when there is a pandemic).

ENCRYPTION DEVICES. The proposed protocol requires that the patients store the secret keys on the cloud machines. More precisely, the patients store their share of secret key on a FPGA and deliver it to the cloud server for the operation. The security concerns regarding key storage on cloud machines may cause patients to continue hosting the encryption/decryption operations using traditional privately-held servers. In this case, even though clients can leverage the computational and storage power of the cloud for the database and analytics, they must still maintain one or more local servers to perform encryption and execute decryption requests from assisting server. Unfortunately, in this case many of the advantages of the cloud are nullified, since patients still need local infrastructure. Furthermore, the cloud service can suffer severe performance issues because inter-site communication is much slower than intra-site communication.

Chapter 6

Conclusions and Future Work

The main goal of this thesis is to present protocols for privacy-preserving query processing over databases that are outsourced to cloud servers. We assume that the textual shape of SQL queries is not private but the constants contain sensitive information and must be protected against curious cloud servers. Therefore, the proposed protocols reveal the attribute names and the type of the aggregate function to the cloud server. The proposed protocols leverage symmetric private information retrieval (SPIR) and private integer comparison to protect the access privacy of clients as well as the database privacy of data owners in the cloud environment. Chapter 4 considers the case where the database owner is trusted for collecting and storing the records that belong to the individuals. We provide a solution for secure storage of the database records on the cloud server. Furthermore, our solution addressed two types of queries: keyword search queries and SQL queries. We proposed to organize the database records as a left-balancing binary search tree. Execution of the query can be performed by traversing the tree in the oblivious manner.

In chapter 5, we consider the scenario where the individuals are not willing that the data owner collects and manages their record. In particular, we propose a query processing model in the context of the cloud-based Personal Health Records (PHRs) where the patients are responsible for managing their medical record, controlling the access and protecting them against the curious cloud server. We propose to organize the patients in smaller group where the members of each group encrypt their records with jointly generated keys. For query execution, we utilized an approach similar to the one we provided in Chapter 4. Note that the proposed model can be utilized in situations where a database has been horizontally partitioned between different parties and the client would like to execute a private query over the distributed database. The results of the complexity analysis indicates that the proposed protocols incur reasonable communication and computation overhead on the client side, considering the added advantage of being able to perform symmetrically-private database search. We tested our implementation in Java 1.6 by employing a real data set. We observed that the running time for executing range queries is more than partial-matching queries. This observation agree well with the result of the complexity analysis. Furthermore, we observed as the number of attributes in the predicate of a query increase, the execution time decreases. Finally, we concluded that our proposed protocol would work well with the small to medium size databases and the queries that contain multiple attributes.

In the remaining of this chapter, we will highlight some of the future works related to this work.

6.1 Future Work

The work presented in this thesis, can be extended in several directions. In what follows, we briefly summarize some possible future research works.

Other SQL Queries. In this thesis, we have considered only static data. Although some applications have relatively stable data contents, many scenarios for database outsourcing require the system to support data dynamics. For example, in the cloud-based PHR case discussed in Chapter 5, patients may conduct new medical examinations and need to add new information to the outsourced database. In addition, they may find that some data have been miscalculated and need to be updated. An extension to this work can explore preserving the privacy of sensitive information within SQL INSERT, UPDATE and DELETE data manipulation statements.

Protecting The Shape of The Query. In this thesis, we assume that the shape of the SQL query is not private and our goal to protect query privacy was based on hiding the constants in the query predicate. Simultaneous protection of both the shape and the constants of a query can be considered as an interesting extension to our work.

Database Partitioning. Recently, the need of the distributed database system increases in order to reduce communication cost and improve reliability. In this case, the original database records are partitioned horizontally [131] or vertically [132] and each partition is outsourced to a database node. Horizontal partitioning allows access methods such as tables, indices and views to be partitioned into disjoint sets of rows that are physically stored and accessed separately. On the other hand, vertical partitioning allows a table to be partitioned into disjoint sets of columns. An interesting extension for our work is to study symmetrically-private search on vertical/horizontal partitioned databases.

