Protocols for Secure Computation on Privately Encrypted Data in the Cloud

Feras Abdulaziz Aljumah

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy at

Concordia University

Montréal, Québec, Canada

April 2017

CONCORDIA UNIVERSITY

Division of Graduate Studies

This is to certify that the thesis prepared

By: **Feras Abdulaziz Aljumah**

Entitled: **Protocols for Secure Computation on Privately Encrypted Data in the Cloud**

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

Doctor of Philosophy

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

Signed by the final examining committee:

______________________ Dr. Khaled Galal	Chair
______________________ Dr. Mohammad Zulkernine	External Examiner
______________________ Dr. Anjali Agarwal	External Examiner to Program
______________________ Dr. Peter Grogono	Examiner
______________________ Dr. Terry Fancott	Examiner
______________________ Dr. Mourad Debbabi	Thesis Co-Supervisor
______________________ Dr. Makan Pourzandi	Thesis Co-Supervisor

Approved by ______________________________________
Chair of the Computer Science and Engineering Department

______ 2017 ______________________________________
Dean, Faculty of Engineering and Computer Science

ABSTRACT

Protocols for Secure Computation on Privately Encrypted Data in the Cloud

Feras Abdulaziz Aljumah

Concordia University, 2017

Cloud services provide clients with highly scalable network, storage, and computational resources. However, these service come with the challenge of guaranteeing the confidentiality of the data stored on the cloud. Rather than attempting to prevent adversaries from compromising the cloud server, we aim in this thesis to provide data confidentiality and secure computations in the cloud, while preserving the privacy of the participants and assuming the existence of a passive adversary able to access all data stored in the cloud.

To achieve this, we propose several protocols for secure and privacy-preserving data storage in the cloud. We further show their applicability and scalability through their implementations. we first propose a protocol that would allow emergency providers access to privately encrypted data in the cloud, in the case of an emergency, such as medical records. Second, we propose various protocols to allow a querying entity to securely query privately encrypted data in the cloud while preserving the privacy of the data owners and the querying entity. We also present cryptographic and non-cryptographic protocols for secure private function evaluation in order to extend the functions applicable in the protocols.

DEDICATION

To my parents, whose love and support paved the way for

the knowledge necessary to complete this work.

To my wife, for her inspiration and love.

To my brothers and sisters, for being there when I needed them the most.

ACKNOWLEDGEMENTS

First and foremost, all praises to Allah for blessing, protecting and guiding me throughout my studies. I could never have accomplished this without my faith.

I would like to express my sincere gratitude to my supervisors Prof. Mourad Debbabi and Dr. Makan Pourzandi for their continuous support, patience, motivation, and immense knowledge. The research objectives would not have been achieved without the professional and experienced guidance and support of my supervisors. My gratefulness extends to members of the examining committee including Dr. Mohammad Zulkernine, Dr. Peter Grogono, Dr. Terry Fancott, and Dr. Anjali Agarwal for critically evaluating my thesis and giving me valuable feedback.

My gratitude also goes to my colleagues, Andrei Soeanu, Dr. Dima Alhadidi, Raymond Hei Man Leung, Samira Barouti, Dr. Sujoy Ray, and Dr. Wen Ming Liu. I thank them for their collaborations and their friendship. I am forever thankful to my brothers and colleagues Hamad BinSalleeh and Sultan AlQahtani, whose friendship made this journey and my stay in Canada truly memorable.

I would like to acknowledge the financial support from the Government of Saudi Arabia under the scholarship of Imam Mohammed Bin Saud University, which enabled me to undertake my PhD studies.

Last but not the least, I take this opportunity to express my deepest gratitude to my parents, my wife, my brothers, and my sisters for their unconditional love, prayers, and support. Without them, this journey would not have been possible, and to them I dedicate this milestone.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

ABE	Attribute Based Encryption
CP-ABE	Ciphertext Policy Attribute Based Encryption
CS	Cloud Server
DO	Data Owners
ECP	Emergency Care Providers
EHR	Electronic Health Record
EMR	Emergency Medical Records
FHE	Fully Homomorphic Encryption
HR-ABE	Hybrid ABE Revocation Scheme
IBE	Identity Based Encryption
KP-ABE	Key Policy Attribute Based Encryption
LSSS	Linear Secret Sharing Scheme
MPC	Multi-Party Computation
PHE	Partially Homomorphic Encryption
PHR	Personal Health Record
PII	Personally Identifiable Information
PKG	Public Key Generator
QE	Querying Entity
SMC	Secure multi-party computation
SSE	Searchable Symmetric Encryption
SWHE	SomeWhat Homomorphic Encryption
TLS	Transport Layer Security

Chapter 1

Introduction

The idea of cloud computing was first suggested in 1961 by John McCarthy during a speech given to celebrate MIT's centennial. In that speech, he envisioned that computer time-sharing technology might result in a future in which computing power and even specific applications could be sold through the utility business model (like water or electricity). The idea became popular during the 1960s, but faded in the 1970s because the infrastructure needed for such an idea was not possible at the time [1]. Cloud computing as a term was used for the very first time in 1996 by George Favaloro in an internal Compaq analysis titled "Internet Solutions Division Strategy for Cloud Computing". In 1997, Netcentric attempted to trademark the term for educational purposes, but the application was never approved [2]. In 2006, the term was used by Google's CEO Eric Schmidt during a conference note at Google where he suggested that "Data services and architecture should be on servers. We call it cloud computing" [3].

The National Institute of Standards and Technology (NIST) later defined cloud computing as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (*e.g.,* networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [4].

In recent years, storing data in the cloud has become popular due to its simplicity, integration, affordability, and the wide range of features offered by service providers. Within minutes of buying a new smart phone, a user can pull backup data from the cloud and restore call logs, messages, photos, app data, media, and emails to the new phone within minutes without needing anything other than a username and a password. Although these backups contain private and personal data, many users find that the convenience out-ways the privacy concerns.

However, there are many applications, where the privacy of the data remains a concern due to the sensitive nature of the data. Such systems include cloud-based personal health record (PHR) or financial management systems. Researchers have addressed this issue by proposing numerous schemes to ensure the confidentiality of user data in the cloud [5].

When users upload data to the cloud or the web, their control over the data is lost. Most of the time, it is unclear to the users where the data is stored, how many copies of the data exist, or who actually has access to the data. Moreover, users are unable to detect or prevent a malicious user from accessing their data. Although data is usually transferred using encrypted channels to protect user data from malicious eavesdroppers, the data is decrypted in most cases when received by the cloud service provider. This leaves the data vulnerable to malicious insiders, or any attack on the provider's system.

There are various methods for securing sensitive data stored in the cloud [5]. Data encryption ensures that access is only possible when the correct decryption key is provided. However, although cloud providers claim that sensitive data in the cloud is securely encrypted, in most cases trust is given to the cloud service providers to manage the encryption keys on behalf of the users. This implies that, although data is encrypted, the cloud service provider has the ability to see the actual data.

It is possible to build a secure privacy-preserving system that allows users to store privately encrypted data in the cloud. In such a system, data is encrypted and key management becomes the responsibility of the users, thus ensuring cloud service providers cannot access the unencrypted data. This approach is not popular due to the limited features providable by a cloud service provider, such as sharing or querying data to provide statistical information. Cloud-based PHRs are the most common applications used by researchers when discussing this problem [6, 7, 8].

In the market today, there are many solutions, which give users the ability to cryptographically secure their cloud storage drives. Such solutions include Boxcryptor, Cryptomator, Tarsnap, cryptelo, and Seafile [9]. These systems guarantee that access is only possible through the user's encryption keys, which are not accessible or stored by the cloud provider. These solutions provide data confidentiality, but also prevent the system from generating useful reports or statistics based on users' data.

In this thesis, we will address the security issues of preserving the privacy of data owners in the cloud by empowering data owners and giving them control over their data in the cloud. Our goal is to protect the confidentiality of the data in the cloud and the privacy of the data owners, while providing services to the data owners such as the ability to run queries or compute secure functions with each other or querying entities.

The rest of the chapter is organized as follows: Section 1.1 presents the motivations. Section 1.2 lists the objectives. Section 1.3 lists the contributions of the thesis. The structure of the thesis is given in Section 1.4.

1.1 Motivations

Cloud computing provides many opportunities for users by offering a wide range of computing services. These services include high storage and computation resources, dynamism, elasticity, and many other choices offered by this highly scalable technology are too attractive to overlook. Dynamic elasticity, allows for rapid scaling of resources so it appears to the client that the available capabilities are unlimited, while paying for used resources only. However, these services do not come without challenges. In this section, we discuss the challenges and motivations of the current study and identify cloud storage challenges:

1.1.1 Trust in the Cloud

Cloud services present us with new challenges introduced by a different type of trust scenario. In this section, we will discuss the problem of trust in the cloud, which is a major concern for users. The problem does not only lie in the lack of trust in the cloud providers intentions, but in the provider's ability to protect the security of the stored data and computations from internal or external adversaries. Trusting the cloud can become difficult due to lack of transparency, loss of control over data, and unclear security assurances [5].

There have been many cases in the past couple of years to make users and enterprises fearful of uploading any sensitive data to the cloud. These include cases where confidential user data was abused intentionally or unintentionally by service providers, corporations, or governmental agencies.

In 2014, Community Health Systems, which operates 206 hospitals in 28 states across the United States, announced that hackers gained unauthorized access into its network and stole data on 4.5 million patients. Cyber-security experts from Mandiant were hired to consult on the hack, and later determined that malware was used by Chinese hackers to launch the attacks [10].

In 2015, the second largest health insurer Anthem in the United States, announced that hackers broke into their systems and stole over 78.8 million records. The stolen data contained personally identifiable information such as names, birth dates, medical IDs, social security numbers, and income data. [11]. According to Anthem Inc., the data breach extended to into many of its brands, including Anthem Blue Cross, Anthem Blue Shield, Amerigroup, Caremore, and UniCare.

In 2016, it was revealed that the Red Cross blood service in Australia was affected by a data breach, which affected 550,000 blood donors. The stolen data included personal data about blood donors between 2010 and 2016, including information about “at-risk sexual behavior” [12].

In 2014, a vulnerability was discovered in the user authentication system of Apple’s iCloud. The vulnerability allowed users to try logging into the system an unlimited number of times regardless of the number of failed attempts [13]. A script was later published on github that allowed users to brute force passwords to gain access to iCloud accounts [14]. This led to hackers to publish hundreds of private and intimate images collected from iPhone image backups of a large number of famous public figures.

In 2014, a vulnerability was found in the cloud storage providers Dropbox, Box, and Google Drive [15]. The vulnerability allowed third parties to read private files stored in the cloud. It was discovered that when users type the URL of a shared file in the search engine field rather than the URL field, the files are then automatically indexed by search engines and become publicly available.

In a 2015 report, by Imperva’s Hacker Intelligence Initiative, a "man-in-the-cloud" attack was presented [16]. The attack allows attackers to gain access to a users’ file synchronization cloud storage accounts without needing to compromise the users’ login credentials. Tests were conducted on Microsoft’s OneDrive, Dropbox, Google Drive, and Box.

These solutions work by connecting user devices to a cloud server through the same user account. When a file is added to the repository, then all devices associated with the account synchronize and download the file. To make the system simpler for users, many of these popular applications do not require users to enter their credentials each time. Instead, authentication depends on a token usually stored locally on the users' devices. Attackers can add their devices to be synchronized with an account by accessing an authentication token of that account. As a proof of concept, Imperva researchers developed a tool to steal credential tokens by using phishing or drive-by download attacks.

Based on the aforementioned, it is clear that relying on the trust and reputation of a school, corporation, cloud service provider's employees, or an authentication system is neither a strong nor is it a sustainable way forward for the cloud computing industry.

1.1.2 Privacy in the Cloud

In this section, we discuss the issue of preserving the privacy of data owners in the cloud. In the market today, there are many cloud service providers that offer data storage and analysis for specific applications such as financial or medical applications [17, 18, 19, 20]. Microsoft HealthVault [17] stores Personal Health Records (PHRs) in the cloud and allows users to access them from location or device. These records include allergies, chronic disease, prescription records, surgeries, imaging reports, family history, vaccinations, etc. The system conveniently allows patient to give access to doctors or pharmacist in moments, thus helping medical professionals give more accurate diagnoses. Microsoft HealthVault also provides useful analysis on user data, such as suggested doctor checkups based on age and chronic diseases, or notifying users when two conflicting prescription drugs are added. Dossia [18] is a similar solution, which is based on open-source software. Dossia is an initiative led by from AT&T, Intel, BP America, Walmart, and many other companies to offer a cloud-based PHR system to their employees. Given the number of their employees, Dossia could be the largest PHR system in the world.

Quicken and Mint [19, 20] are financial applications that offer cloud-based services that allow users to track their bank accounts and credit card balances and spending. They also offer services such as reminding user of upcoming bills, irregular account activity, or spending statistics (*e.g.,* gas, groceries, shopping, entertainment). TurboTax and H&R Block [21, 22] are applications that analyze the users' financial data to help with income tax preparation by filling in forms for the users and searching for applicable deductions and credits.

Although all of these features simplify users' lives, they find themselves with a difficult choice. Should users use theses services and hand their sensitive data to these corporations. Can users trust them with their complete medical history and all their financial records? Can users trust them with data on every doctors visit, every prescription drug, or every dollar they've ever spent?

1.1.3 Security in the Cloud

In this section, we discuss the importance of protecting the confidentiality of user data in the cloud and the challenges introduced by the cloud environment. By relying on cryptography, data owners can encrypt their data before uploading it to the cloud to guarantee data confidentiality. Although this approach would prevent the cloud provider from accessing the data, it would also prevent the cloud provider from analyzing the data to produce useful statistics (e.g., searching data, computations on data).

In some cloud services, user data is encrypted and decrypted when access to the data is needed. For example, financial cloud applications need to access user data to produce spending reports or help calculate taxes. However, since the service providers have access to the encrypted data and the decryption keys, user data is still accessible to the service provider. The data in this case is also vulnerable to "man-in-the-cloud" attacks [16], or attacks on the provider's infrastructure targeting the encrypted data and the decryption keys.

Homomorphic cryptography can perform mathematical operations on encrypted data directly. This would help providers analyze user data without having to decrypt the data first. Moreover, the results of the computations can not be accessed without the corresponding cryptographic keys. Unfortunately, there is not a one size fits all solution. Partially homomorphic cryptography is limited in the type of operations that can be computed. Fully homomorphic cryptography on the other hand is expensive and slow to the point where it is not practical [23]. A detailed discussion on homomorphic cryptography is presented in section 2.1.4.

1.2 Objectives

Our main objective is to allow users to securely store their data online while allowing the following querying features:

- Design and implement a security protocol to allow users to securely store their data in the cloud, while preventing cloud providers from accessing user data.
- Propose a protocol for querying privately encrypted user data in the cloud during an emergency (such as emergency medical records) while preserving the privacy of the user.
- Design and implement a security protocol to allow querying privately encrypted data in the cloud using aggregate and comparison queries, while preserving the privacy of the querying entity and the users.

- Elaborate a protocol to allow users to jointly compute a private polynomial function on their data while keeping their values private.

  To this end, we will have the following requirements:

- Protocols can not rely on a trusted entity in the environment able to access unencrypted user data.
- Users need to be given complete control over their data, enabling them to decline participating in certain queries.

1.3 Contributions

To address the objectives mentioned above. The contributions of this thesis can be summarized as follows:

- **Design a protocol for secure mobile emergency access of privately encrypted data in the cloud:** We propose a protocol to allow emergency responders to query a subset of cloud stored private data in the case of an emergency in a mobile environment, such as critical medical data. The proposed protocol allows access only in the case of an emergency, preventing emergency responders from abusing their privileges. We propose a protocol based on attribute based encryption and symmetric key threshold encryption and solves the problem without requiring the participation of the patient in the protocol. We experimentally evaluate the performance of the proposed protocol and report on the results of implementation.
- **Design a protocol for secure and privacy-preserving querying protocol on privately encrypted data:** We propose a protocol that would allow querying entities

such as health organizations to produce statistical information about privately encrypted data, which is data encrypted by the data owner using keys not accessible to the service provider, such as PHRs stored in the cloud. The protocol depends on two threshold homomorphic cryptosystems: Goldwasser-Micali (GM) [24] and Paillier [25]. It executes queries on KD-trees that are constructed from encrypted health records. It also prevents patients from inferring what health organizations are concerned about. We experimentally evaluate the performance of the proposed protocol and report on the results of implementation.

- **Design an improved protocol for secure and privacy-preserving querying protocol on privately encrypted data:** We propose a second privacy-preserving protocol that would allow a third party, such as a health organization, to query privately encrypted data without relying on a trusted entity. The improved protocol does not store data in a KD-Trees, computations by the DO are reduced, the query size is reduced, and DOs can choose not to participate in a query. The protocol relies on homomorphic, threshold cryptography, and randomization to allow for secure, distributed, and privacy-preserving queries. We also present two variations of the protocol, the first aims to hide the query attributes, the second aims to prevent data inference by applying differential privacy. We evaluate the performance of our protocol and report on the results of the implementation.

- **Design a protocol for a privacy-preserving private function evaluation protocol in the cloud:** We propose cryptographic and a non-cryptographic privacy-preserving protocols that allow a participant to collaboratively compute a private polynomial function with at least two other participants using semantically secure cryptosystems. We experimentally evaluate the performance of the proposed protocol and report on the results of our implementation.

The work in this thesis is published in [26], [27], [28], and [29], and a patent has been filed in June of 2014 [30] on a collaborative research with Ericsson Canada Research and Development.

1.4 Thesis Organization

The remainder of the thesis is organized as follows. Chapter 2 gives an overview of the necessary knowledge required throughout our work. In addition, it provides a discussion on the current literature about the subjects that are related to the problems addressed in this thesis. Chapter 3 describes our solution for accessing privately encrypted data in the case of an emergency. Chapter 4 and Chapter 5 propose solutions to solve the problem of querying privately encrypted data stored in the cloud. Chapter 6 discusses and solves the problem of private multi-party computation and proposes a protocol to achieve PFE in a reasonable amount of time. Chapter 7 presents concluding remarks on this thesis together with a discussion of future research.

Chapter 2

Background and Related Work

2.1 Background

In this section, we review some of the required concepts that are used throughout our thesis. This section is organized as follows. Section 2.1.1, discusses cloud security models and the assumptions we make throughout this thesis. Section 2.1.2, presents an overview on Attribute Based Encryption (ABE), which provides cryptographically enforced access control to data in the cloud. Sections 2.1.3, presents the concept of threshold cryptography, which can aid in splitting trust in the cloud between multiple users. Section 2.1.4, presents the cryptographic primitives, which allow users to run secure computations while guaranteeing the confidentiality of the computation inputs.

2.1.1 Security Models and Assumptions in the Cloud

In this section, we discuss the goals of security in the cloud, cloud trust models, cloud adversary models, and the assumptions we follow throughout this thesis. Although there are many benefits to cloud storage and computing, there are many risks introduced by the cloud to sensitive data. These risks come from the need to trust a third party provider to securely store sensitive data. Cloud providers may have nodes administered or controlled by untrusted entities. In some cases, cloud storage may be vulnerable to attacks from external adversaries or malicious insiders such as employees or from other cloud tenants. Security guarantees are required to assure data owners that their data is properly protected. These security guarantees are legally binding promises between the cloud provider and the data owner as outlined in a Service Level Agreement (SLA). Data owners can proactively protect their data by relying on cryptography rather than relying on SLA agreements, which can be difficult to enforce.

Three cryptographic security guarantee goals are considered in the context of cloud computing security:

- **Data Confidentiality:** This property ensures that data contents are not accessible by unauthorized users. When data is stored in the cloud, outsourced data becomes out of the data owners' direct control. Access to sensitive data should only be given to authorized users, while others, including cloud service providers, should not gain any information about the data. Data owners should be able to fully take advantage of cloud data services such as data search, computation, or sharing, without leakage of any information about the data to any adversaries including the cloud provider.

- **Data Access Controllability:** This property enables data owners to selectively restrict access to data outsourced to the cloud. This means data owners can grant users to some users, while restricting others from accessing the data without permission. Moreover, it is desirable to enforce fine-grained access control; this would enable the data owner to create different access privileges to granted to different users in regards to different data pieces. Access authorization should only be controlled by the data owner untrusted cloud environments.

- **Data Integrity:** This property demands ensuring the accuracy and completeness of outsourced data in the cloud. Data owners expect that outsourced data is stored in the cloud correctly and in a trustworthy manor. This implies that data should not be modified, fabricated, or deleted. This property ensures that data owners would be able to detect corruption or loss of data. Moreover, when a subset of the data is corrupted or lost, the remaining data should be retrievable.

- **Privacy Preservability:** This property ensures that the identities of data owners are hidden when using cloud data services. This also includes the protection of the computations executed on the data and the information retrieved from the cloud. For example, the keywords queried by data owners over the outsourced data and the results should not be accessible any party in the cloud. Furthermore, the behaviors and habits of data owners should be protected and not be inferred by others.

In terms of the trust assumptions between the data owner and the cloud resources, we consider the following three trust models:

- **Untrusted cloud**: In this model, data owners do not trust the cloud or any nodes associated with the cloud provider. This implies that the cloud is not trusted to maintain the confidentiality or integrity of the data or the computations outsourced to it. In this model, client-side protections are a necessity to ensure the confidentiality and integrity of the data. This model is commonly associated with public cloud deployment.

- **Trusted cloud**: In this model, the cloud is deployed in an isolated environment. Nodes in this model may be corrupted, but can not access any private data. Nodes can also attempt to violate the data and computation integrity. This model is commonly associated with the private cloud deployment model, and in government use-cases.

- **Semi-trusted cloud**: In this model, it is assumed that the client does not fully trust the cloud. However, the cloud is not assumed to be untrusted. Instead, it is assumed that parts of the cloud may be under the control of an adversary at any given time, but a sufficient fraction of the resources would remain adversary-free. This model is relevant to real-world deployments, where a cloud provider that is trusted to attempt to maintain security, but not guaranteed to successfully protect against all internal and external adversaries. This model may be associated with hybrid, public, or private cloud deployment models.

To model the threat of an adversary, who is able to control chosen nodes in the cloud, we present the two adversary models:

- Passive adversary: In this model, a party is corrupted by an honest-but-curious adversary. The adversary in this model tries to learn additional information about private data by combining and analyzing observations of its set of corrupt parties. In this model, the adversary does not modify any messages or the prescribed protocol steps.
- Active adversary: In this model, a malicious adversary may cause a party to deviate from the prescribed protocols. This may include sending malformed messages, modifying messages, protocol steps, or actively colluding with other malicious parties. The adversary in this model actively controls corrupted parties to violate the confidentiality or integrity of the data or computations.

The goal of cloud security is to preserve the confidentiality and integrity of the data in the cloud in the presents of adversaries. The chosen cloud architecture, cloud trust model, and adversary models dictate the suitable solutions for a given scenario. In this thesis, we focus on preserving the confidentiality of the data and the secure computations in the cloud. The solutions we present in this thesis also assume the semi-honest cloud trust model for its suitability for most real-world cloud deployments and the passive adversary model.

Threats to data integrity in the cloud are out of the scope of this thesis. There have been many works to address integrity challenges in the cloud. Cloud Integrity issues related to data loss or manipulation have been addressed in [31, 32], dishonest computations in cloud servers are addressed in [33, 34], and Provable Data Possession (PDP) are addressed in [35, 31, 32, 36].

2.1.2 Attribute Based Encryption

This section discusses data encryption methods on the cloud, and provides an overview on Attribute Based Encryption (ABE) and its variations. Although cloud storage has many advantages, it also presents us with challenging issues such as security and practicality [37, 38, 39, 40]. One of the most serious challenges is protecting the confidentiality of the data in the cloud. Traditional methods, such as the reliance on a username and password to authenticate users, do not protect the confidentiality of the stored data from the cloud provider. Therefore, it is necessary to utilize an encryption method before uploading the data to the cloud storage server. For example, a user can encrypt the data using a symmetric-key algorithm before uploading the data. However, to share the data, the user needs to share the symmetric-key, thus granting the shared users access to all data encrypted with this key. Another method would be to encrypt the data using an asymmetrical-key encryption scheme to encrypt the data with the private key before uploading the data to the cloud. In this case, to share data, the user needs to download the data, decrypt it using the corresponding private key, and then re-encrypt the file using the shared user's public key before uploading the file again. This method has many drawbacks. First of all, there will be many copies of the same file encrypted with a different key for every shared copy. Secondly, when a file is updated, it needs to be re-encrypted again for every shared copy of the file.

Identity Based Encryption (IBE) was first proposed by Adi Shamir in 1984 [41].

IBE generates users' public keys based the unique user information, such as a user's email address. This allows users to use the public system parameters to generate the public keys of other users without having to store a list of all the users' and their public keys. When using Biometrics as identities for Identity Based Encryption, the values of the biometric feature readings contain noise. Sahai *et al.* [42] tackled this problem and in doing so, they also introduced what is now known as attribute based encryption. They suggested that if we use w as the Biometric Identity, then a second reading of the biometric features would be w', which would be close to the value of w. They also proposed a Fuzzy Identity Based Encryption scheme which can use w' as an Identity to decrypt messages encrypted using w as the Identity. They then explain that Fuzzy-Based Encryption can be used as an application for "Attribute-Based Encryption". In ABE, ciphertexts are associated with attributes. To Decrypt the ciphertexts, a user must have all the attributes needed to decrypt the file.

To achieve Fuzzy IBE they suggested that each user in the system would have a set of attributes. These attributes could be roles, or in the case of Biometrics the features would be split into attributes. A user would be able to decrypt a file if he has at least k attributes of the attributes used by the encrypter to encrypt the file. Since FIBE is intended for error tolerance when using IBE, it only supports access structures in the shape of a threshold gate. ABE, on the other hand supports Linear Secret Sharing Scheme (LSSS) realizable access structure.

Matthew Pirretti, Patrick Traynor, and Patrick McDaniel [43] later proposed a more secure Attribute Based Encryption method. They implement more complex policies for Attribute Based Encryption based on the work done by Sahai *et al.* [42]. In their method, they allow complex policies such as having '*and*' and '*or*' logical policies. They also applied their method to a medical application where the patients' medical records are only

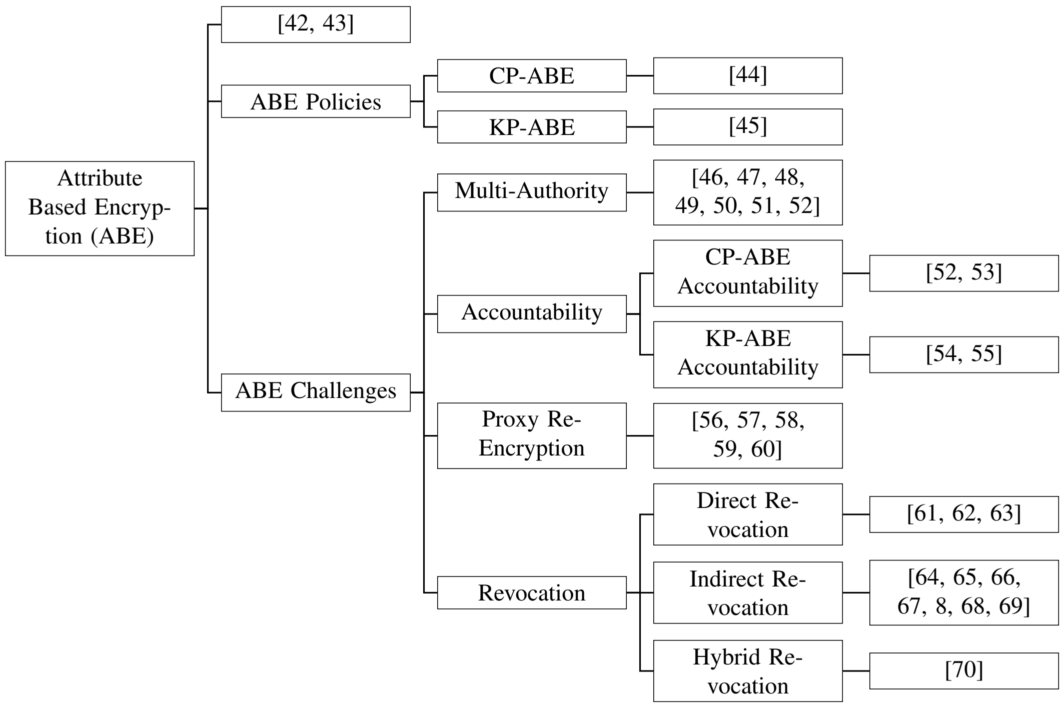


Figure 2.1: Overview of the Surveyed Attribute Based Encryption Research Works Taxonomy

available to entities with the proper attributes.

There are two basic ABE schemes, namely cipher-based attribute based encryption (CP-ABE) [44] and key-policy attribute based encryption (KP-ABE)[45]. The difference between these two schemes is what the attributes describe as shown in Figures 2.2 and 2.3. In CP-ABE the attribute access structures are used to describe the encrypted data, while in KP-ABE the attribute access structures are used to describe the user's key. CP-ABE is similar to RBAC, while KP-ABE is similar to ABAC.

