HOMOMORPHIC ENCRYPTION FOR DATA SECURITY IN CLOUD COMPUTING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ASNDAR WAINAKH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
CRYPTOGRAPHY

JUNE 2018

Approval of the thesis:

HOMOMORPHIC ENCRYPTION FOR DATA SECURITY IN CLOUD COMPUTING

submitted by **ASNDAR WAINAKH** in partial fulfillment of the requirements for the degree of **Master of Science in Department of Cryptography, Middle East Technical University** by,

Prof. Dr. Ömür Uğur
Director, Graduate School of **Applied Mathematics** _____

Prof. Dr. Ferruh Özbudak
Head of Department, **Cryptography** _____

Prof.Dr. Ersan Akyıldız
Supervisor, **Mathematics, METU** _____

Assoc.Prof.Dr. Murat Cenk
Co-supervisor, **IAM, METU** _____

Examining Committee Members:

Assoc.Prof.Dr. Ali Doğanaksoy
Mathematics, METU _____

Assoc.Prof.Dr. Zülfükar Saygı
Mathematics, TOBB ETU _____

Assist.Prof.Dr. Fatih Sulak
Mathematics, ATILIM UNIVERSITY _____

Prof.Dr. Ersan Akyıldız
Mathematics, METU _____

Assoc.Prof.Dr. Murat Cenk
IAM, METU _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ASNDAR WAINAKH

Signature :

ABSTRACT

HOMOMORPHIC ENCRYPTION FOR DATA SECURITY IN CLOUD COMPUTING

Wainakh, Asndar

M.S., Department of Cryptography

Supervisor : Prof.Dr. Ersan Akyıldız

Co-Supervisor : Assoc.Prof.Dr. Murat Cenk

June 2018, 64 pages

Recently, cloud computing has grown into a popular aspect of the IT industry. Cloud computing provides a range of hardware and software resources to its customers, which they can access through the internet. With the rapid development of cloud computing, various security issues related to confidentiality, and integrity are appearing. Traditional encryption techniques provide security to data while it is stored and transmitted, but not while it is processed. Hence traditional encryption techniques are not enough to secure data completely. Homomorphic encryption presents a resolution to this obstacle by allowing computation on encrypted data. Within the thesis, we present a summary of cloud computing security concerns plus the possibility of applying homomorphic encryption for data security.

Keywords : Cloud Computing, Homomorphic Encryption, Cryptography, Cloud Security, Encryption

ÖZ

BULUT BİLİŞİMDE VERİ GÜVENLİĞİNİ AMAÇLAYAN HOMOMORFİK ŞİFRELEME

Wainakh, Asndar
Yüksek Lisans, Kriptografi Bölümü
Tez Yöneticisi : Prof.Dr. Ersan Akyıldız
Ortak Tez Yöneticisi : Doç.Dr. Murat Cenk

Haziran 2018, 64 sayfa

Son zamanlarda, bulut bilişimi, bilişim teknolojileri sektöründe yaygın bir fenomen haline gelmiştir. Bulut bilişim, müşterilerine İnternet üzerinden erişebilecekleri bir dizi donanım ve yazılım kaynağı sunar. Bulut bilişimin hızla gelişmesiyle, gizlilik ve bütünlük ile ilgili çeşitli güvenlik sorunları ortaya çıkıyor. Geleneksel şifreleme teknikleri, depolanırken ve iletirken verilere sağladığı güvenliği işlenirken sağlamaz. Bu nedenle, verileri tamamen korumak için geleneksel şifreleme teknikleri yeterli değildir. Homomorfik şifreleme, şifrelenmiş veriler üzerinde hesaplamaya izin vererek bu soruna bir çözüm sağlar. Bu tezde, bulut bilişim güvenliği konularının bir özetini ve veri güvenliği için homomorfik şifreleme uygulama olasılığını sunuyoruz.

Anahtar Kelimeler : Bulut Bilişim, Homomorfik Şifreleme, Kriptografi, Bulut Güvenliği, Şifreleme

To My Beloved Family.
My father who first taught me the value of education and critical thought.
My mother for her constant, unconditional love and support.
My brother and sister for always supporting, helping, and standing by me.

ACKNOWLEDGMENTS

I would like to express my very great appreciation to my thesis supervisors Prof. Dr. Ersan Akyıldız and Assoc.Prof.Dr. Murat Cenk for their patient guidance, enthusiastic encouragement and valuable advices during the development and preparation of this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

In today's world, creating data is multiplying every 18 months [67]. The International Data Corporation (IDC) predicted that from 2013 to 2020, the digital universe would expand by a factor of 10 - from 4.4 trillion to 44 trillion gigabytes and the data volume handled by organizations is increasing 50 times. Hence, processing and storing data, by using conventional solutions, will get too expensive. To avoid the high expenses organizations and individuals tend further to outsource data to other companies, which have enough resources to perform the task in a shorter time and with a lower cost [19, 78].

Cloud Computing is thought to be the highest progressive innovations of the century. By delivering computing services such as storage, servers, software, networking and more over the Internet. Cloud Computing Providers are companies providing these services to the customers. Nevertheless, security is an essential challenge concerning the adoption of Cloud services. The Cloud provider can access the data that is in the Cloud at any time. It could modify or delete the data, and it could share the data with other parties. That makes the Cloud customers worried about losing control of their sensitive and high-risk data, such as medical records and financial details.

Several procedures can be adapted to reduce data security issues and help to provide data confidentiality, integrity, and availability to the Cloud customer. Cloud customers could encrypt the data on the Cloud in order to block unauthorized access, but the encrypted data under processing needs a particular sort of encryption. Encryption that enables the Cloud provider to process computation over encrypted data. This encryption called Homomorphic Encryption (HE).

1.1 Objectives

This thesis aims to explain the data security issues in the Cloud. How encryption could solve some issues. Moreover, we study the Homomorphic Encryption (HE) and its implementations. By covering the following objectives:

Define Cloud Computing and give a summary of the security matters influencing Cloud Computing.

Describe the performance of traditional encryption as a possible threat countermeasure and the challenges facing its implementation.

Manifest the basics of Homomorphic Encryption (HE) then present some of its real-life implementations.

1.2 Thesis Structure

In recent years, many experts presented the concepts of Cloud Computing, and Homomorphic encryption separately. Therefore based on the available literature, we tried to present the connection between the two concepts with more details. We hope that our research will be helpful to other researchers in this area.

The thesis is structured in a way that helps to achieve a broad and connected analysis of Cloud Computing and Homomorphic encryption. The thesis formed in 6 chapters:

In chapter 2, "Cloud Computing", we introduce the idea of Cloud Computing through presenting many definitions in the literature, then we discuss the essential characteristics of Cloud Computing together with a variety of existing services and deployment models.

Chapter 3, "Encryption and Cloud Computing" presents a review of the security concerns of Cloud Computing. Then we introduce the use of cryptography in general for supporting security in the Cloud. More specifically, we analyze conventional encryption as a potential step for solving the data security concerns. Then the chapter ends by reviewing the challenges involved when the traditional encryption is implemented in a Cloud environment.

In chapter 4 "Homomorphic Encryption", we describe in some detail Homomorphic Encryption (HE), Partially Homomorphic Encryption (PHE), Somewhat Homomorphic Encryption (SWHE), and Fully Homomorphic Encryption (FHE), and we present some example schemes.

In chapter 5 "Implementation of Homomorphic Encryption", we present some different implementations of Homomorphic Encryption (HE) as viewed in the literature.

CHAPTER 2

Cloud Computing

Cloud Computing is one of the quickest developing technologies that entice researchers to add and enhance its services. This technology is changing the traditional IT services into remote and on-demand paid hardware and software services. Usually, these services configured for specific needs. The Cloud provider manages and hosts these services and provide them to other organizations or customers. Organizations benefit from this by increasing their flexibility and efficiency without the need to have a devoted IT staff or buying special hardware equipment or software licenses.

2.1 What Is Cloud Computing?

Cloud Computing enables the customers to use applications installed on other Internet-connected devices separate from their computers. These devices end up to be a distant data center.

With the traditional IT model, the organization needs to purchase an application on Cds or DVDs then install it on its computers and with each new update the organization has to install that update on each computer. In addition to the expenses of all these licenses, since most likely the organization will use the application only once in a while, yet it has to pay as much for the license as if it used daily. The advantage of Cloud Computing is that the cloud provider will host the application, manage the software updates and handle the hardware (servers, networks). This will reduce the expenses as the organization will pay for using the services only when it uses them. Moreover, no need to purchase hardware equipment, or devote an IT team to operate the equipment [102].

2.1.1 Definition

Although the name Cloud Computing has been animated by the cloud icon to symbolize the World Wide Web in network diagrams the definitions that researchers and organizations provide often differ substantially. The Cloud was defined as “set of services hosted online can be accessed via the Internet, which is metaphorically represented as a ‘Cloud’.” [58].

The Cloud Security Alliance (CSA) contributes another definition as " Cloud describes the use of a collection of services, applications, information, and infrastructure comprised of pools of computing, network, information, and storage resources that can be acquired or released as necessary." [18].

The US National Institute of Standards and Technology (NIST) provided the common popular definition. Where it defined the Cloud as "Cloud Computing is 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" [79] . NIST definition also includes the essential characteristics, the service models, and the deployment models, see Figure 2.1.

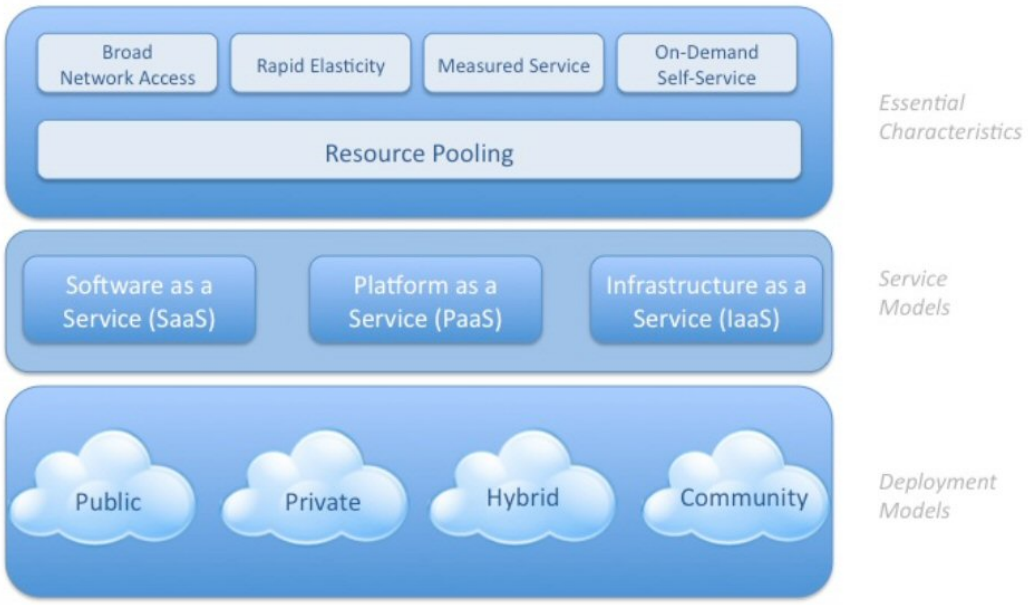

Figure 2.1: Cloud Computing [79]

2.1.2 Cloud Components

Cloud Computing is composed of several components: the clients, the high-speed broadband, and the data center.

Each component has a function and performs a specific part in providing an efficient cloud service [118].

> **The Clients:** The clients inside a Cloud Computing structure are similar to the clients in any local area network (LAN). These are, usually, the personal computers. However, they might also be laptops, tablets and mobile phones, as a result of their mobility, these play a significant rule in Cloud Computing. In other words, clients are the devices which the customers use to manage their data on the cloud [118]. Clients classified into three classes:

- Mobile smartphones.
- Thin clients are computers without hard drives. The server does all the data processing, then present the results.
- Thick clients are the regular computers, which use an Internet browser to connect to the cloud.

High-speed broadband: An essential component, which offers the media to connect all the other components. Broadband accessibility has become widely available, especially wireless access (e. g., WiFi, mobile) [76].

Data center: The data center is typically a combination of servers host the services that the customer subscribed. However, the servers are not necessarily located in one location. Usually, servers are in different places. Nevertheless, from a customer perspective, these servers act as one system.

2.2 Essential Characteristics

NIST has identified five essential characteristics of the Cloud Computing model [79], as in Figure 2.2. That shows how Cloud Computing model is different from the traditional computing model. The characteristics are:

On-demand self-service: In the Cloud Computing model, there are customers and Cloud providers. That allows the customer to self-provision additional computing capabilities, such as storage, processing capability, applications, memory and network resources as much as wanted automatically without any direct contact with the cloud provider.

Broad network access: Most of the data storage, and data processing is done in the Cloud. Therefore these capabilities are accessible through the network using standard protocols such as HTTP and IP. Which, permits the use of various types of thin and thick clients.

Resource pooling: Unlike traditional computing models, which dedicate computing facilities to one user, Cloud Computing is a multi-tenant model. The computing sources are combined to serve many customers.

Rapid elasticity: Additional resources can be elastically provisioned and released by customers according to their demands. The available resources usually seem to the customer as endless resources which can use in any capacity at any moment.

Measured service: There are several payment types in the Cloud, such as fixed price plans, pay-as-you-go, and free. There are many metering functionalities can be used in the cloud, based on the provided service (e.g., storage, processing, and bandwidth), a metering functionality is selected. The customers pay only for the used resources, only for the time they use them. To provide transparency records to the customer and the provider, the usage of resources can be monitored, controlled, and reported.

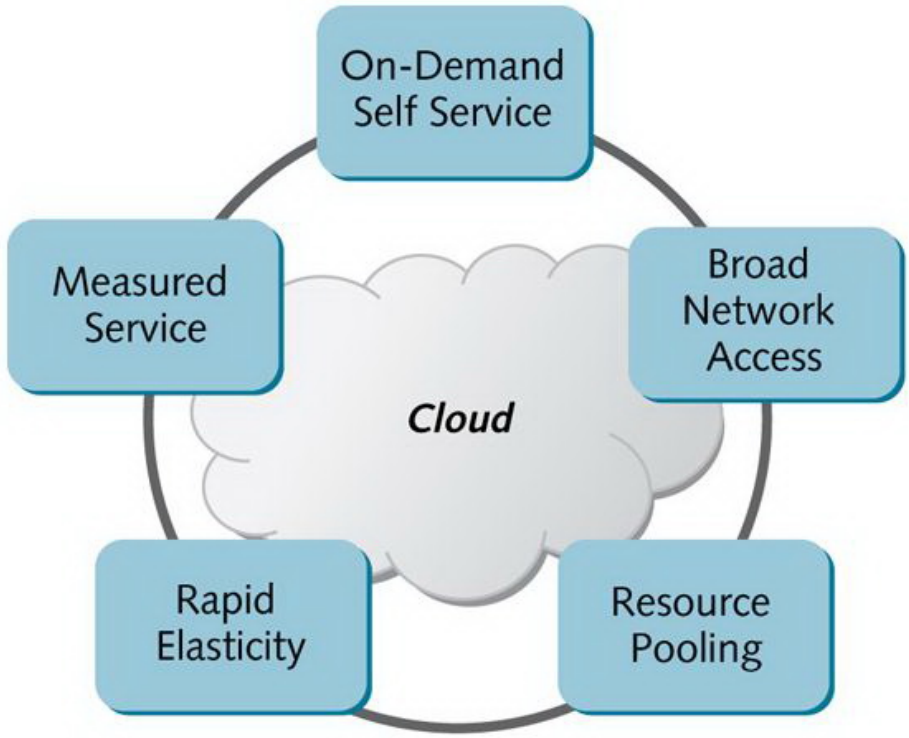

Figure 2.2: Essential Characteristics [57]

2.3 Service Models

The concept of Cloud Computing services is to provide many reusable sources and components across the vendor's network to the customer. The main services are widely known for Software, Platform, Infrastructure (SPI) model. NIST has categorized the services into three main approaches, software-as-a-service (SaaS), platform-as-a-service (Paas), and infrastructure-as-a-service (IaaS) [79], as shown in Figure 2.3. SaaS primarily focuses on end-users, PaaS targets software developers, while IaaS is enterprise oriented [83]. Each one of these services could split into more specific types of implementations, such as Security as a Service (SECaaS), Storage as a Service (STaaS), Application Programming Interface (API) as a Service (APIaaS), and Data as a Service (DaaS).