Malicious Adversaries. The assumed adversary model in this thesis was semi-honest. This is a common adversary model used in the SMC literature [108]. A common approach to secure a protocol against malicious adversaries is done by requiring each party to use zero-knowledge proofs [37] to prove that it is following the protocol specifications. Since this generic approach adds considerable overhead to each step of the protocol, it is inefficient and more practical techniques are required to be developed.

Bibliography

[1] Amazon Relational Database Service (Amazon RDS), 2012. http://aws.amazon.com/rds/.

[2] SQL Azure: Highly Available and Scalable Cloud Database Service, 2012. http://www.windowsazure.com/en-us/home/features/sql-azure/.

[3] H. Hacigumus, B. Iyer, and S. Mehrotra. Providing Database as a Service. In *Proceedings. 18th International Conference on Data Engineering*, ICDE'02, pages 29–38, 2002.

[4] Mei Hui, Dawei Jiang, Guoliang Li, and Yuan Zhou. Supporting Database Applications as a Service. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ICDE '09, pages 832–843, 2009.

[5] Carlo Curino, Evan Jones, Raluca Ada Popa, Nirmesh Malviya, Eugene Wu, Samuel Madden, Hari Balakrishnan, and Nickolai Zeldovich. Relational Cloud: A Database Service for the Cloud. In *5th Biennial Conference on Innovative Data Systems Research*, January 2011.

[6] D. McCullagh. Privacy leaks hit Facebook, Google, AT&T, December 2010. http://news.cnet.com/2702-1009_3-986.html.

[7] M. J. Schwartz. Twitter Finalizes FTC Security Settlement, March 2011. http://www.informationweek.com/news/security/attacks/229301037.

[8] J. M. Arrington . AOL Proudly Releases Massive Amounts of Private Data, August 2006. http://techcrunch.com/2006/08/06/aol-proudly-releases-massive-amounts-of-user-search-data/.

[9] R. Jones, R. Kumar, B. Pang, and A. Tomkins. I Know What You Did Last Summer: Query Logs and User Privacy. In *Proceedings of the ACM Conference on Conference on Information and Knowledge Management*, CIKM '07, pages 909–914. ACM, 2007.

[10] F. Olumofin and I. Goldberg. Privacy-Preserving Queries Over Relational Databases. In *Proceedings of the International Conference on Privacy Enhancing Technologies*, PETS'10, pages 75–92, 2010.

[11] Hakan Hacigümüs, Balakrishna R. Iyer, Chen Li, and Sharad Mehrotra. Executing SQL Over Encrypted Data in The Database-Service-Provider Model. In *SIGMOD Conference*, pages 216–227, 2002.

[12] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. *Order Preserving Encryption for Numeric Data*, pages 563–574. ACM Press, 2004.

[13] Murat Kantarcıoğlu and Chris Clifton. Security Issues in Querying Encrypted Data. In *Proceedings of the 19th annual IFIP WG 11.3 Working Conference on Data and Applications Security*, DBSec'05, pages 325–337, 2005.

[14] Jun Li and Edward R. Omiecinski. Efficiency and Security Trade-off in Supporting Range Queries on Encrypted Databases. In *Proceedings of the 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, DBSec'05, pages 69–83. Springer-Verlag, 2005.

[15] Erez Shmueli, Ronen Waisenberg, Yuval Elovici, and Ehud Gudes. Designing Secure Indexes for Encrypted Databases. In *Proceedings of the 19th Annual IFIP*

WG 11.3 Working Conference on Data and Applications Security, DBSec'05, pages 54–68, 2005.

[16] Zhiqiang Yang, Sheng Zhong, and Rebecca N. Wright. Privacy-Preserving Queries on Encrypted Data. In *Proceedings of the 11th European Conference on Research in Computer Security*, ESORICS'06, pages 479–495, 2006.

[17] Ernesto Damiani, S. De Capitani Vimercati, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Balancing Confidentiality and Efficiency in Untrusted Relational DBMSs. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, CCS '03, pages 93–102. ACM, 2003.

[18] Bijit Hore, Sharad Mehrotra, and Gene Tsudik. A Privacy-Preserving Index for Range Queries. In *Proceedings of the 13th International Conference on Very Large Databases - Volume 30*, VLDB '04, pages 720–731. VLDB Endowment, 2004.