Many extensions have been proposed by researches to address the problems of multi-authority, accountability, proxy re-encryption, and revocation. In an ABE scheme, an authority is responsible for generating a key pair for every user after verifying the user's identity. However, there are many cases where having a single trusted authority for all users

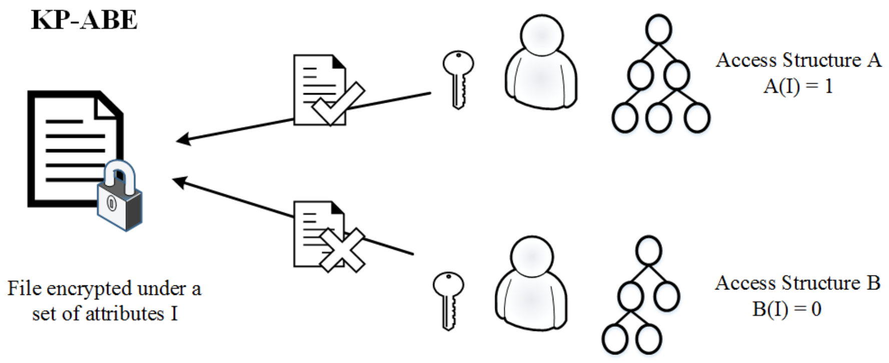


Figure 2.2: KP-ABE

in the system is not possible. Chase *et al.* [46] was the first to propose a Multi-authority ABE scheme, other multi-authority schemes were later proposed in [47, 48, 49, 50, 51, 52]. Accountability in ABE is needed to prevent key abuse. To achieve accountability in ABE, two kinds of abuses are considered in the literature. The first, is when users collude and illegally share their private keys. The second is when a semi-trusted ABE authority misbehaves by illegally generates legitimate keys and distributes them to unauthorized users. CP-ABE accountability was address in [52, 53], and KP-ABE accountability was addressed in [54, 55].

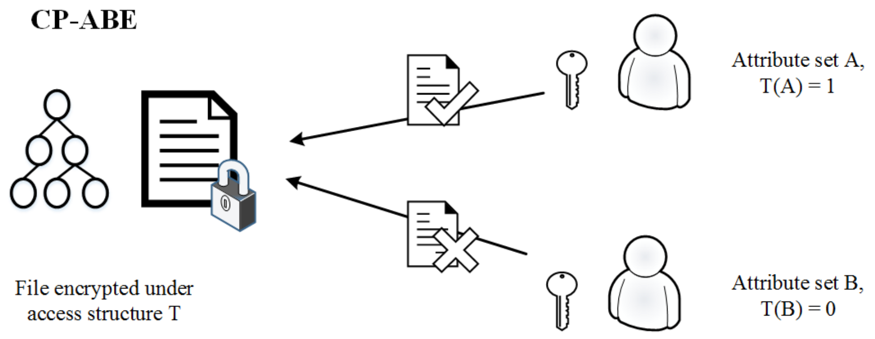


Figure 2.3: CP-ABE

Proxy re-encryption (PRE) allows a proxy to convert a ciphertext encrypted for a user to a ciphertext, which may be decrypted by a different user, without having to know the private key of either of these users. For example, this would allow a user to go on a vacation and delegate his work load to another colleague. The proxy in this case would temporarily forward all incoming encrypted emails to the colleague after re-encrypting them. Proxy re-encryption in ABE has been addressed in [56, 57, 58, 59, 60]. ABE-PRE allows a user to delegate designated users to decrypt proxy re-encrypted ciphertexts using the associated attributes of the delegated user.

Revocation is also a challenging problem in an ABE system. The key pairs issued by the authority in an ABE system correspond to the users' attributes. This allows users to decrypt any file where their attributes satisfy a ciphertext's access policy. Access revocation becomes difficult because once a user is granted a key, users can store their keys anywhere and make an as many copies they want. Since the system is based on enforcing access control cryptographically, revocation also has to be enforced the same way and not by the system. The complexity of the problem also depends on whom access is being revoked from. If access to a file is being revoked from an entire attribute, such as revoking access to a file from an entire department, the simple solution would be to re-encrypt the files using a new symmetric key after updating the access policy tree. The problem becomes more complex and difficult when access is being revoked from a single user that has many attributes shared with other users.

For example, a user that works in the IT department in Montreal, would have access to all files that permit access to employees in the IT department, in Montreal, or both. In this case, the keys of all users that share attributes with the revoked user need to be changed. All files related to these attributes would also need to be re-encrypted and the access policies would need to be updates with the keys. Proposed works to address revocation are split into two main categories:The first is the indirect revocation method [64, 65, 66, 67, 8, 68, 69], and the other is the direct revocation method [61, 62, 63].

In the indirect revocation method, the data owner delegates a third party to execute revocation. This releases a key update periodically preventing revoked users from updating their keys. The drawback of this method is that it requires the authority periodically communicate with all non-revoked to execute the key update phase.

In the direct method of revocation, it is the responsibility of the data owner is to revoke access. This is done by the data owner by specifying a revocation list when encrypting the ciphertext. The advantage of this method is that it does not require all non-revoked users to update their keys. The drawback of this method is that the data owner needs to keep and manage a list of all revoked users, which can become long and troublesome with time. Attrapadug *et al.* [70] proposed a hybrid ABE revocation scheme (HR-ABE), which takes advantage of the direct and indirect methods. It allows the data owner to select the encryption scheme including specifying whether to use the direct or indirect revocation method. However, their solution only supports user revocation but not attribute revocation.

2.1.3 Threshold Cryptosystems

In a (t,n) threshold scheme [71], a secret S is split into n shares and distributed to the participants. To reveal the secret S, any t participants must work together and use their share to calculate the secret S. Threshold schemes also ensure that when $t-1$ participants work together, it would not be possible for them to calculate S nor gain any information about it.

2.1.4 Secure Computation

In this section, we discuss the research works related to secure computations, which is a subfield of cryptography that aims to create methods to enable parties to compute mathematical operations while keeping the inputs private.

Rivest *et al.* [72] introduced the concept of homomorphic cryptography, which allows

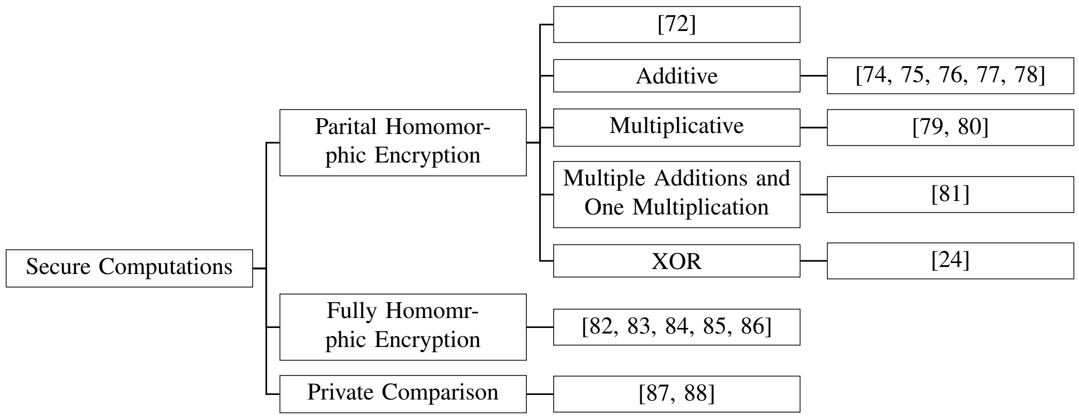


Figure 2.4: Overview of the Surveyed Secure Computation Research Works Taxonomy

certain algebraic operations on two encrypted values, without any intermediate decryptions. Given two ciphertexts $c1 = E_{pk}(m1)$ and $c2 = E_{pk}(m2)$ of messages $m1$ and $m2$ with the same public key pk. A cryptographic homomorphic scheme can compute E_{pk} ($m1$ op $m2$), an arbitrary operation op on two encrypted values without having to decrypt either value. Variants of homomorphic algorithms were designed by researchers to do computations on encrypted data. All these proposal could be broadly classified into Partial Homomorphic Encryption (PHE) schemes and Fully Homomorphic Encryption (FHE) schemes [73].

Partially Homomorphic Encryption (PHE)

Partial homomorphic encryption schemes are limited to the number of operations they can securely compute. For example, some schemes either support addition or multiplication operations on encrypted ciphertexts due to its inability to decrypt after a threshold of noise it reached by the operations. Some of the most commonly used partial homomorphic crytosystems are Paillier [74], Unpadded RSA and ElGamal [89], Goldwasser-Micali [24], Benaloh [76], Damgard-Jurik [75], and Boneh-Goh-Nissim [81].

The Paillier scheme [74] is one of the most popular PHE methods. Using Paillier's protocol, given two ciphertexts $E(x)$ and $E(y)$, an encryption of their sum $E(x+y)$ can be efficiently computed by multiplying the ciphertexts modulo a public key N, i.e., $E(x+y) = E(x).E(y) mod N$. The cost of Paillier encryption is not high, but the cost of decryption is higher because it needs one exponential modulo n to the power $\lambda(n)$ and a multiplication modulo n. In 2002, Cramer*et al.* [77] proposed an extension to the Pallier cryptosystem to prevent chosen ciphertext attacks. Damgard-Jurik *et al.* [75] propose a generalization of the Paillier scheme to reduce the expansion value. The solution they propose aims to reducing the expansion value to one in some cases, but the solutions was computationally more intensive compared to the Paillier scheme. Galbraith *et al.* [78] propose an Elliptic Curve Cryptosystem (ECC) for homomorphic encryption based on a one-way trapdoor function. Their scheme uses the algebraic structure of elliptic curves over finite fields. ECC has the advantage of having a smaller key size, thus reducing the storage and communication requirements. However, ECC has a very high computational cost for key generation and decryption. Boneh *et al.* [81] propose a scheme based on bilinear pairings on elliptic curves and is able to compute multiple additions and a single multiplication operation. Their scheme required a larger message space to compute discrete logarithms during the encryption process. The Goldwasser-Micali (GM) cryptosystem is a semantically-secure protocol based on the quadratic residuosity problem [24]. It has XOR homomorphic properties, in the sense that $E(b).E(b') = E(b \oplus b')$ mod N, where b and b' are bits and N is the public key. Variations of the homomorphic Paillier and GM cryptosystems are the distributed threshold decryption protocols in which the decryption is performed by a group of participants rather than one party [90, 91]. In this case, each participant would obtain a share of the secret key by executing the distributed key generation algorithm detailed in [92].

Fully Homomorphic Encryption (FHE)

Cryptosystems that can compute addition and subtraction on encrypted data are considered fully homomorphic. Many FHE works have been proposed [82, 83, 84, 85, 86]. Gentry *et al.* [82] discusses FHE in his doctoral thesis and proposes a scheme that can be used to compute complex operations on encrypted data. His scheme consists of three modules: Ideal lattices based on SomeWhat Homomorphic Encryption (SWHE), a bootstrapping module that transfers SWHE to FHE using bootstrapping. The scheme is based on hardness approximating problems within sub exponential factor. It uses the parameter per gate evaluation time (the ratio of time for homomorphic evaluation of the circuit to the time for evaluating circuit on plaintext). Gentry's work became the blueprint, which researchers are building on to improve the efficiency and practicality of FHE. Smart *et al.* [85] present a FHE scheme that improves efficiency by using smaller keys and ciphertexts. However, the scheme they propose requires more time to generate keys. Gentry *et al.* [23] continued the work done by Smart *et al.* [85]. They improve the key generation process and simplify the decryption, which help reduce the per-gate-evaluation time. However, the processing resources required are too high for the solution to be feasible for most applications. In an optimized implementation on a high-end workstation for a large setting, key generation takes 2.2 *hours* and encryption takes 3 $minutes$. Stehle *et al.* [93], propose a faster FHE scheme by improving Gentry's FHE scheme on ideal lattices [94]. They implement a probabilistic decryption algorithm with a circuit of a low multiplicative degree. The hardness assumption of the security is stronger in their work compared to Gentry's scheme. In spite of the many improvements to Gentry's scheme, FHE remains too slow and expensive when compared to PHE.

Private Comparison

Yao's classical millionaires' problem [87] involves two millionaires who wish to know who is richer. However, they do not want to inadvertently find out any additional information about each other's wealth. More formally, given two input values x and y, which are held as private inputs by each party respectively, the problem is to securely evaluate the Greater Than (GT) condition $x > y$ without exposing the inputs. In this paper, we employ Fischlin's protocol [88] for private comparison because it allows us to compare two ciphertexts encrypted with the GM crytosystem using the same public key. Fischlin's protocol takes as input two ciphertexts encrypted using the GM cryptosystem and produces ciphertext sequences, namely Δ and c, that are encrypted by the same public key. The decryption of these sequences reveals the result of comparing the private inputs without revealing anything beyond the result of the comparison. Fischlin's protocol utilizes the XOR-homomorphic GM cryptosystem to privately compute:

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

where $|x| = |y| = n$.

2.2 Related Work

In this section, we present a review of state-of-the-art techniques developed in the areas of secure and privacy-preserving querying of privately encrypted data in the cloud. This section is organized as follows:

In Section 2.2.1, we present the contributions related to electronic health records (EHRs) and securing storing and querying them in the cloud. Section 2.2.2, we discuss the works related to searching and querying encrypted data in the cloud. Finally, section 2.2.3, presents the contributions related to secure multi-party computation and garbled circuits.

2.2.1 Electronic Health Records

In this section, we present an overview on electronic health records and the literature related to securely storing and analyzing health data on the cloud. Healthcare has evolved throughout the past two decades. Rising service costs and an aging population are challenges that face healthcare systems. To overcome these challenges, governments and healthcare providers are searching for ways to improve the efficiency of their healthcare services. Moreover, they are investing vast sums of money into information and communications technologies, including decision support systems, telemedicine, and electronic records.

Furthermore, there are many benefits to using electronic health records with mobile networks. Using an efficient mobile healthcare system on the cloud can dramatically reduce the cost of healthcare. It eliminates a healthcare provider's need to buy and maintain servers. Mobile health can provide high availability of sensitive medical records remotely, which can help saving a patient's life. It would also enable doctors to remotely monitor their patients' conditions.

Mobile health can also help during epidemic and disease outbreaks by warning patients that might be prone to catch the disease. The Global Health Organization recently released a survey on mobile health, which shows that most mobile health projects are on a small scale that are tailored to address specific problems in data sharing and access[95].

There have been many fatal and highly contagious diseases throughout history. The Bubonic plague swept through Europe in the 14th century and killed an estimated 25 million people, or 30-60% of the European population. In 1994, there was a plague outbreak in India causing an estimated 700 infections, which resulted in 52 deaths. In that same year, there were also cases reported in Peru. Another case was reported in 2010 in Oregon, United States [96].

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 a few days. There are many organizations, which work on studying epidemiology and preventing them from spreading around the world. Recent examples of newer diseases include mad cow disease, SARS, swine flu, etc.

To better understand what caused a disease, researchers need as much data as possible about the infected patients. After a link is found between the disease and the infected patients, it becomes crucial to prevent the disease from spreading further by informing and taking care of the people that might be vulnerable to the disease.

In a perfect world, electronic health records would be stored in a single location for every person in the world. Unfortunately, electronic health records are usually managed by hospitals, making it even more difficult to get a single patient's history due to the fact that it might be spread between multiple hospitals and pharmacies. What makes combining these databases together even more difficult, is the cost of creating and enforcing the use of a standard that can be trusted by all institutions.

EHR (Electronic Health Records) are groups of electronic health information about patients or populations. The law in Canada gives patients the right to access their Personal Health Information (PHI) that is stored in medical records systems [97, 98]. However, not all the information about a patient in EHR is PHI. EHR records may include data such as medical history, medication and allergies, immunization status, laboratory test results, radiology images, vital signs, personal stats like age and weight, and billing information. Personal patient data in the EHR are referred to as Personal Health Records (PHR).

Since patients have the right to access and control their PHR, many researchers and service providers began to implement solutions to make use of PHRs [99]. These solutions have been able to use PHR to give patients statistics about their health records, raise flags if the patient is vulnerable to a new medical epidemic, remind patients to perform certain procedures if needed, raise flags if the patient is taking interacting drugs that can cause side effects, etc.. In this section we will compare some of the solutions available in the market from a security point of view.

In the field of secure personal health records which are managed and controlled by patients, there are many research proposals [18, 17, 100, 101, 102, 103, 104, 105, 106]. Hembroff *et al.* [102] and Neubauer *et al.* [105] propose the use of smart cards for encryption. Microsoft HealthVault [17], Dossia [18], and Ueckert *et al.* [106] encrypt health records using symmetric keys, while all communication is encrypted using Transport Layer Security (TLS). Recently, the executive office of President's Council of Advisors on Science and Technology (PCAST) has reported that more extensive cloud-based solutions and services in the future should support public health reporting and basic clinical research [107].

Indivo is a free open-source open-standard personally controlled health record (PCHR) system that gives patients control over their medical records [108] . Indivo provides users with the ability to share their records with different physicians, hospitals, and clinic. In 1998, researchers at the Children's Hospital Informatics Program (CHIP) at Children's Hospital in Boston introduced the concept of Indivo in a planning grant. In 2005, the project was implemented under the name personal inter-networked notary and guardian (PING) and renamed to Indivo in 2006. The project was designed since Indivo 1.0 as a distributed, cloud-based online system that provides an API for connecting to user specific Person Health Applications (PHA). Indivo also provides an API that enables healthcare providers to develop modules and services to the existing system. In the latest release, Indivo is actively deployed as the backbone of various commercial PHR systems such as the Dossia consortium and the Children's Hospital in Boston.

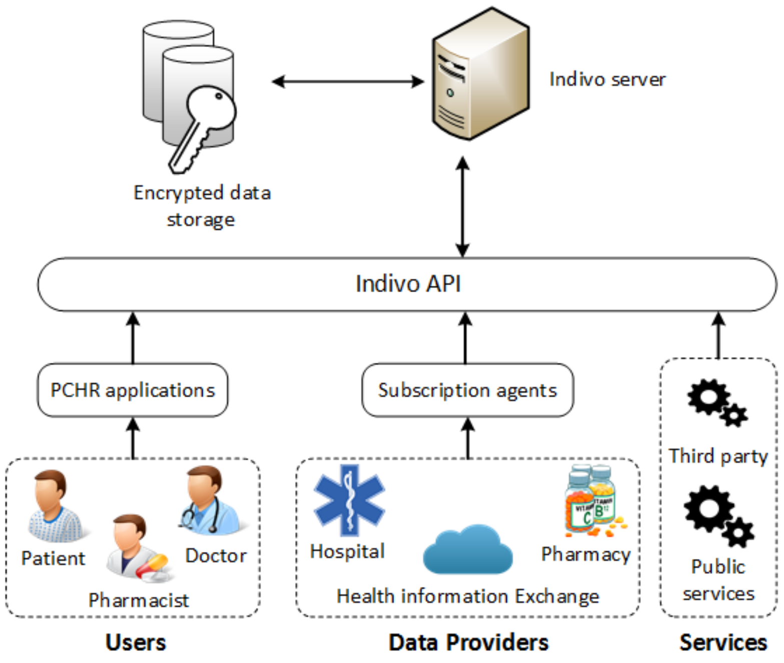


Figure 2.5: Indivo System Architecture

The main components of the Indivo system as shown is figure 2.5 include the Indivo API, an encypted storage, and a staging server [108]. The PCHRs are stored on a backend server in encrypted form. Unlike Microsoft Health Vault and Google health, Indivo stores all data in XML documents. Indivo uses PostgreSQL relational database for data storage. To maximize performance, database level encryption is used to provide data security. However, this does not protect against an adversary who has access to the data or the server. Although adding column or table level encryption can increase security, it is not the optimal way to provide patient-centric, fine-grained access control. The reasoning behind the weakness of adding column or table level encryption is that the keys would be on the server while they are being decrypted and sent to the client. This provides the adversary with a window to intercept the data and the keys. Moreover, the client would need to use multiple asymmetric key pairs to achieve fine-grained access control.

In the domain of securely storing health records in the cloud, Narayan *et al.* [7] show how Attribute-Based Encryption (ABE) scheme can be used to construct a secure and a privacy-preserving Electronic Health Record (EHR) system that enables patients sharing data among healthcare providers in a flexible manner. Whereas an electronic health record (EHR) is a computer record that originates from and is controlled by doctors, a personal health record (PHR) can be generated by physicians, patients, hospitals, pharmacies, and other sources but is controlled by the patient. The EHRs are stored on an untrusted cloud storage and encrypted using symmetric key cryptography. The proposed keyword search functionality allows the patient choosing what terms may be searched, and who may be able to access the search terms. For example, a patient may allow a certain hospital searching the terms "Diabetes" and "male". Narayan *et al.* [7] also assume that there is a trusted authority that generates keys for users of the system. Similarly, Li *et al.* [109] and Akinyele *et al.* [110] leverage ABE techniques to encrypt health records under the assumption of the existing of the trusted authority.

2.2.2 Searchable Encryption

In this section, we present the literature in the field of searchable encryption, which includes secure single keyword search, Multi-keyword search, ranking encrypted search results, subset and range queries on encrypted data, and searching privately encrypted databases.

The primary concern of data owners in the cloud, is protecting their privacy and the confidentiality of their data. Homomorphic cryptosystems address the issue of computing data confidentiality. In this section, we discuss the contributions related to the problem of

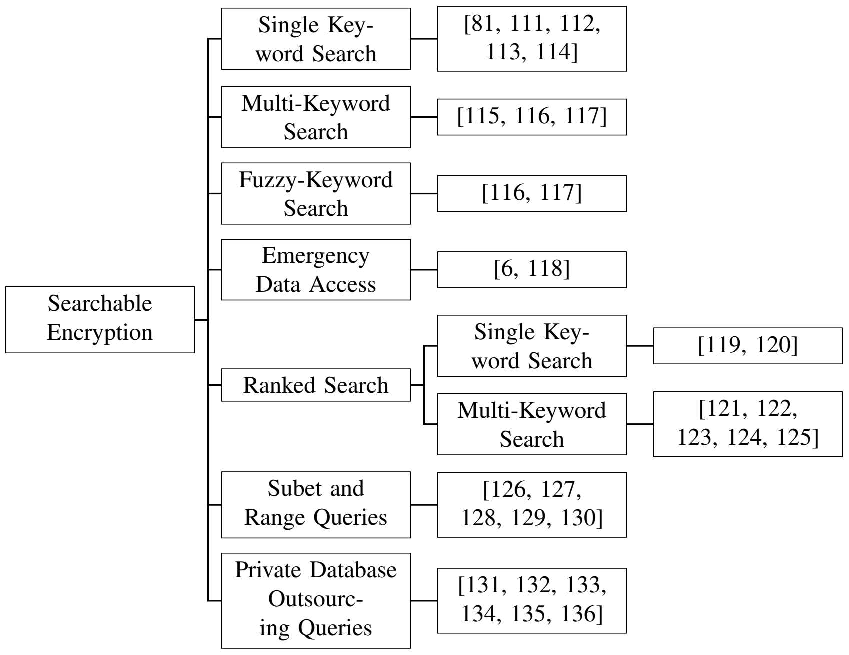


Figure 2.6: Overview of the Surveyed Searchable Encryption Research Works Taxonomy

preserving the privacy of data owners when searching encrypted data in the cloud. Researchers have proposed numerous works to address the issue of querying encrypted data on the cloud.

Single Keyword Search

The most trivial search functionality is the single keyword search. Song *et al.* [111] are the first to address the concept of searching encrypted data. Their scheme used two layers of symmetric encryption to encrypt each word independently in the file. The first layer uses a symmetric key with a secret key, while the second uses a pseudo-random number generator and two pseudo-random functions as shown in figure 2.7. However, their method only allowed querying for equality. Goh *et al.* [112] later proposed an efficient search technique based on bloom filters, which uses a separate index file for each document. However, the scheme results in space inefficiency. Chang *et al.* [113], proposed a scheme based on the Searchable Symmetric Encryption (SSE) method. However, adaptive queries by an adversary can lead exposing the privacy of the data owners. Curtmola *et al.* propose an extension in [114] to solve the privacy issue. They propose using hash indexing for the entire file. Their scheme had the drawback of hiding the access patterns of specific users. Boneh *et al.* [81] later proposed the first searchable encryption "Public key encryption with keyword search", which supported a multi-user model. In the construction of their scheme, any user with the public key can write data to the cloud, but only users with the private key can search the encrypted data.

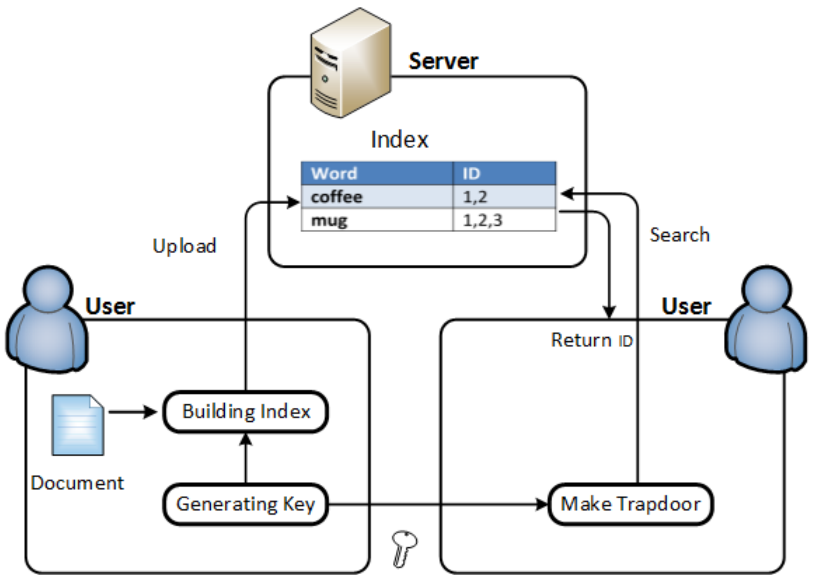


Figure 2.7: Searchable Encryption General Architecture

Multi-Keyword Search

Multi-Keyword gives users the ability to search encrypted collections of files for keywords in a conjunctive matter. Golle *et al.* [115] propose two schemes, which both supported conjunctive keyword searching. Li *et al.* [116] proposed a fuzzy keyword searching scheme, which tries to match a keyword to an encrypted file. If the keyword is not found, then a search for a close match of the keyword is executed and the corresponding file is returned.

An improved method that works on symmetric key encrypted data is proposed by Jianhua *et al.* [117]. Ranking keyword search was first proposed by Cao *et al.* [121]. In their proposed scheme, multi-keyword queries are searched linearly in the database and the top-k most relevant documents are returned to the user. The relevance is based on the number of keyword matches between the encrypted files and the search query. Wang *et al.* [119, 120], proposed a ranked keyword scheme, where the relevance was computed using an inverted index for keywords in the database. The scores are encrypted with an order-preserving symmetric encryption scheme. Sun *et al.* [122, 123], proposed a multi-keyword ranked encryption scheme based on similarity ranking. The scheme ranks search indexes on term frequency and vector space. Efficiency is improved in this scheme by using tree structures. Bouabana-Tebibel *et al.* [124], proposed a method combining attribute based encryption with searchable encryption and adopted the access control mechanism to protect confidentiality. Xia *et al.* [125], proposed a scheme that applies latent semantic analysis to find the relationships between terms and documents.

Emergancy Access

Ming Li*et al.* [6] proposed a method for providing access to emergency private health records (PHRs) encrypted with ABE during an emergency. They suggested using a break-glass approach, where each patient's PHR's access rights are also delegated to an emergency department. To protect the PHR break-glass option from being abused they suggested that the ER staff need to contact the emergency department to verify their identities and verify the emergency case. The emergency department would then give temporary read keys. After the emergency, the patient may restore normal access by computing another key and then submit it to the emergency department and the server to update the files.

Although their proposed solution provides access in emergency cases, they do make the assumption of trusting emergency departments. Any employee in an emergency department would be able to access any emergency PHRs in the system. In our work, we do make the assumption that none of the entities in the system are completely trusted. We also make the assumption that the access to PHRs is not achieved by emergency staff accessing them using their personal keys. In our work, we simplify the process by giving all Emergency Care Providers (ECPs) the same attributes and keys.
Gardner *et al.* [137] proposed a different method which did not rely on attribute based encryption. In their work they assume that the PHRs are to be stored on a smart phone, and their goal is to protect the PHRs stored on the phone from being accessed by an adversary. Moreover, their approach might be modified to access Emergency Medical Records (EMRs) stored on an untrusted cloud. They propose using secret sharing [71] to split the decryption key and distribute it to different entities. By dividing the access capabilities, it becomes possible to associate the credentials of different entities with varying weights. The partial weighted rights are then distributed in a way where no one credential is sufficient to obtain access, but appropriate combinations are.

They used secret sharing to distribute the decryption key k_r between the following entities:

- Break-The-Glass (BTG): a share that is used only as a last-resort access mechanism
- Password: a share accessible with the PHR owner's password
- EP: a share that is available to ECPs
- Face: a share that is accessible with the PHR owner's face
- Finger: a share that is accessible with the PHR owner's fingerprint

To access the PHRs, the decryption key k_r could only be recalculated by satisfying the following policy:

$$\{BTG \vee \{password \wedge EP\} \vee \{\{password \vee EMT\} \wedge \{face \vee finger\}\}\}$$

This means that patients would be able to view their PHRs at anytime by using the password and taking a photo of their face, or using the password and their fingerprint. However, in case of an emergency an adversary would not be able to recover the PHRs by using the patient's face or fingerprint since the password is needed. However, an EP would be able to use their password to gain access to the patients' PHRs.