2.3.1 Software as a Service (SaaS):

The traditional computing model based on purchasing software for a license fee paid by the customer who has to load the software onto his hardware. The customer should take care of the operating systems installations, and the license agreements. To obtain additional utilization, the customer has to purchase a maintenance service.

In the SaaS model, the provider hosts applications as a service to the customer, who

does not purchase applications, but instead uses them on a pay-per-use model. The customer uses the service by any approved device wherever they have web access. The customer does not have to maintain or support the applications. However, the customer cannot manage or control the servers, network, storage, operating systems, or even the configuration settings, except for few specific settings. Applications like Google Docs, Gmail, and Microsoft Office365 are all considered as SaaS runs on the Cloud and offer the customer some functions and options using only an Internet browser [118].

The pay-per-use model can be a double-edged sword. Rather than pay for purchasing the software once and be done with it, costs for using the software can be a continuing situation: the more the customer uses it, the more he will pay. On the other hand, the customer is only billed based on his use of the software, which costs less money than buying the software outright. Outsourcing the hosting and management of applications to a Cloud service provider will help the customer to get rid of the cost of hosting the application internally [76].

Despite the advantages of SaaS, some obstacles are facing its implementation, such as the "lock-in" situation. After a customer starts using an application from a particular provider, it is considered difficult to port that application to another provider and migrate existing data [102].

2.3.2 Platform as a Service (PaaS):

The customer gets the required development environment to build applications entirely, including application design, programming languages, testing services, libraries and other development tools. These tools are supported by the provider, and the customer does not manage the cloud infrastructure still the customer can manage the applications deployed by him. Besides, the customer can manage some settings regarding the application-hosting environment [76].

PaaS is beneficial for start-up companies because PaaS systems can help them reduce the cost of deploying web-based applications by eliminating the complexity of managing the servers [102].

A limitation of PaaS is that the application is attached to a singular provider. In other words, customers may not be able to move an application that they have created with one cloud provider to another provider. Another limitation is, if the provider goes out of business, the customer will lose his applications and data [118].

2.3.3 Infrastructure as a Service (IaaS):

While SaaS and PaaS provide customers with applications and programming platforms, IaaS provides hardware that the customer can put whatever he wants onto it [76]. That includes the entire infrastructure, of networks, storage, processing, and other computing resources, which are needed to run and deploy applications [102]. The customer controls the authentication, identity management, operating systems,

and supervision of the resources [20].

Based on the resources needed by the application, the infrastructure provided by IaaS can be dynamically scaled up or down [118]. Still, the customer cannot control or manage the provider's actual physical infrastructure but can install any operating system or any application to run the resources [79]. The Cloud infrastructure hardware is usually shared among distinct Cloud customers using virtual servers, while the provider guarantees that the infrastructures used by various customers are logically separated [65].

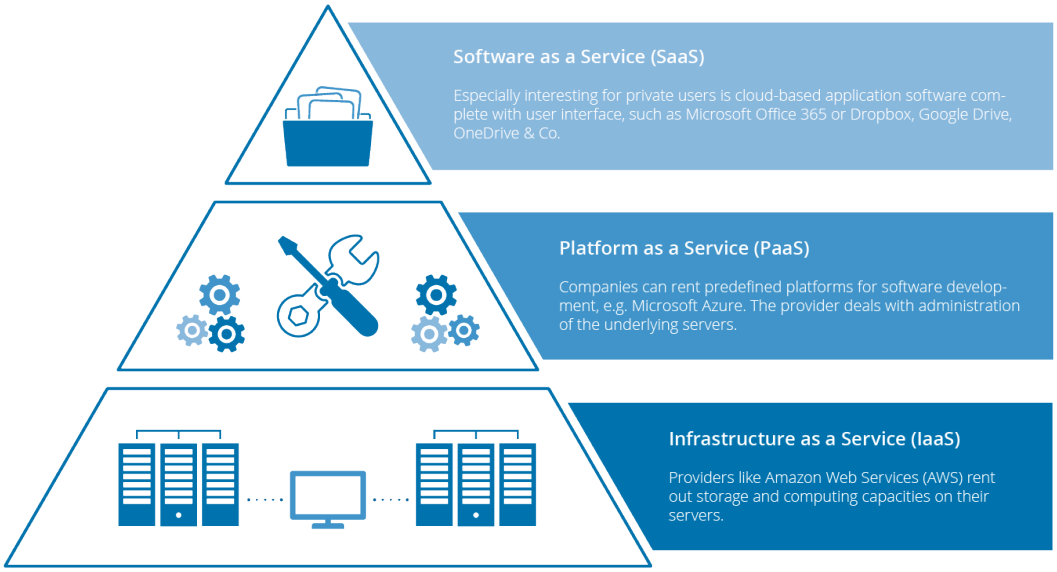

Figure 2.3: Service Models [10]

2.4 Deployment Models

In the Cloud, NIST has defined four deployment models depending on the relationship between the Cloud and the customers who are allowed to use the cloud services (person, organization, enterprise,...) [79]. The deployment models divided into four categories: Private, Community, Public, and Hybrid Clouds (See Figure /reffig:dep). In the following, we present an overview of each of these categories.

Private Clouds: The Private Cloud dedicates the infrastructure for exclusive use by a single organization and does not share it with other customers. This Cloud infrastructure can be owned, built, managed, and operated by the organization.

Thus this infrastructure requires more resources to build and manage. However, this closed environment will assure the privacy and the security of the data. It can also be owned, built, managed, and operated by some other third party, but the organization will still be able to control the underlying infrastructure with the logical and physical security features. The location of the infrastructure could be on or off premises. Thus, we can classify the private clouds into three categories [76]:

- Dedicated Private Clouds where the customer builds his data center, to host the Cloud infrastructure and an internal IT team will manage it.
- Community Private Clouds where the Cloud provider owns and operates the infrastructure. The customer decides the security and compliance requirements, for the provider to follow. The infrastructure can be hosted on the premises of a third party.
- Managed Private Clouds where the Cloud infrastructure is owned and hosted by the customer and managed by the provider.

Community Clouds: The Cloud infrastructure is dedicated to a specific community or a group of customers who have the same concerns or have a business partnership. It can be operated, built, owned and managed by one of the organizations or by another third party. It can be located on or off premises.

Public Clouds: The Cloud infrastructure is available to the general public. It is hosted, operated, and managed by the Cloud provider. Thus more significant cuts in cost can be achieved. The customer can control only some of the logical and physical security features. That might be a challenge for managing data security considering potentially malicious users could use the infrastructure. The infrastructure is located on the premises of the cloud provider.

Hybrid Clouds: A Hybrid Cloud environment consists of linking multiple Cloud deployments models (community, public, or private) together. With a Hybrid Cloud, to manage data security during Cloud interaction, the customer needs to consider the various security aspects of the different Clouds. For instance, organizations might run non-sensitive data and applications in a public cloud, while maintaining sensitive applications and data in a private cloud.

2.5 Examples of Cloud Service Providers

Many companies offer Cloud Computing services (SPI) to other organizations and customers. These companies referred to as Cloud Service Providers (CSPs). In what follows, we provide a short overview of the most prominent three providers with their service offerings and use cases (See Figure 2.5) [76].

Amazon Web Services (AWS): Provides (IaaS) solutions like storage and computing power among other services. These services allow organizations to get

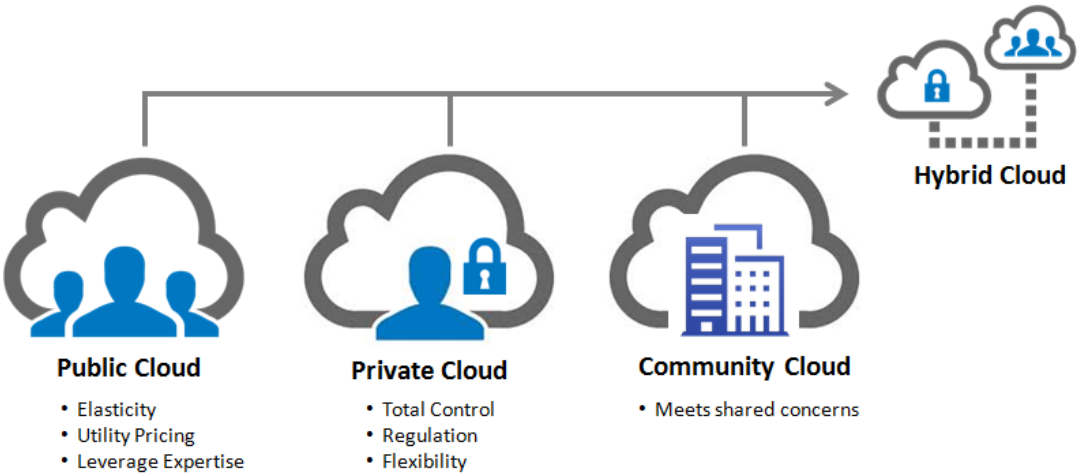

Figure 2.4: Deployment Models [96]

the benefit of Amazon's infrastructure. AWS offers many infrastructure-related services, including the following: Elastic Compute Cloud (EC2), Simple Storage Service (S3), Simple Queue Service (SQS), CloudFront, and SimpleDB [102].

Google Cloud Platform: Is a (PaaS) providing the customers with hosting and building applications on Google's infrastructure. On the other hand, Google Apps is a (SaaS) featuring several applications, including Gmail, Google Calendar, Talk, Docs, and Sites.

Microsoft Azure Services Platform: works as a runtime operating system for the applications and provides a collection of services that support management, development, and hosting of applications. The platform covers the following services: SQL Services, .NET Services, and Live Services.

Figure 2.5: Cloud Computing Providers

CHAPTER 3

Encryption and Cloud Security

Apart from the advantages of using the Cloud, we also need to look at the problems or disadvantages that it could cause. Security in the Cloud is a primary problem, considering that the customer does not have immediate control over the infrastructure and the data. As a consequence, the customer demands to have a guarantee that his data is secure. In other words, to guarantee the confidentiality, integrity, and availability of the data [111].

The data security threats categorized into three classes, the threats to confidentiality, availability, and integrity. Confidentiality indicates that the data should not be exposed to an unauthorized entity. Integrity means that the data should not be modified or deleted by unauthorized users. The issues related to the availability of data stored on the Cloud services are critical. Such issues, even if temporary, are a severe information security threats. Given that makes the system or data unusable or unavailable [116].

Depending on the deployment models and the type of the services, the provider, and the customer agreed on the security responsibilities. For example, in SaaS, the cloud provider usually manages the security of the infrastructure and the applications. In IaaS, the provider manages the infrastructure security, while the customer is subject to make sure of the software and the platform security. In PaaS, the provider controls the platform security, but the application security is controlled by the customer [102].

3.1 Cloud Security Threats

Through the years, many organizations like the Cloud Security Alliance (CSA) and the European Network and Information Security Agency (ENISA) have taken up the task of assessing the security risks in Cloud Computing models. They regularly evaluate the security threats that Cloud Computing can be subject to [61, 101]. This section introduces the most important Cloud security threats.

3.1.1 Abuse and Nefarious Use of Cloud Computing:

The Cloud infrastructure can be abused for nefarious purposes, or for performing other malicious activities. For instance, an attacker can use the virtually unlimited computing power and resources to host malicious code or to launch an attack against customers. Service providers may provide their potential customers with an illusion offer of unlimited computing, storage, and network capacity. Some providers even offer a free 30-days trial, with an easy registration process where the customer is asked to provide a legitimate credit card to join and instantly start using the Cloud services. These registration and usage models keep the customer identity relatively unknown, which can be abused by malicious code authors and spammers. IaaS and PaaS providers have experienced several of these kinds of attacks, where the customer has enhanced control over the infrastructure. SaaS, on the other hand, suffered fewer attacks since it has a system in which the customer can use the resources only by a hosted application in more controlled conditions.

3.1.2 Insecure Application Programming Interfaces:

Customers control and communicate with Cloud services using a set of Application Programming Interfaces (APIs) or other software interfaces. The Cloud providers use these interfaces to manage, monitor and synchronize data. Therefore, the security and availability of the general services will be affected, if the security of the underlying APIs is jeopardized. The APIs are exposed over the network, so the design of these interfaces must consider protection against threats and activity monitoring of the interfaces.

3.1.3 Malicious Insiders:

The threat of malicious insiders at the provider's data center is well-known to most organizations. These insiders have access to the data to manage the Cloud services, but they might also take advantage of the absence of transparency in the provider's procedures. For instance, a provider may not explain how the operators grant access to the Cloud infrastructure, or what is the hiring measures and manners for the workers. The level of access granted to employees could create an attractive opportunity for an adversary to collect confidential data or obtain full control over the cloud services with no chance of discovery. That is why, the provider is obligated to do all the needed procedures, such as monitoring employees and auditing all their actions.

3.1.4 Shared Technology Issues:

In order to deliver the services in a scalable way, the IaaS vendors share the Cloud infrastructure with different customers. The Cloud infrastructure tries to separate the activities of the different customers in the Cloud. However, the infrastructure (the

CPU caches and GPUs) was not created to provide secure separation for a multi-tenant Cloud infrastructure. That vulnerability could be a threat when a tenant operating system gains unauthorized control or influence on the underlying Cloud infrastructure [20]. To address this gap, the separation between these tenants is managed by a virtualization hypervisor, which interferes between the tenant operating systems and the hardware infrastructure [73]. However, virtualization hypervisors have shown some flaws. Therefore, to assure that some customers cannot affect the operations of other customers running on the same infrastructure, a robust defense strategy should be implemented by enforcing compute, storage, and network security. For instance, an individual customer should not have a way to access any other customer's data.

3.1.5 Data Loss or Leakage:

In the traditional IT systems the data is under the customer's direct control and more isolated. However, the data in the Cloud can be compromised in more severe ways due to the operational characteristics of the Cloud environment. The data in the Cloud could get deleted or modified without a backup of the original data. Also, storing data on unreliable media could cause loss of data confidentiality, this includes threats to encryption, such as using inappropriate encryption algorithms and processes, and bad key management practices. Finally, only authorized parties should have access to sensitive data.

3.1.6 Account or Service Hijacking:

Attackers are still working on practices like fraud, phishing, and exploitation of software vulnerabilities to hijack accounts or services. Reusing credentials and passwords often amplify the impact of such attacks [115]. If attackers obtain the credentials to the Cloud, then they can manipulate the data, return falsified information, eavesdrop on the activities and transactions, and redirect the customers to illegal sites. Weak authentication and access control pose a significant threat to the Cloud. Moreover, the attacker might also use the account or service as a base to launch further attacks.

3.1.7 Unknown Risk Profile:

The term "unknown risk profile" is another threat to the Cloud. Usually, the customer believes that the provider manages the Cloud infrastructure security, without knowing exactly how risks are being assessed and managed by the provider. The Cloud provider might lose track of the security consequences by using some nonstandard security practices, versions of software, code updates, vulnerability profiles, or security design that are not sufficiently verified and tested. Besides that, the provider might have configured the Cloud infrastructure inappropriately, that information about who are the other tenants using the infrastructure or other logs may be exposed. Consequently, further unknown risks are exposed.

3.2 Cryptography in The Cloud

After identifying the main security threats to Cloud Computing. It is useful to view a few cryptographic techniques to address some of the security issues. Cryptography performs an essential part in delivering data security in the Cloud. It can be used efficiently to handle some of the security issues, mainly when it used in collaboration with other countermeasures [7]. Various Cryptographic techniques can be practiced such as confidentiality, data integrity, user authentication, non-repudiation, and data origin authentication. In the following, we briefly present some of these techniques.