[19] Valentina Ciriani, Sabrina De Capitani Di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Keep A Few: Outsourcing Data While Maintaining Confidentiality. In *Proceedings of the 14th European Conference on Research in Computer Security*, ESORICS'09, pages 440–455. Springer-Verlag, 2009.

[20] O. Goldreich B. Chor, E. Kushilevitz and M. Sudan. Private Information Retrieval. *Journal of the ACM*, 45(6):965–981, November 1998.

[21] E. Kushilevitz and R. Ostrovsky. Replication Is Not Needed: Single Database, Computationally-Private Information Retrieval. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, FOCS '97, pages 364–373, 1997.

[22] Sh. Wang, D. Agrawal, and A. El Abbadi. Towards Practical Private Processing of Database Queries over Public Data with Homomorphic Encryption. Technical Report 2011-06, Department of Computer Science, University of California at Santa Barbara, Nov 2011.

[23] Yehuda Lindell and Benny Pinkas. Privacy Preserving Data Mining. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '00, pages 36–54. Springer-Verlag, 2000.

[24] J. Katz and M. Yung. Threshold Cryptosystems Based on Factoring. In *Proceedings of the Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '02, pages 192–205. Springer-Verlag, 2002.

[25] Marc Fischlin. A Cost-Effective Pay-Per-Multiplication Comparison Method for Millionaires. In *Proceedings of the 2001 Conference on Topics in Cryptology: The Cryptographer's Track at RSA*, CT-RSA 2001, pages 457–472, 2001.

[26] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, pages 223–238. Springer-Verlag, 1999.

[27] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword Search and Oblivious Pseudorandom Functions. In *TCC*, pages 303–324, 2005.

[28] Sanjeev Kumar Mishra and Palash Sarkar. Symmetrically Private Information Retrieval. In *Proceedings of the First International Conference on Progress in Cryptology*, INDOCRYPT '00, pages 225–236, 2000.

[29] Sonia Bergamaschi, Francesco Guerra, Silvia Rota, and Yannis Velegrakis. KEYRY: A Keyword-Based Search Engine over Relational Databases Based on

a Hidden Markov Model. In *Advances in Conceptual Modeling. Recent Developments and New Directions - ER 2011 Workshops*, volume 6999, pages 328–331. Lecture Notes in Computer Science, 2011.

[30] Ruoyu Wu, Gail-Joon Ahn, and Hongxin Hu. Towards HIPAA-Compliant Healthcare Systems. In *Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium*, IHI'12, pages 593–602. ACM, 2012.

[31] Ming Li, Shucheng Yu, Kui Ren, and Wenjing Lou. Securing Personal Health Records in Cloud Computing: Patient-Centric and Fine-Grained Data Access Control in Multi-owner Settings. In *Security and Privacy in Communication Networks*, volume 50, pages 89–106. Springer Berlin Heidelberg, 2010.

[32] Ming Li, Shucheng Yu, Yao Zheng, Kui Ren, and Wenjing Lou. Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. *IEEE Trans. Parallel Distrib. Syst.*, 24(1):131–143, January 2013.

[33] Emmanuel Abbe, Amir E. Khandani, and Andrew W. Lo. Privacy-Preserving Methods for Sharing Financial Risk Exposures. *CoRR*, abs/1111.5228, 2011.

[34] Bee-Chung Chen, Daniel Kifer, Kristen LeFevre, and Ashwin Machanavajjhala. Privacy-Preserving Data Publishing. *Foundations and Trends in Databases*, 2(1-2):1–167, 2009.

[35] Andrew C. Yao. Protocols for Secure Computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164. IEEE Computer Society, 1982.

[36] Benny Pinkas. Cryptographic Techniques for Privacy-Preserving Data Mining. *SIGKDD Explor. Newsletter*, 4(2):12–19, December 2002.

[37] Jean-Jacques Quisquater, Louis Guillou, Marie Annick, and Tom Berson. How To Explain Zero-Knowledge Protocols To Your Children. In *Proceedings on Advances in Cryptology*, CRYPTO '89, pages 628–631. Springer-Verlag New York, Inc., 1989.

[38] O. Goldreich. *Foundations of Cryptography*, volume 2. Cambridge University Press, 2001.

[39] B. Chor, N. Gilboa, and M. Naor. Private Information Retrieval by Keywords. *Report TR CS0917l*, 1997.