Although this solution could be applied to provide access to PHRs stored on an untrusted cloud. We do not think it could be used as a global solution because this would mean that all medical health providers would be using the same key. This implies that any ECP would be able to view all PHRs on the cloud. Gardner *et al.* [118] discussed this problem and suggested three methods for solving this problem.

- The first method aims at providing every EP with a different key pair. However, this would mean that the client software must store a corresponding ciphertext for each key. This would obviously have a huge storage overhead if looked at as a global solution.

- The second solution consists of using a public-key broadcast encryption system such as that proposed by Boneh*et al.* [138], a trusted entity would generate a private key for every EP and a broadcast public key. This entity would program the private keys onto a smart card for each EP and maintain a list of non-revoked keys. The client software would use this list and the broadcast public key to encrypt the EP share.

- The third method consists of updating the EP keys everyday, but this would also require the patient to download the new keys and re-encrypt the PHRs every day.

Subset and Range Queries

Range and subset queries allow users to query encrypted data for values between a given upper and lower bound. A method allowing range queries is suggested by Agrawal *et al.* [126]. In their work, an encryption protocol is proposed allowing for the comparisons to be executed on the encrypted data directly. It is assumed, in their work, that the order of the data was confidential. Boneh *et al.* [127] proposed a scheme based on Hidden Vector Encryption (HVE). The scheme allows queries on encrypted data to produce tokens for testing supported query predicates. Tokens allow users to test the predicate of a ciphertext without leaking any information on the plaintext. This allows for comparison and subset queries. Shi *et al.* [129] proposed a scheme for conjunctive range queries. The scheme uses points of a multidimensional space to represent the queries. To execute a range query, the server tests whether the point is located in a hyperrectangle of the multidimensional space. Boldyreva *et al.* [128] proposed a searchable encryption scheme that supports range queries by improving on previous order-preserving encryption schemes. They introduce the modular order preserving encryption scheme, which is more secure than previous order preserving scheme and allows for range queries. Li *et al.* [130] proposed a scheme for range queries by applying hierarchical predicate encryption to construct a flexible searchable encryption scheme.

Private Database Outsourcing

Private database outsourcing also known as database-as-a-service model deals with the problem of hiding database records from an untrusted cloud server. A common approach in the existing research proposals is to send a set of encrypted records to the client for further processing and filtration [131, 132, 133, 134, 135, 136, 139]. The major problem with this approach is that finding the required records, on which the query is executed, is a significant challenge. In order to solve this problem, current proposals have revealed some information about the query to be used in filtering the required tuples [132, 139]. With a good filtration mechanism, the communication cost of retrieving data from service providers is less. However, the quality of the filtration process depends on the amount of information revealed to the service provider. Instead of using encryption, Aggarwal *et al.* [140] propose secret sharing to hide database records from the service provider. They require multiple non-colluding service providers to keep the share of each database records. These techniques keep the database confidential from service providers and outsider adversaries but they do not protect database privacy. CryptDB [141] works by executing SQL queries over encrypted data using a collection of SQL-aware encryption schemes including order-preserving and deterministic ones. It works by intercepting all SQL queries in a database proxy, which rewrites queries to execute them on encrypted data as shown in figure 2.8. The proxy encrypts and decrypts all data, and changes some query operators, while preserving the semantics of the query. As mentioned previously, the symmetric key and the deterministic cryptosystems are not semantically secure. CryptDB also has a trusted proxy that represents a single point of failure.

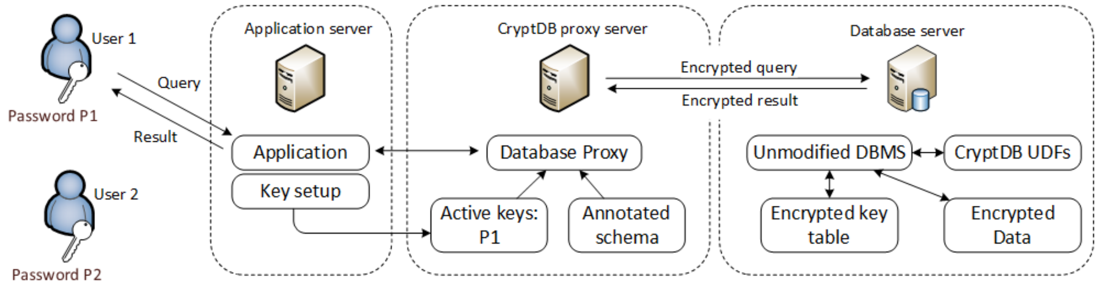


Figure 2.8: CryptDB Architecture

Similar to CryptDB, MrCrypt [141, 142] also uses partial homomorphic cryptography to execute computations on encrypted data. However, neither of them protect the confidentiality of code, nor guarantee the integrity of the computation results or their completeness. Other secure databases systems are also proposed in [143, 144, 145, 146], but unlike CryptDB and MrCrypt, these solutions require trusted hardware. Monomi [145], TrustedDB [146], and Cipherbase [144] use different types of hardware to securely execute queries on encrypted data. These solutions also lack the ability to protect the confidentiality of the code and the integrity of the data. Monomi splits the computations between users and an untrusted server by utilizing PHE. Mylar [143] is a system that allows user to build web applications that support secure searches on encrypted data.

2.2.3 Secure Multi-Party Computation

Secure Multi-Party Computation (MPC) is appropriate for the semi-trusted cloud setting. MPC takes advantage of the participation of honest parties, without having to know which of them are honest, to satisfy the confidentiality and integrity of the data and computation. Security guarantees are higher in FHE, but MPC can be more efficient and practical. The practicality and relative efficiency of MPC, and its applicability to the real world model of a semi-trusted cloud, make it a promising choice for use in secure cloud computations.

Secure computation is initially proposed by Yao [147] and Goldreich *et al.* [148] and later extended to the multi-party case by Chaum *et al.* [149], and Ben-Or *et al.* [150]. The threshold adversary model is used by most MPC schemes; this limits the number of parties that an adversary can corrupt to t out of n, where t is the threshold and n is the total number of parties. These schemes share the input data between the parties in a way that ensures that no set of less than t shares reveals anything about the input data. Computing on the input shares enables the parties to compute shares of the output that the receiver can reconstruct to get the real output. MPC ensures that the parties do not learn about the input or output data because the participating parties only have access to their own shares of the data.

Researchers have proposed a number of practical MPC implementations. For example, a solution was proposed by Bogetoft *et al.* [151], which allowed Danish farmers to agree the price of sugar beets by using MPC in auctions. Several MPC protocols can be run using the VIFF library [152]. Bogdanov *et al.*proposed Sharemind [153]. Burkhart *et al.* proposed [154] Sepia. Ejgenberg *et al.* [155] proposed the Secure Computation Application Programming Interface (SCAPI), which is an open-source Java library for implementing MPC protocols.

The idea of garbled circuits is first proposed by Yao in [156]. In his secure two-party setting, a circuit is created with two inputs and a single output. Each input and output can accept a single bit. In the protocol Yao proposed, Alice wants to securely compute the value of a circuit with two inputs and a single output, where she provides the first input and Bob provides the second input:

- Alice creates the circuit and generates 6 keys corresponding to the 6 possible binary values for the two inputs and the output as shown in Figure 2.9 and Table 2.9.
- Alice then sends the key corresponding to her input bit to Bob along with a garbled truth table of four values corresponding to encrypted values of the two possible outputs keys.
- Alice sends Bob his two possible input keys using the 1-out-of-2 oblivious transfer protocol.
- Bob chooses the key corresponding to his input it and uses it along with Alice's input key to lookup the correct value in the garbled truth table to decrypt the output key.

- Bob then sends the output key to Alice who looks up the actual output bit mapped to that key.

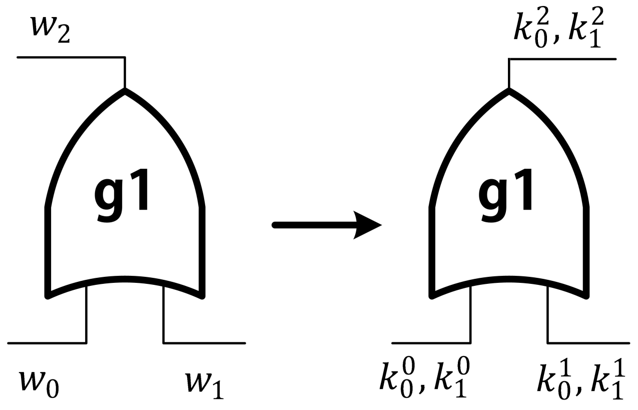


Figure 2.9: Garbling a Circuit

w_0	w_1	w_2
0	0	0
0	1	1
1	0	1
1	1	1

w_0	w_1	w_2	Garbled Values
k_0^0	k_1^0	k_2^0	$H(k_0^0 \| \| k_1^0 \| \| g_1) \oplus k_2^0$
k_0^0	k_1^1	k_2^1	$H(k_0^0 \| \| k_1^1 \| \| g_1) \oplus k_2^1$
k_0^1	k_1^0	k_2^1	$H(k_0^1 \| \| k_1^0 \| \| g_1) \oplus k_2^1$
k_0^1	k_1^1	k_2^1	$H(k_0^1 \| \| k_1^1 \| \| g_1) \oplus k_2^1$

Table 2.1: Garbled Circuit Truth Tables. Original Values (left), Garbled Values (right)

Many works are proposed to optimize Yao's protocol and make it less expensive [157, 158, 159, 160, 161, 162]. These contributions fall into three main categories:

- **Communication optimization** the aim is to reduce the amount of data that must be transferred between the parties. The cost of transmitting a garbled circuit from P_1 to P_2 is the highest of all other communication steps in Yao's protocol. This is due high number of gates a circuit can contain; in some cases it can grow to billions of gates.

Each gate in the circuit is connected via a wire, which is represented by four multi-byte strings. The problem gets worse in the case of a malicious setting, where the cut-and-check strategy requires the first party to send multiple copies of the circuit. Reducing the size of the data communicated between the parties and the number of communication steps is a significant issue in making Yao's protocol efficient.

A solution is proposed by Goyal *et al.* [157] to optimize Yao's protocol. Their scheme included two modifications to reduce the communication costs of the protocol. In first modification, the first party deterministically calculates the values for the wires in the circuit by using a random seed, instead of having to assign a random value for each wire. In the second modification, the first party does not need to send m number of copies of the circuit during the cur-and-check phase. Instead, commitments are used, where the first party sends "commitments" for each version of the circuit. This allows P_2 to generate the garbled circuit from the random seed and the structural information for the circuit sent by P_1. This technique reduces the communication overhead of the protocol significantly because the sizes of the commitments are constant and much smaller than the size of the circuit.

- **Execution optimization** aims at reducing the computation time needed to execute the same number of gates. These optimizations deal with the efficiency of securely evaluating a garbled circuit without having to alter the circuit's structure. The first method reduces the time needed to evaluate the circuit by introducing faster table lookups. This method was proposed by Milkhi *et al.* [158] and further optimization is proposed by Huang *et al.* [160]. This technique does not require P_2 to decrypt each row of the truth table until a value is found that decrypts correctly. Instead, a bit is added to each garbled output. This bit also serves as half of an index to the following gate's garbled truth table. Combining the index bits can identify which of the four rows can decrypt from the following gate's garbled truth table. The index values do not reveal any information about the underlying value due to the randomization of the rows in the garbled truth table during the construction of the circuit.

The second execution optimization method reduces the execution time by pipelining circuit execution. Garbled circuits can be large for even simple function, making it difficult to store the circuit for the generating and computing parties. Evaluating the circuits can also take a large amount of time to evaluate because P_2 waits in the protocol while P_1 garbles the circuit. Huang *et al.* [160] proposed that the execution and garbling processes may be partially done in parallel. Their solution had the advantages of requiring less storage because none of the parties need to store the entire circuit in the memory. The solution was also faster due the execution pipelining. However, this solution is only possible in the semi-honest model due to the cut-and-choose technique needed in the malicious model, where P_1 needs to create multiple variations of the circuit as previously mentioned. Kreuter *et al.* [161] proposed a circuit pipelining method for the malicious model. In their solution, P_1 generates each garbled circuit twice, the first is done before P_2 chooses which circuit to verify, and the second after P_2 makes a choice. This solution also optimizes the execution by not requiring either party to store the entire circuit and the time needed to execute a circuit is reduced due to pipelining.

- **Circuit optimization** this aims at reducing the number of gates needed to compute a function. The objective of the first method is to simplify the circuit. Circuit optimization techniques were proposed by Pinkas *et al.* [159] and Kreuter *et al.* [161]. Reducing the gates in a pre-garbled circuit can be achieved by adding a pre-processing step to optimize the circuit before garbling. This is done by looking for unnecessary gates, which do not have an effect on the final output. These gates can be identity gates or sub-circuits that are guaranteed to result in the same output. In fact, Pinkas *et al.* [159] found that optimizing circuits generated from a common circuit generator can result in a 60% reduction to the size of the circuit.

A "free XOR" method was proposed by Kolesnikov *et al.* [162] to reduce the number of gates needed to evaluate a garbled circuit. This is done by replacing all garbled XOR gates in the circuit with a simple XOR operation. Instead of needing to lookup and decrypt the output value in a garbled truth table, four garbled values from the circuit are removed for each XOR gate.

Moreover, Pinkas *et al.* [159] also proposed a solution to further reduce the number of garbled values that need to be stored in the garbled circuit. Their solution removes the need to store a garbled value from each AND and OR gate in the circuit. This is achieved by building on the fast table lookup technique and choosing one of the four indexes in the gate's garbled truth table for reduction. Instead of needing a new pseudo-random value, the garbled output value for the chosen wire is designed to be a function of the input wire values.

Implementations and Practicality: Multiple implementations of Yao's garbled circuit protocol are developed in [158, 160, 161]. In 2004, Fairplay [158] is introduced, making it one of the first executions of Yao's protocol. Fairplay allows a user to describe function using a high level language, which is then transformed into a circuit ready for garbling. The system included a variation of the cut-and-choose technique to protect malicious adversaries. The system also takes advantage of the fast table lookup technique to optimize the execution of the garbled circuit. In 2011, another implementation is presented by Huang *et al.* [160]. Their implementation allowed users to write their functions in Java, and the code would then be transformed to a circuit. One of the differences between their approach and Fairplay, is their focus on the semi-honest model. This meant that they chose to trade off less security for faster performance. They also propose additional optimizations to their system including, free XORs, garbled circuit reduction, and pipelined circuit execution. In 2012, Kreuter *et al.* [161] presented an implementation that takes advantage of all optimization techniques mentioned above. Their system addresses the malicious model, and the system is able to produce significantly larger circuits using less computations than Fairplay.

Chapter 3

Emergency Mobile Access to Privately Encrypted Data on an Untrusted Cloud

When storing files on an untrusted cloud, attribute based encryption is the cryptosystem usually chosen to securely encrypt the files while allowing fine grained access control. When storing sensitive privately encrypted data such as medical records, we find that allowing access to a subset of the users' data in the case of an emergency to be a difficult task to achieve while preserving the users' privacy. In this chapter we present a protocol to aid in providing remote mobile access to privately encrypted data in the case of a emergency only. In this chapter, we will be using access to sensitive patient medical files during an emergency as a running example. Our aim is to provide emergency responders with access to patients' emergency medical records in the case of an emergency only, without requiring the participation of the patient or allowing emergency responders to abuse their privileges.

3.1 Introduction

As patients our medical records are stored in many locations; hospitals, pharmacies, clinics, medical labs, etc. This makes it difficult for a doctor to get the complete medical history for a patient. Personal health records (PHR) have become popular. In such a system, patients' data would be gathered in one location and is accessible to the patients.

With cloud storage we usually expect the data on the cloud to be protected by trusted servers. However, if an attacker were able to penetrate the security of the authentication server on such a cloud, he would be able to access all patient records. Since medical health records are sensitive, we assume the data is stored on an untrusted server.

Although this might seem unsecure, each file in the system is to be encrypted using an attribute based cryptosystem which would only allow users with the right credentials to decrypt the files. To better understand what we mean by attributes we could think of them as roles or privileges which that entity has.

In a cloud based patient centric PHR management system, the patients have control over their medical records. When we assume the cloud to be untrusted, then this means that the files are to be encrypted with the patient's private key and access control is to be controlled by the patient.

Before a user encrypts a medical record, he associates that record with a set of attributes which describe the users who should have access to the file. When a record is requested from the cloud, the user would be able to decrypt the record if and only if he has all the attributes associated with the encrypted record.

Our proposed solution solves the problem of granting an Emergency Care Provider (ECP) access to a patient's Emergency Medical Records (EMRs) in case of an emergency while preventing them from abusing that power to access patients' records without an emergency.

3.2 Problem Statement

In case of an emergency, we cannot assume the patient would be able to give consent to ECPs in order to grant them access to EMRs stored on the cloud. EMRs contain basic information such as the patient's blood type, terminal illnesses, emergency contacts, etc. Granting access to these files may be easily done by adding an attribute to represent an ECP. Also, when the patients encrypt their emergency files, they must set the emergency access attribute. This solution would easily allow ECP to access any patient's emergency medical records. However, it does give too much power to ECPs. An employee at an ECP would have the power to access all patient emergency files in the system without being detected or stopped since we're assuming the cloud to be untrusted.

In our research, our goal is to come up with a method that would enable us to allow ECPs access to a patient's emergency medical records without giving them the ability to view every medical record in the system.

This chapter is organized as follows. In section 3.3, we illustrate the problem by providing several scenarios. Section 3.4 presents a secure privacy-preserving PHR system, which we base our work on. In section 3.5, we present our proposed solution.

3.3 Motivational Scenarios

In this section, we present two possible scenarios. The first scenario presents the behavior of a PHR system during an emergency. The second scenario shows how an adversary might be able to abuse such a system to invade a user's privacy.

In the first scenario, a hospital gets a call about a car accident. They are told that the patient is unconscious, and that he has been severely injured. The hospital rapidly sends an emergency team to the location of the accident and retrieve the patients file before he gets to the emergency. An employee at the hospital enters the patient's ID into the system and then the system would retrieve the patient's EMRs in a reasonable amount of time.

In the second scenario, an authenticated employee at a hospital with malicious intentions retrieves a public figure's EMRs. The purpose of this request is to smear that person's reputation by convincing the public he is not healthy enough for a higher position. The employee would enter the patient's ID into the system. Rather than getting the medical records back, the system should reject that request and flag the hospital for abusing the system.

3.4 Privacy-preserving EHR system

Narayan et al [7] proposed a scheme for Privacy preserving electronic health records (EHR) using an attribute-based infrastructure. The system they proposed is patient-centric which means that the patient has complete control over the EHR. In their scheme the EHR are stored in untrusted cloud storage. The EHR in this system are seen as a directory of files stored in sub-folders. They assumed EHR to have the following structure:

- Patients' health data in the form of encrypted files. These files are encrypted with symmetric keys which can be found in the entry files.
- A table consisting of entries corresponding to the patient's files. The contents of the entry files include the following:
  - Meta data describing the files and their locations, all in encrypted form. The data is encrypted using broadcast ciphertext-policy attribute-based encryption and includes the following information:
    - File description.

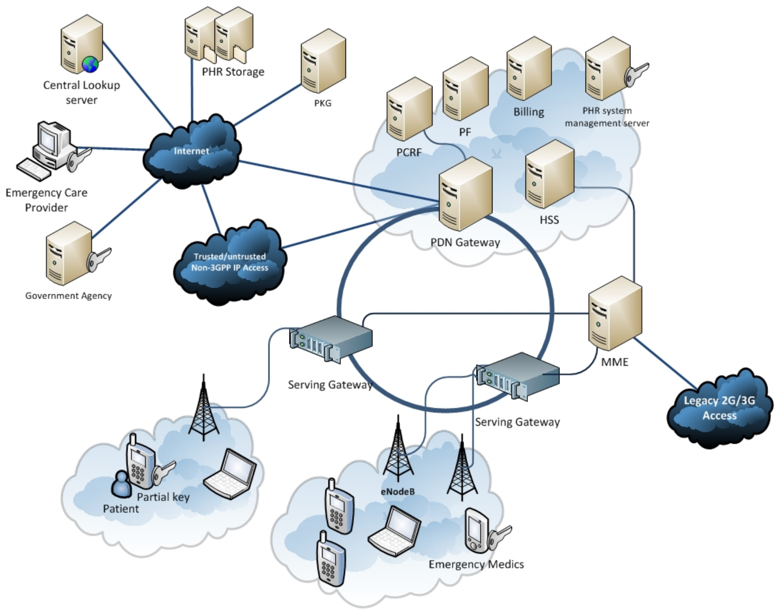


Figure 3.1: Proposed System Architecture Under an LTE Mobile network

    * A random locator tag which is also the file name used to locate the file in the cloud.
    * A symmetric key used to encrypt the health data.
- An access policy in plain text form which specifies who decrypt the encrypted data in the entry file.
- A search-index for keywords within the encrypted file used for keyword search. The search index is the encryption of the keyword.

3.5 Proposed Solution

We base our work on the privacy preserving PHR scheme proposed by Narayan et al. [7] described in section 3.4. However, in our work we introduce a method that would allow ECPs to get access to a patient's EMRs in case of an emergency without giving them access to all EMRs in the system.

In our scheme, rather than generating a single key for every user in the system, we propose generating another ABE key pair to aid in the emergency access protocol. The public key generator (PKG) would generate another key k_{ep} for every patient in the system for use in case of an emergency. This key would be generated under the policy:

$$\{emergency \wedge patient'sID = x\}.$$

In our proposed scheme, we encrypt all EMRs with k_{ep}. We also note that no one in the system would have that key, not even the ECPs or the patient. The patient would not be able to decrypt the records without having the emergency attribute.

The PKG would then split the k_{ep} key into n shares, where k shares would be needed to recalculate k_{ep}. To achieve this, we propose using Shamir's *(k,n)* threshold scheme [71]. For our running example, we propose splitting the keys into four shares, where any two shares are needed to compute k_{ep}. However, increasing the number of shares needed to access an EMR would make it more difficult for an attacker to conspire with key share holders to unlawfully access a patient's EMRs. All the key shares would also be encrypted with k_e, which is a key with the policy $\{emergency\}$. This would make the key shares accessible to ECPs only. The key shares will be given to the following entities:

- One share would be in the patient's entry file. This would enable any ECP to access the patient's first k_{ep} key share. However, the ECPs would not be able to decrypt the EMRs because they would still need another share of k_{ep} to be able to decrypt the EMRs.

- A second share of k_{ep} is given to the patient for cases where the patient needs emergency medical care, but is also able to participate in the protocol. In this case, the protocol would enable the patient to provide the ECP with the second share enabling them to retrieve and decrypt the emergency medical records. This share could be stored on the patient's mobile device and sent to the ECP after the patient acknowledges the emergency.

  Another method of storing the key share could be to give the patient an RFID chip with the key share stored on it. This could even save time if found in the patient's wallet during an emergency.

- The third share would be giving to a third party, which would provide the third share after verifying that the patient needs emergency medical attention. In this scheme, we assume the third party to be the patient's telecom provider. We do not assume that the telecom provider is completely trusted, which is why all the key shares are also encrypted with k_{ep}. In section *4.2.5* we will further discuss why we chose the telecom providers to be the third party and the methods they may use to verify an emergency situation. After verifying the patient's need for emergency medical attention, the partial key is sent to the ECP allowing them to calculate k_{ep} and decrypt the emergency medical records.

- The fourth share would be given to a government agency which would provide the fourth share after verifying that the patient is in need of emergency medical care. This verification would be done manually by a government employee. After making sure that the patient is in need of emergency medical attention, the government agency would authorize the system request to send the k_{ep} key share to the ECP.

Encrypting the EMRs with an ABE key as in our proposed solution would enable the patient to modify the medical records at anytime. This is mainly because the system would automatically generate the EMRs from the patient's medical records and encrypt the EMRs with the policy set to:

$$\{emergency \wedge patient'sID = x\}.$$

Although encrypting the EMRs with an ABE key gives us the advantages we mentioned above, we find that a problem could arise after a patient has received the needed emergency care. Since the same key would always be used to encrypt the patient's EMR, the ECP would always be able to access the patient's medical records, because they can keep the key to use it at any time in the future. If we assume that this could be a security risk, we propose the following solutions to solve this problem:

- Session attributes could be added to the ABE access policy. The new policy would be:

  $$\{emergency \wedge patient'sID = x \wedge session = y\}.$$

  It would be enough for the session attributes to be a counter where every time the patient gets emergency treatment the counter y is increased by one. The PKG would also need to split the new key to four share and redistribute them.

- Rather than using an ABE key k_{ep} to encrypt the EMRs, a random symmetric key Kr would be used to encrypt the patients' EMRs. Similar to our protocol, the first share of Kr would also be encrypted with k_e and placed in the patient's entry file. The other shares would be distributed to the other parties after encrypting them with k_e.

Although using a symmetric key might seem more appropriate than using an ABE key as in our proposed solution, we find that when using the latter, the shares could always be recomputed by the PKG in case one of the key shares is lost. However, if one of the symmetric key shares is lost then nothing can be done unless a copy of the entire symmetric key is stored somewhere.

3.5.1 Key Share Providing Protocol

We propose an authentication protocol in place for sending a key share to the ECP. The purpose of this protocol is to authenticate both parties to each other, and ensure that the key share is securely sent to the ECP. ABE is used in this protocol to ensure the confidentiality and integrity of the data.

Notations

- ECP: Emergency Care Provider
- KSH: Key Share Holder (this could be any entity with a key share, ex. patient or telecom provider)
- PID: Patient's ID
- $ECPID$: ECP's ID
- $KSHID$: KSH's ID
- $\{..\}_{K_{KSH}}$: Encrypted with a key which has the identity attributes set to the KSH's ID
- $\{..\}_{K_{ECP}}$: Encrypted with a key which has the identity attributes set to the ECP's ID
- K_{epi}: A key share of K_{ep}

The Key Share Request Protocol

- $ECP \rightarrow KSH$ $\quad \{PID, ECPID RAND1\}_{K_{KSH}}$
- $KSH \rightarrow ECP$ $\quad \{PID, RAND1, RAND2\}_{K_{ECP}}$
- $ECP \rightarrow KSH$ $\quad \{PID, RAND2\}_{K_{KSH}}$

- $KSH \rightarrow ECP \qquad \{PID, \{K_{epi}\}\}_{K_{ECP}}$

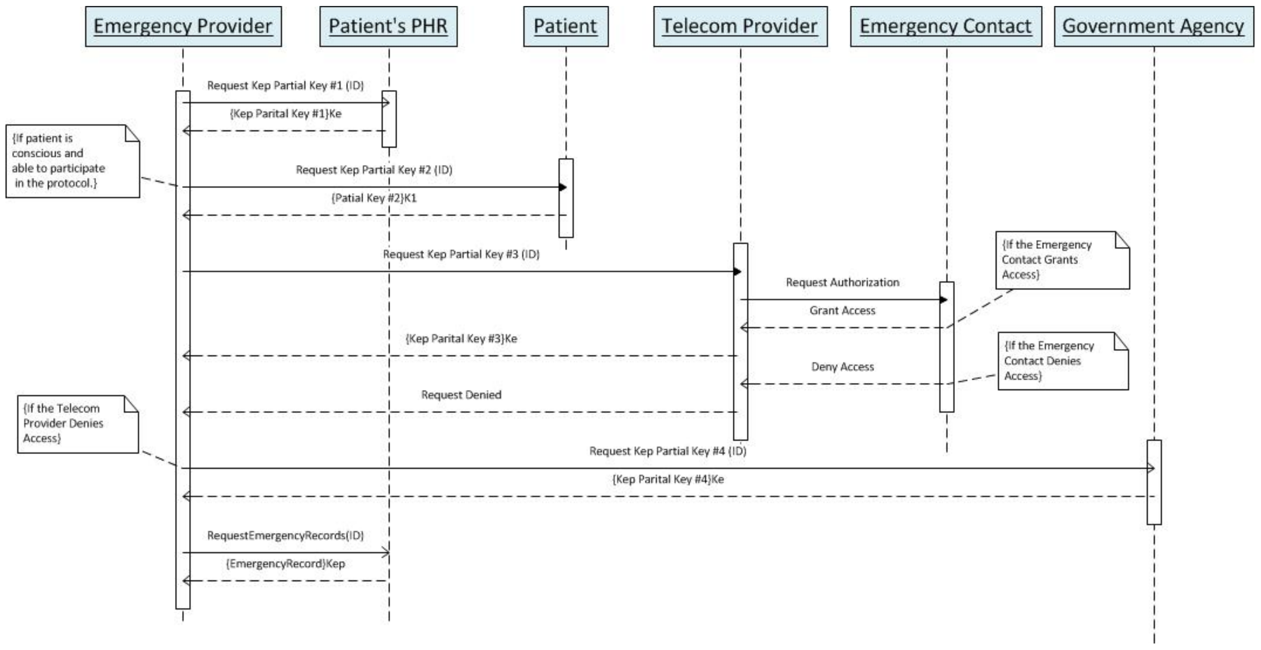

Emergency Provider
Patient's PHR
Patient
Telecom Provider
Emergency Contact
Government Agency
Request Kep Partial Key #1 (ID)
{Kep Parital Key #1}Ke
{If patient is conscious and able to participate in the protocol.}
Request Kep Partial Key #2 (ID)
{Patial Key #2}K1
Request Kep Partial Key #3 (ID)
Request Authorization
{If the Emergency Contact Grants Access}
Grant Access
{Kep Parital Key #3}Ke
{If the Emergency Contact Denies Access}
Deny Access
Request Denied
{If the Telecom Provider Denies Access}
Request Kep Partial Key #4 (ID)
{Kep Parital Key #4}Ke
RequestEmergencyRecords(ID)
{EmergencyRecord}Kep

Figure 3.2: Sequence Diagram of Proposed Solution

3.5.2 Building a Patient-Centric EHR System

In this section, we outline the algorithms needed to construct a PHR system with the properties described in our proposed solution. The protocol is presented in a sequence diagram in figure 3.2.