User authentication: Is the process of verifying that the identities of the entities participated in a transaction, are true at that point in time. Alternatively, data origin authentication is used to confirm the source of a specific message. In a Cloud environment, digital signatures are used to perform data origin authentication, given that creating such signatures require information about the secret key.

Data integrity: Ensure that other unauthorized entities did not modify the data. Using a key known only for the authorized entities, that can generate digital signatures and Message Authentication Codes (MAC) for the messages. A changed message will return invalid signatures or check-sums. In the Cloud, for instance, such cryptographic methods can be applied to preserve the integrity of forensic data used during an investigation.

Non-repudiation: Prevent an entity from making an argument that he did not make a transaction or send a specific message. The recipient can present a proof that a particular sender created this message. Digital signatures can achieve non-repudiation considering that the signature can only be created by an entity that knows the secret key, over a specific message.

Confidentiality: Means making data available exclusively to authorized entities, assuring it is not revealed through unauthorized entities. Cryptography works extremely well to achieve confidentiality by using encryption. Other techniques based on cryptography used to generate a pseudorandom number to generate keys for encryption schemes.

3.3 Encryption as a Threat Countermeasure

As we have seen in the previous section, a number of Cloud security threats were identified. In the section, we evaluate how encryption can assist to minimize such threats.

Abuse and Nefarious Use of Cloud Computing: For an attacker, the Cloud infrastructure is a desirable platform to use for conducting attacks. Because it is adjustable and upon demand can afford large volumes of computing power and

resources. To manage the use of the Cloud resources, some cryptographic procedures may help to stop unauthorized use. For example, to identify a customer on the Cloud, a digital signature generated by the customer can be used without revealing the secret key. The provider can easily identify malicious users. Encryption cannot solve this problem, because encryption works only on the data and cannot work on the Cloud infrastructure.

Insecure Interfaces and Application Programming Interfaces: To manage the Cloud services, the APIs are opened over the network. To avoid unauthorized customers, these interfaces need to be protected. By encrypting the communications, an unauthorized customer cannot eavesdrop the Cloud communications. Other cryptographic methods can be used, as explained in Section 4.1. This guarantee the integrity of the Cloud management messages, which also preserves against message forgery [7].

Malicious Insiders: For controlling and managing the data, some of the Cloud provide's employees might have access to the stored data. A malicious employee can easily abuse such rights. One of the methods of facing this threat is by encrypting the data, so it becomes insignificant to the employees. Then the data will be preserved against insiders who got unauthorized access to the data. Nevertheless, the encryption should be accompanied with proper key management methods, because if the insider gain access to the keys then he can decrypt the data [7, 61].

Shared Technology Issues: Since Cloud is a resource pooling model and the infrastructure is shared among many tenants, a customer can find a vulnerability in the virtualization hypervisor to get unauthorized access to the Cloud infrastructure. To address this threat some cryptographic methods can be used such as code signing, which allows the provider to verify that a trustworthy source creates the applications on the Cloud and that it has not been modified. Data encryption cannot help that a lot against this kind of threat as data confidentiality is not the main interest in this situation.

Data Loss or Leakage: Encryption can be used to achieve the data confidentiality in the cloud, and to protect the data against unauthorized modification or deletion. However, it cannot protect against the other threats [61].

Account or Service Hijacking Authentication tools and access control methods are very important in the Cloud. Because if an attacker managed to manipulate these methods he can get access to the customer's data and also use this data to launch more attacks. Digital signatures can handle the bypass alterations of access control information by attackers and verify identity. However, in such situations encryption, has a limited effect.

Unknown Risk Profile: The Cloud customer usually is not informed about how the provider evaluates and manage threats. And what security measures are used by the provider. Therefore, the customer should also take some steps, to increase the security afforded by the provider. Better than just blindly depend on the provider's standards to manage the data security. Encrypting the data before sending it to the provider is a very good step the customer should take.

3.4 Cloud Encryption

The customer loses control over the data after he transmits the data to the Cloud, and it becomes under the Cloud provider's control. Therefore, encryption plays an essential role in protecting data confidentiality. Cloud encryption is a complicated process that can be achieved in various techniques, depending on the Cloud service models (section ...). SaaS presents the least risks since the customer has minimum control over the infrastructure, while in IaaS the customer has more control over the infrastructure and he is responsible for the security measures. In this Section, we review the applicability of encryption in the Cloud service models.

3.4.1 Encryption in SaaS

In SaaS, the provider usually implements the security measures directly, even though in some cases before transmitting the data to the Cloud the customer can perform some security measures. Three main strategies can be adopted when considering encryption in SaaS [29]:

Cloud provider encrypt the data after transmission: The Cloud provider encrypts the data after receiving it from the customer. By doing this, data confidentiality would be protected against any threats and then the provider can re-transmit or store this data in the encrypted state. Transmitting the data, from the customer to the provider, done by using secure SSL/TLS channels between the customer and the provider. However, allowing the same provider to manage the data encryption and the data storage, causes additional security issues.

Customer encrypt the data before transmission: The customer can encrypt the data before sending it to the Cloud, using an encryption scheme independently from the provider. The encrypted data is transmitted to the provider and when the customer retrieves it, the data is decrypted. In this case, the customer is in complete control of the encryption scheme and the keys. Since the provider does not have the decryption key, that means the SaaS application can only execute limited operations on the encrypted data (such as searching). In some cases, the SaaS application will not be able to read or identify the encrypted data. For example, if the email sender encrypts the message before sending it to the email provider (SaaS provider), then the message will not be delivered to the recipients. The email provider is unable to interpret the message to find the receiver email address, with knowing the decryption key.

Third party encrypt the data during transmission: The data can be automatically encrypted by a network-based encryption scheme, located between the provider and the customer. The third party proxy could encrypt the transmitted data from the customer to the Cloud provider, also can decrypt any data received from the provider. This shifts the pressure of encryption and key management from the customer, and separate the encryption - storage services introduced by the provider. Of course, the third party proxy needs to be trusted.

3.4.2 Encryption in PaaS

In PaaS, the customer or the cloud provider can encrypt the data [29]. PaaS encryption can be done in two ways:

Application-Level Encryption: In PaaS, the customer is usually a software developer who writes applications that uses several PaaS services to provide the needed purposes. The PaaS encryption is implemented directly at the application layer by the developer. For instance, applications developers can program a software to encrypt the data before adding it to a database. Then when the data is retrieved from the database the application decrypted it again. The benefit of this technique is that the developer completely controls the encryption process, and can control it as he wants. Also, the Cloud provider does not get the data original form. However, this method could be complicated and slow to implement securely.

Infrastructure-Level Encryption: Typically, the Cloud provider in PaaS is responsible for secure the data stored on the Cloud infrastructure. Encrypting the data files stored on the disk protects them from being opened by unauthorized users. However, the Cloud provider can encrypt the data files when they arrive at the infrastructure. The problem with this practice is that the provider again will be responsible for both data storage and encryption.

3.4.3 Encryption in IaaS

The customer is responsible for the data security while the provider is responsible for the security of the infrastructure by using firewalls and other measures [29]. Encryption in IaaS implemented in many ways:

Volume-Based Encryption: The whole storage volume is encrypted in the Cloud infrastructure. So all the data which will be sorted on that volume (Hard disk) will be encrypted directly and this makes it so difficult to understand the data on the disk without the decryption keys. This guarantees the data confidentiality and that the customer and the Cloud provider cannot access the volume without keys. Usually, to handle the configuration the boot volume cannot be encrypted.

File-Based Encryption: In File-based encryption, each file is directly encrypted by the Cloud provider before it is recorded on the disk. This method enables the customer to decide which data must be encrypted, using which encryption scheme with the needed key-strength. Nevertheless, this method works just with files, and it cannot be used for data stored in a database.

Application-Level Encryption: In IaaS, we can also implement encryption directly within the applications operating on the Cloud infrastructure, as in PaaS. This encryption is recommended when IaaS is used to develop applications with built-in encryption.

3.5 Encryption and Data Life Cycle

Regardless of the fact that using encryption can help to maintain data confidentiality, implementing encryption is not easy and needs much more thinking and attention. We need to figure out at which point we should encrypt the data and when it should be decrypted. Encrypting data at the wrong stage could cause an undesirable cost regarding performance, computation, and complexity. However, not implementing encryption on data when it is needed could compromise the data at its most vulnerable state.

The data stored on the Cloud infrastructure referred to as data at rest. Data transmitted between the customer and the provider or transmitted among different providers or different customers called data in transit. Data is in use when the customer is currently displaying or processing the data [8]. In the following, we discuss how encryption can be applied to data in these stages.

Encryption of Data at Rest: The encryption of the stored data provides confidentiality for data, but also raises some difficulties such as falling to execute particular processes on encrypted data and performing a search for an object. The customer can encrypt data that is at rest before sending it to the Cloud, or the Cloud provider can encrypt data directly. This method ensures that the data is secure since it is encrypted. Also, the provider does not have the keys, which will limit the operations provider can perform on the data [8, 61].

In IaaS and PaaS the data at rest is usually encrypted. However, in PaaS, depending on the user data storage system, that could require extra complexity. In SaaS, the encryption should be integrated into the software's source code.

Encryption of Data in Transit: To protect the confidentiality of the transmitted data over the cloud network, Transport Layer Security (TLS) and Secure Socket Layer (SSL) protocols can establish secure channels. In order to encrypt the data, the asymmetric keys of SSLv3 and TLS are distributed using symmetric encryption. Other techniques can be used, depending on how the data is transmitted in the Cloud.

In SaaS, a proxy can be located between the customer and the provider, to encrypt the transmitted data. This will ensure that the provider uses, and store only encrypted data. In the public and the hybrid Cloud models, it is important to encrypt data in transit, while in the private Cloud the data is transmitted over an internal network, while data is transmitted over the Internet in a public Cloud [8].

Encryption of Data in Use: In some cases, a fault in the Cloud infrastructure could expose the data in use. For example, the unencrypted data currently in use in the Cloud infrastructure's memory might be disclosed to a malicious user. Therefore, this data should be encrypted to guarantee that it is not compromised. Still, this could cause few difficulties while processing data, because only the original plaintext of the data can be processed correctly. Processing encrypted ciphertext data gives wrong results. Before load and process encrypted data

in memory, it needs to be decrypted. Which makes it challenging to keep the data encrypted at all time. Therefore, it is very important to have the ability to perform meaningful processing operations on encrypted data without decrypting it [8].

3.6 Challenges of Implementing Cloud Encryption

Encryption might be suitable to be used to assure data confidentiality in the Cloud. However, we need first to recognize the difficulties that could face implementing encryption in the Cloud. In the following, we highlight two of these challenges.

Searching Encrypted Data: In order to access a specific file, when there are large amounts of data in the Cloud, the customer needs to search for it. However, searching encrypted data will be more complicated, unless the Cloud provider knows the decryption key. Still, in order to perform the search, the data the provider needs to decrypt the data continuously which is very computationally expensive [6].

Processing Encrypted Data: At this point, encrypted data cannot be processed, it must be decrypted in order to be processed and used. However, to get the data decrypted is a costly operation concerning the time and the required processing power. Therefore other methods which allow using encrypted data, and maintain its confidentiality are being researched. One alternative is Fully Homomorphic Encryption (FHE). FHE has the ability to allow performing complex mathematical operations on encrypted data, without decrypting it and without jeopardizing the encrypted data security in the process.

CHAPTER 4

Homomorphic Encryption

Homomorphism, in abstract algebra, is described as a function that maps an input element from the domain set, to an element as an output in the range of an algebraic set. In cryptography, Homomorphic Encryption (HE) is an encryption scheme that enables a cloud service provider to operate particular computation functions on the data while it is encrypted. With traditional encryption schemes, customers have to compromise their data security to use cloud services, since the traditional schemes cannot allow the provider to operate on encrypted data. Therefore, it is necessary to develop a scheme where computation operations can be performed on encrypted data while it is still encrypted.

The idea of using Homomorphic Encryption to do some computations on encrypted data was proposed for the first time after RSA sheme [104] by Rivest, Adleman, and Dertouzous in 1978. It was called "Privacy Homomorphism" [103]. This system inspired a lot of efforts by other researchers to create a homomorphic scheme supports a lot of computations [9, 11, 30, 35, 49, 63, 64, 85, 89, 90, 104, 107, 124]. All the proposed schemes have managed to perform one or few numbers of operations on the encrypted data. The actual breakthrough happened with Gentry's based on ideal-lattices scheme [40] that can perform an unlimited number of operations with arbitrary functions. However, it was not a practical scheme due to its high cost concerning computation. In the following years, new optimizations and improvements with other schemes were proposed.

There are three different types of the HE schemes depending on computational operations and the number of times it can be performed:

> Partially Homomorphic Encryption (PHE) supports only operations of one type performed as many times as wanted.
>
> Somewhat Homomorphic Encryption (SWHE) supports few types of operations performed only for few times.
>
> Fully Homomorphic Encryption (FHE) supports any kind of operations performed as many times as wanted.

We start by introducing the basics of Homomorphic Encryption. Then, introduce examples of the well-known schemes.

4.1 Definition of Homomorphic Encryption

Homomorphic encryption is a particular type of encryption, has the ability to execute computation on ciphertexts and gives the result of the same computation executed on the plaintexts. The result will be encrypted.

Definition 4.1. A homomorphic encryption scheme with encryption algorithm E over an operation ' ' supports the following equation:

$$E(m_1) \quad E(m_2) = E(m_1 \quad m_2), 8m_1, m_2 \; 2 \; M,$$

where M is the messages space [125].

The HE scheme has four main algorithms: (Figure 4.1):

The Key Generation Algorithm, *KeyGen*: inputs a security parameter. For an asymmetric HE scheme outputs a pair of secret key and public key, and for symmetric HE scheme a single key.

The Encryption Algorithm, *Enc*: inputs plaintext message m from message space M with the encryption key. To generate the ciphertext $c = E(m)$ from the ciphertext space C.

The Decryption Algorithm, *Dec*: inputs the ciphertext c with the decryption key, to get the plaintext message $D(c) = m$.

The Evaluation Algorithm, *Eval*: takes inputs ciphertexts (c_1, c_2) and performs the function $f(...)$ over the ciphertexts to output evaluated ciphertexts $f(c_1, c_2) = E(f(m_1, m_2))$, without seeing the messages (m_1, m_2), i.e. $D(f(c_1, c_2)) = f(m_1, m_2)$.

It is critical to preserving the format of the ciphertexts after an evaluation process to decrypt it correctly. Also, to perform an unlimited number of operations, the size of the ciphertext should stay constant. Otherwise, the ciphertext size will increase, and that will restrict the number of performed operations. Eval function in PHE schemes supports only either addition or multiplication, while it supports only a limited number of operations in SWHE schemes. In FHE schemes, Eval function supports the evaluation of arbitrary functions for an unlimited number of times over ciphertexts.

4.2 Partially Homomorphic Encryption

Many traditional encryption schemes can be classified as PHE due to their ability to perform one type of computation operation on encrypted data.

Definition 4.2. Partial Homomorphic Encryption (PHE) is either an additive homomorphic scheme, which supports only additive operations or a multiplicative homomorphic scheme, which supports only multiplicative operations on encrypted data.

We will intreduce examples for the important PHE schemes, such as RSA [104], Goldwasser-Micali [49], Elgamal [35], Benaloh [9], and Paillier [90].