[40] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting Data Privacy in Private Information Retrieval Schemes. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, STOC '98, pages 151–160, 1998.

[41] R. Rivest, L. Adleman, and M. Dertouzos. On Data Banks and Privacy Homomorphisms. In *Foundations of Secure Computation*, pages 169–177, 1978.

[42] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption & How To Play Mental Poker Keeping Secret All Partial Information. In *Proceedings of The 14th Annual ACM Symposium on Theory of Computing*, STOC '82, pages 365–377, 1982.

[43] Tomas Sander, Adam Young, and Moti Yung. Non-Interactive Crypto-Computing For NC1. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 554–567, 1999.

[44] Yvo Desmedt. Some Recent Research Aspects of Threshold Cryptography. In *Information Security*, volume 1396 of *Lecture Notes in Computer Science*, pages 158–173. Springer Berlin Heidelberg, 1998.

[45] Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing Decryption in The Context of Voting or Lotteries. In *Proceedings of the 4th International Conference on Financial Cryptography*, FC '00, pages 90–104. Springer-Verlag, 2001.

[46] Ivan Damgård and Mats Jurik. A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography*, PKC '01, pages 119–136. Springer-Verlag, 2001.

[47] Adam Barnett and Nigel P. Smart. Mental Poker Revisited. In *Cryptography and Coding*, volume 2898 of *Lecture Notes in Computer Science*, pages 370–383. Springer Berlin Heidelberg, 2003.

[48] Takashi Nishide and Kouichi Sakurai. Distributed Paillier Cryptosystem Without Trusted Dealer. In *Proceedings of the 11th International Conference on Information Security Applications*, WISA'10, pages 44–60. Springer-Verlag, 2011.

[49] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust Efficient Distributed RSA-Key Generation. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, STOC '98, pages 663–672. ACM, 1998.

[50] Dan Boneh and Matthew Franklin. Efficient Generation of Shared RSA Keys. *J. ACM*, 48(4):702–722, July 2001.

[51] Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178. ACM, 2009.

[52] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. In *Proceedings of the 31st Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'12, pages 483–501. Springer-Verlag, 2012.

[53] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. Cloud-Assisted Multiparty Computation from Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2011:663, 2011.

[54] Andrew C. Yao. How to Generate and Exchange Secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, SFCS '86, pages 162–167, 1986.

[55] O. Goldreich, S. Micali, and A. Wigderson. How to Play ANY Mental Game. In *Proceedings of The 19th Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, 1987.

[56] Felix Brandt. Efficient cryptographic protocol design based on distributed el gamal encryption. In *ICISC'05*, pages 32–47. Springer-Verlag, 2005.

[57] Hsiao-Ying Lin and Wen-Guey Tzeng. An Efficient Solution to the Millionaires' Problem Based on Homomorphic Encryption. In *The 3rd International Conference on Applied Cryptography and Network Security*, ACNS'05, pages 456–466, 2005.

[58] Giovanni Di Crescenzo. Private Selective Payment Protocols. In *Proceedings of the 4th International Conference on Financial Cryptography*, FC '00, pages 72–89. Springer-Verlag, 2001.

[59] Ian F. Blake and Vladimir Kolesnikov. Conditional Encrypted Mapping and Comparing Encrypted Numbers. In *Proceedings of the 10th International Conference on Financial Cryptography and Data Security*, FC'06, pages 206–220, 2006.

[60] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[61] J. Andreas Bærentzen. On left-balancing binary trees, August 2003. http://www2.imm.dtu.dk/pubdb/p.php?2535.

[62] Mark De Berg, Otfried Cheong, Marc Van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, 3rd ed. edition, 2008.

[63] J. L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Commumications of the ACM*, 18(9):509–517, September 1975.

[64] Dmitri Asonov. Private Information Retrieval - An Overview And Current Trends. In *ECDPvA Workshop,Informatik*, 2001.

[65] Erica Y. Yang, Jie Xu, and Keith H. Bennett. Private Information Retrieval in The Presence of Malicious Failures. In *26th Annual International Computer Software and Applications Conference, 2002*, pages 805–810. IEEE, August 2002.

[66] Donald E. Knuth. *The Art of Computer Programming, volume 3: (2nd ed.) sorting and searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[67] Edward Fredkin. Trie Memory. *Commun. ACM*, 3(9):490–499, September 1960.