The main concept of this work is to build on current ABE PHR systems to allow emergency access. Access to the EMRs is to be provided only in case of a verified emergency. Granting access should not allow ECPs the right to view every emergency medical record in the system. The encryption scheme of our PHR system is based on CP-ABE [44] and contains the following functions:

PHR-Setup: Runs CP-ABE-Setup(): It is used to obtain the system public key PK and the master secret MK. The setup algorithm will choose a bilinear group $\mathbb{G}_0$ of prime order p with generator g. Next it will choose two random exponents $\alpha, \beta \in \mathbb{Z}_p$. The public key is published as:

$$PK = \mathbb{G}0, g, h = g^{\beta}, e(g,\ g)^{\alpha}$$

and the master key MK is (β, g^{α}).

PHR-KG(S,ID): Run CP-ABE-KeyGen(*MK, S, ID*). The key generation algorithm will take as input a set of attributes *S* along with the *ID* of the user, and outputs a secret key SK_{id} that identifies with the set of user attributes.

PHR-ER-KG(ID): Run CP-ABE-KeyGen(*MK, emergency, ID*) to generate an emergency key k_{ep} for a patient, the key generator will generate a key k_{ep} with the attributes $\{$*emergency, patientID*$\}$.

PHR-Split-ER(k_{ep}) It is used to split the key k_{ep} into four shares where any two shares are enough to reconstruct the key k_{ep}. The algorithm first chooses a random value *rand*. Then it forms the polynomial:

$$f(x) = k_{ep} + rand\, x$$

Finally, it computes and returns the key shares $k_{epi} = f(i)$, $1 \leq i \leq 4$, along with the index i.

PHR-Combine-ER(k_{ep1},k_{ep2}) it combines any two shares to reconstruct k_{ep}. The algorithm computes the coefficient a_1 of f(x) by using Langrangian interpolation. The algorithm then can compute $f(0) = a_0 = k_{ep}$.

PHR-Encrypt: Run CP-ABE-Encrypt(*PK,M,T*) to generate a cipher text of a message *M* under the tree access structure *T*.

PHR-Decrypt: Run CP-ABE-Decrypt(*CT*, SK_{id}). The decrypt algorithm takes as input the ciphertext *CT* and the private key SK_{id}.

3.5.3 Emergency Verification

In our proposed solution, we suggested that a third party would provide a key share of k_{ep}, which is also encrypted with k_e, to the ECP after verifying that the patient is in need of medical attention. It is difficult to come up with an automated method to achieve this task because a false negative could delay the medical care a patient needs. Also, a false positive would be an invasion of a patient's privacy. There are numerous ways of verifying an emergency. We propose in this section three possible methods of emergency verification.

Willkomm et al. [163] analyzed data collected over three weeks at hundreds of base stations in highly populated areas. The dataset consisted tens of millions of calls and billions of minutes of talk time. In their study they found that 10% of RF voice channels were allocated for the duration of about 27 seconds. They later confirmed that these were calls that were either not answered and then redirected to voice mail. This means that 90% of all phone calls were answered.

With a 90% probability of having phone calls answered, we propose giving the third share to a third party that would contact the patient and emergency contacts to verify the emergency. The third party could be a telecom provider given the resources they have. Moreover, with the high percentage of mobile subscribers, we are assuming every user in the system would be a mobile subscriber.

We propose the following three methods of verification, each with its advantages and disadvantages.

Location Based Verification

Telecom providers have the ability to find a mobile subscribers approximate location given the cell towers providing the mobile subscriber with a connection to the network.

When a ECP requests a key share for a patient, the telecom provider may provide it if the patient's telecom device is in a location within a reasonable distance from the hospital's location.

This solution has the benefit of being completely automated. This would lead to a response within seconds, allowing the ECPs the chance to act rapidly to save the patient's life.

Although this solution would deny access from ECPs which are not in the same location as the patient's mobile device, it would still be possible for an ECP to target a specific person within the same location where the patient's mobile device is located. Access might also be denied if the patient's mobile device's battery runs out in a location far from the location of the emergency. This would mean that the telecom provider's data shows the last known location of the patient's device in a different area. However, that could also be solved with an algorithm that would estimate the possibility a patient could travel from the last known location to the location of the emergency. Also, access to a patient's key share might be denied because the patient did not have his/her mobile device at the location where the emergency occurred.

Emergency Contact Verification

To overcome the disadvantages in the location based solution, we propose contacting the patient's emergency contacts. The telecom provider could either send a text message or generate an automated call to the patient asking for permission to provide access to the ECP. In case the patient does not reply within a reasonable amount of time, the same request would be sent to the patient's emergency contacts.

This method has the benefit of being more thorough in verifying whether the patient is in need of emergency care or not. However, it does require more time, which could be dangerous to the patient's health in an emergency. Moreover, a request might be denied if the emergency contacts are contacted during late hours of the night.

Location-Based and Emergency Contact Verification

Both of the emergency verification methods we mentioned above had their advantages and disadvantages. Thus, we propose using a hybrid solution that combines both methods. We propose using the emergency contact verification method in the beginning, and in case no response is returned, the location based method would take over. This means that a request would only be denied if the emergency contacts do not respond within a reasonable time interval and the patient's last location is not in proximity to the ECPs location [164].

3.6 Implementation

In this section, we show some preliminary results as a proof of concept for our proposed encryption-based solution. The implementation code is written in Java and run on an Intel Core i5 with 4GB RAM. The attribute based encryption implementation is based on the open-source Advanced Crypto Software Collection (ACSC) [1]. The threshold implementation is based on a java implementation by Tim Tiemens [2]. To test the practicality and scalability of our proposed solution, we test for the time needed to encrypt the EMRs, combining multiple key shares, and the time needed to decrypt the EMRs. However, communication delays are out of the scope of this work and are not taken into account.

According to a technical report by TechTarget [165], the size of an electronic health

[1] http://www.cs.berkeley.edu/ bethenco/
[2] https://github.com/timtiemens

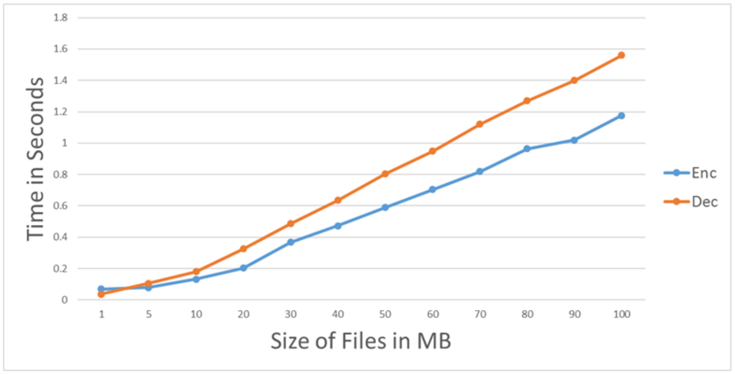


Figure 3.3: CP-ABE Encryption and Decryption Execution Times

record without images is typically less than 1MB for a relatively healthy adult patient and 40MB for a patient with major medical issues. To calculate the time needed to encrypt the EMRs, we create a user with a simple access policy and run the encryption algorithm assuming various file sizes. We show in figure 3.3, the execution times for encrypting and decrypting files of various sizes.

Assuming that the EMRs are for a patient with major health issues, it would take 0.47 seconds to encrypt the 40MB records. It would also take the system 0.16 seconds to split a 256 bit symmetric key into 4 shares, where at least 2 shares are needed to reconstruct the secret. To combine the shares, It would take the ECPs 0.2 seconds to reconstruct the symmetric key. Finally, decrypting the 40MB EMRs takes 0.64 seconds. Moreover, our experiments show that the execution time for a large EMR of 200MB, would require the ECPs less than 2 seconds to combine the key shares and decrypt the medical records. Therefore, our experimental results show the practicality and scalability of our protocol.

3.7 Conclusion

In this work we discussed the problem of providing emergency response providers access to emergency PHRs stored on an untrusted cloud. We proposed a solution based on using ABE and threshold cryptography to secure the data on the cloud while providing fine grained access control and confidentiality. In our solution we proposed a protocol capable of providing access to these records, but only during a verified emergency. We achieved this by splitting the decryption key between four parties where two are needed to calculate the decryption key. We also discussed how verifying the occurrence of an emergency can be automated by a third party such as a telecom provider. We believe that this method can be generalized to storing sensitive data in untrusted storage systems such as cloud storage systems.

Chapter 4

Secure and Privacy-Preserving Querying of Privately Encrypted Data in the Cloud

In cases where users need to store sensitive data in an untrusted cloud, cryptography can help protect the confidentiality of the users' data. By encrypting the data before uploading it to the cloud, users can benefit from having secure access to their data when needed as long as they safely store their private keys. Since access to the data is not possible without the corresponding private keys, analyzing the data is also prevented. In this chapter, we present a scheme to allow a querying entity to securely query privately encrypted data while preserving the privacy of the users and the querying entity. We will continue using privately encrypted records (PHRs) as a running example for our scheme. Our goal is to allow querying entities such as health organizations to produce statistical information about privately encrypted data such as PHRs stored in the cloud. The protocol depends on two threshold homomorphic cryptosystems: Goldwasser-Micali (GM) and Paillier. It executes queries on KD-trees that are constructed from encrypted health records. It also prevents patients from inferring what health organizations are concerned about. We experimentally evaluate the performance of the proposed protocol and report on the results of implementation.

4.1 Introduction

Electronic health records are usually managed by different healthcare providers including primary care physicians, therapists, hospitals and pharmacies. Consequently, it is difficult to get a single patient's history due to the fact that it is spread between multiple providers. It has become a recent trend for patients to take these matters into their own hands by managing their own records using a Personal Health Record (PHR) system. PHRs systems allow patients to manage their medical data, giving them the ability to create, view, edit, or share their medical records with other users in the system as well as with healthcare providers [166]. 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 PHRs and alert patients when a preventive checkup is needed. These providers include Microsoft HealthVault [17] and Dossia [18]. Due to the sensitivity nature of health data, security concerns have prevented many patients from using PHR systems. Many of the providers of the current PHR systems have the ability to access all patient records.

Recently, architectures for storing PHRs in the cloud have been proposed [166]. However, this does not solve the privacy problem and the latter remains an issue for many patients. Since these records are stored on cloud servers, it means that these servers have the ability to read any medical record in the system. In addition, if an attacker is able to compromise a cloud server, then all the PHRs would be exposed. For these reasons, researchers have begun searching for a way to allow patients storing their medical records in the cloud using a Database-as-a-Service (DaaS) model while preserving their privacy. DaaS is a category of cloud computing services that enables IT providers to deliver database functionality as a service. Encrypting PHRs before outsourcing appears to be a promising solution in this domain. However, it prevents health organizations from analyzing medical data for research purposes. To better understand what caused a disease, health organizations and researchers need as much data as possible about the infected patients.

In this chapter, we propose a protocol that allows health organizations producing statistical information about encrypted PHRs stored in the cloud. The proposed protocol also does not enable patients to infer about what health organizations are concerned about; not to worry or panic patients targeted by the queries. Intuitively, the proposed protocol works as follows: Patients are organized in small groups. The patients of a given group jointly generate public keys for encryption. They later encrypt their PHRs using their public keys and send the ciphertext to the cloud server. Encrypted records are stored in KD-trees constructed by the cloud server for each group. To execute SQL queries, KD-trees are traversed in the cloud server. Finally, the cloud server aggregates the results and sends the final query result to the health organization. However, realizing this seemingly simple system presents a number of significant challenges. First, the search should be performed on encrypted records. To achieve this, our proposed protocol depends on the 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 Fischlin's protocol [88] whereas query aggregate functions are computed using Paillier cryptosystem [25]. Second, the engagement of the patients 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 patients, namely the threshold k. Finally, the searching should be efficient and can be done in logarithmic time. For this purpose, we use balanced binary KD-trees.

The contributions of this work can be summarized as follows:

- We propose a protocol, which allows health organizations producing statistical information about PHRs stored in the cloud and encrypted using semantically-secure encryption schemes. The main characteristics of the protocol are the following:

  - It preserves the privacy of health organizations and patients.
  - It supports aggregate queries such *count*, *sum*, *max* and *min*.

- We design and implement a prototype of the proposed protocol and we also report on the experimental results.

The rest of the chapter is organized as follows. Section 4.2 discusses the execution environment. Section 4.3 presents a protocol to find the maximum/minimum value of encrypted inputs without resorting to deterministic encryption schemes. In Section 4.4, the proposed protocol is presented. The security and complexity analysis of the protocol as well as the experimental results are discussed in Section 4.5. Finally, concluding remarks as well as a discussion of future work are presented in Section 4.8.

4.2 Execution Environment

In this section, we first identify the involved entities. Then, we present the assumptions underlying the system design.

4.2.1 Entities

There are three main entities:

- *Patients* who own health records and want to store them on a cloud server while keeping them confidential from the cloud server and health organizations.
- *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 will assign an assisting server to each group. The assisting server is a cloud computing

instance, which will be responsible for storing the encrypted records of the patients and executing the SQL queries of the health organization.

- *Health organization* that execute queries over the encrypted records of the patients and produce statistical information.

4.2.2 Assumptions

We assume that there is no fully trusted entity in the environment and all entities are semi-honest. Semi-honest adversaries follow the protocol steps, but they try to extract information about other entities' input or output. This is a common security assumption used in secure multiparty computation literature [167] and it is realistic in our problem scenario since different organizations are collaborating for mutual benefits. Hence, it is reasonable to assume that parties will not deviate from the defined protocol. However, they may be curious to learn additional information from the messages they receive during protocol execution. The cloud server and the assisting servers are modeled as "honest but curious" in the sense that they will execute requests correctly, but they are not reliable to maintain data confidentiality. Regarding query privacy, we assume that the shape of SQL queries submitted by health organizations is public whereas the constants contained in the query predicates are private [168, 169]. We also assume that there is no collusion between the different parties and that there are mechanisms that ensure integrity and availability of the remotely stored data. Our scheme focuses only on confidentiality and privacy issues and does not provide protection against attacks such as data tampering and denial of service.

4.3 Secure Maximum/Minimum Computation

As a major step in our proposed protocol, we need to execute the *max*/*min* aggregate queries over encrypted records. In this section, we provide a cloud-based solution that calculates the maximum/minimum of encrypted values owned by some parties. Formally, given n inputs $v_1, \ldots, v_n$ owned by the parties $P_1, \ldots, P_n$ respectively, the cloud server wishes to securely compute $max(v_1, v_2, \ldots, v_n)$ and $min(v_1, v_2, \ldots, v_n)$. We assume that the parties are not malicious and they correctly carry out the prescribed steps. The proposed cloud-based solution relies on Fischlin's protocol [88] and the threshold GM cryptosystem [90]. We explain the technique to find the *max* but it can be easily modified to find the *min*.

Parties jointly generate the public key pk for k-out-of-n threshold GM cryptosystem such that at least k parties are required to fully decrypt a ciphertext [90]. Parties then encrypt their values and outsource them to the cloud server. The cloud server initializes the current maximum to the encryption of a small negative value with the threshold GM cryptosystem. Afterwards, the cloud compares the current maximum with the encrypted value of the party P_i using Fischlin's protocol. The outputs of Fischlin's protocol are sequences of λ elements. To decide whether the encrypted value of P_i is greater than the current maximum (If there exists a sequence of λ quadratic residues), the cloud server contacts k other parties and sends the generated sequences for them. Each party performs calculation on the sequences using her share of the factors of the public key [90] using the threshold GM decryption. The results are then submitted to the cloud server. Afterwards, the cloud server combines the results received from k parties and decide if the encrypted value of P_i is greater than the current maximum based on the quadratic residuosity of the combinations. If the output indicates that the current maximum is greater, there is no need to update the current maximum. Otherwise, the cloud server sets the current maximum to the value of P_i and repeats the same process with P_{i+1}. After comparing the current maximum with all the existing values, the cloud server sends the encrypted maximum value to k members for decryption. This idea can be extended to enable a cloud server to sort the encrypted values without knowing the secret key and the plaintexts. In this case, any comparison-based sorting algorithm can be utilized and the comparison is performed on the encrypted values by exploiting Fischlin's protocol.

4.4 Secure Execution of Queries in Cloud

In this section, we present a protocol that enables health organizations to produce statistical information about encrypted personal health records stored in the cloud server. The proposed protocol prevents patients from inferring what health organizations are concerned about. The health organization's input is an aggregate SQL query that consists of exact-matching and interval-matching predicates over multiple attributes combined with logical operators (AND/OR/NOT). The cloud server's input is the encrypted health records of the patients. The naive approach to achieve these objectives is that the health organization communicates with each patient and securely evaluates queries on her record. This can be achieved by exploiting Fischlin's protocol for private comparison. However, this approach incurs excessive communication and computation overhead on the health organization side that is linear to the number of patients.

To reduce this overhead, patients are organized into smaller groups. The patients in each group jointly generate two public keys for Goldwasser-Micali [24] and Paillier [25] 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. Assisting servers are responsible for securely executing health organization's queries on the database of each group to obtain partial results. Assisting servers then collaborate to combine the partial results into the query result and report it to the health organization. In the following, we elaborate the basic steps of our protocol that protects the data privacy of patients and the query privacy of the health organization.

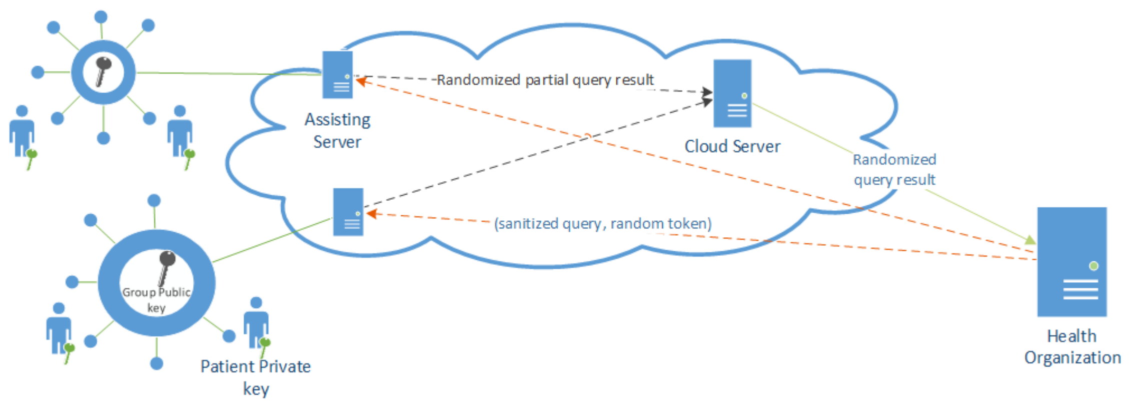


Figure 4.1: System Architecture

4.4.1 Key Generation and Tree Construction

Assuming the total number of N patients, the cloud server defines $L = \lfloor \sqrt{N} \rfloor$ groups. It then randomly maps and assigns 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 health organizations' queries over the medical database of patients as shown in figure 4.1. Assisting servers collaborate with each other to obtain the query result from the partial results and send it to the health organization. In the i-th group, the patients execute the distributed key generation algorithms for the threshold Paillier and the threshold GM cryptosystems to obtain the public keys $pk_i^{\prime}$ and pk_i for Paillier and GM cryptosystems. We utilize the protocols explained in [90] and [170] for the threshold GM and the threshold Paillier cryptosystems, respectively. These protocols depend on distributed RSA key-generation protocols [171, 92] without the need for a trusted dealer. Following the execution of the key-generation protocols, each patient obtains a single public key and a share of the secret key. The threshold cryptosystems enable 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.

Example 1. *Consider health records with the attributes* Age *and* Surgery, *where the value* *of* Surgery *specifies the type of the surgery that a patient undergoes (e.g. 1: Transgender, 2: Plastic, 3:Vascular, 4: Urology). The total number of patients is* $N = 10$*; therefore, these patients are organized in* $L = \lfloor\sqrt{10}\rfloor = 3$ *groups, namely,* G_1*,* G_2 *and* G_3*. Assume that patients 1,9 and 10 are assigned to* G_1*; patients 2, 4, 5 and 8 to* G_2 *and patients 3, 6 and 7 to* G_3*, randomly. Furthermore, the patients in the group* G_i *jointly generate the public key* pk_i *and* pk'_i *for the GM and the Paillier cryptosystems, respectively. The members of* G_i *encrypt their records with* pk_i *and* pk'_i *as presented in Fig. 4.2. The encrypted tables are then outsourced to the cloud server.*

To store the shares of a secret key, we assume the secret key is stored on secure hardware such as FPGA [172]. These devices are designed in such a way that after a patient places a key into the on-board key memory on the device, it cannot be read externally. After the secret key of each patient and the group public keys (for Paillier and GM cryptosystems) have been written, the FPGA can be delivered to the cloud operator for installation.

The patients encrypt each record using both Paillier and GM cryptosystems by the group public keys. Therefore, the encrypted record of each patient 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 the threshold Paillier cryptosystem, and another column that stores the GM encryption of the attribute value using the group public key for the threshold GM cryptosystem. Finally, the cloud server assigns an assisting server to each group. Each assisting server collects the encrypted health records and organizes them as a KD-tree [173].

Example 2. *(Continued from Example 1) The generated KD-trees for each group are shown in Fig. 4.2. The partitioning attributes may be different in each group depending on the data within that group.*

	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 4.1: Health Records

	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 Data Set

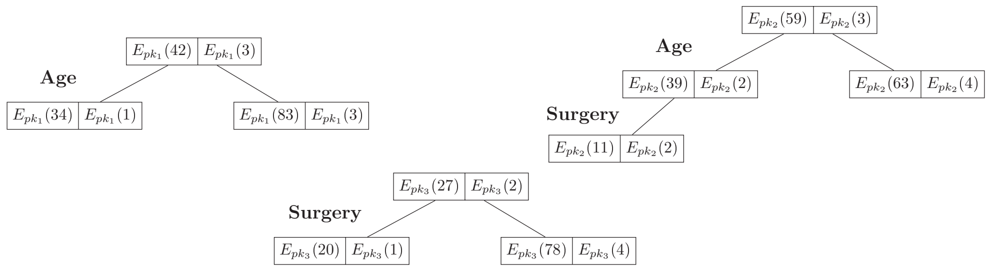
	Age$_{GM}$	**Surgery**$_{GM}$	**Age**$_{P}$	**Surgery**$_{P}$
Patient 2	$E_{pk_2}(39)$	$E_{pk_2}(2)$	$E_{pk'_2}(39)$	$E_{pk'_2}(2)$
Patient 4	$E_{pk_2}(59)$	$E_{pk_2}(3)$	$E_{pk'_2}(59)$	$E_{pk'_2}(3)$
Patient 5	$E_{pk_2}(63)$	$E_{pk_2}(4)$	$E_{pk'_2}(63)$	$E_{pk'_2}(4)$
Patient 8	$E_{pk_2}(11)$	$E_{pk_2}(2)$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

(b) G_2 Data Set

	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 Data Set

Table 4.2: Outsourced Health Records in Groups



$E_{pk_1}(42)$ $E_{pk_1}(3)$
Age
$E_{pk_1}(34)$ $E_{pk_1}(1)$
$E_{pk_1}(83)$ $E_{pk_1}(3)$
$E_{pk_2}(59)$ $E_{pk_2}(3)$
Age
$E_{pk_2}(39)$ $E_{pk_2}(2)$
$E_{pk_2}(63)$ $E_{pk_2}(4)$
Surgery
$E_{pk_2}(11)$ $E_{pk_2}(2)$
$E_{pk_3}(27)$ $E_{pk_3}(2)$
Surgery
$E_{pk_3}(20)$ $E_{pk_3}(1)$
$E_{pk_3}(78)$ $E_{pk_3}(4)$

Figure 4.2: Generated KD-trees

4.4.2 Query Sanitization and Token Generation

The health organization wishes to execute an SQL query such that the constants in the predicates are not revealed to the patients and the cloud server. Therefore, the health organization sanitizes the query by replacing the constants contained in the predicates by their GM encryption using the public key of each group. Furthermore, the health organization uses a token for each group that is encrypted by the group's public key. This token can be manipulated by assisting servers to produce a noisy query result. Generating the token depends on the type of the aggregate function. For *count* and *sum*, the health organization generates L random numbers $R_1, R_2, \ldots, R_L$ 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 i-th group. For the *max* and *min*, the health organization generates a random number R as the token for all groups. The health organization then encrypts the token of each group by the Paillier cryptosystem using 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.

Example 3. *Suppose that the health organization's query is:*

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

The sanitized query that will be forwarded to the i-th group would be

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

In addition, since the function is max*, the health organization generates a random number R and uses it as the token for all groups. The health organization then encrypts the token using the Paillier encryption by the public key of each group and forwards the encrypted token together with the sanitized query to the corresponding assisting server.*

4.4.3 Tree Traversal and Query Execution

To execute the health organization's query, the assisting servers must traverse the KD-trees, constructed from the encrypted records of the patients. To do so, the assisting servers follow the tree traversal algorithm. The search begins from the root; the assisting server uses Fischlin's protocol and the threshold GM decryption to evaluate the query predicate on the root record. Based on the result of the query evaluation, the search is continued on the left tree or the right subtree or both. 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*: The assisting servers encrypt 0 (as the current sum) by the Paillier encryption using the group public key. While traversing the tree, 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, min*: Initially, the assisting servers pick up a small negative number (or a large positive in case of *min*) that denotes the current *max/min* value and encrypts it by the GM and the Paillier cryptosystems using the group public keys. 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's protocol using the encrypted current *max/min* value and the GM-encrypted value, to find out if this record has greater (resp. smaller) value or not. If so, the assisting server initializes the current *max/min* value to the GM- and the Paillier-encrypted ciphertexts. Otherwise, the current *max/min* value remains unchanged. At the end, the assisting servers end up with the query result encrypted with the Paillier and GM cryptosystems. 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 the Paillier scheme by the public key of the group).

Example 4. *(Continued from Example 3) Considering the KD-trees presented in Fig. 4.2 and the sanitized query*

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

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 Fischlin's protocol and the threshold GM cryptosystem. The record of Patient 1 in G_1 and *the record of Patient 3 in G_3 satisfy the predicate. Therefore, the output of the tree traversal for the assisting servers of the 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.*

4.4.4 Query Result Decryption

So far, the assisting servers have obtained the encrypted partial query result, which we will call group results from now on. Therefore, the assisting servers must collaborate with each other to compute the final query result and submit it to the health organization. The group results are encrypted with different keys. Therefore, in order to compute the final query result, the group results must be in plaintext. The assisting servers first obfuscate the group results because they are not willing to reveal these results to each other. The obfuscation is performed by the mean of multiplying the group 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 group result. 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. Afterwards, each assisting server decrypts the noisy group result (that is encrypted by the Paillier cryptosystem) by contacting patients in its group.

The assisting servers then need to obtain the final noisy query result by aggregating their group results. In this context, the assisting servers send the noisy group results in plaintext to the cloud server. The cloud server then aggregates the partial noisy query results to obtain the noisy query result. In the case of *count* and *sum*, the cloud server adds up all the group results and submits the summation to the health organization. In the case of max and min aggregate functions, the cloud server executes the *maximum/minimum* algorithm on the plaintexts and sends the resulted value to the health organization. Notice 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 5. *(Continued from Example 4) We have seen that the result of executing the SQL query*

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

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.*

4.5 Security Analysis

Assuming the semi-honest adversary model and no collusion between the patients, the security of the protocol depends on the steps where the parties exchange information and it is conducted as follows:

- *Health organization-Cloud server*: The health organization sends the sanitized query and the token that are encrypted by the semantically-secure encryption schemes using patients' public keys. Therefore, the cloud server is not able to decrypt it [24, 25].
- *Patient-Cloud server*: The patients sends their medical records, encrypted using semantically-secure encryption schemes that are secure against semi-honest adversary [24, 25].
- *Cloud server-Patient*: The cloud server communicates with the patients in order to execute Fischlin's protocol, that is proven to be secure in the presence of semi-honest adversaries [88, 90, 91].
- *Assisting servers*: The interactions between assisting servers are required to aggregate the noisy group results and acquire the randomized final query result. Since the query results have been randomized by a number that is generated by the health organization, the assisting servers are not able to extract the actual query results from the noisy results.

Moreover, the output of each subprotocol is the input to another subprototcol. Therefore, according to the Composition Theorem [174], the entire protocol is secure. The main concern with threshold cryptosystems comes from the collusion attack. We address in the following the possible attacks resulted from the collusion between the different parties. The threshold decryption will be compromised if the number of colluding clients under the control of an adversary exceeds the threshold k. 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

Health Organization		Assisting Servers	
Communication	Computation	Communication	Computation
$O(\sqrt{N})$	$O(k\sqrt{N})$	$O(kT(N))$	$O(kT(N))$

Table 4.3: Communication and Computation Cost

patients in any group collude to infer constants in the query. In practice, we can increase the threshold k such that attackers will not be able to compromise too many patients. Despite simplicity, this mechanism has two disadvantages: First, the number of the required online patients is increased. Second, the communication cost on the assisting servers is increased because they need to communicate with more parties for decryption. Therefore, there should be a trade-off between availability/efficiency and security by choosing a proper value for k.