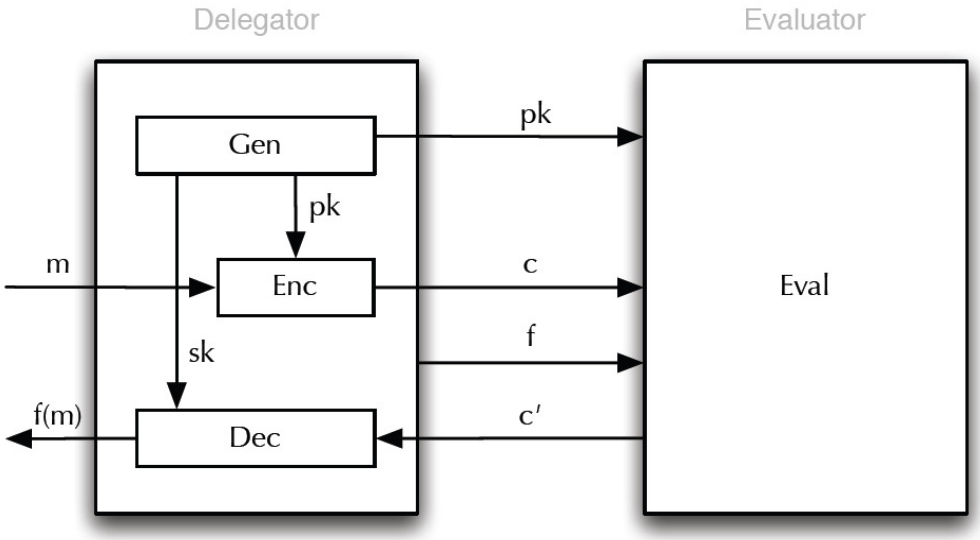

Figure 4.1: Homomorphic Encryption [110]

4.2.1 RSA

Shortly after Diffie Helman [31] developed public key cryptography in 1976, Rivest, Shamir, and Adleman [104] introduced RSA in 1978. The hardness of the RSA scheme was based on the problem of factoring a product of two large prime numbers. The homomorphic property of RSA was introduced later by, Rivest, Adleman, and Dertouzous using the term "privacy homomorphism" [103], which was an early example of PHE. The RSA scheme involves four algorithms as follows:

KeyGen Algorithm The public key is two integers (n, e), where $n = pq$ and p, q are large primes and e chosen such that $gcd(e, \varphi(n)) = 1$,where $\varphi(n) = (p \quad 1)(q \quad 1)$ and namely e is invertible $\pmod{\varphi(n)}$. The secret key is (d, n), where d is determined such that d is the inverse of e (i.e. $ed \quad 1 \pmod{\varphi(n)}$).

Encryption Algorithm First, the message is converted into a plaintext m *2* Z_n, then computes the ciphertext c as follows:

$$E(m) = m^e \pmod{n} = c, \tag{4.1}$$

where the ciphertext c *2* Z_n.

Decryption Algorithm Takes the secret key (d, n) with ciphertext c to decrypt

$$D(c) = c^d \pmod{n} = m, \tag{4.2}$$

because d is the multiplicative inverse of e in Z_n then $ed \quad 1 \pmod{\varphi(n)}$.

Homomorphic Property For m_1, m_2 *2* Z_n,

$$\begin{aligned} E(m_1) \quad E(m_2) &= (m_1^e \pmod{n}) \quad (m_2^e \pmod{n}), \\ &= (m_1 \quad m_2)^e \pmod{n}, \\ &= E(m_1 \quad m_2). \end{aligned} \tag{4.3}$$

As we can see, the multiplication homomorphic property of RSA can evaluate $E(m_1$ $m_2)$ directly from $E(m_1)$ and $E(m_2)$ without decrypting them.

4.2.2 Goldwasser-Micali

The first probabilistic public key encryption scheme was introduced in 1982 by Goldwasser and Micali [49]. Using the quadratic residuosity problem [47] , we say a number a is quadratic residue modulo n if we find an integer x satisfy $x^2 \quad a \pmod{n}$.

KeyGen Algorithm Takes $n = pq$ where p and q two large prime. Then finds x non-residue that the Legendre symbols $\left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) = \quad 1$ and hence the Jacobi symbol $\left(\frac{x}{n}\right) = 1$. The public key consists of (x, n), while the secret key is (p, q).

Encryption Algorithm The algorithm encodes the message m into bits $(m_1, \ldots, m_r)$, then for every bit m_i, we choose y_i such that $gcd(y_i, n) = 1$. To encrypt a bit:

$$E(m_i) = y_i^2 x^{m_i} \pmod{n} = c_i, \tag{4.4}$$

$m_i \; 2 \; f0, 1g$, and $m = m_0, m_1, \ldots, m_r$, $c = c_0, c_1, \ldots, c_r$ where $x \; 2 \; Z_n$ and the block size is r.

Decryption Algorithm We do not need a separate algorithm for decryption, because $x \; 2 \; Z_n$, c_i is quadratic residue modulo n only when $m_i = 0$. So if c_i is a quadratic residue modulo n then $m_i = 0$, otherwise $m_i = 1$. Which gives the plaintext $m = (m_1, \ldots, m_r)$.

Homomorphic Property For every bit $m_i \; 2 \; f0, 1g$;

$$\begin{aligned} E(m_1) \quad E(m_2) &= (y_1^2 x^{m_1} \pmod{n}) \quad (y_2^2 x^{m_2} \pmod{n}), \\ &= (y_1 \quad y_2)^2 x^{m_1 + m_2} \pmod{n}, \\ &= E(m_1 + m_2). \end{aligned} \tag{4.5}$$

The homomorphic property of GM scheme, allow us todirectly compute the encryption of the sum $m_1 + m_2$ from the encrypted $E(m_1)$ and $E(m_2)$. Since the message and ciphertext in GM scheme are both binary numbers, GM is additive homomorphic scheme over binary numbers.

4.2.3 ElGamal

In 1985 a new public key scheme based on the hardness of the discrete logarithm problem [47] was proposed by Taher ElGamal [35]. ElGamal scheme is considered to be an advanced variant of the Diffie-Hellman algorithm [31]. ElGamal is mostly used to encrypt the secret key of asymmetric encryption schemes. The ElGamal scheme algorithms are:

KeyGen Algorithm g generate a cyclic group G of order n. Then compute, $h = g^y$ for random $y \in \{1, 2, \ldots, n-1\}$. Outputs the public key (G, n, g, h) and y is the secret key.

Encryption Algorithm Using g and $x \in \{1, 2, \ldots, n-1\}$, convert the messgage m to $m' \in G$ then in output the ciphertext as a pair $(c = (c_1, c_2))$ as follows:

$$E(m) = c = (c_1, c_2) = (g^x, m' h^x) = (g^x, m' g^{xy}). \tag{4.6}$$

Decryption Algorithm To decrypt a ciphertext (c_1, c_2), first, compute $s = c_1^y$. Then:

$$c_2 s^{-1} = m' g^{xy} g^{-xy} = m', \tag{4.7}$$

where s^{-1} is the inverse of s in the group G. Then we get the plainttext message m by converting back m'.

Homomorphic Property

$$\begin{aligned} E(m_1) \cdot E(m_2) &= (g^{x_1}, m_1 h^{x_1}) \cdot (g^{x_2}, m_2 h^{x_2}), \\ &= (g^{x_1} g^{x_2}, m_1 h^{x_1} m_2 h^{x_2}), \\ &= (g^{x_1 + x_2}, (m_1 m_2) h^{x_1 + x_2}), \\ &= E(m_1 \cdot m_2). \end{aligned} \tag{4.8}$$

As we can see the ElGamal is a multiplicative homomorphic scheme.

4.2.4 Benaloh

Instead of encrypting the message bit by bit as in the GM scheme, Benaloh proposed an extension to the GM scheme to encrypt the message as a block [9]. Benaloh's security is using the higher residuosity problem (x^n), a generalization of thequadratic residuosity problems (x^2) [47].

KeyGen Algorithm Given a block size r, the two large primes p and q are chosen such that $r | (p-1)$, $gcd(r, (q-1)) = 1$ and $gcd(r, (p-1)/r) = 1$. Compute, $n = pq$ and $\varphi(n) = (p-1)(q-1)$. Choose $y \in \mathbb{Z}_n$ such that $y^{\varphi/r} \neq 1 \pmod{n}$. Finally, set $x = y^{\varphi/r} \pmod{n}$ then the secret key is (p, q, x) and the public key is (y, r, n).

Encryption Algorithm Encrypting $m \in \mathbb{Z}_r$, needs to take $u \in \mathbb{Z}_n$ then:

$$E(m) = y^m u^r \pmod{n} = c. \tag{4.9}$$

Decryption Algorithm Compute $a = c^{\varphi/r} \pmod{n}$, since $u \in \mathbb{Z}_n$ and $m \in \mathbb{Z}_r$:

$$\begin{aligned} a = c^{\varphi/r} &\equiv (y^m u^r)^{\varphi/r}, \\ &\equiv (y^m)^{\varphi/r} (u^r)^{\varphi/r}, \\ &\equiv (y^{\varphi/r})^m (u)^{\varphi}, \\ &\equiv x^m (u)^0 \equiv x^m \pmod{n}. \end{aligned} \tag{4.10}$$

To recover m from a, we find $m = \log_x(a)$ i.e., find m such that $x^m \quad a \pmod n$. When r is small, by checking if $x^i \quad a \pmod n$ for all $0 \ldots (r \quad 1)$ we can recover m. If r is large, to recover m in $O(\sqrt{r})$ time ,the Baby-step giant-step algorithm can be used.

Homomorphic Property

$$\begin{aligned} E(m_1) \quad E(m_2) &= (y^{m_1} u_1^r \pmod n) \quad (y^{m_2} u_2^r \pmod n), \\ &= y^{m_1 + m_2} (u_1 \quad u_2)^r \pmod n, \\ &= E(m_1 + m_2). \end{aligned} \tag{4.11}$$

The Benaloh scheme is an additively homomorphic. Because any multiplication operation on encrypted data corresponds to the addition on plaintext.

4.2.5 Paillier

Pascal Paillier invented the Paillier encryption scheme [90] in 1999. It is another probabilistic public-key scheme using composite residuosity problem [47]. It examines if an integer x satisfy $x^n \quad a \pmod{n^2}$ for an integer a. The Paillier scheme algorithms are:

KeyGen Algorithm Let p and q two large prime numbers, such that $gcd(pq, (p \quad 1)(q \quad 1)) = 1$. Compute $n = pq$ and $\lambda = lcm(p \quad 1, q \quad 1)$, then, select a random $g \in \mathbb{Z}_{n^2}$ such that $gcd(n, L(g^\lambda \pmod{n^2})) = 1$, and $L(u) = (u \quad 1)/n$ for every $u \in \mathbb{Z}_{n^2}$. Output the public key (n, g) and (p, q) as secret key.

Encryption Algorithm To encrypt a message $m \in \mathbb{Z}_n$. Select random r where $r \in \mathbb{Z}_n$. Compute ciphertext as:

$$E(m) = g^m r^n \pmod{n^2} = c. \tag{4.12}$$

Decryption Algorithm To decrypt $c \in \mathbb{Z}_{n^2}$. Compute the plaintext message as:

$$D(c) = \frac{L(c^\lambda \pmod{n^2})}{L(g^\lambda \pmod{n^2})} \pmod n = m. \tag{4.13}$$

Homomorphic Property

$$\begin{aligned} E(m_1) \quad E(m_2) &= (g^{m_1} r_1^n \pmod{n^2}) \quad (g^{m_2} r_2^n \pmod{n^2}), \\ &= g^{m_1 + m_2} (r_1 \quad r_2)^n \pmod{n^2}, \\ &= E(m_1 + m_2). \end{aligned} \tag{4.14}$$

Paillier is homomorphic over addition scheme. And it has some extra operations on plaintexts $m_1, m_2 \in \mathbb{Z}_{n^2}$ by using the public key pair (n, g) and $E(m_1), E(m_2)$:

$$E(m_1) \quad E(m_2) \pmod{n^2} = E(m_1 + m_2 \pmod n). \tag{4.15}$$

$$E(m_1) \quad g^{m_2} \pmod{n^2} = E(m_1 + m_2 \pmod{n}). \tag{4.16}$$

$$E(m)^k \pmod{n^2} = E(km \pmod{n}). \tag{4.17}$$

Paillier's encryption scheme is widely used in electronic voting protocols and biometric applications.

4.2.6 Other PHE schemes

PHE schemes were not considered as solutions for Cloud Computing, due to the limited range of computation operations [68, 109]. Several efforts have tried to enhance the adaptability of the existing PHE schemes. Examples of such efforts include a new PHE scheme proposed by Okamoto-Uchiyama (OU) [89] to enhance the computational efficiency. Naccache-Stern [85] introduced a generalization of the Benaloh scheme to improve its computational performance by changing the decryption algorithm of the scheme. Similarly, a generalization of Paillier was presented by Damgard-Jurik (DJ) [30]. Likewise, Kawachi (KTX) et al. [64] proposed an additively homomorphic scheme based on lattice problems. Finally, Galbraith [38] presented a generalization of Paillier's scheme on elliptic curves.

Table 4.1: Well-known Partial Homomorphic Encryption Schemes (PHE)

PHE Schemes	Encryption Scheme				HE Properties		
	Symmetric	Asymmetric	Probabilistic	Determinstic	Additive	Multiplicative	Algorithm
RSA [104]							$(m_1^e \pmod{n}) \quad (m_2^e \pmod{n}) = (m_1 \quad m_2)^e \pmod{n}$
GM [49]							$(y_1^2 x^{m_1} \pmod{n}) \quad (y_2^2 x^{m_2} \pmod{n}) = (y_1 \quad y_2)^2 x^{m_1+m_2} \pmod{n}$
Elgamal [35]							$(g_1^x, m_1 h^{x_1}) \quad (g^{x_2}, m_2 h^{x_2}) = (g^{x_1+x_2}, m_1 8 m_2 h^{x_1+x_2})$
Benaloh [9]							$(y^{m_1} u_1^r \pmod{n}) \quad (y^{m_2} u_2^r \pmod{n}) = y^{m_1+m_2}(u_1 \quad u_2)^r \pmod{n}$
Paillier [90]							$(g^{m_1} r_1^n \pmod{n^2}) \quad (g^{m_2} r_2^n \pmod{n^2}) = g^{m_1+m_2}(r_1 \quad r_2)^n \pmod{n^2}$

4.3 Somewhat Homomorphic Encryption

Before the first usable FHE was released in 2009 [46], there were some practical SWHE examples, such as [11, 63, 107, 124]. Some of the SWHE schemes are used to build FHE schemes as we will see in FHE section. For now, we will talk about one of the main SWHE schemes.

Definition 4.3. Somewhat Homomorphic Encryption (SWHE) can perform both additive and multiplicative operations, but only for limited number of repetitions. This limitation is defined by the scheme's ability to decrypt ciphertexts associated with the homomorphic operations correctly.

Generally, the ciphertext of the homomorphic encryption scheme has a noise parameter, and to decrypt it properly the noise must be less than a specific limit. A SWHE scheme can perform both additive and multiplicative homomorphism on encrypted data, but that will increase the noise in the generated homomorphic ciphertext with each operation. In order to keep the noise parameter as small as possible, SWHE schemes can execute only a limited number of operations.

4.3.1 Boneh-Goh-Nissim (BGN)

Until 2005, all the proposed Homomorphic encryption schemes limited to only either addition or multiplication operation. The initial actions approaching new scheme were presented by Boneh-Goh-Nissim (BGN) [11]. BGN scheme supports one multiplication and an unlimited number of additions and with a constant-size ciphertext. The scheme based on the subgroup decision problem [47]. It is about deciding if an element belongs to a subgroup G_p of the group G of order $n = pq$, and p, q are distinct primes. The scheme algorithms are:

KeyGen Algorithm The public key is (n, G, G_1, e, g, h), where G, G_1 are cyclic groups of order $n = q_1 q_2$, q_1 and q_2 are two distinct large primes, and e is a bilinear map such that $e : G \quad G \; ! \; G_1$. Pick two random generators g, u of G, let $h = u^{q_2}$. Then h is a random generator of the subgroup of G with order q_1. The secret key is q_1.