[68] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical Techniques for Searches on Encrypted Data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, SP '00, pages 44–56, 2000.

[69] J. Reardon, J. Pound, and I. Goldberg. Relational-Complete Private Information Retrieval. Technical Report CACR 2007- 34, University of Waterloo, 2007.

[70] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill, Inc., New York, NY, USA, 3 edition, 2003.

[71] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient Private Matching and Set Intersection. In *EUROCRYPT*, pages 1–19. Springer, 2004.

[72] Lea Kissner and Dawn Song. Private and Threshold Set-Intersection. In *Proceedings of CRYPTO '05*, August 2005.

[73] Wakaha Ogata and Kaoru Kurosawa. Oblivious Keyword Search. *J. Complex.*, 20(2-3):356–371, April 2004.

[74] R. Sion. Secure Data Outsourcing. In *Proceedings of the conference on Very large data bases*, VLDB '07, pages 1431–1432, 2007.

[75] Raluca Ada Popa, Catherine M S Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB : Protecting Confidentiality with Encrypted Query Processing. *In Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, pages 85–100, 2011.

[76] Divyakant Agrawal, Amr El Abbadi, Fatih Emekçi, and Ahmed Metwally. Database Management as a Service: Challenges and Opportunities. In *ICDE*, pages 1709–1716, 2009.

[77] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-Preserving Data Mining. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 439–450. ACM, 2000.

[78] Dakshi Agrawal and Charu C. Aggarwal. On The Design and Quantification of Privacy Preserving Data Mining Algorithms. In *Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '01, pages 247–255. ACM, 2001.

[79] Charu C. Aggarwal and Philip S. Yu. A Condensation Approach to Privacy Preserving Data Mining. In *Advances in Database Technology - EDBT 2004*, volume 2992 of *Lecture Notes in Computer Science*, pages 183–199. Springer Berlin Heidelberg, 2004.

[80] Wenliang Du and Zhijun Zhan. Building Decision Tree Classifier on Private Data. In *Proceedings of the IEEE International Conference on Privacy, Security and Data Mining - Volume 14*, CRPIT '14, pages 1–8. Australian Computer Society, Inc., 2002.

[81] Keke Chen and Ling Liu. Privacy Preserving Data Classification with Rotation Perturbation. In *Proceedings of the 5th IEEE International Conference on Data Mining*, ICDM '05, pages 589–592. IEEE Computer Society, 2005.

[82] Olvi L. Mangasarian, Edward W. Wild, and Glenn M. Fung. Privacy-Preserving Classification of Vertically Partitioned Data via Random Kernels. *ACM Trans. Knowl. Discov. Data*, 2(3):12:1–12:16, October 2008.

[83] Shariq J. Rizvi and Jayant R. Haritsa. Maintaining Data Privacy in Association Rule Mining. In *Proceedings of the 28th International Conference on Very Large Databases*, VLDB '02, pages 682–693. VLDB Endowment, 2002.

[84] Murat Kantarcioglu and Chris Clifton. Privacy-Preserving Distributed Mining of Association Rules on Horizontally Partitioned Data. *IEEE Trans. on Knowl. and Data Eng.*, 16(9):1026–1037, September 2004.

[85] Jaideep Vaidya and Chris Clifton. Privacy Preserving Association Rule Mining in Vertically Partitioned Data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 639–644. ACM, 2002.

[86] Jaideep Vaidya and Chris Clifton. Privacy-Preserving k-means Clustering over Vertically Partitioned Data. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 206–215. ACM, 2003.

[87] Geetha Jagannathan and Rebecca N. Wright. Privacy-Preserving Distributed k-means Clustering Over Arbitrarily Partitioned Data. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 593–599. ACM, 2005.

[88] Srujana Merugu and Joydeep Ghosh. Privacy-Preserving Distributed Clustering Using Generative Models. In *Proceedings of the Third IEEE International Conference on Data Mining*, ICDM '03, pages 211–218. IEEE Computer Society, 2003.

[89] Jaideep Vaidya, Murat Kantarcıoğlu, and Chris Clifton. Privacy-Preserving Bayes Classification. *The VLDB Journal*, 17(4):879–898, July 2008.