It should be noted that the proposed protocol does not protect the privacy of patients from being identified through the query result. There is a significant body of works on distributed privacy preserving data mining (e.g. constructing decision trees [175] and differential privacy [176, 177, 178]) that provide a rich and useful set of tools to protect the record owner (i.e., patients) from being identified through query results. 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 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 health organization's queries.

4.6 Complexity Analysis

Let N denotes the number of patients in a PHR system. These patients are organized into L groups and each group contains $n = \frac{N}{L}$ patients. Recall that the execution time of range queries, exact matching queries and partial matching queries on a KD-tree with $\sqrt{N}$ records, will be $O(N^{0.5-1/2d}+m)$, $O(\log N)$ and $O(N^{0.5-s/2d}+m)$, respectively [179] where d is the number of the attributes in the table, s is the number of attributes in the query predicate and m denotes the number of records, reported as the query result. The communication and the computation cost of the protocol is summarized in Table 4.3 where $T(N)$ indicates the execution time of different types of queries (e.g., exact-matching, partial matching and range matching). The most communication- and computation- intensive operation on the assisting servers is the tree traversal.

4.7 Performance Evaluation

To evaluate the performance of the proposed protocol, we implement a prototype relying on some existing open source projects [180] in Java 1.6. The secret keys p and q of the GM cryptosystem are 256-bit long. Moreover, we employ the publicly available *Breast Cancer* dataset [181]. It has 286 records with 9 categorical attributes. The patient's and the server's side experiments are conducted on an Intel core i5 2.3GHz notebook with 4GB of RAM. The number of patients in each group is fixed at $L = 286$ leading to approximately $81,800$ patients in the PHR system. The shares of the secret key are stored on FPGAs. Decrypting a ciphertext by the cloud server is performed by sending a ciphertext to the FPGAs. Since the communication is intra-site, we ignore communication delays in the performance evaluation.

To understand the source of the overhead, we measure the query execution time for

Query	**Query Time**(ms)
Select by =	41.93
Select range	216.14
Select sum	33.02
Select max/min	217.32

Table 4.4: Assisting Server Latency for Different Types of SQL Queries ($k = \frac{n}{4} = 71$).

different types of aggregate SQL queries, but running with only one core enabled. The result is presented in Table 4.4 and Fig. 4.3. 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. According to a similar analysis, for small values of k there is a small change in the execution time but as k increases the query time becomes linear with k. Finally, we calculate the execution time of an arbitrary SQL query $k = \frac{n}{4} = 71$. In addition, 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. 4.4. The results indicate that as the number of attributes in the query increases, the execution time decreases due to the smaller search space and the reduction of the number of comparisons. 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. The implementation and the above results are meant to prove the feasibility. Further optimizations may lead to a better performance. It is worth to mention that health organizations

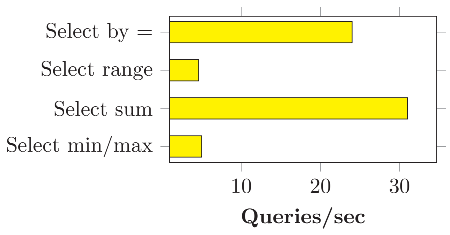


Figure 4.3: Query Time

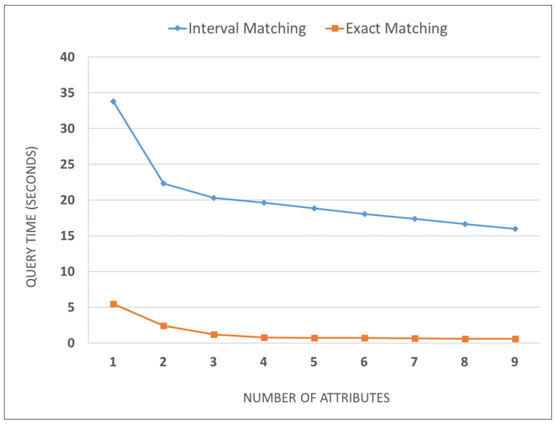


Figure 4.4: SQL Query Time (Exact Matching vs Interval Matching)

do not frequently conduct statistical studies on the medical databases (every month or when there is a pandemic). Therefore, the performance of the protocol is acceptable for this type of applications whose goal is to perform search while the absolute privacy of the patients and the health organization is preserved.

4.8 Conclusion

In this chapter, we have presented a protocol that allows executing various types of SQL queries on PHRs stored in the cloud while preserving the privacy of the patients and the health organization as well. The health records are encrypted using probabilistic encryption schemes, which are semantically secure. The protocol supports aggregate, exact matching and range matching queries. It is based on Fischlin's protocol for private comparison and on two threshold cryptosystems. The implementation result has indicated that the protocol works well with medium size databases and the queries that contain multiple attributes. We have shown that we can execute queries over encrypted data using probabilistic cryptosystems.

Chapter 5

Enhanced Privacy-Preserving Querying Mechanism on Privately Encrypted Data in the Cloud

The affordability of cloud data storage has made it simpler for users to store and access data online from any location or operating system. These services may be used by users to store sensitive data, such as personal health records or financial data. Many service providers offer features such as analyzing the users' private data to generate useful reports for medical data. Storing such sensitive data on the cloud raises many privacy concerns. While encryption can ensure data confidentiality, it introduces the challenge of analyzing the privately encrypted data while preserving the privacy of the users and the querying entity. In this paper, we address this problem by elaborating a cryptographic protocols that allows a third party, such as a health organization, to query privately encrypted data without relying on a trusted entity. The proposed protocol preserves the privacy of the users and the querying entity. The protocol relies on homomorphic, threshold cryptography, and randomization to allow for secure, distributed, and privacy-preserving queries. We evaluate the performance of our protocol and report on the results of the implementation.

5.1 Introduction

There are various methods for securing sensitive data stored in the cloud. Data encryption ensures that access is only possible when the correct decryption key is provided. However, although users are assured that sensitive data in the cloud is securely encrypted, in most cases trust is given to the cloud service providers to manage the encryption keys on their behalf. This implies that, although the data is encrypted, the cloud service provider has the ability to see the actual data. It is possible to build a secure privacy-preserving system where data is encrypted and key management becomes the responsibility of the users, thus ensuring cloud service providers cannot see the plaintext data. This approach has not been popular due to the limited features providable by a cloud service provider, such as sharing or querying data to provide statistical information. There are many solutions to provide these features while preserving the users' privacy. Cloud-based PHRs are the most common application used by researchers when discussing this problem.

Although the use of an ABE system preserves the privacy of the users in the cloud, it prevents third party entities such as health organizations from querying privately encrypted data (e.g. PHRs) on the system. To produce statistical information on privately encrypted data on the cloud, users would have to give the query requester access to all private data using ABE. According to a report from the consulting firm PwC [182], health organizations are falling short in protecting the privacy and security of patient information. Additionally, according to the same report, more than half of health organizations have had at least one issue with information security and privacy since 2009. The most frequently observed issue is the improper use of protected health information by an employee in the organization.

We therefore propose a solution to querying privately encrypted data under the assumption of not having a trusted entity in the system. This implies that the user trusts neither the system nor the Querying Entity (QE) to have direct access to the data. This also means that the QE trusts neither the system nor the users to see the query details in the query. We achieve this by means of a protocol, which uses private comparison protocols along with semantically secure cryptography to compare the encrypted values. This allows us to support equality, range, and aggregate queries such as *count*, *sum*, and *average*. We also use threshold Goldwasser-Micali cryptography to prevent the QE or the cloud server (CS) from viewing the actual user data. Finally, we use the randomization and partial decryption of the results array to prevent the CS from correlating a result to a specific user.

The enhanced protocol we propose in this chapter differs from the solution we present in Section 4.4. Rather than requiring DOs to assist in the decryption process for each comparison in the KD-Tree, the enhanced protocol requires DOs to participate in a single communication step. The computations that need to be executed by the DOs are also reduced. Finally, the enhanced protocol does not require the DOs data to have the same structure.

Variations of our approach may solve a variety of problems, other than the PHR example mentioned in the introduction, where a party needs to query multiple data sources that are privately encrypted. Moreover, neither party trusts the other with any data aside from the final results of the query. Examples of some possible use cases include:

- Credit card companies querying private e-commerce data to find cases of credit card fraud or identity theft.
- Insurance providers querying pharmacies for illegal abuse by patients, doctors, or pharmacists related to drug prescriptions.

The remainder of the chapter is organized as follows. In Section 5.2, we present the threat model and the security guarantees. Section 5.3, describes our protocol in detail.

Section 5.6, provides a security analysis of our protocol. Section 5.7, presents the results of our implementation.

5.2 Security Overview

Figure 1 shows the architecture of our proposed solution. Our solution consists of outsourcing the homomorphic computations to a semi-trusted Cloud Server (CS). The QE encrypts the query constants to preserve its privacy. Threshold encryption is used to protect the confidentiality of the QE's query constants. Threshold encryption along with randomization preserves the privacy of the data owners'. The CS runs all the homomorphic computations on the encrypted data, and it only sends the final result back to the QE. This ensures that our system guarantees the confidentiality of the QE's query constants and the Data Owners' data. The solution does not provide confidentiality if the QE and the CS collude and share their encryption keys.

Although our solution guarantees data confidentiality, it does not ensure the integrity, freshness, or completeness of the results returned to the QE. An adversary that compromises the cloud server or a data owner can modify the data and the final result. We now describe the entities in our system, the threat model, and the security guarantees provided under the threat model.

5.2.1 Entities

There are four main entities in our protocol (as shown in Figure 5.1):

- *Data Owners (DO)*, own the data and want to store it in a cloud server while keeping it confidential from the cloud server and Querying Server.

- *Cloud Storage Servers*, are used by the DOs to store their encrypted data, which are confidential from the cloud service providers. In our protocol, DOs' data may be stored with different cloud providers, under different standards, or encrypted using different cryptosystems. We are using the cloud storage servers as an example, although the privately encrypted data may be stored locally by the DO on an offline storage device.

- *Cloud Server (CS)*, is an intermediary entity in the cloud between the DOs and the QE, trusted to help manage the execution. The DOs and the QE trust the CS to execute the queries, but neither entities trust the CS to see unencrypted data.
- *Querying Entity (QE)*, runs queries over the encrypted data of the DOs to produce statistical data.

5.2.2 Problem Statement

Given a set of n data owners $DO\{DO_1, DO_2, \ldots, DO_n\}$, each storing their privately encrypted data d_i on an offline storage device or on a semi-trusted cloud with different encryption keys or cryptosystems. An external querying entity QE is securely allowed to run a query Q with a set of m attributes $Q_{a_i} \in \{Q_{a_1}, Q_{a_2}, \ldots, Q_{a_m}\}$, $i = 1, \ldots, m$ and a set of j constants $Q_{c_i} \in \{Q_{c_1}, Q_{c_2}, \ldots, Q_{c_j}\}$, $i = 1, \ldots, j$, on the encrypted data d_i while ensuring the confidentiality of the QE's query constants Q_c. The QE cannot ascertain any information about the DOs' data other than the final result Q_r and the number of DOs that satisfy the QE's query Q.

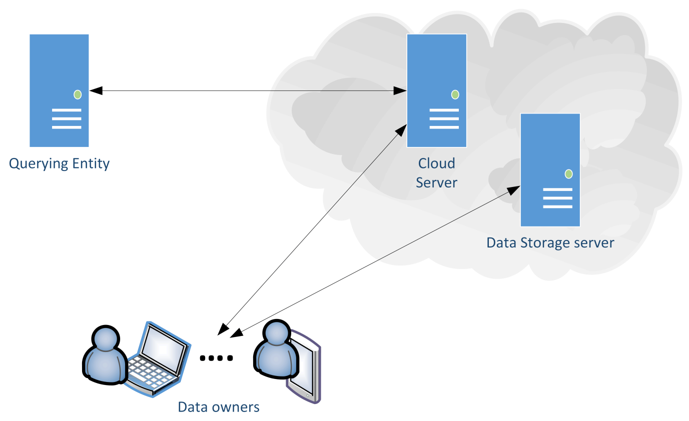


Figure 5.1: System Architecture

5.2.3 Threat Models

Threat 1: Compromising The Survey Server or The Querying Entity

In this threat, our solution guards against a curious cloud server, a curious querying entity, or other external attackers with full access to either server. Our goal is confidentiality (data secrecy), not integrity or availability. Since we are assuming the semi-honest model, the attacker is assumed to be passive, which means that the attacker wants to learn about the confidential data in the querying entity's query, the data owners' responses, or the final result. The semi-honest model implies that the attacker is not able to modify the data in the query, the data owner responses, or the computation results. This threat is becoming increasingly important with the increasing popularity of migrating data centers to the cloud. We address this threat in Section 5.6.

Threat 2: Personally Identifiable Information Queries

In this threat, the querying entity's goal is to take advantage of the secure querying mechanism to create targeted queries that expose Personally Identifiable Information (PII). For example, a malicious QE may create a query with the goal of finding out if a specific person has cancer. By requesting a **count** query for the number of data owners with "diabetes", living in "location" x, are y years old, and are z cm tall. In most cases, a query this specific would usually be unique to a single DO. Our solution guards against such attacks and ensures the data owners that their PII would not be exposed to a querying entity. Our system achieves this while providing the querying entity with an approximate result using differential privacy. We address this threat in section 5.5.

Threat 3: Exposing the Query Attributes to the Data Owners

In this threat, our solution guards against a DO attempting to learn about the attributes in the QE query. To address this threat, our solution hides the query attributes along with the query predicates from the data owners. If the QE were to request a query after a major health outbreak, then exposing the attributes in the query might cause DOs to panic. For Example, if a new fatal virus was to spread and a health organization chose to query asking about DOs' with "Diabetes" and "Pregnancy" attributes, then that might cause all DOs with Diabetes and Fever to panic and unnecessarily flood hospitals for unneeded test. The initial solution we present in Section 5.3 hides the query predicates from the data owners to prevent the data owners from knowing the specifics of what the querying entity is interested in. We chose not to hide the query attributes in the initial solution because one of our goals is to return control of data to the DOs themselves. DOs should have the right to prevent the querying of any attribute or any combination of attributes of their data. In Section 5.4, we present a second solution as an extension to the initial protocol to address this threat by hiding the attributes from the DOs.

5.3 Proposed Protocol

The purpose of this approach is to propose a protocol that enables querying privately encrypted data on a semi-trusted cloud while preserving the privacy of both the DOs and the

QE without requiring a trusted entity.

The protocol uses the Goldwasser-Micali (GM) cryptosystem along with Fischlin's protocol to enable DOs to securely compare values in their data to the encrypted constants in the QE's query.

The protocol additionally uses threshold encryption to allow the QE and the CS to cooperate in order to execute the query without compromising the privacy of either the QE or the DO. Threshold encryption allows us to split an encryption or decryption key into n shares, where only k of the n shares are needed to encrypt or decrypt a message [71]. Threshold encryption allows the CS to calculate the response to the query over encrypted data without trusting it to see the query constants, the DO data, or the result of the query.

5.3.1 Notations

Throughout this chapter, we will be using the following notations:

- QE: Querying entity.
- CS: Cloud server.
- Q: Query sent by QE with the query attributes and the encrypted constants.
- Q_c: Query constants.
- Q_a: Query attributes.
- PK_{gm_i}: The i^{th} private GM-threshold key share.
- pk_{gm}: Public GM-threshold key.
- PK_{gm_i}: The i^{th} private Paillier-threshold key share.
- pk_{pa}: Public Paillier-threshold key.

5.3.2 Functions

Throughout this chapter, we will be using the following functions:

- $PaillierEnc_{pk}(m)$: Encrypts a message m using the Paillier cryptosystem and the session public key [25].
- $GmEnc_{pk}(m)$: Encrypts a message m using the GM cryptosystem and the session public key [24].
- $PaillierSum\,(a,b)$: Runs the Paillier secure sum algorithm adding the two encrypted values a and b, where a and b are both encrypted with the same key using the Paillier cryptosystem [25]. The function outputs an encrypted value c, which is also encrypted with the same key used for a and b.
- $RunFischlin\,(a,b)$: Runs Fischlin's algorithm to compare a and b, which are both encrypted with the same key and using the GM cryptosystem [88]. The functions outputs two values (Δ,c), which are also encrypted with the same key used for a and b. Decrypting (Δ,c) allows us to analyze their values to determine whether a is greater than or equal to b.
- $FischlinResAnalysis(\Delta,c)$: Analyzes the decrypted values of (Δ,c) to determine the comparison result of the two input values to the $RunFischlin\,(a,b)$ function that was used to generate (Δ,c) [88].
- $PaillierDecrypt(m,PK_a)$: Runs the Paillier-threshold decryption algorithm to partially decrypt the message m using a key share a of the private key PK [25].
- $GMDecrypt(m,PK_a)$: Runs the GM-threshold decryption algorithm to partially decrypt the message m using a key share a of the private key PK [24].

5.3.3 Protocol Phases

The protocol is split into four main phases:

- **Setup phase:** In this phase, the CS and QE agree on the session encryption keys required for the QE's query.
- **Query distribution phase:** In this phase, the QE encrypts the queries and sends them to the CS. The CS then processes the queries and forwards them to the DOs.
- **Data Owner Query execution phase:** In this phase, the DOs run the comparison and sum algorithms on the QE's data and their own encrypted data, and send the results to the CS.
- **Cloud server query execution phase:** In this phase, the CS analyzes and calculates the final result of the query with the help of the QE.

The protocol differs in its operations according to the query type. In other words, the overall structure stays the same; however, there are three deviations according to three query types (*Sum/Avg*, *Range*, and *Hybrid* queries).

5.3.4 Setup Phase

This section discusses how the QE and the CS start a new query session in our protocol using the GM-threshold cryptosystem along with Fischlin's protocol.

In the first step of the protocol, the secure key agreement protocol is run jointly by the QE and the CS to generate the session public key pk for k-out-of-n threshold GM cryptosystem or the Pailler cryptosystem such that at least k of n participants are required to fully decrypt a ciphertext [88, 90]. In this solution, we set the threshold k to 2 and the number of participants n to 2. The generated threshold keys will be kept privately by the QE and the CS.

The QE sends a query session request to the CS along with the type of query (range, sum, or hybrid). The identities of the QE and the CS must be verified at this stage (e.g. by using digital signatures). The type of query determines which keys are generated using the secure key agreement protocol as follows:

- For sum or average queries, the QE and CS generate a Paillier-threshold key pair, splitting the private key PK_{pa} into two shares PK_{pa1} and PK_{pa2}. The key PK_{pa1} will be used by the QE and the key PK_{pa2} will be used by the CS.

Phase 1: Sum/Avg query example

To find the average age of DOs in the system, the following query Q will be sent by the QE to the CS:

$AVG\,(age)\,from\,DOs$

- For range queries, which include queries such as equality, range, and Max/Min queries, the QE and CS generate a GM-threshold key pair, splitting the private key PK_{gm} into two shares PK_{gm1} and PK_{gm2}. The key PK_{gm1} will be used by the QE and the key PK_{gm2} will be used by the CS.

Phase 1: Range query example

To find the number of DOs under the age of 20 that have diabetes, the following query Q will be sent by the QE to the CS:

$COUNT\ (*)\,from\,DOs\,WHERE\,age < (Enc_{pk_{gm}}(20) \wedge Diabetes = Enc_{pk_{gm}}(1))$

- For hybrid queries with range and sum operations, the QE and CS generate a GM-threshold and a Paillier-threshold key pair, splitting each private key into two shares.

Phase 1: Hybrid query example

To find the average age of female DOs with diabetes in the system, the following query Q will be sent by the QE to the CS:

$AVG\,(age)\,from\,DOs\,WHERE\,gender = Enc_pk_{gm}(1)\,AND\,Diabetes = Enc_pk_{gm}(1)$

The session public keys pk_{gm} for the GM-Theshold cryptosystem and pk_{pa} for Paillier-Theshold cryptosystem will be separately signed by both parties using their trusted certificates before being sent to the DOs with the queries in the upcoming steps of the protocol.

5.3.5 Query Distribution Phase

The QE encrypts the constants Q_c in the query Q using the session public key pk_{gm} or pk_{pa} depending on the type of query. The QE then forwards the query to the CS.

The CS then distributes the attributes Q_a in the query Q, the encrypted constants Q_c, the public session key pk_{gm} or pk_{pa}, and the cryptosystem to be used (*Paillier-threshold* or *GM-threshold*) to all the DOs.

- In the case of sum or average queries,the DOs would be requested to use the *Paillier cryptosystem*.

Phase 2: Sum/Avg queries

```
1: $QE \rightarrow CS : Q$
2: $CS \rightarrow DOs :\ Q_a[\,]; Paillier; pk_{pa}$
```

- In the case of range queries, the DOs would be requested to use the *GM cryptosystem*.

Phase 2: Range queries

```
1: $QE \rightarrow CS : Q$
2: $CS \rightarrow DOs :\ Q_a[\,]; GMEnc_{pk_{gm}}(Q_c[\,]); GM; pk_{gm}$
```

- In the case of a hybrid query, both of the above actions are required. The CS sends two messages to the DOs: one for the comparison part of the query and another for the Avg/Sum part of the query.

Phase 2: Hybrid queries

```
1: $QE \rightarrow CS : \{Q\}$
2: $CS \rightarrow DOs : \{ Q_a[];\ Paillier;\ pk_{pa}\}$
3: $CS \rightarrow DOs : \{ Q_a[];\ GMEnc_{pk_{gm}}(Q_c[]);\ GM,\ pk_{gm}\}$
```

The queries are not sent to the DOs, but the attributes and the encrypted constants are sent in arrays of the same size. In the case of comparison queries, the constants in the array correspond to the attributes in the same positions in the attributes array.

5.3.6 Data Owner Query Execution Phase

The DOs retrieve the values related to the attributes in the QE's query from the cloud.

- If the DOs are requested to use the Paillier cryptosystem, they encrypt the values retrieved from the cloud and encrypt them using the Paillier cryptosystem. The final encrypted result is then forwarded to the CS.

Phase 3: Sum/Avg queries

```
1: $for\, each\, Q_a\, do$ {
2: $results_{pa}[i] = Enc_{pk_{pa}}(DOdata.Q_{a_i}))$ }
3: $DOs \rightarrow CS :\ results_{pa}[]$
```

- If the DOs are requested to use the GM cryptosystem, they encrypt the values retrieved from the cloud and run Fischlin's protocol on their values and the encrypted values sent from the QE. The DOs then forward the results (Δ, c) to the CS.

Phase 3: Range queries

```
1: $foreach\,Q_a\,do\,\{$
2: $results_{gm}[i] = RunFischlin(GMEnc_{pk_{gm}}(Q_{c_i}), GMEnc_{pk_{gm}}(DOdata.Q_{a_i}))\ \}$
3: $DOs \rightarrow CS:\ results_{gm}[\,]$
```

- In the case of a hybrid query, both of the above actions are required, and the DOs send the CS the final two arrays $results_{gm}$ and $results_{pa}$.

5.3.7 Cloud Server Query Execution Phase

The CS stores the results received from the DOs into an array and waits until it receives the required number of results from the DOs. Depending on the query type, the CS then takes the following steps:

- In the case of a sum/average query, the CS runs the addition algorithm on all the Paillier encrypted values received from the DOs. The CS then partially decrypts the result using its share of the private key. The partially decrypted result is then sent to the QE along with the number of DOs. The QE then decrypts the results using its private key share. If the query was an average query, the QE also divides the result by the number of DOs.

Phase 4: Sum/Avg queries

```
1: $CS: EncResult = PaillierSum(results_{pa}[\,])$
2: $CS: PartiallyDecResult = PaillierDecrypt(EncResult; PK_{pa_2})$
3: $CS \rightarrow QE: PartiallyDecResult; NumberOfDOs$
4: $QE: FinalResult = PaillierDecrypt(PartiallyDecResult; PK_{pa_1})$
   If average is needed:
5: $QE: Avg = FinalResult/NumberOfDOs$
```

- In the case of range queries, all results are sent to the QE in an array. The QE then randomizes the order of the elements in the array. The purpose of the randomization by the QE is to prevent the CS from correlating the result of any comparison to a specific DO. The QE then decrypts all the elements in the array using its share of the private key PK_{gm_1}. The new array is then sent to the CS.

  The CS then uses its share of the private key to complete the decryption of the $(\varDelta, c)$ elements in the array. The CS can then use $(\varDelta, c)$ to determine the results of the comparisons between the encrypted values in the DO's data and the encrypted constants Q_c in the QE's query. The QE then counts the number of results that satisfy the conditions in the QE's query. The final result is then sent to the QE.

Phase 4: Range queries

```
1: $CS \rightarrow QE\,; results_{gm}[\,]$
2: $QE : PartiallyDecResult[\,] = GMDecrypt(Rand(results_{gm}[\,]); PK_{gm_1})$
3: $QE \rightarrow CS; PartiallyDecResult[\,]$
4: $CS : DecryptedResult[\,] = GMDecrypt(PartiallyDecResult[\,]; PK_{gm_2})$
5: $CS : if(FischlinResAnalysis(DecryptedResult[i]))\ counter{+}{+};$
6: $CS \rightarrow QE\,; counter$
```

- In the case of a hybrid query, the CS sends the GM-threshold and Paillier-threshold encrypted result arrays to the QE. The QE then randomizes the order of the elements in both arrays. The same randomization order is applied to both arrays to maintain the relationships between them. The QE then decrypts all the elements in the GM-threshold array using its share of the private key. The QE then chooses a new random value, encrypts it using the public Paillier key, and runs the homomorphic addition algorithm on all the values in the Paillier encrypted results array. The new arrays are then sent to the CS. The CS then uses its share of the private key to complete the decryption of the (Δ, c) elements in the GM-threshold encrypted array.

The CS can then use (Δ, c) to determine the results of the comparisons between the encrypted values in the DO's data and the encrypted values sent by the QE. The CS then runs the addition homomorphic protocol on the values in the Paillier array, which are in the positions of the value that satisfy the condition in the GM-threshold encrypted array. The CS then partially decrypts the encrypted sum result using its share of the Paillier private key. The CS then sends the QE the partially decrypted sum result along with the number of DOs, which satisfy the query condition if the QE's query required the average value. The QE then decrypts the result using its private key share and deducts the random value from the result, which is the random value multiplied by the number of DOs. If the query was an average query, the QE also divides the result by the number of DOs. The flowchart in Fig.5.2 shows the complete process of executing a hybrid query.

Phase 4: Hybrid queries

```
1: $CS \rightarrow QE : results_{gm}[\,]; results_{pa}[\,]$
2: $QE : PaillierSum(rand; results_{pa}[\,])$
3: $QE : Randomize(results_{gm}[\,]; results_{pa}[\,])$
4: $QE : PartiallyDecGMResult[\,] = GMDecrypt(results_{gm}[\,]; PK_{gm_1})$
5: $QE \rightarrow CS : PartiallyDecGMResult[\,]; results_{pa}$
6: $CS : DecryptedResult[\,] = GMDecrypt(PartiallyDecGMResult[\,]; PK_{gm_2})$
7: $CS : if(FischlinResAnalysis(DecryptedResult[i]))$
8:     $\{\ sumResult += results_{pa}[i];$
9:     $counter{+}{+}; \}$
```

Phase 4: Hybrid queries (continued)

```
10: $CS : PartiallyDecResult = PaillierDecrypt(sumResult; PK_{pa_2})$
11: $CS \rightarrow QE : PartiallyDecResult; counter$
12: $QE : FinalResult = PaillierDecrypt(PartiallyDecResult; PK_{pa_1}) - (rand * counter)$
    If average is needed:
13: $QE : Avg = FinalResult / counter$
```

5.4 Hiding Query Attributes

In this section we will address the threat we described in Section 5.2.3. In the solution we presented so far, the query attributes are sent to the clients in plaintext. The purpose of keeping them in plaintext was to give the control to the data owners to decide which attributes can be queried. However, hiding the query attributes will further preserve the privacy of the QE without decreasing the privacy of the data owners.

We now present an extension to the protocol to also hides the query attributes from the DOs. To hide the query attributes, we modify the query distribution phase and the data owner query execution phase.

5.4.1 Modified Query Distribution Phase

To hide the attributes from the DOs we will first send a query request to the DOs. Each DO then generates a Paillier key pair (DO_pk_{pa} and DO_PK_{pa}) and uses the public key to encrypt all their data and send it to the CS in an array along with the generated public key. The CS would then be able to choose the attributes related to the QE's query from the array sent by the DO. The CS then creates a response array with only the encrypted values of the attributes in the query.