Encryption Algorithm To encrypt m, pick a random number $r \; 2 \; f0, 1, \ldots, ng$ and encrypt it using g and h as follows:

$$E(m) = g^m h^r = c \; 2 \; G. \tag{4.18}$$

Decryption Algorithm To decrypt a ciphertext c using the secret key q_1, notice that:

$$c^{q_1} = (g^m h^r)^{q_1} = (g^{q_1})^m. \tag{4.19}$$

Note that $h^{q_1} \quad 1 \pmod{n}$. To decrypt m, compute:

$$D(c) = \log_{g^{q_1}}(c^{q_1}) = m. \tag{4.20}$$

Since the discrete logarithm cannot be computed quickly, the message space should be kept small.

Homomorphism over Addition: To add m_1 and m_2 by using ciphertexts $E(m_1) = c_1$ and $E(m_2) = c_2$:

$$c = c_1 c_2 h^r = (g^{m_1} h^{r_1})(g^{m_2} h^{r_2}) h^r = g^{m_1+m_2} h^{r_1+r_2+r}. \tag{4.21}$$

From the resulting ciphertext c, it easy to recover $m_1 + m_2$.

Homomorphism over Multiplication: Homomorphic multiplication, uses bilinear map. Let $g_1 = e(g, g)$ and $h_1 = e(g, h)$ then g_1 with order n and h_1 with order q_1. With $\alpha \ 2 \ Z$ such that $h = g^{\alpha q_2}$. Then, the multiplication of m_1 and m_2 by using $c_1 = E(m_1) = g^{m_1} h^{r_1} \ 2 \ G$ and $c_2 = E(m_2) = g^{m_2} h^{r_2} \ 2 \ G$ are computed as follows:

$$\begin{aligned}
c &= e(c_1, c_2) h_1^r \ 2 \ G_1, \\
&= e(g^{m_1} h^{r_1}, g^{m_2} h^{r_2}) h_1^r, \\
&= e(g^{m_1+\alpha q_2 r_1}, g^{m_2+\alpha q_2 r_2}) h_1^r, \\
&= e(g, g)^{(m_1+\alpha q_2 r_1)(m_2+\alpha q_2 r_2)} h_1^r, \\
&= e(g, g)^{m_1 m_2 + \alpha q_2 (m_1 r_2 + m_2 r_1 + \alpha q_2 r_1 r_2)} h_1^r, \\
&= e(g, g)^{m_1 m_2 + (m_1 r_2 + m_2 r_1 + \alpha q_2 r_1 r_2)\alpha q_2} h_1^r, \\
&= e(g, g)^{m_1 m_2} e(g, g)^{(m_1 r_2 + m_2 r_1 + \alpha q_2 r_1 r_2)\alpha q_2} h_1^r, \\
&= e(g, g)^{m_1 m_2} e(g^{m_1 r_2 + m_2 r_1 + \alpha q_2 r_1 r_2}, g^{\alpha q_2}) h_1^r, \\
&= e(g, g)^{m_1 m_2} e(g^{m_1 r_2 + m_2 r_1 + \alpha q_2 r_1 r_2}, h) h_1^r, \\
&= e(g, g)^{m_1 m_2} e(g, h)^{m_1 r_2 + m_2 r_1 + \alpha q_2 r_1 r_2} h_1^r, \\
&= e(g, g)^{m_1 m_2} h_1^{m_1 r_2 + m_2 r_1 + \alpha q_2 r_1 r_2} h_1^r, \\
&= e(g, g)^{m_1 m_2} h_1^{r + m_1 r_2 + m_2 r_1 + \alpha q_2 r_1 r_2}.
\end{aligned} \tag{4.22}$$

$r + m_1 r_2 + m_2 r_1 + \alpha q_2 r_1 r_2$ is distributed uniformly like r then $m_1 m_2$ can be recovered correctly from the c. But, c is now in G_1 instead of G. That's why nomore homomorphic multiplication operation can be done in G_1. However, it allows unlimited number of homomorphic additions.

4.3.2 Other SWHE schemes

Melchor et al. [77] depending on some well-known schemes that have some homomorphic properties, introduced an encryption scheme supports constant depth circuit homomorphic evaluation. Another scheme suggested by Sander, Young, and Yung (SYY) [107], supports only one OR/NOT gate with many ANDing gates. However, the circuit depth evaluation was limited, because the ciphertext size was increasing with each OR/NOT gate evaluation. Some of the other efforts reported as broken such as, [32, 51, 62, 84].

4.4 Fully Homomorphic Encryption Schemes

During 2009, Craig Gentry introduced the earliest FHE scheme in his Ph.D. thesis [40]. His proposed scheme was also considered as a framework for how to obtain an FHE scheme. Therefore, based on Gentry's work several researchers tried to create other practical FHE schemes.

Definition 4.4. Fully Homomorphic Encryption scheme (FHE) has the ability to perform an unlimited number of both additive and multiplicative homomorphic operations on encrypted data [37].

Even though the FHE scheme introduced by Gentry was encouraging, it was very difficult to use in real life due to its computational cost. Consequently, many optimizations have implemented his scheme to make it more usable in real life applications. While the efforts continued to create new FHE schemes most of them were based on the lattices problems.

Fully Homomorphic Encryption schemes can be categorized into four main categories based on the problems:

1. **Ideal lattice based** was proposed by Gentry [46], then several researchers tried to improve Gentry's ideal lattice-based FHE scheme, such as Smart and Vercauteren [112].
2. **Over integers based** Van Dijket al [117] proposed a scheme based on the Approximate-GCD problems.
3. **Ring Learning with Error (RLWE) problems-based** was proposed by Brakerski and Vaikuntanathan [15].
4. **NTRU-like** NTRUEncrypt is an old lattice-based encryption scheme who has homomorphic properties recognized recently [74].

We will explain more about these four categories in the following sections.

4.4.1 Preliminaries

This section states some necessary definitions for the schemes described later.

4.4.1.1 Lattice

Definition 4.5. R^m is an Euclidean space with m-dimensional, with n linearly independent vectors $b_1, \ldots, b_n$ in R^m $(m \quad n)$. The following set in R^m defined as a Lattice

$$L(b_1, \ldots, b_n) = \left\{ \sum_{i=1}^{n} x_i b_i : x_i \; 2 \; \mathsf{Z} \right\}, \tag{4.23}$$

We call m dimension of the lattice and n the rank of the latticeand, when $m = n$, the basis vectors equals the number of coordinates, we say it is a full rank or full dimensional lattice [80]. The lattice basis is the sequence of vectors $b_1, \ldots, b_n$ is called, and it is conveniently represented as a matrix

$$B = [b_1, \ldots, b_n] \; 2 \; \mathsf{R}^{m \;\; n}. \tag{4.24}$$

Using the previous matrix notation, with the matrix-vector multiplication we can write 4.23 as:

$$L(B) = \mathsf{f}Bx : x \; 2 \; \mathsf{Z}^n \mathsf{g}. \tag{4.25}$$

Graphically, in a n-dimensional infinite regular grid a lattice is the set of intersection points of the grid. The grid not necessarily orthogonal. A 2-dimensional example is shown in Figure 4.2. A lattice point is a result of the bases vectors linearly combining, and the repeated pattern "Fundamental Parallelepiped" shown in red area, is bounded by lattice points.

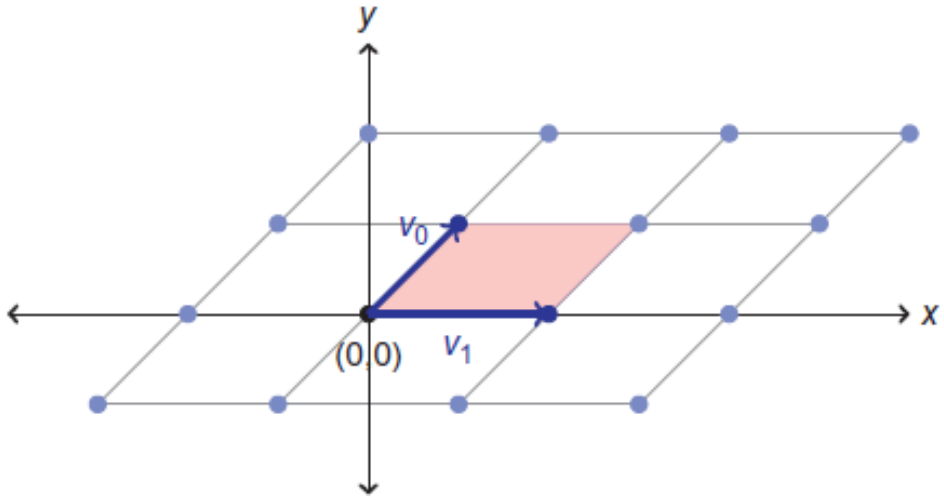

Figure 4.2: A lattice in R^2 [66]

The lattice's basis is not unique. A lattice can have many different bases. If the vectors of the basis are almost orthogonal, we call it a "good" basis. Otherwise, it is called "bad" basis. Usually, the "bad" bases are shorter than the "good" bases. The lattice theory was introduced in 1910 by Minkowski [82]. Then in 1996, Ajtai [5] defined a class of random worst-case lattice problems, Shortest Vector Problem (SVP) and Closest Vector Problem (CVP) were used later for lattice-based schemes [80, 91].

Given a basis, SVP problem finds the shortest nonzero vector in the lattice.

Definition 4.6. (Shortest Vector Problem, SVP) Given a basis $B \; 2 \; \mathsf{Z}^{m \;\; n}$, find a nonzero lattice vector Bx (with $x \;\; 2 \;\; \mathsf{Z}^n \; n \; \mathsf{f}0\mathsf{g}$) such that $\mathsf{k}Bx\mathsf{k} \quad \mathsf{k}By\mathsf{k}$ for any other $y \; 2 \; \mathsf{Z}^n \; n \; \mathsf{f}0\mathsf{g}$ [80].

The CVP problem, finds the nearest lattice point to a given point of the lattice.

Definition 4.7. (Closest Vector Problem CVP) Let $B \in \mathbb{Z}^{m \times n}$ be a lattice basis. Find a lattice vector Bx closest to the target vector $t \in \mathbb{Z}^m$, i.e., find an integer vector $x \in \mathbb{Z}^n$ such that $\|Bx - t\| \leq \|By - t\|$ for any other $y \in \mathbb{Z}^n$ [80].

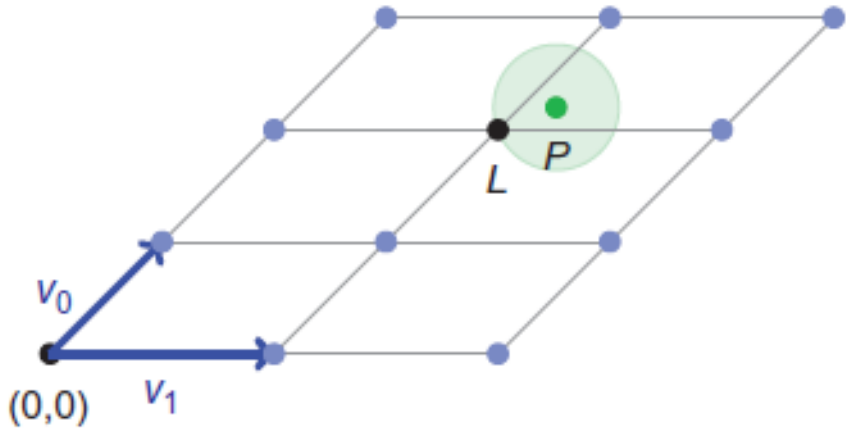

Figure 4.3: Finding the nearest lattice point [66]

4.4.1.2 Circuits

Definition 4.8. (Circuits) A circuit is an acyclic directed graph has edges and nodes. The edges named wires and nodes named gates. Depending on the input contents of the circuit (Boolean, Integers, ...), the gates can represent logic gates (AND, OR, NOR, NAND, ...) or arithmetic operations [50].

When a function f is needed to be evaluated, f is represented as a circuit where the gates are arranged into levels according to the function's operations, to be executed sequentially.

Definition 4.9. The number of the non-input gates, is called the size of a circuit C [50].

Definition 4.10. The length of the longest path, from an input gate to the output gate, is the depth of a circuit C [50].

As shown in Figure 4.4 the function f computes $A.B + B.C.(B + C)$ using logic gates AND and OR.

4.4.2 Ideal Lattice-based FHE schemes

Craig Gentry [46] introduced the first encryption scheme that can evaluate arbitrary depth circuits. Gentry first created a SWHE scheme based on ideal lattices. That can

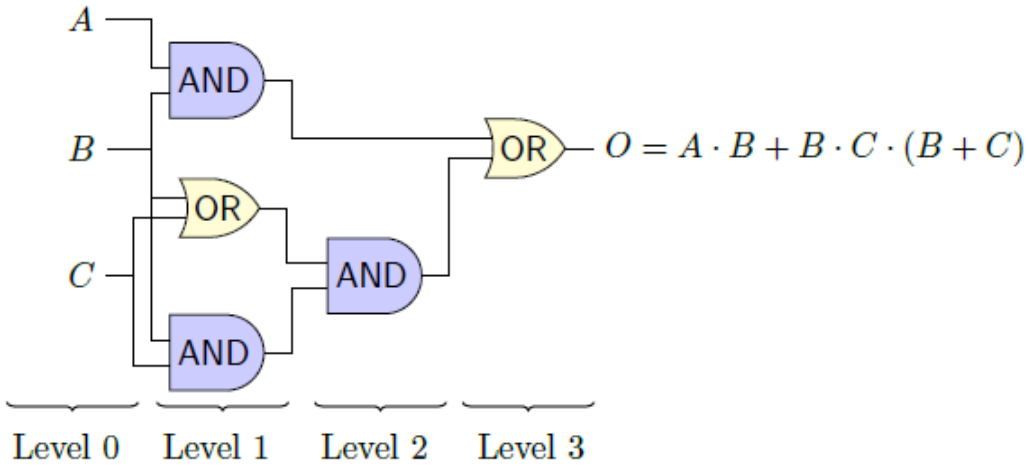

Figure 4.4: Circuit representation [50]

evaluate homomorphically the ciphertext for only a limited number of times, i.e., for low depth circuits. This SWHE scheme adds some noise to the plaintext to get the ciphertext. This noise increases with each addition and multiplication operation, i.e., with each gate in the circuit, performed on ciphertexts until eventually, the noise makes the decryption algorithm unable to retrieve the plaintext from the ciphertext correctly.

To get a ciphertext that allows an arbitrary number of homomorphic operations, Gentry used a method called bootstrapping. Bootstrapping reduces the noise in the ciphertext to an acceptable level. To perform the bootstrapping the SWHE scheme needs to have a low complexity decryption algorithm, i.e., low depth decryption circuit so that it can evaluate its own decryption algorithm. RSA and ElGamal, have more complex decryption algorithms than the lattice-based encryption schemes, which make it an excellent choice for FHE schemes.

The bootstrapping efficiently refresh the ciphertext from the noise to be used again in other addition and multiplication operations. Gentry's meaning of "refreshing" is first to generate a pair of encryption-decryption keys then encrypt the data with the encryption key, perform as much as computations on the encrypted data that the SWHE scheme can perform correctly. Then generate a second pair of encryption-decryption keys and encrypt the results under the second encryption key. Now to reduce the noise from the encrypted results, Gentry encrypted the first decryption key with the second encryption key. Then run the decryption algorithm of the SWHE scheme through the evaluation algorithm, which will decrypt the results encrypted under the first key but will keep it encrypted under the second key. Continue with the computations as much as wanted. When the computations are completed, decrypt the results ciphertext with the last decryption key generated to obtain the computations result. This processes can be repeated again and again, to make the scheme capable of evaluating an unlimited number of operations on the ciphertexts. However, this scheme was not practical, because the operations time and the ciphertext volume keep increasing. Gentry's blueprint consist of three main steps:

1. **Step 1: Constructing a SWHE scheme based on ideal lattices.** As with Goldreich, Goldwasser, and Halevi (GGH) scheme [48], Gentry's SWHE scheme was based on the lattice reduction problems [91]. For a given lattice, the lattice reduction tries to find a relatively short and orthogonal basis, i.e. "good" basis of the lattice. Usually, If the "good" bases of a lattice are known, the SVP and CVP problems can be solved in polynomial time. However, if the "good" bases of the lattice are not known, the best-known algorithms can solve these problems, like LLL algorithm [69] can solve it in exponential time.