[90] Zhiqiang Yang and Rebecca N. Wright. Improved Privacy-Preserving Bayesian Network Parameter Learning on Vertically Partitioned Data. In *Proceedings*

of the 21st International Conference on Data Engineering Workshops, ICDEW '05, pages 1196–1206. IEEE Computer Society, 2005.

[91] W. Du, Y. S. Han, and S. Chen. Privacy-Preserving Multivariate Statistical Analysis: Linear Regression and Classification. In *Proceedings of 2004 SIAM International Conference on Data Mining (SDM04)*, April 2004.

[92] W. Du and M. Atallah. Privacy-Preserving Cooperative Statistical Analysis. In *Proceedings of the 17th Annual Computer Security Applications Conference*, ACSAC '01, pages 102–. IEEE Computer Society, 2001.

[93] Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. Incognito: Efficient Full-Domain k-Anonymity. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 49–60, 2005.

[94] L. Sweeney. k-Anonymity: A Model for Protecting Privacy. In *International Journal on Uncertainty, Fuzziness and Knowledge-Based Systems*, volume 10, pages 557–570, 2002.

[95] Roberto J. Bayardo and Rakesh Agrawal. Data Privacy through Optimal k-Anonymization. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 217–228, 2005.

[96] Adam Meyerson and Ryan Williams. On The Complexity of Optimal k-Anonymity. In *Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '04, pages 223–228, 2004.

[97] Steven P. Reiss, Mark J. Post, and Tore Dalenius. Non-Reversible Privacy Transformations. In *Proceedings of the 1st ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, PODS '82, pages 139–146, 1982.

[98] Steven P. Reiss. Practical Data-Swapping: The First Steps. *ACM Trans. Database Syst.*, 9(1):20–37, March 1984.

[99] Zhiqiang Yang, Sheng Zhong, and Rebecca N. Wright. Privacy-Preserving Classification of Customer Data without Loss of Accuracy. In *Proceedings of the 5th SIAM International Conference on Data Mining*, 2005.

[100] Cynthia Dwork. Differential Privacy: A Survey of Results. In *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation*, TAMC'08, pages 1–19. Springer-Verlag, 2008.

[101] Frank D. McSherry. Privacy Integrated Queries: An Extensible Platform for Privacy-Preserving Data Analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, SIGMOD '09, pages 19–30. ACM, 2009.

[102] Indrajit Roy, Srinath T. V. Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. Airavat: Security and Privacy for MapReduce. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 297–312. USENIX Association, 2010.

[103] Hyun-A Park, Justin Zhan, and Dong Hoon Lee. Privacy Preserving SQL Queries. In *Proceedings of the 2008 International Conference on Information Security and Assurance (ISA 2008)*, ISA '08, pages 549–554. IEEE Computer Society, 2008.

[104] Markus Jakobsson and Moti Yung. Proving Without Knowing: On Oblivious, Agnostic and Blindolded Provers. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, pages 186–200. Springer-Verlag, 1996.

[105] Radu Sion. On the Computational Practicality of Private Information Retrieval. In *In Proceedings of the Network and Distributed Systems Security Symposium, 2007. Stony Brook Network Security and Applied Cryptography Lab Tech Report*, 2007.

[106] Changyu Dong, Giovanni Russello, and Naranker Dulay. Shared and Searchable Encrypted Data for Untrusted Servers. In *Proceeedings of the 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, pages 127–143, 2008.

[107] S. Moro, R. Laureano, and P. Cortez. Using Data Mining for Bank Direct Marketing: An Application of the CRISP-DM Methodology. In *Proceedings of the European Simulation and Modelling Conference - ESM'2011*, pages 117–121, Guimaraes, Portugal, October 2011. EUROSIS.

[108] Wei Jiang and Chris Clifton. A Secure Distributed Framework for Achieving k-Anonymity. *The VLDB Journal*, 15(4):316–333, November 2006.

[109] Benjamin C. M. Fung, Ke Wang, Rui Chen, and Philip S. Yu. Privacy-Preserving Data Publishing: A Survey of Recent Developments. *ACM Comput. Surv.*, 42(4), June 2010.

[110] C. Gentry and Z. Ramzan. Single-Database Private Information Retrieval with Constant Communication Rate. In *Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 103–103. Springer Berlin Heidelberg, 2005.