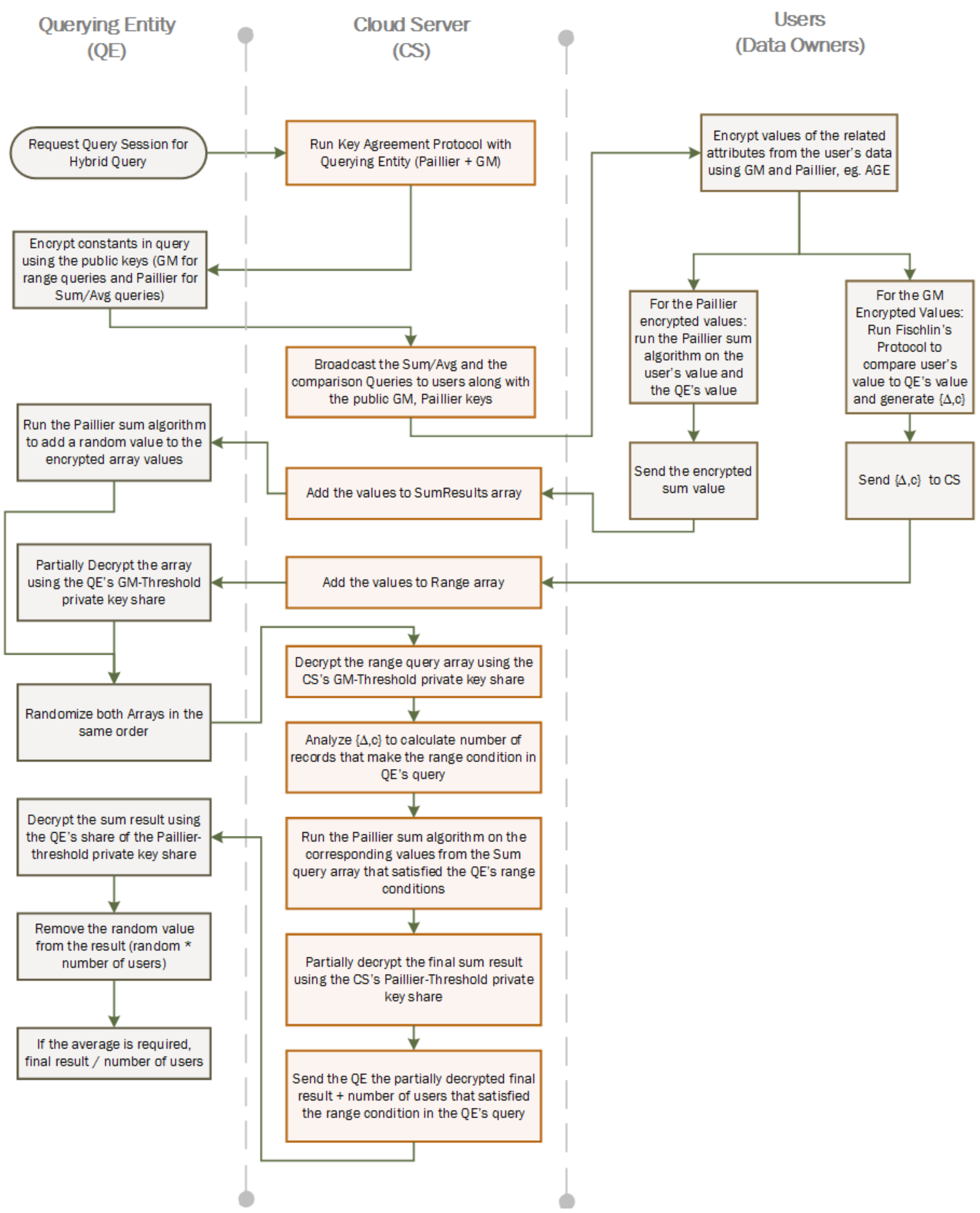

Querying Entity (QE)
Cloud Server (CS)
Users (Data Owners)
Request Query Session for Hybrid Query
Run Key Agreement Protocol with Querying Entity (Paillier + GM)
Encrypt values of the related attributes from the user's data using GM and Paillier, eg. AGE
Encrypt constants in query using the public keys (GM for range queries and Paillier for Sum/Avg queries)
For the Paillier encrypted values: run the Paillier sum algorithm on the user's value and the QE's value
For the GM Encrypted Values: Run Fischlin's Protocol to compare user's value to QE's value and generate {Δ,c}
Broadcast the Sum/Avg and the comparison Queries to users along with the public GM, Paillier keys
Run the Paillier sum algorithm to add a random value to the encrypted array values
Add the values to SumResults array
Send the encrypted sum value
Send {Δ,c} to CS
Partially Decrypt the array using the QE's GM-Threshold private key share
Add the values to Range array
Randomize both Arrays in the same order
Decrypt the range query array using the CS's GM-Threshold private key share
Analyze {Δ,c} to calculate number of records that make the range condition in QE's query
Decrypt the sum result using the QE's share of the Paillier-threshold private key share
Run the Paillier sum algorithm on the corresponding values from the Sum query array that satisfied the QE's range conditions
Remove the random value from the result (random * number of users)
Partially decrypt the final sum result using the CS's Paillier-Threshold private key share
If the average is required, final result / number of users
Send the QE the partially decrypted final result + number of users that satisfied the range condition in the QE's query

Figure 5.2: Hybrid Query Execution Flowchart

The CS then chooses a random value for each attribute. For example, if the QE query was concerned with the DOs' "age" and "salary", we would have a one random value for "age" and another for "salary". The same random value would be used for the "age" attribute for all DOs.

The CS then uses the generated public Paillier key to encrypt the random values and then add them to the encrypted values in the response array it created. The response array is then sent to the DO.

Phase 2: Hybrid queries

```
1:  $QE \rightarrow CS : Q$
2:  $CS :$
3:  $foreach\, Q_a\, in\ Q\ \ do\,\{$
4:  $Rand[i] = GenerateRandomValue();\}$
5:  $CS \rightarrow DOs : QueryRequest()$
6:  $Each\ DO :$
7:  $GeneratePaillierSessionKey()$ % Generates Public $S\_pk_{pa}$ and Private $S\_PK_{pa}$ Keys
8:  $foreach\, Q_a\, in\ DOdata\ \ do\,\{$
9:      $DATA_{pa}[i]\ =\ Enc_{S\_pk_{pa}}(DOdata.Q_{a_i}))\ \}$
10: $DOs \rightarrow CS :\ DATA_{pa}[\,];S\_pk_{pa}$
11: $CS :$
12: Extract the elements from $DOdata[\ ]$ that correspond to the $Q_a$s in $Q$
13: $foreach\ DOdata.Q_a\ in\ Q_a\ \ do\,\{$
14:     $attribute[i] = PaillierSum(DATA_{pa}[i], Enc_{S\_pk_{pa}}(Rand[i]))$
15:     if $Q_a$ is related to Sum/Avg part of the query {
16:         $crytosystem[i] = Paillier\}$
```

Phase 2: Hybrid queries (continued)

```
17:     else if $Q_a$ is related to the comparison part of the query {
18:         $crytosystem[i] = GM$}
19: }
20: $CS \rightarrow DO : attribute[]$ ; $cryptosystem[]; pk_{pa}; pk_{gm}; GMEnc_{pk_{gm}}(Q_c[])$
```

5.4.2 Modified Data Owner Query Execution Phase

In this phase, the data owners are to execute the query without knowing which of their attributes are being queried. The participation of the DOs is needed in this phase because it is not possible to run the encrypted homomorphic operation without having the data encrypted with the session public key. For that reason, the DO needs to decrypt the values in the $attribute[]$ array to be able to execute the homomorphic operations on the QE's query.

For the sum/avg part of the query, the DO simply re-encrypts the value of the attribute using the public Paillier session key pk_{pa} and add the result to $results_{pa}[]$. For the comparison part of the query, the DO needs to compare the re-encrypted value to the GM

Phase 3: Hybrid queries Sum/Avg queries

```
1: $for\,each\,attribute[]\,do${
2: $if\ (cryptosystem[i] == Paillier)$ {
3:     $results_{pa}[counterPA] = Enc_{pk_{pa}}(PaillierDecrypt(attribute[i], S\_PK_{pa})$
4:     $counterPA++$ }
5: $else\ if\ cryptosystem[i] == GM$ {
6:     $results_{gm}[counterGM] = RunFischlin(GMEnc_{pk_{gm}}(Q_{c_{counterGM}}),$
7:     $GMEnc_{pk_{gm}}(PaillierDecrypt(attribute[i], S\_PK_{pa}))$
8:     $counterGM++$ }}
9: $DOs \rightarrow CS:\ results_{pa}[]\,;\,results_{gm}[]$
```

5.5 Differentially Private Query Results

In this section, we will be addressing the threat model described in Section 5.2.3. The mechanism we presented in Section 5.3 securely executes aggregate and comparison queries while preserving the privacy of the QE and the DOs. The mechanism produces accurate results for each query. However, if the QE were to create a query with the goal of exposing PII about a specific DO, then the identity and data of a specific DO could be revealed.

To address this issue, we present an extension to the mechanism proposed in Section 5.3 to add differential privacy. Rather than returning the accurate result, the result becomes an approximation of the actual result. Formally, differential privacy is defined as follows:

A randomized function K gives ε-differential privacy if for all data sets D and D' differing on at most one row, and all $S \subseteq Range(K)$, $Pr[K(D) \in S] \leq exp(\varepsilon) \times Pr[K(D') \in S]$

Differential privacy ensures that the risk to a DO's privacy should not substantially decrease when responding to statistical queries. This means the knowledge a curious QE can gain about a specific DO, is not affected by the participation of the DO in responding to the query. This provides DOs with the assurance that the risk to the DO's privacy for participating and responding to a query is low. One method for adding noise involves the use of the Laplace mechanism to add random noise that conforms to the Laplace statistical distribution [183].

The Laplace distribution has two parameters, the location and scale parameters. The value of the location parameter will be set to zero to keep the noisy result close to the accurate result. The scale parameter is directly proportional to its standard deviation, or noisiness. The value we choose as the scale depends on the value of the privacy parameter ε, and the QE's query. Moreover, it corresponds to the maximum difference a DO can have on the result of the query f. This is known as the sensitivity of the query f, and can be defined mathematically as follows:

$$\triangle f =_{D,D'}^{max} \| f(D) - f(D') \|_1$$

$$=_{D,D'}^{max} \sum_{i=1}^{d} |f(D)_i - f(D')_i|$$

for all D, D' databases differing in at most one row. Dwork *et al.* [184] presents a proof showing that by adding a random Laplace($\Delta f/\varepsilon$) variable to a query, ε-differential privacy is guaranteed.

5.5.1 Query Sensitivity

In the case of count queries, the maximum effect a DO can have on a query result is 1, for that reason the query sensitivity would always be equal to 1. For sum queries, the query sensitivity would be equal to the maximum value an attribute could have. In our mechanism, the CS would have a list of all attributes and the maximum value an attribute can have. For that reason, the query sensitivity would also be equal to 1 for binary value attributes. For larger value attributes, the sensitivity would be equal to the maximum value of that attribute. For example, if we wanted to execute a sum query on the "Diabetes" attribute, which specifies whether a DO has diabetes, then the sensitivity would be equal to 1. If we wanted to execute a sum query on the "Salary" attribute, it would be equal to the maximum defined salary value. Finally, for average queries the value of the sensitivity is equal to the maximum value defined value for the queried attribute divided by the number of DOs responding to the query.

5.5.2 Modified Protocol

To modify the protocol we introduced in Section 5.3, we first need to modify the noisy query response before returning it to the QE. There are two modifications that need to be made to the noise value. First of all, for the noisy query response to make sense, the value must be an integer, and for that reason we will be rounding the laplace noise before adding it to the actual result.

The second modification is to ensure the noisy query response is not a negative value. The laplace noise can be positive or negative, but for the query response to make sense we need the noisy response to be zero or more. In our mechanism, the query is executed by the CS, which means that for comparison queries, the CS would know the number of entities that satisfy the query conditions. We do not consider this a security concern because the CS will not be able to know what the query constants are or which DOs satisfy the query conditions. Since the CS knows the noise value and the number of DO that satisfy the query conditions, it would also know whether the noisy response is positive or negative. In the case of a negative noisy query response, the CS would always respond with a zero. However, in the case of sum or average queries, we propose aggregating the query sensitivity to the noisy response. This would enable the QE to decrypt the noisy query response. If the result is negative, the QE would consider the result to be zero. The modifications to the protocol would only change the cloud server execution phase we present in Section 5.3.7:

Phase 4: Hybrid queries

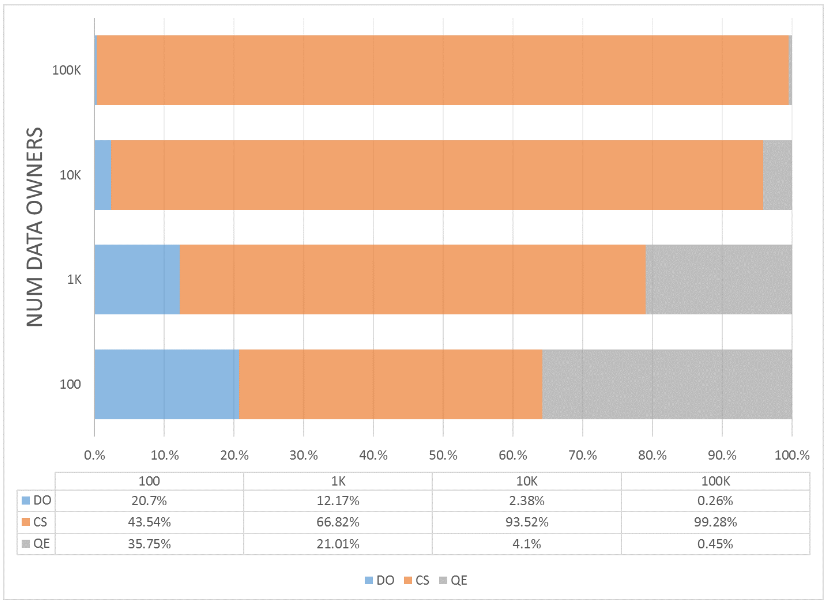
```
1: $CS \rightarrow QE : results_{gm}[]; results_{pa}[]$
2: $QE : PaillierSum(rand; results_{pa}[])$
3: $QE : Randomize(results_{gm}[]; results_{pa}[])$
4: $QE : PartiallyDecGMResult[] =$
5: $GMDecrypt(results_{gm}[]; PK_{gm_1})$
6: $QE \rightarrow CS : PartiallyDecGMResult[]; results_{pa}$
7: $CS : DecryptedResult[] =$
8: $GMDecrypt(PartiallyDecGMResult[]; PK_{gm_2})$
9: $CS : if(FischlinResAnalysis(DecryptedResult[i]))$
10:     $\{sumResult += results_{pa}[i];$
11:     $counter++;\}$
12: $sumResult = PaillierSum(sumResult, LaplaceNoise(0, querySensitivity(Q)))$
13: $sumResult = PaillierSum(sumResult, querySensitivity(Q))$
14: $CS : PartiallyDecResult =$
15: $PaillierDecrypt(sumResult; PK_{pa_2})$
16: $CS \rightarrow QE : PartiallyDecResult; counter; querySensitiviy(Q)$
17: $QE : FinalResult = PaillierDecrypt(PartiallyDecResult; PK_{pa_1}) - (rand * counter) -$
    $querySensitiviy(Q)$
    If average is needed:
18: $QE : Avg = FinalResult / counter$
```

5.6 Security Analysis

In this section, we address the security threats mentioned in Section 6.2.3:

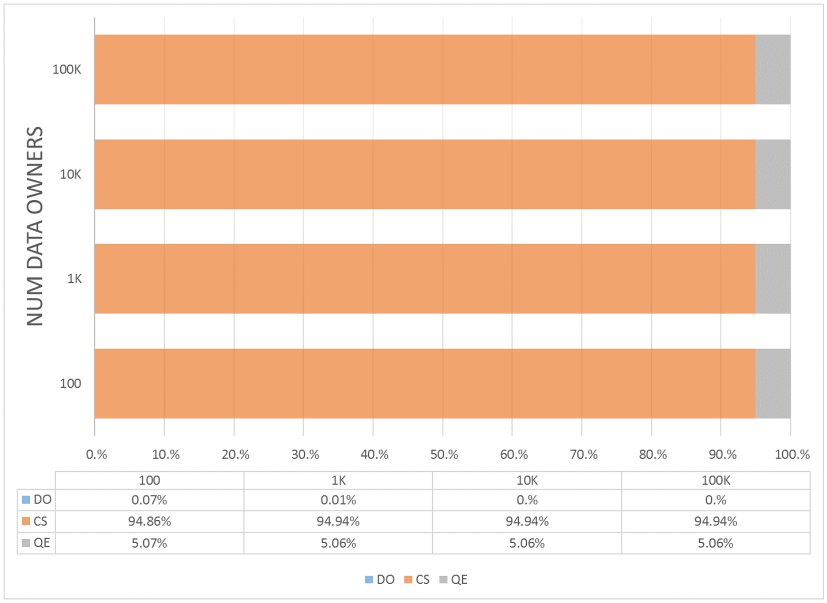
Approach

Our solution aims at protecting the confidentiality of the querying entity data by preventing the CS from accessing the private keys. The querying entity encrypts the query constants with the session public key. The CS and the data owners can encrypt data and run homomorphic computations on the data using only public keys. The querying entity is also unable to access the final result without having the private key. However, the CS needs the private keys to be able to preform comparisons on encrypted data. To enable the CS to complete these computations, we rely on threshold encryption with a randomization algorithm. Together, they allow the CS to complete the secure computations with the assistance of the QE while protecting the confidentiality of the data owners' data and the QE's query constants.



	100	1K	10K	100K
DO	20.7%	12.17%	2.38%	0.26%
CS	43.54%	66.82%	93.52%	99.28%
QE	35.75%	21.01%	4.1%	0.45%

Figure 5.3: The Processing Load Distribution for Sum Queries



	100	1K	10K	100K
DO	0.07%	0.01%	0.%	0.%
CS	94.86%	94.94%	94.94%	94.94%
QE	5.07%	5.06%	5.06%	5.06%

Figure 5.4: The Processing Load Distribution for Hybrid Queries

Guarantees

Confidentiality is provided by our solution for query constants, DOs data, and the secure computation results. The solution does not hide the attributes in the query or the number of data owners responding to a query. However, the solution does protect the identities of the data owners that satisfy the query range and prevents the CS from correlating a any results to specific DOs. The security of our solution is not perfect: The number of DOs that satisfy the range conditions in the query is revealed to the CS. Also, the number of DOs that responded to a query is revealed to the QE, but the number of those that satisfy the range conditions is not revealed. Finally, the solution does not protect against queries that target specific data owners. For example, a QE can send a query to find if a specific person has specific disease. By sending a sum query of all DOs that have that disease and combining the query with other identifiers such as location, age, gender, etc. The result of the query can reveal the data of a specific DO. To minimize this problem, the solution prevents the CS from sending the final result of the query to the QE in case the result of the query condition only involves a single data owner. This technique is not prefect because a malicious QE can craft complex queries to reveal data of specific DOs, and this attack is not addressed by our solution. More intuitively, our solution provides the following security properties:

- ***The query constants and the random values cannot be recovered without the participation of the QE***: The QE sends the sanitized query constants and the random values encrypted using a semantically secure cryptosystem: Paillier-threshold in the case of sum queries, or GM-threshold in the case of range queries. This means that the query constants and the random values cannot be recovered without the participation of the QE [24, 25].

- ***The cloud server cannot correlate any of the results to specific data owners***: In the *cloud server query execution phase*, the CS sends the results array $results_{gm}$ to the QE for partial decryption in the case of range queries, and sends both arrays $results_{gm}$ and $results_{pa}$ in the case of a hybrid query. Randomizing the order of $results_{gm}$ along with the partial decryption prevents the CS from correlating a result to a specific DO. However, in the case of hybrid queries, the QE has to randomize both arrays in the same order before sending them back to the CS. This renders the partial decryption ineffective in preventing the CS from correlating a result to a specific DO. It is for this reason that we must add a random value to all the elements in the $results_{pa}$ array using the Paillier sum algorithm. This ensures that the CS executes the query using the randomized array $results_{pa}$ and the partially decrypted $results_{gm}$ without being able to correlate any of the results to a specific DO.

- ***Data owner data cannot be decrypted or calculated by the cloud server or the querying entity***: In regard to range queries, the DOs do not send their actual data encrypted, but instead send the results of the Fischlin's private comparison protocol (Δ, c), which are then used to compare the encrypted values.

In the case of sum/avg or hybrid queries, the DOs send their data encrypted using the Paillier-threshold cryptosystem after adding a random value to it using the Paillier sum algorithm. Adding the random value prevents the CS from calculating the DO data in case the CS sends the QE the $results_{pa}$ array in place of the $results_{gm}$ array in the hybrid case of the *cloud server query execution phase*, thus receiving a partially decrypted $results_{pa}$ array, which it can then decrypt, using its Paillier-threshold private key share, to access the actual DO data. Under our semi-honest adversary model assumption, we do not need to add this extra step; however, due to the simplicity of this attack and due to the fact that it would expose the DOs' actual data, we felt that adding this step is a necessity.

- ***Data owners cannot know what attributes are being queried by the querying entity***: In the extension we present in section 5.4, the DO send their data encrypted using a public key, which they generated for that session. No entity has access to the corresponding private key, which means the confidentiality of the DO's data is protected. The DO also sends the public keys to allow the CS to perform the homomorphic addition of random values to the DO's values corresponding to the attributes related to the QE's query. The CS then sends the noisy values back to the DO in random order. This ensures that the DO will receive noisy values in random order, thus preventing the DO from knowing which attributes the QE was interested in.

- ***The QE cannot learn about Personally Identifiable Information (PII) about a specific data owner***: In the extension we presented in Section 5.5, the CS adds a noisy value to the final response to protect the privacy of DOs in the system. The noise is chosen according to the type of query to ensure that the response is as accurate as possible while satisfying the definition of differential privacy. The noise is chosen from a laplace distribution where the maximum amount of noise is equal to the maximum difference a single DO can have on the response of query. This ensures that even if the QE manages to create a query that should result in PII, it cannot be sure whether the final result of the query is PII or noise.

5.7 Performance Evaluation

In this section, we present our experimental result. We rely on the projects used in Section 4.7, which are open-source [27]. The code is written in Java and executed on Amazon's elastic cloud (EC2) [185]. Amazon allows users to run their applications on virtual machines, called instances. Amazon offers a variety of instance types with different configurations of CPU, RAM memory, and ROM storage. Amazon EC2 is built on commodity hardware, and over time there may be several different types of physical hardware underlying EC2 instances. Using this method allows Amazon to provide consistent amounts of processing power regardless of the actual underlying physical hardware. A single EC2 compute unit produces the equivalent CPU capacity of 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. We run our implementation on an Windows instance with the following specs:

- 128 vCPUs
- 1,952 GiB of memory
- 2 x 19,20 GB of SSD instance storage
- Instance Type: x1.32xlarge

The execution time in our work depends greatly on the type of query and the number of attributes in the query. To demonstrate the required processing time, we take sum and hybrid queries as an examples. Sum queries simply aggregate a value of a common attribute among data owners. Hybrid queries are aggregate queries that also include comparisons. For that reason, they are the most expensive ones since they utilizes all the building blocks. As previously mentioned, our goal is to allow a QE to query privately encrypted distributed data with the assistance of a semi-trusted cloud. We run our code assuming 100, 1000, 10k, and 100k data owners to demonstrate the practicality of our approach. We do not take network delays into account. We also assume data owners run their computations in parallel. To test the performance we execute the following two queries:

$$AVG\,(doctorVisitsPerYear)\,FROM\,data\_owners$$

$$AVG\,(doctorVisitsPerYear)\,FROM\,data\_owners\,WHERE\,age > 30\,AND\,diabetes = 1$$

Our implementation shows how the overall load on the CS increases with the number of data owners as shown in Figures 5.3 and 5.4. Our results show that executing the sum query on 10k DOs requires 0.023 seconds. However, hybrid queries require 2.34 minutes due to the high computational overhead of Fischlin's protocol. These computations can be done in parallel, which allows us to reduce the computation time with the use of cluster computing solutions such as Apache Spark [186]. In fact, with the assistance of 100 clusters of the same instance used, the hybrid query can be executed for 1 million data owners in less that 3 minutes. These results show that our approach can be reasonably used for large set of data owners to provide the results in timely manner.

5.8 Conclusion

In this Chapter, we described the problem and challenges of securely querying privately encrypted data, where data owners store their data in the cloud encrypted with their own private keys, while preserving the privacy of all the parties involved. We proposed a protocol that allows third parties to execute various types of queries on privately encrypted data stored in an untrusted cloud, while preserving the privacy of the data owners as well as the querying entity. The protocol relies on semantically secure probabilistic cryptosystems. It also allows for range, sum, average, and hybrid queries. The protocol relies on two cryptosystems along with Fischlin's protocol for private comparisons. We also present the results of our implementation, which show the feasibility and scalability of our approach. Finally, we analyzed the security of our protocol.

Chapter 6

Secure Multi-Party Private Function Evaluation

Secure multi-party computation (SMC) allows multiple parties to jointly and securely compute a function while preserving the privacy of all the parties involved. Homomorphic cryptosystems allow users to perform addition or multiplication operations on encrypted values without having to decrypt the function input. In this chapter, we propose a cryptographic and a non-cryptographic privacy-preserving protocols that allow a participant to collaboratively compute a polynomial function with at least two other participants using semantically secure cryptosystems. We experimentally evaluate the performance of the proposed protocol and report on the results of our implementation.

6.1 Introduction

Secure function evaluation can be useful in solving problems where multiple parties would like to assist each other, but are bound by privacy policies. Currently, most homomorphic cryptosystems either allow the parties to calculate addition or multiplication operations, but not both. In 2009 Gentry et al. [82] proposed a solution to support fully homomorphic operations. However, the processing resources required are too high for the solution to be feasible for most applications. In an optimized implementation on a high-end workstation for a large setting, key generation takes 2.2 hours and encryption takes 3 minutes [23]. In this work, we propose a privacy-preserving protocol that would allow multiple parties to jointly compute polynomial functions with addition and multiplication operations.

In this work, we address the trust issue by proposing a cryptographic protocol that relies on a semantically-secure cryptosystem to allow an entity to request the assistance of at least two other entities to securely calculate a private polynomial function while preserving the privacy of the function and all the values provided by the other entities. Since homomorphic encryption is known to need heavy computation, we also presented a scalable non-encryption-based solution.

Using the solutions we propose in this work, analysts or organizations would be able to calculate, for instance, the weighted financial effect a phishing attack had on corporations while preserving the privacy of the affected organizations and the corporations. The solution may also be useful in medical studies where a research center would like to securely compute a function jointly with the assistance of medical centers nationwide while preserving the privacy of the patients' data. Moreover, the protocol may also be useful in allowing distributed decision makers receiving plan execution monitoring feeds to generate parameters of interest for situational awareness.

6.2 Execution Environment

In this section, we identify the involved entities and then, we present more details about the assumptions underlying the system design. The entities involved are decision makers who own private data and want to keep it confidential. A common way to protect private data is through encryption.

6.2.1 Assumptions

We assume that there is no fully trusted entity in the environment and that all entities are semi-honest. Semi-honest players follow the protocol steps, but they try to extract information about other entities' input or output. This is a common security assumption used in secure multi-party computation literature [167] and it is realistic in our problem scenario since different decision makers are collaborating for mutual benefits. Hence, it is reasonable to assume that parties will not deviate from the defined protocol. However, they may be curious to learn additional information from the messages they receive during protocol execution. We also assume that there is no collusion between the different parties and that there are mechanisms that ensure integrity and availability of data. Our scheme focuses only on confidentiality and privacy issues and does not address issue such as data tampering and denial of service.

6.2.2 Problem Statement

Given a set of N parties $D_i \in \{D_1, D_2, \ldots D_N\}$, $i = 1..N$, each with access to private variables x_i, securely compute polynomial form non-trivial secret functions $f_{D_j}(x_i)$ known only to $D_j, j = 1..N$ such that the value of $f_{D_j}(x_i)$ is known only to D_j and x_i remain known only to D_i.

In particular, $f_{D_j}(x_i)$ may be the same for all decision makers in the case where only the arguments of the function are private for the decision makers.

6.2.3 Threat Model

The proposed solution guards against a curious function owner or a participant. Our goal is to guarantee data confidentiality, ensuring the secrecy of the private values of all the participants and the function owner's private function. The solution does not guarantee the integrity of the data. Since we assume the semi-honest model, the attacker is assumed to be passive. This implies that the attacker will attempt to analyze all the messages received or sent by the compromised entity. The attacker's goal in this threat is to learn about the private values of the participants, the function owner's private function, the results of any intermediate steps, or the final result. This threat is becoming increasingly important with the increasing popularity of migrating data centers to the cloud. We address this threat in Sections 6.3 and 6.4.

6.3 Approach

In order to allow a participating decision maker to securely calculate a wide range of possible functions with multiple multiplication and sum operations based on private values accessible only to other decision makers, we propose the following protocol:

Given a multivariate function in the form of a polynomial, with an expression known to only one of the decision makers, we need to collaboratively compute its value without exchanging the values of the underlying variables.

For instance, let us consider the following example: Given a situation where three decision makers (e.g. incident response teams) A,B,C with their respectively secret values x,y,z and function $f(x,y,z) = x^2y + xz$, known only to A, jointly calculate f in a secure manner such that the values of x,y,z remain known only to A,B,C respectively and the value of f is known only to A.

In order to securely calculate f, we need the capability to perform homomorphic multiplication. In this pursuit, we have to leverage appropriate crypto-system primitives. Thus, each party needs to generate its own ElGamal cryptosystem key pair, which would allow the participants to carry out homomorphic multiplication. We can assume that each of the A, B, C participants has a pair of keys (private and public) for homomorphic multiplication. Moreover, we assume that each participant knows the public keys of every other. Then, we can proceed in the following manner:

We use the notation $\rightarrow$ to denote sending a message along with $E_{m,B}(\ldots)$ and $E_{m,C}(\ldots)$ as representing the multiplicative homomorphic encryption with the corresponding public keys of B and C. Conversely, we use $D_{m,B}(\ldots)$ and $D_{m,C}(\ldots)$ as representing homomorphic decryption with the corresponding private keys of B and C respectively. To better explain the proposed approach, we present the key idea of the protocol in simplified form. We discuss thereafter the weaknesses in the simple protocol, and subsequently we propose two other protocol variants that are improving the simplified variant.