   A "bad" basis of the lattice is used to generate the public key, while a "good" basis generates the secret key. Finally, the noise is added to a lattice point to get the ciphertext. To decrypt the ciphertext, we need to find the closest lattice point by using the secret key.

   In this scheme, lattices are used to represent the ideals. For instance, in an ideal lattice, a column of lattice with basis B_I of length n is used to present an ideal $I \ 2 \ \mathsf{Z}[x]/(f(x))$ with $f(x)$ of degree n. The SWHE scheme algorithms are:

   KeyGen Algorithm Takes as input a fixed ring R and the basis B_I of small ideal $I \trianglelefteq R$ to embed the message into noise vector.
   Additionally, an $IdealGen(R, B_I)$ algorithm generates the keys. Using a "good" basis B^{sk} of J for the secret key. A "bad" basis B^{pk} of the ideal lattice J is used as the public key is. J is an ideal lattice such that I, J are relatively prime and $I + J = R$.
   A $Samp()$ algorithm used in the Encryption Algorithm to sample a short vector from a coset of the ideal, by shifting an ideal a specific quantity.
   The public key is $(R, B_I, B_J^{pk}, Samp())$ and the secret key is B_J^{sk} .

   Encryption Algorithm Takes the public key B^{pk} and the message $\vec{m}$. The plaintext space P is a subset of $R \pmod{B_J}$, where $R \pmod{B_J}$ is the set of distinguished representatives of $\vec{r} + I$ over $\vec{r} \ 2 \ R$, with respect to the basis B_I of I.
   Using $Samp(\vec{m}, B_I)$ algorithm to sample a vector $\vec{e} = \vec{m} + \vec{i}$ which will be reduced modulo the public basis B_J^{pk}

$$E(\vec{m}) = \vec{e} \pmod{B_J^{pk}} = (\vec{m} + \vec{i}) \pmod{B_J^{pk}} = \vec{c}. \tag{4.26}$$

   The ciphertext in this case is a vector $\vec{c}$ where it is encoded in the distance to the closest lattice point.

   Evaluation Algorithm Takes as input the public key B_J^{pk}, a set of ciphertexts $\Psi = \mathsf{f}\vec{c_1}, \ldots, \vec{c_t}\mathsf{g}$, and a circuit C with gates operations modulo B_I from a permitted set C . To compute the output ciphertext $\vec{c}$, it performs Add_{B_I} and $Mult_{B_I}$ in the specific sequence .

$$Add(B_J^{pk}, \vec{c_1}, \vec{c_2}) \text{ outputs } \vec{c_1} + \vec{c_2} \pmod{B_J^{pk}}.$$
$$Mult(B_J^{pk}, \vec{c_1}, \vec{c_2}) \text{ outputs } \vec{c_1}.\vec{c_2} \pmod{B_J^{pk}}.$$

   In the process of computing $\vec{c}$, the algorithm applies a $\pmod{B_I}$ circuit C to the plaintexts, then by replacing C's Add_{B_I} and $Mult_{B_I}$ operations with the ring operations addition $+$ and multiplication . in the ring R [50].

Decryption Algorithm

$$\begin{aligned} \vec{m} &= (\vec{c} \pmod{B_J^{sk}}) \pmod{B_I}, \\ &= \vec{e} \pmod{B_I}, \end{aligned} \tag{4.27}$$

because $\vec{e} = \vec{m} + \vec{i}$ with $\vec{i}\ 2\ I$.

Key Generation: $\mathsf{KeyGen}(R, \mathbf{B}_I)$

Input: R and basis $\mathbf{B}_I$ of $I \trianglelefteq R$.

Run $\mathsf{IdealGen}(R, \mathbf{B}_I)$
$(\mathbf{B}_{pk}, \mathbf{B}_{sk}) \leftarrow \mathsf{IdealGen}(R, \mathbf{B}_I)$, where $I + J = R$.
$(\mathbf{B}_{pk}, \mathbf{B}_{sk})$ are bases of J

Output: (pk, sk)
public key: $pk = (R, \mathbf{B}_I, \mathbf{B}_{pk}, \mathsf{Samp})$
secret key: $sk = \mathbf{B}_{sk}$

Encryption: $\mathsf{Enc}(\vec{m}, pk)$

Input: $\vec{m} \in P$ and pk.

Run $\mathsf{Samp}(\vec{m}, \mathbf{B}_I)$
$\vec{e} \leftarrow \mathsf{Samp}(\vec{m}, \mathbf{B}_I)$
Compute $\vec{c} = \vec{e} \bmod \mathbf{B}_{pk}$

Output: $\vec{c} \in R \bmod \mathbf{B}_{pk}$

Evaluation: $\mathsf{Eval}(\mathbf{B}_{pk}, C, \Psi)$

Input: $pk = \mathbf{B}_{pk}$, a circuit $C \in \mathcal{C}_{\mathcal{E}}$ and $\Psi = \{\vec{c}_1, ..., \vec{c}_t\}$

Compute $\vec{c} = g(C)(\Psi) \bmod \mathbf{B}_{pk}$
$g(C)$ denotes the generalized circuit

Output: $\vec{c} \in R \bmod \mathbf{B}_{pk}$

Decryption: $\mathsf{Dec}(\vec{c}, sk)$

Input: $\vec{c} \in R \bmod \mathbf{B}_{pk}$

Compute $\vec{m} = (\vec{c} \bmod \mathbf{B}_{sk}) \bmod \mathbf{B}_I$

Output: $\vec{m} \in P$

Figure 4.5: The SWHE based on ideal lattices [50]

As long as the noise parameter is very close to a lattice point, more addition and multiplication can be applied on the ciphertext. After a threshold point, it is not possible to decrypt the ciphertext properly. With each addition, the noise parameter grows linearly, and with each multiplication, it grows exponentially. The noise grows much faster with the multiplication

operations, that is why the number of multiplication operations is limited. To turn the SWHE scheme into a FHE scheme, Gentry used the bootstrapping technique. To apply the bootstrapping technique, the SWHE scheme should have a small decryption circuit depth. The depth of the circuit is squashed to make the SWHE scheme bootstrappable.

2. **Step 2: Squashing.** In order to squash the decryption circuit, a hint about the secret key is placed in the public key. That shifted some of the computations from the decryption stage to the encryption stage. However, this procedure weakened the security of the initial scheme. To treat that the hardness of the secret key recovering using Sparse Subset Sum Problem (SSSP) [60]. The squashing procedure split the decryption algorithm into two phases [50]:

   (a) The encrypter performs an initial computationally intensive preprocessing phase without the secret key.

   (b) The decrypter, using the secret key, performs lightweight computational phase.

   The squashed scheme introduces the changed decryption algorithm and two new algorithms $ExpandCT$, $SplitKey$.

   The $SplitKey$ algorithm puts in the public key, a hint about the secret key.

   The $ExpandCT$ algorithm prepares the ciphertext for the changed decryption algorithm.

3. **Step 3: Bootstrapping.** The scheme which can evaluate its decryption algorithm circuit is called bootstrappable [40]. Bootstrapping is basically "refreshing" procedure to decrease the noise of a ciphertext after performing homomorphic operations on it. To do so, first generate key pairs, (pk_1, sk_1) and (pk_2, sk_2). Then, encrypts m under the first public key pk_1 that $c = E_{pk_1}(m)$, again we reencrypt c under the second public key pk_2 i.e. $E_{pk_2}(c) = E_{pk_2}(E_{pk_1}(m))$, and encrypt the first secret key under the second public key $E_{pk_2}(sk_1)$. Then we transmit $E_{pk_2}(sk_1)$ and $E_{pk_2}(c)$ to the Cloud provider. Since the squashed SWHE scheme can evaluate its own decryption circuit, the cloud provider can apply the decryption circuit to decrypt the noisy ciphertext homomorphically using the encryption of the first secret key sk_1 under pk_2 i.e. $E_{pk_2}(sk_1)$. Therefore $E_{pk_2}(D_{sk_1}(c)) = E_{pk_2}(m)$, which can be decrypted by the customer using sk_2, i.e.,$D_{sk_2}(E_{pk_2}(m)) = m$.

In summary, first the noise was removed from the noisy ciphertext using the homomorphic decryption, and the second encryption offers new ciphertext with small noise as if the ciphertext is just encrypted, see figure 4.6. Then the "fresh" ciphertext can go through more homomorphic operations until it reaches a threshold point again. The bootstrapping technique increases the computational cost, which makes the scheme impractical in real life.

Later, a new KeyGen algorithm was introduced by Gentry [41] with an enhancement to the security of the hardness assumption of SSSP. However, Stehle and Steinfeld [113] presented a more competitive analysis of SSSP.

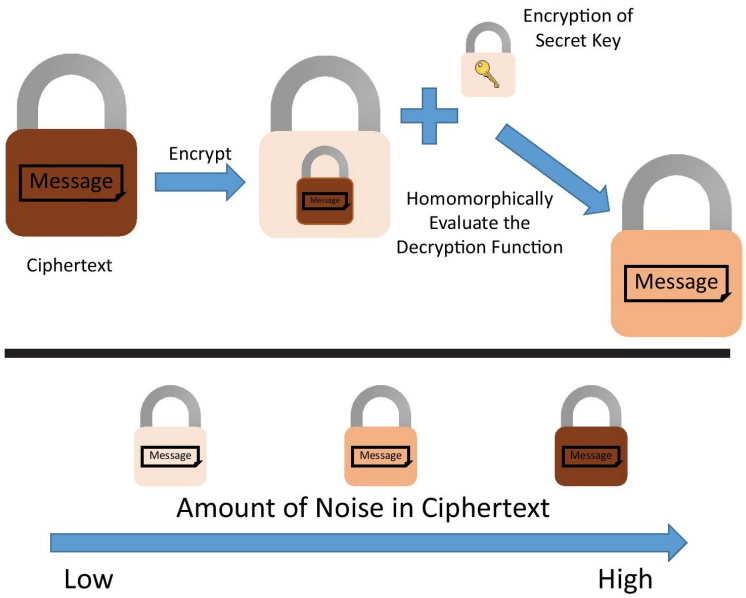

Figure 4.6: Step 3: Bootstrapping. [121]

Also, a new FHE scheme using a smaller key size and generating smaller ciphertext size was introduced by Smart and Vercauterenin [112]. Some other works focused on applying the FHE scheme effectively by optimizing the key generation algorithm [42, 88, 108]. Moreover, to improve the number of homomorphic operations, a new SWHE scheme by Mikuš [81] with bigger plaintext space and a small increase in the key generation algorithm complexity.

4.4.3 FHF schemes Over Integers

In 2010, Dijk et al. [117] presented another fully homomorphic encryption scheme, which uses many of Gentry's, but they replaced Gentry's ideal lattice-based SWHE scheme with a SWHE scheme using integers. The scheme is based on the Approximate-Greatest Common Divisor (AGCD) problem [39]. The AGCD problem seek to find p from the set of $x_i = pq_i + r_i$. The scheme is therefore conceptually simpler than Gentry's ideal lattice scheme but has similar properties concerning homomorphic operations and efficiency. The proposed symmetric SWHE scheme is described as follows:

KeyGen Algorithm The key is an odd integer p, chosen from some interval $p \in [2^{\eta-1}, 2^{\eta})$.

Encryption Algorithm To encrypt a bit $m \in \{0, 1\}$, set the ciphertext as

$$E(m) = m + 2r + pq = c, \tag{4.28}$$

where the q, r are random, and $r < p/2$.

Decryption Algorithm To decrypt the ciphertext:

$$D(c) = (c \pmod p) \pmod 2 = m. \tag{4.29}$$

Evaluation Algorithm Given the (binary) circuit C with t inputs and t ciphertexts c_i apply the (integer) addition and multiplication gates of C to the ciphertexts, performing all the operations over the integers, and return the resulting integer.

Homomorphism over addition:

$$\begin{aligned} c_1 + c_2 &= E(m_1) + E(m_2) = m_1 + 2r_1 + pq_1 + m_2 + 2r_2 + pq_2, \\ &= (m_1 + m_2) + 2(r_1 + r_2) + (q_1 + q_2)p = E(m_1 + m_2). \end{aligned} \tag{4.30}$$

Clearly, $E(m_1 + m_2)$ can be decrypted if the noise $r_1 + r_2 < p/2$,

Homomorphism over Multiplication:

$$\begin{aligned} c_1c_2 &= E(m_1)E(m_2) = (m_1 + 2r_1 + pq_1)(m_2 + 2r_2 + pq_2), \\ &= m_1m_2 + 2(m_1r_2 + m_2r_1 + 2r_1r_2), \\ &+ (pq_1q_2 + 2q_1r_2 + 2q_2r_1 + m_1q_2 + m + 2q + 1)p, \\ &= E(m_1m_2). \end{aligned} \tag{4.31}$$

The encrypted data can be decrypted if $2r_1r_2 + m_1r_2 + m_2r_1 < p/2$.

The scheme runs less homomorphic multiplication operations than addition, because the noise grows exponentially with the multiplication operation.

$$r_1 + r_2 < p/2$$

$$2r_1r_2 + m_1r_2 + m_2r_1 < p/2$$

That truly reduces the number of the performed operations on the ciphertext, plus the decryption of $C(c_1, c_2, \ldots c_t)$ could not be $C(m_1, m_2, \ldots, m_t)$. To transform this SWHE scheme into a FHE scheme, the same squashing and bootstrapping techniques were used by Dijk et al. [117].

The scheme introduced was symmetric homomorphic encryption. To transform it to an asymmetric HE scheme, Dijk et al. [117] compute many "encryptions of zero" $x_i = pq_i + 2r_i$, then sharted many x_i as the public key, where p is private key. The encryption algorithm adds the message to a subset sum of x_i. Without changing the decryption algorithm. Since the scheme now uses different keys to encrypt and decrypt, it has become a public key encryption scheme.

Many other tried to follow-up by decreasing the size of public keys [27, 28, 123]. Without reducing the security of the scheme, others suggested more efficient public key generating (e.g., Rahmaiah et al. [98]) and re-encryption (e.g., Chen et al. [22]). Later, Chen et al. [22] proposed batch FHE over integers [24], which could pack multiple ciphertexts into a single ciphertext.

Some new over integers schemes were also suggested, like a new scheme with integer plaintexts introduced by Ramaiah et al. [97]. Another symmetric FHE scheme without bootstrapping by Aggarwal et al. [3]. Coron et al. [26] introduced a new scale invariant FHE over integers. A new SWHE scheme proposed by Pisa et al. [93] without converting the numbers to bits can compute arithmetic operations on large integers. Also introduced FHE for non-binary message spaces Nuida et al. [87].

4.4.4 LWE-based FHF schemes

In recent years, Learning With Error (LWE) has become the foundation of post-quantum cryptography. Oded Regev presented LWE as an extension of a problem known as "learning from parity with error" problem [99]. The main contribution of Regev was reducing the difficulty of worst-case lattice problems such as SVP problem to LWE problems, i.e., in case an algorithm could solve LWE problems efficiently with respect to time. It can efficiently as well solve SVP problems. The problem of LWE requires to find a secret s 2 Z_q^n, knowing that there is a set of approximate random linear equations on s are given. An example of the input can be

$$\begin{aligned}
14s_1 + 15s_2 + 5s_3 + 2s_4 &\quad 8 \pmod{17}.\\
13s_1 + 14s_2 + 14s_3 + 6s_4 &\quad 16 \pmod{17}.\\
6s_1 + 10s_2 + 13s_3 + 1s_4 &\quad 3 \pmod{17}.\\
10s_1 + 4s_2 + 12s_3 + 16s_4 &\quad 12 \pmod{17}.\\
9s_1 + 5s_2 + 9s_3 + 6s_4 &\quad 9 \pmod{17}.\\
3s_1 + 6s_2 + 4s_3 + 5s_4 &\quad 16 \pmod{17}.\\
&\vdots\\
6s_1 + 7s_2 + 16s_3 + 2s_4 &\quad 3 \pmod{17}.
\end{aligned}$$

Knowing that each of these equations is valid considering small additive error (assume, 1). The goal now is recovering s, and the solution in this example is $s = (0, 13, 9, 11)$. Recovering s can be very easy, if not for the error: considering n equations, we can use Gaussian elimination to find s in a polynomial period of time. However, taking the error into account increases the difficulty of the problem significantly. E.g., linear collections of n equations are taken by the algorithm of Gaussian elimination. Thus, magnifying the error to high levels leads to eliminate all information in the output equations [100].