[111] J. Trostle and A. Parrish. Efficient Computationally Private Information Retrieval from Anonymity or Trapdoor Groups. In *Information Security*, volume 6531, pages 114–128. Springer Berlin Heidelberg, 2011.

[112] K. Philipkoski. Black Death's Gene Code Cracked, 2001. http://www.wired.com/medtech/health/news/2001/10/47288.

[113] Centers for Disease Control and Prevention (CDC). Updated CDC Estimates of 2009 H1N1 Influenza Cases, Hospitalizations and Deaths in the United States, April 2009 - April 10,2010, 2010. http://www.cdc.gov/h1n1flu/estimates_2009_h1n1.htm.

[114] WHO — World Health Organisation, Accessed in 2012. http://www.who.int/en/.

[115] Marco Eichelberg, Thomas Aden, Jörg Riesmeier, Asuman Dogac, and Gokce B. Laleci. A Survey and Analysis of Electronic Healthcare Record Standards. *ACM Comput. Surv.*, 37(4):277–315, December 2005.

[116] Jane Grimson, Gaye Stephens, Benjamin Jung, William Grimson, Damon Berry, and Sebastien Pardon. Sharing Health-Care Records over the Internet. *IEEE Internet Computing*, 5(3):49–58, May 2001.

[117] Hassan Takabi, James B. D. Joshi, and Gail-Joon Ahn. Security and Privacy Challenges in Cloud Computing Environments. *IEEE Security and Privacy*, 8(6):24–31, November 2010.

[118] Microsoft health vault, 2012. http://www.healthvault.com/Personal/index.html.

[119] Dossia personal health platform, 2012. http://www.dossia.org/.

[120] H. Löhr, A. Sadeghi, and M. Winandy. Securing The E-Health Cloud. In *Proceedings of the International Health Informatics Symposium*, IHI'10, pages 220–229. ACM, 2010.

[121] Judy Foreman. At risk of exposure, 2006. http://articles.latimes.com/2006/jun/26/health/he-privacy26.

[122] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data. In *Proceedings of The Conference on Computer and Communications Security*, CCS '06, pages 89–98. ACM, 2006.

[123] Consulting:PwC, Accessed in 2012. http://www.pwc.com/gx/en/consulting-services/index.jhtml.

[124] Privacy & data protection update. *Respecting Patient Privacy, Building Patient Trust. NewsLetter of the office of HIPAA Provacy & Security. Miller School of Medicine. University of Miami*, (21), 2012.

[125] Ken Eguro and Ramarathnam Venkatesan. FPGAs for Trusted Cloud Computing. In *22nd International Conference on Field Programmable Logic and Applications (FPL), Oslo, Norway, August 29-31, 2012*, pages 63–70, 2012.

[126] Bruce Schneier. *Applied Cryptography (2nd Ed.): Protocols, Algorithms, and Source Code in C* . John Wiley & Sons, Inc., New York, NY, USA, 1995.

[127] Chris Clifton, Murat Kantarcioglu, Jaideep Vaidya, Xiaodong Lin, and Michael Y. Zhu. Tools for Privacy Preserving Distributed Data Mining. *SIGKDD Explor. Newsl.*, 4(2):28–34, December 2002.

[128] Elaine Shi, T.-H. Hubert Chan, Eleanor G. Rieffel, Richard Chow, and Dawn Song. Privacy-Preserving Aggregation of Time-Series Data. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011*, 2011.

[129] Martin Geisler. Cryptographic Protocols: Theory and Implementation. In *PhD Thesis*.

[130] UCI Machine Learning Repository: Breast Cancer Data Set, April 2012. http://archive.ics.uci.edu/ml/datasets/Breast+Cancer.

[131] S. Ceri, M. Negri, and G. Pelagatti. Horizontal Data Partitioning in Database Design. In *Proceedings of the 1982 ACM SIGMOD International Conference on Management of Data*, SIGMOD '82, pages 128–136, 1982.

[132] Shamkant Navathe, Stefano Ceri, Gio Wiederhold, and Jinglie Dou. Vertical Partitioning Algorithms for Database Design. *ACM Trans. Database Syst.*, 9(4):680–710, December 1984.