We start with a simplified protocol variant to illustrate the key idea. Given that A needs to calculate its function f (where f can be any non-trivial polynomial function), we assign the role of $MultiplicationAssistant$ to B. The latter is responsible for running the homomorphic multiplicative algorithm to assist A to securely calculate the polynom. To this end, B will be the only party with access to the homomorphic multiplication private encryption key ($e_{m,B}$). However, for the homomorphic multiplication algorithm, the public keys are known to all parties.

Considering $D_{m,B}(\ldots)$ as denoting the multiplicative homomorphic decryption with the corresponding private key, we can proceed as follows:

1. $B \rightarrow A : E_{m,B}(y)$

2. $C \rightarrow A : E_{m,B}(z)$

3. $A : E_{m,B}(x^2y) = E_{m,B}(x) * E_{m,B}(x) * E_{m,B}(y)$

4. $A : E_{m,B}(xz) = E_{m,B}(x) * E_{m,B}(z)$

5. $A \rightarrow B : E_{m,B}(xz), E_{m,B}(x^2y)$

6. $B : D_{m,B}(E_{m,B}(xz)) = valueOf(xz)$

7. $B : D_{m,B}(E_{m,B}(x^2y)) = valueOf(x^2y)$

8. $B \rightarrow A : valueOf((xz) + (x^2y))$

Although the foregoing steps allow *A* to compute its private function using private variables owned by other entities, there is an issue concerning the information that *B* may be able to infer about *A*'s function and the values of the variables used. The purpose of requiring an assistant to help the function owner with the calculations, is to prevent *A* from being able to decrypt the values of the used private variables. In this context, the assistant *B* will receive multiple values from *A*, which it has to decrypt, then sum, and return back to *A*. The fact that the function is private to *A* and unknown to *B* prevents the latter from knowing what the values correspond to. However, *B* would be able to see the final result of the function. To solve this issue, we require *A* to multiply each term with a random value *r* to prevent *B* from seeing the final result or the values of the function terms:

1. $B \rightarrow A : E_{m,B}(y)$

2. $C \rightarrow A : E_{m,B}(z)$

3. A : chooses a large random value r

4. $A : E_{m,B}(x^2yr) = E_{m,B}(x) * E_{m,B}(x) * E_{m,B}(y) * E_{m,B}(r)$

5. $A : E_{m,B}(xzr) = E_{m,B}(x) * E_{m,B}(z) * E_{m,B}(r)$

6. $A \rightarrow B : E_{m,B}(xzr), E_{m,B}(x^2yr)$

7. $B : D_{m,B}(E_{m,B}(xzr)) = valueOf(xzr)$

8. $B : D_{m,B}(E_{m,B}(x^2yr)) = valueOf(x^2yr)$

9. $B \rightarrow A : valueOf((xzr) + (x^2yr))$

10. $A : valueOf((xz) + (x^2y)) = valueOf((xzr) + (x^2yr))/r$

The aforementioned steps solve the problem of sending the values of the function terms and potentially the final result of *A*'s private function to the assistant *B*. Since we assumed that all entities in the system are honest but curious, then *B* should not be able to calculate the values of the terms nor the final result. However, because *A* multiplies all the terms by the same random value *r*, it is still possible for *B* to infer the terms and even the final result. This is a result of *A* multiplying all the terms by the same random value *r*, which means that all the terms are also divisible by *r*. The assistant *B* can infer the terms by listing all the common divisors for the terms sent by the function owner *A*, and one of those values would be the correct random value. This gives *B* the ability to generate a list of possible results, which definitely contains one value as the correct result. Moreover, as the number of terms increases, the possibility of the random value *r* being the greatest common divisor (gcd) of terms also increases. To solve this issue, we introduce another modification to the protocol in order to prevent *B* from inferring the terms or the final result. If *A* could multiply each term by a different random value before sending the terms to *B*, then the terms would not necessarily have the random value as a common divisor. However, the function owner would not be able to remove the random values from the final result if each term was multiplied by a different random value. To address this, we add another assistant to help *A* in calculating the private function. In the final variant of our protocol, the function owner chooses a large random r and splits it into t dissimilar shares (the sum of the shares equals r) and inserts the values into an array $rand_1$ of size t, where t is the number of terms. It then creates a second array $rand_2$ of the same size as $rand_1$, where the value at each position in $rand_2$ represents the difference between the random value r and the value in the same position in the first array $rand_1$. This scheme allows A to multiply each term by a different random, while being able to calculate the final result by adding the results sent by the two assistants and then dividing by r to get the actual final result.

1. $B \rightarrow A : E_{m,B}(y), E_{m,C}(y)$
2. $C \rightarrow A : E_{m,B}(z), E_{m,C}(z)$
3. A : chooses a large random value r
4. A : creates two arrays $rand_1[\,]$ and $rand_2[\,]$ of size t, where t is the number of terms in the function.
5. A : splits r into t shares, and adds the values to $rand_1[\,]$
6. A : for($i = 0$; $i < t$; $i = i + 1$) $rand_2[i]$ = r - $rand_1[i]$
7. $A : E_{m,B}(x^2y * rand_1[0]) = E_{m,B}(x) * E_{m,B}(x) * E_{m,B}(y) * E_{m,B}(rand_1[0])$
8. $A : E_{m,B}(xz * rand_1[1]) = E_{m,B}(x) * E_{m,B}(z) * E_{m,B}(rand_1[1])$
9. $A \rightarrow B : E_{m,B}(xz * rand_1[0]), E_{m,B}(x^2y * rand_1[1])$
10. $B : D_{m,B}(E_{m,B}(xz * rand_1[0])) = valueOf(xz * rand_1[0])$
11. $B : D_{m,B}(E_{m,B}(x^2y * rand_1[1])) = valueOf(x^2y * rand_1[1])$
12. $B \rightarrow A : valueOf((xz * rand_1[0]) + (x^2y * rand_1[1]))$

13. $A : E_{m,C}(x^2y * rand_2[0]) = E_{m,C}(x) * E_{m,C}(x) * E_{m,C}(z) * E_{m,B}(rand_2[0])$

14. $A : E_{m,C}(xz * rand_2[1]) = E_{m,C}(x) * E_{m,C}(z) * E_{m,C}(rand_2[1])$

15. $A \rightarrow C : E_{m,C}(xz * rand_2[0]), E_{m,C}(x^2y * rand_2[1])$

16. $C : D_{m,C}(E_{m,C}(xz * rand_2[0])) = valueOf(xz * rand_2[0])$

17. $C : D_{m,C}(E_{m,C}(x^2y * rand_2[1])) = valueOf(x^2y * rand_2[1])$

18. $C \rightarrow A : valueOf((xz * rand_2[0]) + (x^2y * rand_2[1]))$

19. $A : valueOf((xz * r) + (x^2y * r)) = valueOf((xz * rand_1[0]) + (x^2y * rand_1[1])) + valueOf((xz * rand_2[0]) + (x^2y * rand_2[1]))$

20. $A : valueOf((xz) + (x^2y)) = valueOf((xz * r) + (x^2y * r))/r$

6.4 Non-Encryption Solution

As previously discussed, we utilize homomorphic encryption in order to achieve secure multi-party computation. However, such encryption schemes are known to be computationally demanding. In this section, we discuss a lightweight non-encryption solution to achieve operations, such as, summation, average, and multiplication.

6.4.1 Protocol for Secure Summation

For privacy-preserving multi-party summation, let us assume we have N participants $(D_1, D_2, \ldots, D_N)$, each with a secret value v_i where $i = 1, \ldots, N$, collaborate to securely calculate the summation of v_i. The protocol is composed of two stages: off-line key setup and on-line secure summation.

Stage 1: Key Setup

1. Each D_i ($i \in [1,N]$) randomly selects an integer value $r_{i,j}$ $((j \in [1,N]) \wedge (i \neq j))$ and sends it to j, and sets $r_{i,i} = 0$ (as shown in Figure 6.1).

2. The key k_i for D_i is calculated as follows:

$$k_i = \sum_{k=1}^{N} (r_{k,i}) - \sum_{j=1}^{N} (r_{i,j})$$

	D_1	D_2	…	D_i	…	D_{N-1}	D_N
D_1		$r_{1,2}$	…	$r_{1,i}$	…	$r_{1,N-1}$	$r_{1,N}$
D_2	$r_{2,1}$		…	$r_{2,i}$	…	$r_{2,N-1}$	$r_{2,N}$
…	…	…		…	…	…	…
D_i	$r_{i,1}$	$r_{i,2}$	…		…	$r_{i,N-1}$	$r_{i,N}$
…	…	…	…	…		…	…
D_{N-1}	$r_{N-1,1}$	$r_{N-1,2}$	…	$r_{N-1,i}$	…		$r_{N-1,N}$
D_N	$r_{N,1}$	$r_{N,2}$	…	$r_{N,i}$	…	$r_{N,N-1}$	

Figure 6.1: Key Setup

Stage 2: Secure Summation The naive way is, for each participant, to broadcast the value $(v_i + k_i)$ to the other participants, and then each participant can independently calculate the summation.

1. Each D_i ($i \in [1,N]$) sends $v'_i = v_i + k_i$ to other participants.

2. Each participant then calculates the summation:

$$sum = \sum_{i=1}^{N} (v'_i)$$

This aforementioned procedure requires high communication overhead. To achieve better efficiency, an alternative way is as follows: Each participant sends the intermediate value to one of its neighbors. The last participant sums up the final result and broadcasts it to all the others.

1. D_1 sends $sum_1 = v_1 + k_1$ to D_2.
2. For $i = 2$ to $N-1$: D_i sends $sum_i = sum_{i-1} + v_i + k_i$ to D_{i+1}.
3. D_N sends the result (sum_N) to other participants.

Note that we can simply utilize certain additional data structure to further reduce the whole communication and computation overhead. For example, we can organize the participants in a binary-tree architecture.

6.4.2 Analysis

Correctness In this section, we analyze the protocol to ensure that the computations are properly calculated. It is straightforward to note that:

$$\sum_{i=1}^{N}(k_i) = 0$$

since:

$$\sum_{i=1}^{N}(k_i) = \sum_{i=1}^{N}(\sum_{k=1}^{N}(r_{k,i}) - \sum_{j=1}^{N}(r_{i,j}))$$

$$= \sum_{i=1}^{N}\sum_{k=1}^{N}(r_{k,i}) - \sum_{i=1}^{N}\sum_{j=1}^{N}(r_{i,j}) = 0$$

Informally, each r_{ij} appears exactly twice in two keys which cancel out each other. Therefore,

$$\sum_{i=1}^{N}(v_i') = \sum_{i=1}^{N}(v_i + k_i)$$

$$\sum_{i=1}^{N}(v_i + k_i) = \sum_{i=1}^{N}(v_i) + \sum_{i=1}^{N}(k_i) = \sum_{i=1}^{N}(v_i)$$

Complexity In this section, we analyze the complexity of the protocol. For the key setup phase, the complexity of the communication for each participant is $O(N)$ ($N-1$ for receiving, $N-1$ for sending). The complexity of the computations in the key setup phase is $O(N)$ for each participant ($2 \times N$ summation operations).

For the secure summation in the naive method, the complexity of the communications for each participant is $O(N)$ ($N-1$ for receiving value from others, $N-1$ for sending). The complexity of the computations in the naive method are $O(N)$ ($N+1$ summation operations). The complexity in the secure summation in the alternative method is $O(1)$ (1 for receiving value from D_N, 1 for sending to the next) for all participants other than D_N. However, the complexity of the communications for D_N is $O(N)$ (1 for receiving value D_N, $N-1$ for sending to others). Finally, the complexity of the computations in the alternative method is $O(1)$ (2 summation operations) for all participants other than D_1. The computations complexity for D_1 is $O(1)$ (1 summation operation).

Note that from a practical implementation stand point, $O(N)$ for summation operation is notably different than $O(N)$ for homomorphic encryption or decryption since in the latter case each operation is significantly more costly.

Security Each participant only knows those random values sent to him and those he sends. This means that participant D_i only knows those values in the cells filled in gray color as shown in Figure 6.1. Since the random values are drawn randomly without bias, no participant can infer any of the others' keys unless all the N participants collude.

6.4.3 Enhanced Key Setup

Note that, in the aforementioned key setup process, the key for each participant is determined once. Given that k_i, v_i is one-to-one mapped to v_i', this scheme is not semantically secure. One possible solution to achieve semantic security is as follows. Instead of determining the key once off-line, each time for a summation operation, each participant reselects the random values for each participant by repeating the key setup stage. As analyzed above, the communication and computation overhead will be $O(N)$ for each participant, which may be an challenging for near real-time application.

Actually, in many real applications, it is acceptable that less than K participants cannot collude to disclose a participant's secure value. To reduce the overhead, we can slightly reduce the resistance degree of collusion attack by reducing all the N participants to a given parameter K (which means that at least K participants should collude to disclose others' value). In such case, each participant only sends K random values r_{ij} to the other K participants. For example, D_i only sends to $D_{(i+1)\ mod\ N}, \dots, D_{(i+K)\ mod\ N}$. Now the complexities are $O(K)$.

6.4.4 Protocol for Secure Multiplication

The protocol for secure summation can be slightly revised in order to achieve secure multiplication. In the remainder of this section, we briefly introduce the required changes. The analysis and key setup enhancement are similar with secure summation, and are omitted.

Stage 1: Key Setup In this stage, each D_i ($i \in [1,N]$) randomly selects one positive integer value $r_{i,j}$ ($(j \in [1,N]) \wedge (i \neq j)$) and sends it to j, and then sets $r_{i,i} = 1$. The key k_i for D_i is calculated as follows:

$$k_i = \frac{\prod_{k=1}^{N}(r_k, i)}{\prod_{j=1}^{N}(r_{i,j})}$$

Stage 2: Secure Multiplication In this stage, to securely calculate the product, each D_i ($i \in [1,N]$) sends $v_i' = v_i \times k_i$ to other participants. Each participant then calculates the multiplication in the follow way:

$$prod = \prod_{i=1}^{N} (v_i')$$

It must be noted here that, fractions may lead to an accuracy errors, which can be controlled by decimal digits.

Example 6. *Secure Summation*

	D_1	D_2	D_3
D_1	*0*	*5*	*-3*
D_2	*7*	*0*	*10*
D_3	*8*	*-9*	*0*

Table 6.1: The Values of r_{ij}

As shown in Table 6.1, D_1 sends $r_{1,2} = 5$ to D_2. Based on the r_{ij} values as shown in Table 6.1, the key for each participant is shown as follows:

$$D_1 : k_1 = r_{2,1} + r_{3,1} - r_{1,2} - r_{1,3} = 7 + 8 - 5 - (-3) = 13$$

$$D_2 : k_2 = r_{1,2} + r_{3,2} - r_{2,1} - r_{2,3} = 5 + (-9) - 7 - 10 = -21$$

$$D_3 : k_3 = r_{1,3} + r_{2,3} - r_{3,1} - r_{3,2} = (-3) + 10 - 8 - (-9) = 8$$

Suppose that the value for participant D_1, D_2 and D_3 is $v_1 = 10$, $v_2 = 30$, and $v_3 = 15$, respectively. Then, the summation is $sum = v_1 + v_2 + v_3 = 55$.

D_1, D_2 and D_3 broadcasts $v_1' = v_1 + k_1 = 10 + 13 = 23$, $v_2' = v_2 + k_2 = 30 - 21 = 9$, and $v_3' = v_3 + k_3 = 15 + 8 = 23$, respectively.

Each participant will calculate the summation as:

$$sum = v_1' + v_2' + v_3' = 23 + 9 + 23 = 55.$$

Example 7. *Case Study: Secure Multiplication*

	D_1	D_2	D_3
D_1	*1*	*5*	*3*
D_2	*7*	*1*	*10*
D_3	*8*	*9*	*1*

Table 6.2: The Values of r_{ij}

As shown in Table 6.2, D_1 sends $r_{1,2} = 5$ to D_2. Based on the r_{ij} values as shown in Table 6.2, the key for each participant is shown as follows.

$$\begin{aligned} D_1 : k_1 = \frac{r_{2,1} \times r_{3,1}}{r_{1,2} \times r_{1,3}} &= \frac{7\times8}{5\times3} &= 3.733333 \\ D_2 : k_2 = \frac{r_{1,2} \times r_{3,2}}{r_{2,1} \times r_{2,3}} &= \frac{5\times9}{7\times10} &= 0.642857 \\ D_3 : k_3 = \frac{r_{1,3} \times r_{2,3}}{r_{3,1} \times r_{3,2}} &= \frac{3\times10}{8\times9} &= 0.416667 \end{aligned}$$

Suppose that the value for participants D_1, D_2 and D_3 is $v_1 = 10$, $v_2 = 30$, and $v_3 = 15$, respectively. Then, the product is $prod = v_1 \times v_2 \times v_3 = 10 \times 30 \times 15 = 4500$.

D_1, D_2 and D_3 broadcasts $v'_1 = v_1 \times k_1 = 37.33333$, $v'_2 = v_2 \times k_2 = 19.28571$, and $v'_3 = v_3 \times k_3 = 6.250005$, respectively.

Each participant will calculate the product as $prod = v'_1 \times v'_2 \times v'_3 = 37.33333 \times 19.28571 \times 6.250005 = 4500.0022$. Note that the accuracy can be increased by increasing decimal digits of the keys.

6.5 Implementation

In this section, we will show some preliminary results as a proof of concept for our proposed encryption-based solution. The implementation code is written in Java and run on an intel Core i7 with 8GB RAM. The homomorphic implementation is based on an open-source Java implementation of the ElGamal cryptosystem by the Computing and Software Systems

Department at the University of Washington [1].

To test the practicality and scalability of our proposed solution, we test the solution assuming that 3 entities participate in the protocol. We then run multiple iterations of the protocol increasing the number of participants by 100 at each stage. We also tested each stage using four different key sizes for the ElGamal Homomorphic cyrptosystem. Within our solution, we are concerned with three main operations, encryption, homomorphic multiplication, and decryption.

6.5.1 Encryption

We test the time needed to encrypt thousands of random integers under four key sizes (128 bits, 256 bits, 512 bits, and 1024 bits). Since encryption in our protocol is executed by the participants in parallel, we are only concerned with the time it takes to encrypt a single value using each of the above keys. As such, the encryption time is really insignificant.

Key size	128 bits	256 bits	512 bits	1024 bits
Avg. encryption time	0.04 ms	0.088 ms	0.109 ms	0.859 ms

6.5.2 Homomorphic Multiplication

To get accurate results, we run multiple iterations of our programs, testing the time needed to homomorphiclly multiply two ElGamal encrypted values with each other, and then we increase the number of values by 100 for each iteration to test the scalability of homomorphic algorithm. In our approach, the homomorphic multiplications will be executed by the function owner. The results of our testing in Figure 6.2, show the scalability of our approach. They also show that that the average time per multiplication operation is significantly lower than the time needed to encrypt or decrypt a value using the ElGamal cryptosystem with

[1] http://faculty.washington.edu/moishe/javademos/Security/ElGamal.java

an average time of 0.00972 ms per multiplication operation on two values encrypted with a 1024 bit encryption key.

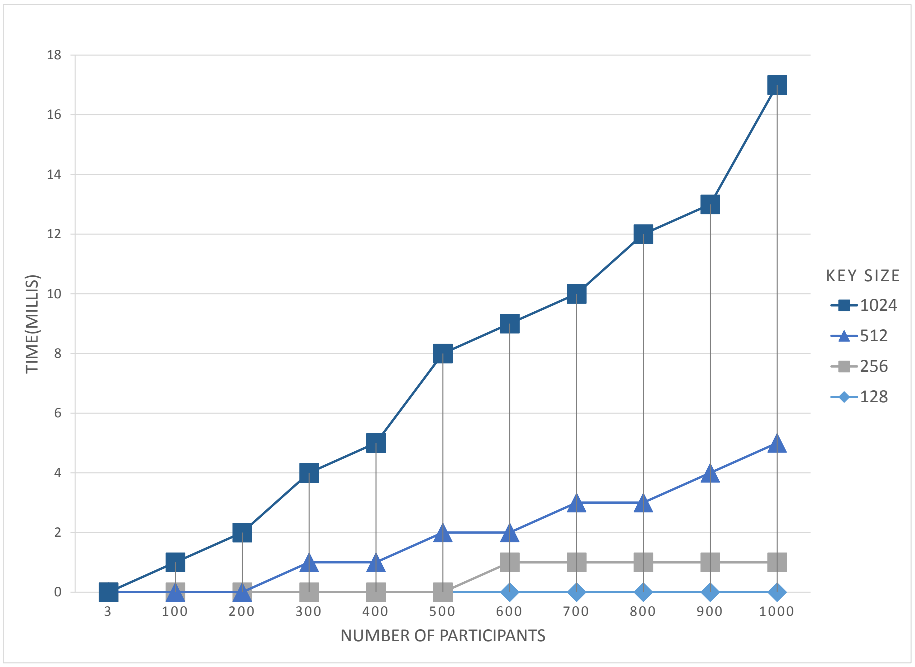


Figure 6.2: Homomorphic Multiplication

6.5.3 Decryption

We test the time needed to decrypt thousands of random integers under four key sizes (128 bits, 256 bits, 512 bits, and 1024 bits). The decryption operations will be executed by the assistants in the protocol. The number of values each assistant needs to decrypt is equal to the number of terms in the function owner's polynomial function. The following table shows the average time needed to decrypt each term encrypted with the following key sizes (128 bits, 256 bits, 512 bits, and 1024 bits):

Key size	128 bits	256 bits	512 bits	1024 bits
Avg. decryption time	0.0168 ms	0.041 ms	0.108 ms	0.365

6.6 Conclusion

In this chapter, we presented a protocol that leverages homomorphic encryption primitives in order to securely compute a polynomial function with private variables provided by at least two other parties while preserving the privacy of the function itself and the variables. Since homomorphic encryption is known to need heavy computation, we also presented a more scalable non-encryption based solution. In this respect, illustrative examples have been presented. The proposed protocol is suitable for collaborative environments where the participants are deemed semi-honest.

Chapter 7

Conclusion and Future Work

7.1 Summary and Conclusion

Cloud storage is becoming increasingly popular due to its high computational power, storage capabilities, convenience, availability, cost, and dynamism. Security concerns and lack of trust have many users and service providers weary of adapting such solutions. Cryptography can address confidentiality by allowing users to privately encrypting their data in the cloud. However, this approach leads to the problem of querying privately encrypted data. Solving this problem while preserving the privacy of the users and the confidentiality of their data, would enable cloud providers to provide secure services such as analyzing sensitive medical or financial data.

In Chapter 3, we introduced a protocol to allow a querying entity access to a subset of a user's privately encrypted data in the cloud. This would aid emergency responders to get critical medical information about a patient in the case of an emergency. The proposed protocol prevents emergency responders from abusing their privileges. The protocol relies on attribute based encryption and symmetric key threshold encryption to solve the problem without requiring the participation of the patient in the process. We also evaluated the performance of the protocol and presented the results of our experimental results.

To allow a querying entity to execute more complex queries, we introduced in Chapter 4 a privacy-preserving querying protocol on privately encrypted data in the cloud. The proposed protocol would allow a querying entity such as a health organization to query privately encrypted data such as private health records. The protocol allows for sum, average, or comparison queries. The protocol executes queries on Kd-trees that are constructed from encrypted health records. It also prevents patients from inferring what health organizations are concerned about. We experimentally evaluated the performance of the protocol and reported on the results of implementation.

In Chapter 5, we introduced enhancements to the protocol in Chapter 4. The enhanced protocol removed the preprocessing steps needed to gather the encrypted data in a central location for each group and securely sorting the KD-Tree. The protocol also reduced the computational overhead on the cloud server and the data owners. The size of the queries was also reduced. Two variations of the protocol were introduced, the first hides the query attributes to prevent the data owners from knowing what the querying entity is interested in. The second variation prevents the querying entity from being able to query for personally identifiable information. We also evaluated the performance of the protocol and reported on the results of the implementation. We also filed a patent on a mechanism based on a variant of this protocol designed for an LTE mobile network environment.

In Chapter 6, we introduced a cryptographic and a non-cryptographic privacy-preserving protocol to allow multiple parties to jointly compute a private polynomial function. The protocol allowed a user to request the assistance of at lease two other participants in computing a polynomial function using their privately encrypted data, without exposing the values of the participating parties or the function. Additionally, We experimentally evaluated the performance of the protocol and reported on the results of our implementation.

In this thesis, we proposed several protocols for secure and privacy-preserving data storage in the cloud. We further show their applicability and scalability through their implementations. The proposed protocols provide an efficient way forward to guarantee user data confidentiality in the cloud, while preserving their privacy and providing them with a range of computational features.

7.2 Future Work

Much progress remains to be made in addressing secure cloud computations on privately encrypted data. In what follows, we suggest some research directions to improve on the protocols we proposed in this thesis.

- **Reduce Computation Load on Data Owners:** To increase the practicality of secure cloud computations on privately encrypted data, the load on the DOs should be reduced. This can be done by outsourcing a portion of the cryptographic operations to the cloud.
- **Provable Function Usage:** To enhance the private function evaluation protocol we presented in Chapter 6, we suggest proposing a method to prevent the function owner from crafting a function that aims to expose a participant's private values. This method should prove to the participants that the function owner is being fair, and the function is not designed to focus on a single participant's private values.

Bibliography

[1] Ivetta Abramyan. *Atmospheric Patterns and Asian Typhoon Landfalls*. PhD thesis, University of South Carolina, 2013.

[2] MIT Tech Review. Who Coined 'Cloud Computing'?, October 2011. URL `https://www.technologyreview.com/s/425970/who-coined-cloud-computing`. Accessed March, 2017.

[3] Eric Schmidt. Conversation With Eric Schmidt Hosted by Danny Sullivan, August 2006. URL `https://www.google.com/press/podium/ses2006.html`. Accessed March, 2017.

[4] Peter Mell and Tim Grance. The NIST definition of cloud computing. 2011. URL `https://www.nist.gov/news-events/news/2011/10/final-version-nist-cloud-computing-definition-published`. Accessed March, 2017.

[5] Zhifeng Xiao and Yang Xiao. Security and Privacy in Cloud Computing. *IEEE Communications Surveys & Tutorials*, 15(2):843–859, 2013.

[6] Li Ming, Yu Shucheng, Ren Kui, and Lou Wenjing. Securing Personal Health Records in Cloud Computing: Patient-Centric and Fine-Grained Data Access Control in Multi-owner Settings. In Sushil Jajodia and Jianying Zhou, editors, *Security*

and Privacy in Communication Networks, volume 50 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 89–106. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-16160-5.

[7] Sh. Narayan, M. Gagné, and R. Safavi-Naini. Privacy Preserving EHR System Using Attribute-Based Infrastructure. In *Proceedings of the Cloud computing security workshop*, CCSW '10, pages 47–52, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0089-6. doi: http://doi.acm.org/10.1145/1866835.1866845.

[8] Matthew Pirretti, Patrick Traynor, Patrick McDaniel, and Brent Waters. Secure Attribute-Based Systems. *Journal of Computer Security*, 18(5):799–837, 2010.

[9] Bruce Schneier, Kathleen Seidel, and Saranya Vijayakumar. A Worldwide Survey of Encryption Products. 2016. URL `https://www.schneier.com/academic/paperfiles/worldwide-survey-of-encryption-products.pdf`. Accessed March, 2017.

[10] Jose Pagliery. Hospital Network Hacked, 4.5 Million Records Stolen. *CNN Money, August*, 2014. URL `http://money.cnn.com/2014/08/18/technology/security/hospital-chs-hack/`. Accessed March, 2017.

[11] Anthem. FAQs on the cyber attack against Anthem. 2015. URL `https://www.anthemfacts.com/faq`. Accessed March, 2017.

[12] ABC News Online. Red Cross Blood SerVice Admits to Personal Data Breach Affecting Half a Million Donors. 2016. URL `http://www.abc.net.au/news/2016-10-28/red-cross-blood-service-admits-to-data-breach/7974036`. Accessed March, 2017.

[13] Jose Pagliery David Goldman and Laurie Segall. CNN Tech: How Celebrities' Nude Photos Get Leaked, September 2014. URL http://money.cnn.com/2014/09/01/technology/celebrity-nude-photos/index.html?iid=EL. Accessed March, 2017.

[14] Hackappcom. Github: AppleID Bruteforce Project. 2013. URL https://github.com/hackappcom/ibrute. Accessed March, 2017.

[15] Intralinks. Your Sensitive Information Could Be at Risk: File Sync and Share Security Issue, May 2014. URL https://blogs.intralinks.com/2014/05/sensitive-information-risk-file-sync-share-security-issue/. Accessed March, 2017.

[16] Imperva: Hacker Intelligence Initiative. Man in the Cloud (MITC) Attacks, 2015. URL https://www.imperva.com/docs/HII_Man_In_The_Cloud_Attacks.pdf. Accessed March, 2017.

[17] Microsoft. Microsoft HealthVault. URL http://www.healthvault.com/. Accessed March, 2017.