Notations:

- $\langle a, b \rangle$ refers to the inner product of vector a and vector b.
- $Z[x]/(f(x))$ refers to the ring of all polynomials modulo $f(x)$.
- d $\xleftarrow{\$}$ D means that d is chosen randomly out of the distribution D.
- R_q $Z_q[x]/(f(x))$ denotes the ring of polynomials modulo $f(x)$ with coefficients in Z_q.
- χ refers to the distribution of an error over the ring R_q.

Definition 4.11. (Learn with error problem LWE) Let's have a fixed size of parameter n 1, a modulus q 2, and an 'error' probability distribution χ on Z_q. Let $A_{s,\chi}$ on Z_q^n Z_q be the probability distribution achieved by selecting a vector a 2 Z_q^n uniformly at arbitrary, selecting e 2 Z_q based on χ, and resulting with $(a, \langle a, s \rangle + e)$, and here

additions operations are executed in $\mathbb{Z}_q$, i.e., modulo q. An algorithm solves LWE with modulus q and error distribution χ if, for any $s \in \mathbb{Z}_q^n$, assuming an arbitrary number of individualistic samples out of $A_{s,\chi}$ it results with s (big probability)[100].

Lyubashevsky et al. proposed one more important development for the LWE. [75] introducing the problem of Ring-LWE (RLWE). This problem is an algebraic version of LWE. From practical perspective, RLWE is even more useful for applications with solid security proofs. By this work, it was shown that the RLWE problems can be reduced to worst-case problems on ideal lattices. Those by turn are difficult for the algorithms that are polynomial-time quantum.

Even though LWE and RLWE problems are utilized to build an FHE scheme, RLWE showed more improved efficiency in the LWE-based FHE schemes. Brakerski and Vaikuntanathan [15] made a remarkable improvement on the way to a practical FHE scheme. They presented a novel SWHE scheme that is based on Polynomial-LWE (PLWE). PLWE is, in turn, a simple version of RLWE, and they used two techniques, namely, Gentry's blueprint squashing and bootstrapping aiming at achieving the FHE scheme.

KeyGen Algorithm From the error distribution, sample an item of the ring as a private key, i.e. $s \xleftarrow{\$} \chi$. After that, set a vector of the private key as $\vec{s} = (1, s, s^2, \ldots, s^D) \in R_q^{D+1}$ which will be used in the decryption process.

Encryption Algorithm The message space is the ring of polynomials with binary coefficients $R_2 = \mathbb{Z}_2[x]/\langle x^n+1\rangle$. That is, a message is encoded as a degree n polynomial with coefficients in $\mathbb{Z}_2$. To encrypt, sample $(a, b = as + 2e) \in R_q^2$, where $a \xleftarrow{\$} R_q$ and $e \xleftarrow{\$} \chi$. Compute $c_0 = b + m \in R_q$ and $c_1 = \quad a$

$$\vec{c} = (c_0, c_1) = (as + 2e + m, \quad a), \tag{4.32}$$

and produce the ciphertext $c = (c_0, c_1) \in R_q^2$. Note that only the key s is used by the encryptor to sample (a, b).

Decryption Algorithm Now we have the ciphertext $c = (c_0, c_1)$ and we would like to decrypt it, so we need to compute $c_0 + c_1 s \pmod 2$.

Evaluation Algorithm In the following, we present the way used to add and multiply two elements homomorphically in R_2.

Homomorphism over addition: Given two ciphertexts $c = (c_0, c_1)$ and $c' = (c'_0, c'_1)$

$$\begin{aligned} c_{add} = c + c' &= E(m) + E(m'), \\ &= (c_0, c_1) + (c'_0, c'_1) = (c_0 + c'_0, c_1 + c'_1), \\ &= (as + 2e + m, \quad a) + (a's + 2e' + m', \quad a'), \\ &= ((a + a')s + 2(e + e') + (m + m'), \quad (a + a')), \\ &= E(m + m'). \end{aligned} \tag{4.33}$$

The underlying messages' sum is obtained by applying vector addition on the ciphertext vectors. And it keeps the noise small.

Homomorphism over Multiplication: To deal with the multiplicative homomorphism, we need to be careful. To produce an element which is derived from the two messages production, the c_0 elements of the two ciphertexts must be multiplied. Let's have the two following ciphertexts $c = (c_0, c_1)$ and $c' = (c'_0, c'_1)$. An encryption of their product is computed as shown below:

$$c_0.c'_0 = \quad aa's^2 + (c_0a' + c'_0a)s + 2(2ee' + em' + e'm) + mm'. \tag{4.34}$$

The output ciphertext is $c_{mult} = (c_{mult,0}, c_{mult,1}, c_{mult,2})$, where $c_{mult,2} = c_1c'_1$, $c_{mult,1} = c_0c'_1 + c'_0c_1$, $c_{mult,0} = c_0c'_0$. It is known that $m + 2e = c_0 + c_1s$ and $m' + 2e' = c')0 + c'_1s$, therefore, it is true that

$$(m + 2e).(m' + 2e') = (c_0 + c_1s).(c'_0 + c'_1s). \tag{4.35}$$

It is possible to generate c_{mult} such that

$$(c_0 + c_1s).(c'_0 + c'_1s) = c_{mult,0} + c_{mult,1}s + c_{mult,2}s^2. \tag{4.36}$$

To decrypt a ciphertext that contains 3 element $c = (c_0, c_1, c_2)$, we can perform $c_0 + c_1s + c_2s^2 \pmod 2$.

Aiming at achieving a fully homomorphic scheme Brakerski and Vaikuntanathan [15] used Gentry's "bootstrapping" and "squashing". The basic scheme has a decryption circuit with a high degree that cannot be evaluated homomorphically. Thereby, to reduce the complexity of decryption, they used the posting technique with a sequence of elements, in addition to the public key. These elements mainly used for hiding the secret key as a sparse subset sum. That is, the scheme becomes sufficient for the "bootstrapping". Later, Gentry proposed a BGN-type cryptosystem based on LWE [44]. After that, Brakerski and Vaikuntanathan [16] introduced a different SWHE scheme based on standard LWE problems, where they performed re-linearization processing.

In 2012, Brakerski et al. [14] introduced a leveled-FHE scheme in the absence of the bootstrapping technique. To keep the noise size constant as an optimization they proposed the batching, which use the "modulus switching" technique.

Definition 4.12. Leveled-FHE is able to evaluate homomorphic operations only when there is a predefined circuit depth level.

Later, the necessity of modulus switching was removed by Brakerski [13]. The new FHE scheme is based on the difficulty of GapSVP [91].

Definition 4.13. Given a vector of length d for a B lattice basis, the problem of GapSVP decides roughly whether there is a vector shorter than the given vector. The output is Boolean: true or false.

Later on, the Brakerski scheme was optimized by Fan and Vercauteren, where they used the RLWE problem as the based assumption [36].

The first identity-based FHE scheme was introduced by Gentry et al. [45]. The new scheme managed to preserve the format of the evaluated ciphertext under homomorphic operations, only by keeping some parameters small. Which allows the identity which has the public parameters to apply the homomorphic evaluation on ciphertext, while in the previous schemes, which uses the bootstrapping technique, an encrypted version of the secret key is transmitted to the cloud where a homomorphically evaluation can be conducted on the ciphertext for the bootstrapping. Then, Brakerski and Vaikuntanathan created an FHE scheme secure based on a polynomial LWE assumption as well [17]. Recently, more LWE-based FHE schemes were proposed in [23, 119, 126, 127], then Cheon et al. presented a novel and efficient SWHE scheme. The proposed scheme uses as a basis the polynomial approximate common divisor problem [25].

4.4.5 NTRU-like FHE schemes

For the previous FHE schemes, the data are required to be encrypted by the same encryption key. Consequently, all the homomorphic computations can be applied only to the encrypted data of a single customer. However, there are several situations where different customers need to evaluate a joint function of encrypted data from the customers. For instance, if a customer wants to generate some statistical information about his data, then he needs to use a multi-key FHE scheme. Multi-key FHE can perform operations on ciphertexts encrypted with different independent keys so that each customer can use his encryption key to encrypt his data. Then the cloud provider executes homomorphic operations on the ciphertexts. The customers need to interact only when it comes to obtain a "joint secret key". One of the earliest attempts to obtain a practical multi-key FHE scheme, which was introduced by López-Alt et al. [74], who was building an FHE scheme from NTRU encryption scheme. NTRU encryption scheme is among the early schemes that used lattice problems as a basis. The scheme was designed by Hoffstein et al. [59]. Stehlé and Steinfeld [114] worked on improving the security aspect of the scheme. Particularly, they were able, over ideal lattices, to reduce the scheme's security towards standard worst-case problems. Its high-performance capabilities again attract researchers interest. López-Alt et al. [74] noticed that the NTRU encryption exhibits fully homomorphic properties, and they designed their NTRU-like encryption scheme as follows:

Notations:

Ring $R = \mathbb{Z}[x]/\langle x^n + 1 \rangle$, where n is a power of two,.

q is odd prime number, a B-bounded distribution χ over R, for $B << q$. By "B-bounded", it means that the size of the coefficients of a polynomial sampled out of χ is definitely less than B.

Define $R_q = R/qR$.

KeyGen Algorithm Sample "small" polynomials f' and g from a distribution χ and set $f = 2f' + 1$ so that $f \quad 1 \pmod 2$. Compute the inverse f^{-1} of f in

R_q. The public key $pk = h = 2gf^{-1} \in R_q$ and the secret key $sk = f \in R$.

Encryption Algorithm To encrypt a bit $m \in \{0, 1\}$, samples s and e from the same distribution χ, and outputs the ciphertext:

$$E(m) = hs + 2e + m = c. \tag{4.37}$$

Decryption Algorithm The decryption algorithm computes $\mu = fc \in R_q$ then

$$D(c) = \mu \pmod 2 = m. \tag{4.38}$$

Note that $fc = 2gs + 2fe + fm \in R_q$ and $f \equiv 1 \pmod 2$.

For describing the multi-key homomorphic properties of the scheme. Let $c_1 = h_1 s_1 + 2e_1 + m_1$ and $c_2 = h_2 s_2 + 2e_2 + m_2$ be ciphertexts using two variant keys $h_1 = 2g_1 f_1^{-1}$ and $h_2 = 2g_2 f_2^{-1}$, respectively. The multi-key homomorphism properties using joint secret key $f_1 f_2$ for two parties computation are:

Multi-key Homomorphism over addition: We assume that $c_{add} = c_1 + c_2$ is the encryption of $m_1 + m_2$ under the joint secret key $f_1 f_2$.

$$\begin{aligned} D(c_{add}) &= f_1 f_2 (c_1 + c_2), \\ &= 2(f_1 f_2 e_1 + f_1 f_2 e_2 + f_2 g_1 s_1 + f_1 g_2 s_2) + f_1 f_2 (m_1 + m_2), \\ &= 2e_{add} + f_1 f_2 (m_1 + m_2), \end{aligned} \tag{4.39}$$

for a slightly larger noise element e_{add}.

Multi-key Homomorphism over Multiplication: We claim that $c_{mult} = c_1 c_2$ is the encryption of $m_1 m_2$ under the joint secret key $f_1 f_2$.

$$\begin{aligned} D(c_{mult}) &= f_1 f_2 (c_1 c_2), \\ &= 2(2g_1 g_2 s_1 s_2 + g_1 s_1 f_2 (2e_2 + m_2) + g_2 s_2 f_1 (2e_1 + m_1), \\ &+ f_1 f_2 (e_1 m_2 + e_2 m_1 + 2e_1 e_2)) + f_1 f_2 (m_1 m_2), \\ &= 2e_{mult} + f_1 f_2 (m_1 m_2), \end{aligned} \tag{4.40}$$

for a slightly larger noise element e_{mult}

As noticed, the aforementioned scheme is in fact a SWHE scheme. To turn into bootstrappable, López-Alt et al. [13, 16] used modulus reduction technique. The new scheme had the capability to work with a limited number of public keys. Therefore, it was a leveled-FHE scheme, because limited number of customers can be used in homomorphic operations. However, the complexity of circuits used in homomorphic operations was not effected. The López-Alt et al. [13, 16] scheme was based on the Decisional Small Polynomial Ratio (DSPR) and the RLWE problems. Later, Bos et al.[12] removeed the DSRP from López-Alt et al. [13, 16] scheme, and also showed how to use larger inputs via Chinese Remainder theorem (CRT) . Recently, many hardware improvements of the NTRU-like FHE schemes implementation were proposed in [33, 71].

CHAPTER 5

Implementation of Homomorphic Encryption

The ultimate purpose of working with HE schemes is to have a practical FHE scheme. PHE schemes and SWHE schemes were steps in the direction of that goal. Nevertheless, they have limited applications, PHE has made it possible for several cloud services and systems to obtain advantages [66]. In the following, we introduce a few examples:

> CryptDB [94] is a database privacy layer that provides the ability to execute SQL queries on encrypted data. CryptDB allows the server to execute queries requested by users. The server runs the queries over encrypted data and then the result is sent back to the customer. The customer then can decrypt the result. The machine of the customer does not process any query. The Paillier cryptosystem [90] was implemented, which offers basic addition operations over integers.
>
> Helios 2.0 [2] is an open-audit voting system based on cloud where homomorphic encryption is used to count the votes. This allows to count and submit all the votes in an encrypted manner. The result will be decrypted only after casting and adding the final votes to the tallies. The PHE scheme has utilized a version of ElGamal [35].
>
> Porticor (2014) [95] is a commercial implementation of homomorphic encryption. It was one of the first implementations that offers a secure cloud key management platform based on PHE. Another firma called Intuit has acquired the company Porticor.

On the other hand, most of the SWHE schemes proposed later to Gentry's proposal are actually part of some FHE schemes. Therefore, it is difficult to split the implementations of the SWHE and FHE schemes. In this section, our objective is to review some of these implementations.

Implementing a cryptographic scheme comes as a second step after the designing phase. After implementing the scheme, it can be applied to a real-world service, where it can be used to assess the performance of the designed scheme [1]. Even though the capability and the performance of the implementations of the new proposed FHE schemes have improved significantly, the high cost of the FHE implementations is a major pitfall that prevents applying FHE in real-world services without disturbing

users. Gentry et al. [42] were the first to implement the FHE scheme by optimizing and simplifying the squashing process. They demonstrated four levels of security, namely, toy, small, medium, and large. These levels have a lattice dimension of 2^{15}. However, the implementation performance was not practical in the real life. Considering the setting of the large parameter, generating a key pair took more than 2 hours. The public key's size was of 2.25 GB. Moreover, the bootstrapping was done in 31 minutes.

Later, Coron et al. [27] implemented an integer version of Van Dijk et al. [117]. The key generation took 43 minutes; the public key's size was 802 MB. After that, Coron et al. [28] using a compression technique, improved the size of the public key reaching 10 MB, in addition, the time of both the key generation and the bootstrapping was improved to 10 minutes and 11 minutes, respectively. Some of the proposed FHE implementations, which are evaluated over circuits with random depth is shown in Table 5.1.