[18] Dossia Personal Health Platforfm. URL http://www.dossia.org/. Accessed March, 2017.

[19] Quicken. URL https://www.quicken.com/. Accessed March, 2017.

[20] Intuit. Mint, . URL https://www.mint.com/. Accessed March, 2017.

[21] Intuit. Turbotax, . URL https://turbotax.intuit.ca/tax-software/i. Accessed March, 2017.

[22] H&R Block Tax Software, accessed in 2016. URL https://www.hrblock.ca/.

[23] Craig Gentry and Shai Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 129–148. Springer, 2011.

[24] 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*, 1982.

[25] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology, EUROCRYPT 99*, volume 1592 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1999.

[26] Feras Aljumah, Raymond Hei Man Leung, Makan Pourzandi, and Mourad Debbabi. Emergency Mobile Access to Personal Health Records Stored on an Untrusted Cloud. In *International Conference on Health Information Science*, pages 30–41. Springer, 2013.

[27] Samira Barouti, Feras Aljumah, Dima Alhadidi, and Mourad Debbabi. Secure and Privacy-Preserving Querying of Personal Health Records in the Cloud. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 82–97. Springer, 2014.

[28] Feras Aljumah, Andrei Soeanu, Wen Ming Liu, and Mourad Debbabi. Protocols for Secure Multi-Party Private Function Evaluation. In *Anti-Cybercrime (ICACC), 2015 First International Conference on*, pages 1–6. IEEE, 2015.

[29] Makan Pourzandi Feras Aljumah and Mourad Debbabi. Privacy-Preserving Querying Mechanism on Privately Encrypted Personal Health Records. 2017.

[30] F. ALJUMAH, M. Pourzandi, and M. Debbabi. Privacy-Preserving Querying Mechanism on Privately Encrypted Data on Semi-Trusted Cloud, December 30 2015. URL `https://www.google.ca/patents/WO2015198098A1?cl=en`. WO Patent App. PCT/IB2014/062,636.

[31] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable Data Possession at Untrusted Stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 598–609. Acm, 2007.

[32] Giuseppe Ateniese, Roberto Di Pietro, Luigi V Mancini, and Gene Tsudik. Scalable and Efficient Provable Data Possession. In *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, page 9. ACM, 2008.

[33] Arati Baliga, Pandurang Kamat, and Liviu Iftode. Lurking in the Shadows: Identifying Systemic Threats to Kernel Data. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 246–251. IEEE, 2007.

[34] Cong Wang, Kui Ren, and Jia Wang. Secure and Practical Outsourcing of Linear Programming in Cloud Computing. In *INFOCOM, 2011 Proceedings IEEE*, pages 820–828. IEEE, 2011.

[35] Ari Juels and Burton S Kaliski Jr. PORs: Proofs of Retrievability for Large Files. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 584–597. ACM, 2007.

[36] C Chris Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic Provable Data Possession. *ACM Transactions on Information and System Security (TISSEC)*, 17(4):15, 2015.

[37] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. The Rise of "Big Data" on Cloud Computing: Review and Open Research Issues. *Information Systems*, 47:98–115, 2015.

[38] Kui Ren, Cong Wang, and Qian Wang. Security Challenges for the Public Cloud. *IEEE Internet Computing*, 16(1):69, 2012.

[39] Subashini Subashini and Veeraruna Kavitha. A Survey on Security Issues in Service Delivery Models of Cloud Computing. *Journal of network and computer applications*, 34(1):1–11, 2011.

[40] Dimitrios Zissis and Dimitrios Lekkas. Addressing Cloud Computing Security Issues. *Future Generation computer systems*, 28(3):583–592, 2012.

[41] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 47–53. Springer, 1984.

[42] Amit Sahai and Brent Waters. Fuzzy Identity-Based Encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 457–473. Springer, 2005.

[43] Matthew Pirretti, Patrick Traynor, Patrick McDaniel, and Brent Waters. Secure attribute-based systems. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 99–112, New York, NY, USA, 2006. ACM. ISBN 1-59593-518-5. doi: http://doi.acm.org/10.1145/1180405.1180419. URL `http://doi.acm.org/10.1145/1180405.1180419`.

[44] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-Policy Attribute-Based Encryption. In *Proceedings of the 2007 IEEE Symposium on Security and*

Privacy, SP '07, pages 321–334, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2848-1. doi: http://dx.doi.org/10.1109/SP.2007.11. URL `http://dx.doi.org/10.1109/SP.2007.11`.

[45] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 89–98, New York, NY, USA, 2006. ACM. ISBN 1-59593-518-5. doi: http://doi.acm.org/10.1145/1180405.1180418.

[46] Melissa Chase and Sherman SM Chow. Improving Privacy and Security in Multi-Authority Attribute-Based Encryption. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 121–130. ACM, 2009.

[47] Vladimir Božović, Daniel Socek, Rainer Steinwandt, and Viktória I Villányi. Multi-Authority Attribute-Based Encryption With Honest-but-Curious Central Authority. *International Journal of Computer Mathematics*, 89(3):268–283, 2012.

[48] Seny Kamara and Kristin Lauter. Cryptographic Cloud Storage. In *International Conference on Financial Cryptography and Data Security*, pages 136–149. Springer, 2010.

[49] Jinguang Han, Willy Susilo, Yi Mu, and Jun Yan. Privacy-Preserving Decentralized Key-Policy Attribute-Based Encryption. *IEEE Transactions on Parallel and Distributed Systems*, 23(11):2150–2162, 2012.

[50] Allison Lewko and Brent Waters. Decentralizing Attribute-Based Encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 568–588. Springer, 2011.

[51] Huang Lin, Zhenfu Cao, Xiaohui Liang, and Jun Shao. Secure Threshold Multi Authority Attribute Based Encryption Without a Central Authority. In *International Conference on Cryptology in India*, pages 426–436. Springer, 2008.

[52] Jin Li, Qiong Huang, Xiaofeng Chen, Sherman SM Chow, Duncan S Wong, and Dongqing Xie. Multi-Authority Ciphertext-Policy Attribute-Based Encryption With Accountability. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 386–390. ACM, 2011.

[53] Jin Li, Kui Ren, and Kwangjo Kim. A2BE: Accountable Attribute-Based Encryption for Abuse Free Access Control. *IACR Cryptology ePrint Archive*, 2009:118, 2009.

[54] Shucheng Yu, Kui Ren, Wenjing Lou, and Jin Li. Defending Against Key Abuse Attacks in KP-ABE Enabled Broadcast Systems. In *International Conference on Security and Privacy in Communication Systems*, pages 311–329. Springer, 2009.

[55] YongTao Wang, KeFei Chen, Yu Long, and ZhaoHui Liu. Accountable Authority Key Policy Attribute-Based Encryption. *Science China Information Sciences*, 55(7): 1631–1638, 2012.

[56] Shanqing Guo, Yingpei Zeng, Juan Wei, and Qiuliang Xu. Attribute-Based Re-Encryption Scheme in the Standard Model. *Wuhan University Journal of Natural Sciences*, 13(5):621–625, 2008.

[57] Kaitai Liang, Liming Fang, Willy Susilo, and Duncan S Wong. A Ciphertext-Policy Attribute-Based Proxy Re-Encryption With Chosen-Ciphertext Security. In *Intelligent Networking and Collaborative Systems (INCoS), 2013 5th International Conference on*, pages 552–559. IEEE, 2013.

[58] Xiaohui Liang, Zhenfu Cao, Huang Lin, and Jun Shao. Attribute Based Proxy Re-Encryption With Delegating Capabilities. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, pages 276–286. ACM, 2009.

[59] Song Luo, Jianbin Hu, and Zhong Chen. Ciphertext Policy Attribute-Based Proxy Re-Encryption. In *International Conference on Information and Communications Security*, pages 401–415. Springer, 2010.

[60] Hwa-Jeong Seo and Ho-Won Kim. Attribute-Based Proxy Re-Encryption With a Constant Number of Pairing Operations. *Journal of information and communication convergence engineering*, 10(1):53–60, 2012.

[61] Nuttapong Attrapadung and Hideki Imai. Conjunctive Broadcast and Attribute-Based Encryption. In *International Conference on Pairing-Based Cryptography*, pages 248–265. Springer, 2009.

[62] Xiaohui Liang, Rongxing Lu, Xiaodong Lin, and Xuemin Sherman Shen. Ciphertext Policy Attribute Based Encryption With Efficient Revocation. Technical report, Citeseer, 2010.

[63] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-Based Encryption With Non-Monotonic Access Structures. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 195–203. ACM, 2007.

[64] Yinghui Zhang, Xiaofeng Chen, Jin Li, Hui Li, and Fenghua Li. FDR-ABE: Attribute-Based Encryption With Flexible and Direct Revocation. In *Intelligent Networking and Collaborative Systems (INCoS), 2013 5th International Conference on*, pages 38–45. IEEE, 2013.

[65] Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. Identity-Based Encryption With Efficient Revocation. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 417–426. ACM, 2008.

[66] Junbeom Hur and Dong Kun Noh. Attribute-Based Access Control With Efficient Revocation in Data Outsourcing Systems. *IEEE Transactions on Parallel and Distributed Systems*, 22(7):1214–1221, 2011.

[67] Luan Ibraimi, Milan Petkovic, Svetla Nikova, Pieter Hartel, and Willem Jonker. Mediated Ciphertext-Policy Attribute-Based Encryption and Its Application. In *Information security applications*, pages 309–323. Springer, 2009.

[68] Xingxing Xie, Hua Ma, Jin Li, and Xiaofeng Chen. New Ciphertext-Policy Attribute-Based Access Control With Efficient Revocation. In *Information and Communication Technology-EurAsia Conference*, pages 373–382. Springer, 2013.

[69] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Attribute Based Data Sharing With Attribute Revocation. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 261–270. ACM, 2010.

[70] Nuttapong Attrapadung and Hideki Imai. Attribute-Based Encryption Supporting Direct/indirect Revocation Modes. In *IMA International Conference on Cryptography and Coding*, pages 278–300. Springer, 2009.

[71] Adi Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, November 1979. ISSN 0001-0782. doi: 10.1145/359168.359176. URL `http://doi.acm.org/10.1145/359168.359176`.

[72] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On Data Banks and Privacy Homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[73] Sigrun Goluch. *The Development of Homomorphic Cryptography: From RSA to Gentry's Privacy Homomorphism*. PhD thesis, Vienna University of Technology, 2011.

[74] 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. ISBN 3-540-65889-0.

[75] Ivan Damgård and Mads Jurik. A Generalisation, a Simpli. Cation and Some Applications of Paillier's Probabilistic Public-Key System. In *International Workshop on Public Key Cryptography*, pages 119–136. Springer, 2001.

[76] Josh Daniel Cohen Benaloh. *Verifiable Secret-Ballot Elections*. Yale University. Department of Computer Science, 1987.

[77] Ronald Cramer and Victor Shoup. Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 45–64. Springer, 2002.

[78] Steven D Galbraith. Elliptic Curve Paillier Schemes. *Journal of Cryptology*, 15(2): 129–138, January 2002. ISSN 0933-2790.

[79] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2): 120–126, 1978.

[80] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A Secure and Optimally

Efficient Multi-Authority Election Scheme. *Transactions on Emerging Telecommunications Technologies*, 8(5):481–490, 1997.

[81] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In *Theory of Cryptography Conference*, pages 325–341. Springer, 2005.

[82] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[83] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully Homomorphic Encryption Over the Integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 24–43. Springer, 2010.

[84] Zvika Brakerski and Vinod Vaikuntanathan. Fully Homomorphic Encryption From Ring-LWE and Security for Key Dependent Messages. In *Annual Cryptology Conference*, pages 505–524. Springer, 2011.

[85] Nigel P Smart and Frederik Vercauteren. Fully Homomorphic Encryption With Relatively Small Key and Ciphertext Sizes. In *International Workshop on Public Key Cryptography*, pages 420–443. Springer, 2010.

[86] Guilhem Castagnos. An Efficient Probabilistic Public-Key Cryptosystem Over Quadratic Fields Quotients. *Finite Fields and Their Applications*, 13(3):563–576, 2007.

[87] 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. doi: 10.1109/SFCS.1982.88. URL `http://dx.doi.org/10.1109/SFCS.1982.88`.

[88] M. Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In *Proceedings of the Conference on Topics in Cryptology: The Cryptographer's Track at RSA*, pages 457–472, 2001.

[89] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of Applied Cryptography*. CRC press, 1996.

[90] 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*, pages 192–205. Springer, 2002.

[91] 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. ISBN 978-3-540-20663-7. doi: 10.1007/978-3-540-40974-8_29. URL `http://dx.doi.org/10.1007/978-3-540-40974-8_29`.

[92] Dan Boneh and Matthew Franklin. Efficient Generation of Shared RSA Keys. *J. ACM*, 48(4):702–722, july 2001. ISSN 0004-5411. doi: 10.1145/502090.502094.

[93] Damien Stehlé and Ron Steinfeld. Faster Fully Homomorphic Encryption. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 377–394. Springer, 2010.

[94] Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-506-2. doi: 10.1145/1536414.1536440. URL `http://doi.acm.org/10.1145/1536414.1536440`.

[95] World Health Organization. mHealth: New Horizons for Health Through Mobile Technologies. *Global Observatory for eHealth series*, 3, 2011. URL `http://www.who.int/goe/publications/goe_mhealth_web.pdf`. Accessed March, 2017.

[96] Morbidity and Mortality Weekly Report. Two Cases of Human Plague. *Centers for Disease Control and Prevention*, February 25, 2011. URL `http://www.cdc.gov/mmwr/preview/mmwrhtml/mm6007a4.htm`. Accessed March, 2017.

[97] Jens H Weber-Jahnke James Williams. The Regulation of Personal Health Record Systems in Canada. *Canadian Journal of Law and Technology*, 8, 2010. URL `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1726214`. Accessed March, 2017.

[98] Nicolas P. Terry. Personal Health Records: Directing More Costs and Risks to Consumers. *DREXEL LAW REVIEW*, 2009:216–259, 1.

[99] Robert Steinbrook. Personally Controlled Online Health Data – the Next Big Thing in Medical Care? *The New England Journal of Medicine*, 358:1653–1656, 2008.

[100] J. Choe and S. K. Yoo. Web-Based Secure Access From Multiple Patient Repositories. *International Journal of Medical Informatics*, 77(4):242 – 248, 2008. ISSN 1386-5056. doi: 10.1016/j.ijmedinf.2007.06.001.

[101] S. Haas, S. Wohlgemuth, I. Echizen, N. Sonehara, and G. Muller. Aspects of Privacy for Electronic Health Records. *International Journal of Medical Informatics*, 80(2):e26 – e31, 2011. ISSN 1386-5056. doi: 10.1016/j.ijmedinf.2010.10.001. <ce:title>Special Issue: Security in Health Information Systems</ce:title>.

[102] G. C. Hembroff and S. Muftic. SAMSON: Secure Access for Medical Smart Cards Over Networks. In *World of Wireless Mobile and Multimedia Networks (WoWMoM),*

2010 IEEE International Symposium on a, pages 1 –6, June 2010. doi: 10.1109/WOWMOM.2010.5534982.

[103] Th. Hupperich, H. L*ö* hr, A.-R. Sadeghi, and M. Winandy. Flexible Patient-Controlled Security for Electronic Health Records. In *Proceedings of the SIGHIT International Health Informatics Symposium*, IHI '12, pages 727–732, 2012. doi: 10.1145/2110363.2110448.

[104] M. Jafari, R. Safavi-Naini, Ch. Saunders, and N. P. Sheppard. Using Digital Rights Management for Securing Data in a Medical Research Environment. In *Proceedings of the workshop on Digital rights management*, DRM '10, pages 55–60. ACM, 2010. doi: 10.1145/1866870.1866883.

[105] T. Neubauer and J. Heurix. A Methodology for the Pseudonymization of Medical Data. *International Journal of Medical Informatics*, 80(3):190 – 204, 2011. ISSN 1386-5056. doi: 10.1016/j.ijmedinf.2010.10.016.

[106] F. Ueckert, M. Goerz, M. Ataian, S. Tessmann, and H.-U. Prokosch. Empowerment of Patients and Communication With Health Care Professionals Through an Electronic Health Record. *International Journal of Medical Informatics*, 70(2):99 – 108, 2003. ISSN 1386-5056. doi: 10.1016/S1386-5056(03)00052-2.

[107] Executive Office of the President PresidentÃćâĆňâĎćs Council of Advisors on Science and Technology. Report to the President Realizing the Full Potential of Health Information Technology to Improve Healthcare for Americans: The Path Forward , December 2010. URL `http://www.whitehouse.gov/sites/default/files/microsites/ostp/pcast-health-it-report.pdf`.

[108] Kenneth D Mandl, William W Simons, William CR Crawford, and Jonathan M Abbett. Indivo: A Personally Controlled Health Record for Health Information Exchange and Communication. *BMC Med Inform Decis Mak*, 7, 2007.

[109] M. Li, Sh. Yu, K. Ren, and W. 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*, pages 89–106. 2010.

[110] Joseph A. Akinyele, Christoph U. Lehmann, Matthew D. Green, Matthew W. Pagano, Zachary N. J. Peterson, and Aviel D. Rubin. Self-Protecting Electronic Medical Records Using Attribute-Based Encryption. pages 75–86, 2011.

[111] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical Techniques for Searches on Encrypted Data. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 44–55. IEEE, 2000.

[112] Eu-Jin Goh. Secure indexes for efficient searching on encrypted compressed data. Technical report, Technical Report 2003/216, Cryptology ePrint Archive, 2003, 2003. URL `http://eprint.iacr.org/2003/216`. Accessed March, 2017.

[113] Yan-Cheng Chang and Michael Mitzenmacher. Privacy Preserving Keyword Searches on Remote Encrypted Data. In *International Conference on Applied Cryptography and Network Security*, pages 442–455. Springer, 2005.

[114] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[115] Philippe Golle, Jessica Staddon, and Brent Waters. Secure Conjunctive Keyword

Search Over Encrypted Data. In *International Conference on Applied Cryptography and Network Security*, pages 31–45. Springer, 2004.

[116] Jin Li, Qian Wang, Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou. Fuzzy Keyword Search Over Encrypted Data in Cloud Computing. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–5. IEEE, 2010.

[117] Jianhua Yu. Conjunctive Fuzzy Keyword Search Over Encrypted Data in Cloud Computing. *Indonesian Journal of Electrical Engineering and Computer Science*, 12(3):2104–2109, 2014.

[118] Ryan W Gardner, Sujata Garera, Matthew W Pagano, Matthew Green, and Aviel D Rubin. Securing Medical Records on Smart Phones. In *Proceedings of the first ACM workshop on Security and privacy in medical and home-care systems*, pages 31–40. ACM, 2009.

[119] Cong Wang, Ning Cao, Jin Li, Kui Ren, and Wenjing Lou. Secure Ranked Keyword Search Over Encrypted Cloud Data. In *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, pages 253–262. IEEE, 2010.

[120] Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou. Enabling Secure and Efficient Ranked Keyword Search Over Outsourced Cloud Data. *IEEE Transactions on parallel and distributed systems*, 23(8):1467–1479, 2012.

[121] Ning Cao, Cong Wang, Ming Li, Kui Ren, and Wenjing Lou. Privacy-Preserving Multi-Keyword Ranked Search Over Encrypted Cloud Data. *IEEE Transactions on parallel and distributed systems*, 25(1):222–233, 2014.

[122] Wenhai Sun, Bing Wang, Ning Cao, Ming Li, Wenjing Lou, Y Thomas Hou, and Hui Li. Privacy-Preserving Multi-Keyword Text Search in the Cloud Supporting

Similarity-Based Ranking. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 71–82. ACM, 2013.

[123] Wenhai Sun, Wenjing Lou, Y Thomas Hou, and Hui Li. Privacy-Preserving Keyword Search Over Encrypted Data in Cloud Computing. In *Secure Cloud Computing*, pages 189–212. Springer, 2014.

[124] Thouraya Bouabana-Tebibel and Abdellah Kaci. Parallel Search Over Encrypted Data Under Attribute Based Encryption on the Cloud Computing. *Computers & Security*, 54:77–91, 2015.

[125] Zhihua Xia, Li Chen, Xingming Sun, and Jin Wang. An Efficient and Privacy-Preserving Semantic Multi-Keyword Ranked Search over Encrypted Cloud Data. *Advanced Science and Technology Letters*, 31:284, 2013. doi: 10.1186/s13677-014-0008-2.

[126] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order Preserving Encryption for Numeric Data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563–574. ACM, 2004.

[127] Dan Boneh and Brent Waters. Conjunctive, Subset, and Range Queries on Encrypted Data. In *Theory of Cryptography Conference*, pages 535–554. Springer, 2007.

[128] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions. In *Annual Cryptology Conference*, pages 578–595. Springer, 2011.

[129] Elaine Shi, John Bethencourt, TH Hubert Chan, Dawn Song, and Adrian Perrig. Multi-Dimensional Range Query Over Encrypted Data. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 350–364. IEEE, 2007.

[130] Ming Li, Shucheng Yu, Ning Cao, and Wenjing Lou. Authorized Private Keyword Search Over Encrypted Data in Cloud Computing. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 383–392. IEEE, 2011.

[131] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order Preserving Encryption for Numeric Data. In *Proceedings of the SIGMOD international conference on Management of data*, pages 563–574, 2004.

[132] H. Hacigümüş, B. Iyer, and S. Li, Ch.and Mehrotra. Executing SQL Over Encrypted Data in the Database-Service-Provider Model. In *Proceedings of the SIGMOD international conference on Management of data*, SIGMOD '02, pages 216–227. ACM, 2002.

[133] M. Kantarcıoğlu and Ch. Clifton. Security Issues in Querying Encrypted Data. In *Proceedings of the annual IFIP WG 11.3 working conference on Data and Applications Security*, DBSec'05, pages 325–337, 2005.

[134] J. Li and E. R. Omiecinski. Efficiency and Security Trade-Off in Supporting Range Queries on Encrypted Databases. In *Proceedings of the annual IFIP WG 11.3 working conference on Data and Applications Security*, DBSec'05, pages 69–83. Springer-Verlag, 2005.

[135] E. Shmueli, R. Waisenberg, Y. Elovici, and E. Gudes. Designing Secure Indexes for Encrypted Databases. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 54–68. Springer, 2005.

[136] Zh. Yang, Sh. Zhong, and R. N. Wright. Privacy-Preserving Queries on Encrypted Data. In *Proceedings of the European conference on Research in Computer Security*, pages 479–495. Springer, 2006.

[137] Ryan W. Gardner, Sujata Garera, Matthew W. Pagano, Matthew Green, and Aviel D. Rubin. Securing Medical Records on Smart Phones. In *Proceedings of the first ACM workshop on Security and privacy in medical and home-care systems*, SPIMACS '09, pages 31–40, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-790-5. doi: 10.1145/1655084.1655090.

[138] Dan Boneh, Craig Gentry, and Brent Waters. Collusion Resistant Broadcast Encryption With Short Ciphertexts and Private Keys. In *Proceedings of the 25th annual international conference on Advances in Cryptology*, CRYPTO'05, pages 258–275, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-28114-2, 978-3-540-28114-6.

[139] B. Hore, Sh. Mehrotra, and G. Tsudik. A Privacy-Preserving Index for Range Queries. In *Proceedings of the international conference on Very large data bases - Volume 30*, pages 720–731, 2004.

[140] D. Agrawal, A. El Abbadi, F. Emekçi, and A. Metwally. Database Management as a Service: Challenges and Opportunities. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 1709–1716. IEEE, 2009.

[141] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB : Protecting Confidentiality with Encrypted Query Processing. *In Proceedings of the ACM Symposium on Operating Systems Principles*, pages 85–100, 2011.

[142] Sai Deep Tetali, Mohsen Lesani, Rupak Majumdar, and Todd Millstein. MrCrypt: Static Analysis for Secure Cloud Computations. *ACM Sigplan Notices*, 48(10):271–286, 2013.

[143] Raluca Ada Popa, Emily Stark, Steven Valdez, Jonas Helfer, Nickolai Zeldovich,

and Hari Balakrishnan. Building Web Applications on Top of Encrypted Data Using Mylar. In *NSDI*, pages 157–172, 2014.

[144] Arvind Arasu, Spyros Blanas, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravishankar Ramamurthy, and Ramarathnam Venkatesan. Orthogonal Security with Cipherbase. In *CIDR*. Citeseer, 2013.

[145] Stephen Tu, M Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. Processing Analytical Queries Over Encrypted Data. In *Proceedings of the VLDB Endowment*, volume 6, pages 289–300. VLDB Endowment, 2013.

[146] Sumeet Bajaj and Radu Sion. TrustedDB: A Trusted Hardware-Based Database With Privacy and Data Confidentiality. *IEEE Transactions on Knowledge and Data Engineering*, 26(3):752–765, 2014.

[147] Andrew C Yao. Protocols for Secure Computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.

[148] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to Play Any Mental Game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

[149] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty Unconditionally Secure Protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19. ACM, 1988.

[150] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10. ACM, 1988.

[151] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, et al. Multiparty Computation Goes Live. *IACR Cryptology ePrint Archive*, 2008:68, 2008.

[152] VIFF Developement Team. Viff, the virtual ideal functionality framework, 2009. URL `http://viff.dk/`. Accessed March, 2017.

[153] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security*, pages 192–206. Springer, 2008.

[154] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics. pages 15–15, 2010. URL `http://dl.acm.org/citation.cfm?id=1929820.1929840`.

[155] Yael Ejgenberg, Moriya Farbstein, Meital Levy, and Yehuda Lindell. SCAPI: The Secure Computation Application Programming Interface. *IACR Cryptology ePrint Archive*, 2012:629, 2012.

[156] Andrew Chi-Chih Yao. How to Generate and Exchange Secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

[157] Vipul Goyal, Payman Mohassel, and Adam Smith. Efficient Two Party and Multi Party Computation Against Covert Adversaries. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 289–306. Springer, 2008.

[158] Dahlia Malkhi, Noam Nisan, Benny Pinkas, Yaron Sella, et al. Fairplay-Secure Two-Party Computation System. In *USENIX Security Symposium*, volume 4. San Diego, CA, USA, 2004.

[159] Benny Pinkas, Thomas Schneider, Nigel P Smart, and Stephen C Williams. Secure Two-Party Computation Is Practical. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 250–267. Springer, 2009.

[160] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster Secure Two-Party Computation Using Garbled Circuits. In *USENIX Security Symposium*, volume 201, 2011.

[161] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-Gate Secure Computation With Malicious Adversaries. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 285–300, 2012.

[162] Vladimir Kolesnikov and Thomas Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. In *International Colloquium on Automata, Languages, and Programming*, pages 486–498. Springer, 2008.

[163] D Willkomm, S. Machiraju, J. Bolot, and A. Wolisz. Primary Users in Cellular Networks: A Large-scale Measurement Study. *New Frontiers in Dynamic Spectrum Access Networks, DySPAN*, pages 1 – 11, 2008.

[164] George M Giaglis, Panos Kourouthanassis, and Argirios Tsamakos. Towards a Classification Framework for Mobile Location Services. *Mobile Commerce: Technology, Theory and Applications*, 67, 2003.

[165] TechTarget. Storage Gets a Dose of Medical Data. *Storage Technology Magazine*,

July 2008. URL `http://searchstorage.techtarget.com/magazineContent/Storage-gets-a-dose-of-medical-data`. Accessed March 2017.

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

[167] Wei Jiang and Chris Clifton. A Secure Distributed Framework for Achieving k-Anonymity. *The VLDB Journal*, 15(4):316–333, November 2006. ISSN 1066-8888.

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

[169] Samira Barouti, Dima Alhadidi, and Mourad Debbabi. Symmetrically-Private Database Search in Cloud Computing. *Proceedings of the 5th IEEE International Conference on Cloud Computing Technology and Science*, December 2013.

[170] 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.

[171] 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. ISBN 0-89791-962-9. doi: 10.1145/276698.276882.

[172] K. Eguro and R. Venkatesan. FPGAs for Trusted Cloud Computing. In *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, pages 63–70. IEEE, 2012.

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

[174] O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.

[175] 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. ISBN 3-540-67907-3.

[176] 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. ISBN 3-540-79227-9, 978-3-540-79227-7.

[177] 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. ISBN 978-1-60558-551-2. doi: 10.1145/1559845.1559850.

[178] 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.

[179] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, 2008.

[180] Martin Joakim Bittel Geisler. *Cryptographic Protocols:: Theory and Implementation*. PhD thesis, Aarhus University, Dept. of Computer Science, 2010.

[181] UCI Machine Learning Repository: Breast Cancer Data Set, April 2012. URL `http://archive.ics.uci.edu/ml/datasets/Breast+Cancer`. Accessed March, 2017.

[182] PwC. Putting Data Security on the Top Table: How Healthcare Organisations Can Manage Information More Safely, June 2013. URL `http://www.pwc.com/gx/en/industries/healthcare/publications/how-healthcare-organisations-can-manage-information-more-safely.html`. Accessed March, 2017.

[183] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006. ISBN 3-540-32731-2.

[184] Cynthia Dwork. A Firm Foundation for Private Data Analysis. *Communications of the ACM*, 54(1):86–95, 2011.

[185] Amazon Web Services. Amazon Elastic Compute Cloud (EC2) Documentation. URL `http://aws.amazon.com/documentation/ec2/`. Accessed March, 2017.

[186] Apache Software Foundation. Apache Spark. URL `http://spark.apache.org/`. Accessed March, 2017.