Table 5.1: Fully implemented FHE schemes [1]

Schemes Information			Running Time			
Implemented Scheme	Base Scheme	PK size	KeyGen	Enc	Dec	Recrypt
Gentry and Halevi[42]	Gentry[46]	2.25 GB	2.2 h	3 min	0.66 s	31 min
Coron et. al.[27]	Van Dijk et. al.[117]	802 MB	43 min	2 min 57 s	0.05 s	14 min 33 s
Coron et. al. with compressed PK[28]	Van Dijk et. al.[117]	10.3 MB	10 min	7 min 15 s	0.05 s	11 min 34 s
Coron et. al. Leveled [28]	Van Dijk et. al.[117]	18 MB	6 min 18 s	3.4 s	0.00 s	2 h 27 min

Other researchers worked on implementing leveled-FHE schemes for circuits with small depth and given runtime for both composed and isolated multiplication and addition. The comparison of these small-depth FHE implementations is presented in Table 5.2.

Table 5.2: FHE implementations for circuits with Low-depth [1]

Schemes Information		Running Time			
Implemented Scheme	Base Scheme	Enc	Dec	Mult	Add
Naehrig et. al.[86]	Brakerski and Vaikuntanathan[15]	0.756s	0.057s	1.59s	0.004s
Bos et al.[12]	López-Alt et. al.[74]	0.027s	0.005s	0.031s	0.024 ms
Lepoint and Naehrig[70]	López-Alt et. al.[74]	0.016s	0.015s	0.018s	0.7 ms
Lepoint and Naehrig[70]	Brakerski and Vaikuntanathan[15]	0.034s	0.016s	59s	1.4 ms
Rohloff and Cousins[106]	López-Alt et. al.[74]	0.012s	0.00336s	0.001s	0.56 ms

For the first time, an implementation of BGV scheme was done by Gentry et al. [43]

[14], which is a leveled-FHE scheme without bootstrapping. The implementation was used to assess the AES circuit in homomorphic manner. This evaluation was enough considering a real-life application but not practical since it took 48 hours to evaluate AES circuit. To homomorphically evaluate AES, first, a customer encrypt the key of AES with FHE $FHE(K)$ and sends it. After that, the customer uploads the encrypted (with AES only) data, $AES_K(m)$. To evaluate the data homomorphically, first, the cloud provider computes $FHE(AES_K(m))$, second, AES is decrypted homomorphically to get $FHE(m)$. Then, the homomorphic operations on the encrypted (with FHE) data can be done by the cloud.

Ducas and Micciancio presented recently another significant implementation, which is "Fastest Homomorphic Encryption in the West" (FHEW) [34]. It noticeably enhances the time required for bootstrapping the ciphertext professing homomorphic assessment of a NAND gate in ¡ 1 sec.

Despite the fact that all previously mentioned implementations are publicly published as research papers, there are just a small number available for the research community. The most important and widely used one is HElib by Halevi and Shoup [53]. HElib implements the BGV scheme [13] with ciphertext packing techniques, in addition to novel optimizations. Halevi and Shoup do the design and implementation of HElib in [52] and the algorithms they used in [54]. HElib was implemented based on GPL-licensed C++ library, and it supports bootstrapping [55]. Gentry used HElib to implement the homomorphic evaluation of AES [43]. However, HElib is not easy to use because it is designed using low-level programming. The low-level programming interacts with components of the computer and the hardware constraints without the need to use the functions and commands of a programming language. There are other notable open sources HE libraries such as SEAL [21], NFLlib [4], libScarab [92], and PALISADE [105]. These libraries implement essentially low-level homomorphic operations such as arithmetic addition, subtraction, multiplication, and comparison. These operations are useful in illustrating that the libraries work on small problems, but they are less beneficial when making real-world systems that need to be deployed in the cloud. To build a system using these libraries, it will take a tremendous effort from the developers to figure out how the capabilities of the libraries can fit their requirement and to figure out what the limitations of the libraries are [56].

As an illustration of how to use HELib, we wrote a simple program based on samples provided with the library, to show how to install HELib (See Appendix B) and perform basic computations, e.g., adding and multiplying of two integers (See Appendix A). The initialization part of the program will create the public key and the private key, after that, we will enter the two integers, which will be encrypted by the public key. The Program will perform one homomorphic addition, and one homomorphic multiplication over the encrypted integers then will decrypt the results and print them out. The running time average of the program is 38.444 s, and the output file size was 37.1 MB Although this is a simple example, HElib allows us to do more advanced things that are needed for practical applications. Most of those can be found in the samples that come with HElib,

CHAPTER 6

Conclusion and Future Work

Unfortunately, there are still some open security issues need to be proven about the FHE schemes. As we have reviewed, Gentry's blueprint bootstrapping technique is utilized by most of the FHE schemes. Which means that in order to bootstrap the noisy ciphertexts, an encrypted copy of the secret key is sent to the cloud provider, in that case, an eavesdropper can get the encrypted secret key. Although it is proven that some SWHE schemes are semantically secure, the semantic security of an unlimited FHE is not proven yet regarding functions, therefore, there is no guarantees to protect the private key from being determined from the encrypted public key by adversary. Therefore, FHE schemes need to be examined to prove that it is secure enough. Another open problem is to design an unlimited FHE scheme that allows unlimited operations without bootstrapping. Although the bootstrapping can be essential to reduce the noise in the assessed ciphertexts, it increases the computational cost. Also, designing a noise-free FHE scheme remains as one of the research challenges. Liu [72] presented a noise-free FHE and Yagisawa [122] presented a FHE without bootstrapping, but both schemes were reported as insecure by Wang [120].

As we have presented, the security issues are a significant challenge for Cloud Computing development. In order to protect the privacy of the customer's data, he needs to encrypt his data before uploading it to the cloud. In case he needs to run some operations on the encrypted data, he has to download the data on his machine and decrypt it, then do the computation then encrypt it again and upload it to the cloud. Otherwise, the customer needs to share his key with the cloud service provider to perform the computations for him which is not secure. In cryptography, Homomorphic encryption is considered a very promising concept. It can significantly promote the security of Cloud Computing because HE schemes permit performing computations on encrypted data with no need for sharing the private key. In this thesis, we have defined Cloud Computing and discussed its security issues. Then we described the use of traditional encryption as a possible threat countermeasure, and the challenges facing its implementation. We reviewed some of the Partially Homomorphic Encryption (PHE) schemes like RSA and Paillier schemes which were not sufficient to secure Cloud Computing because these schemes allow to perform only one computation on the encrypted data. The fully Homomorphic encryption is the genuine solution to secure the customer data in the Cloud Computing because it permits to execute random operations on encrypted data with no need to decrypt it. However, as we explored in chapter 5, the FHE schemes suffer from unsatisfactory speed performance and huge

storage size, it can be improved to be practical for real-life applications. Still, the adoption of HE schemes can start with solutions for some specific problems using the capabilities of the available HE libraries.

REFERENCES

[1] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, A survey on homomorphic encryption schemes: Theory and Implementation, arXiv preprint arXiv:1704.03578, 2017.

[2] B. Adida, O. De Marneffe, O. Pereira, J.-J. Quisquater, et al., Electing a university president using open-audit voting: Analysis of real-world use of helios, EVT/WOTE, 9(10), 2009.

[3] N. Aggarwal, C. Gupta, and I. Sharma, Fully homomorphic symmetric scheme without bootstrapping, in *Cloud Computing and Internet of Things (CCIOT), 2014 International Conference on*, pp. 14–17, IEEE, 2014.

[4] C. Aguilar-Melchor, J. Barrier, S. Guelton, A. Guinet, M.-O. Killijian, and T. Lepoint, NFLlib: NTT-based fast lattice library, in *Cryptographers' Track at the RSA Conference*, pp. 341–356, Springer, 2016.

[5] M. Ajtai, Generating hard instances of lattice problems, in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 99–108, ACM, 1996.

[6] G. Anthes, Security in the cloud, Communications of the ACM, 53(11), pp. 16–18, 2010.

[7] S. as a service working group, Defined categories of service 2011, Cloud Security Alliance, 2011.

[8] S. as a service working group, Category 8 // encryption, Cloud Security Alliance, 2012.

[9] J. Benaloh, Dense probabilistic encryption, in *Proceedings of the workshop on selected areas of cryptography*, pp. 120–128, 1994.

[10] B. Blog, Cloud definition – what is a cloud?, `https://www.boxcryptor.com/en/blog/post/what-is-the-cloud-a-beginner-s-guide/`, March 2017.

[11] D. Boneh, E.-J. Goh, and K. Nissim, Evaluating 2-DNF formulas on ciphertexts., in *TCC*, volume 3378, pp. 325–341, Springer, 2005.

[12] J. W. Bos, K. E. Lauter, J. Loftus, and M. Naehrig, Improved security for a ring-based fully homomorphic encryption scheme., in *IMA Int. Conf.*, pp. 45–64, Springer, 2013.

[13] Z. Brakerski, Fully homomorphic encryption without modulus switching from classical GapSVP., in *CRYPTO*, volume 7417, pp. 868–886, Springer, 2012.

[14] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, (leveled) fully homomorphic encryption without bootstrapping, ACM Transactions on Computation Theory (TOCT), 6(3), p. 13, 2014.

[15] Z. Brakerski and V. Vaikuntanathan, Fully homomorphic encryption from ring-LWE and security for key dependent messages, in *Annual cryptology conference*, pp. 505–524, Springer, 2011.

[16] Z. Brakerski and V. Vaikuntanathan, Efficient fully homomorphic encryption from (standard) LWE, SIAM Journal on Computing, 43(2), pp. 831–871, 2014.

[17] Z. Brakerski and V. Vaikuntanathan, Lattice-based FHE as secure as PKE, in *Proceedings of the 5th conference on Innovations in theoretical computer science*, pp. 1–12, ACM, 2014.

[18] G. Brunette, R. Mogull, et al., Security guidance for critical areas of focus in cloud computing v2. 1, Cloud Security Alliance, pp. 1–76, 2009.

[19] A. A. Cardenas, P. K. Manadhata, and S. P. Rajan, Big data analytics for security, IEEE Security & Privacy, 11(6), pp. 74–76, 2013.

[20] D. Catteddu, G. Hogben, et al., Cloud computing information assurance framework, European Network and Information Security Agency (ENISA), 13, p. 14, 2009.

[21] H. Chen, K. Laine, and R. Player, Simple encrypted arithmetic library-SEALv2. 1, in *International Conference on Financial Cryptography and Data Security*, pp. 3–18, Springer, 2017.

[22] L. Chen, H. Ben, and J. Huang, An encryption depth optimization scheme for fully homomorphic encryption, in *Identification, Information and Knowledge in the Internet of Things (IIKI), 2014 International Conference on*, pp. 137–141, IEEE, 2014.

[23] Z. Chen, J. Wang, Z. Zhang, and S. Xinxia, A fully homomorphic encryption scheme with better key size, China Communications, 11(9), pp. 82–92, 2014.

[24] J. H. Cheon, J.-S. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, and A. Yun, Batch fully homomorphic encryption over the integers, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 315–335, Springer, 2013.

[25] J. H. Cheon, H. Hong, M. S. Lee, and H. Ryu, The polynomial approximate common divisor problem and its application to the fully homomorphic encryption, Information Sciences, 326, pp. 41–58, 2016.

[26] J.-S. Coron, T. Lepoint, and M. Tibouchi, Scale-invariant fully homomorphic encryption over the integers., in *Public Key Cryptography*, volume 8383, pp. 311–328, 2014.

[27] J.-S. Coron, A. Mandal, D. Naccache, and M. Tibouchi, Fully homomorphic encryption over the integers with shorter public keys., in *Crypto*, volume 6841, pp. 487–504, Springer, 2011.

[28] J.-S. Coron, D. Naccache, and M. Tibouchi, Public key compression and modulus switching for fully homomorphic encryption over the integers., in *EUROCRYPT*, volume 7237, pp. 446–464, Springer, 2012.

[29] D. Cuschieri, *Cloud Encryption and Key Management Considerations*, Master's thesis, Royal Holloway, University of London, Egham, Surrey TW20 0EX, England, 2014.

[30] I. Damgard and M. Jurik, A generalisation, a simplification and some applications of paillier's probabilistic public-key system, in *Public Key Cryptography*, volume 1992, pp. 119–136, Springer, 2001.

[31] W. Diffie and M. Hellman, New directions in cryptography, IEEE transactions on Information Theory, 22(6), pp. 644–654, 1976.

[32] J. Domingo-Ferrer, A provably secure additive and multiplicative privacy homomorphism, Information Security, pp. 471–483, 2002.

[33] Y. Doröz, E. Öztürk, E. Savaş, and B. Sunar, Accelerating LTV based homomorphic encryption in reconfigurable hardware, in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 185–204, Springer, 2015.

[34] L. Ducas and D. Micciancio, FHEW: Bootstrapping homomorphic encryption in less than a second., EUROCRYPT (1), 9056, pp. 617–640, 2015.

[35] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE transactions on information theory, 31(4), pp. 469–472, 1985.

[36] J. Fan and F. Vercauteren, Somewhat practical fully homomorphic encryption., IACR Cryptology ePrint Archive, 2012, p. 144, 2012.

[37] T. S. Fun and A. Samsudin, A survey of homomorphic encryption for outsourced big data computation., TIIS, 10(8), pp. 3826–3851, 2016.

[38] S. D. Galbraith, Elliptic curve paillier schemes, Journal of Cryptology, 15(2), pp. 129–138, 2002.

[39] S. D. Galbraith, S. W. Gebregiyorgis, and S. Murphy, Algorithms for the approximate common divisor problem, LMS Journal of Computation and Mathematics, 19(A), pp. 58–72, 2016.

[40] C. Gentry, *A fully homomorphic encryption scheme*, Ph.D. thesis, 2009.

[41] C. Gentry, Toward basing fully homomorphic encryption on worst-case hardness., in *CRYPTO*, volume 6223, pp. 116–137, Springer, 2010.

[42] C. Gentry and S. Halevi, Implementing Gentry's fully-homomorphic encryption scheme., in *EUROCRYPT*, volume 6632, pp. 129–148, Springer, 2011.

[43] C. Gentry, S. Halevi, and N. P. Smart, Homomorphic evaluation of the AES circuit, in *Advances in Cryptology–CRYPTO 2012*, pp. 850–867, Springer, 2012.

[44] C. Gentry, S. Halevi, and V. Vaikuntanathan, A simple BGN-type cryptosystem from LWE, Advances in Cryptology–EUROCRYPT 2010, pp. 506–522, 2010.

[45] C. Gentry, A. Sahai, and B. Waters, Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based, in *Advances in Cryptology–CRYPTO 2013*, pp. 75–92, Springer, 2013.

[46] C. Gentry et al., Fully homomorphic encryption using ideal lattices., in *STOC*, volume 9, pp. 169–178, 2009.

[47] K. Gjøsteen, Subgroup membership problems and public key cryptosystems, 2004.

[48] O. Goldreich, S. Goldwasser, and S. Halevi, Public-key cryptosystems from lattice reduction problems, in *Advances in Cryptology-CRYPTO'97: 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 1997. Proceedings*, p. 112, Springer, 1997.

[49] S. Goldwasser and S. Micali, Probabilistic encryption & how to play mental poker keeping secret all partial information, in *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pp. 365–377, ACM, 1982.

[50] S. Goluch, *The development of homomorphic cryptography: from RSA to Gentry's privacy homomorphism*, na, 2011.

[51] D. Grigoriev and I. Ponomarenko, Homomorphic public-key cryptosystems and encrypting boolean circuits, Applicable Algebra in Engineering, Communication and Computing, 17(3), pp. 239–255, 2006.

[52] S. Halevi and V. Shoup, Design and implementation of a homomorphic-encryption library, IBM Research (Manuscript), 6, pp. 12–15, 2013.

[53] S. Halevi and V. Shoup, An implementation of homomorphic encryption, GitHubRepository, https://github.com/shaih/HElib, 2013.

[54] S. Halevi and V. Shoup, Algorithms in HElib, in *International Cryptology Conference*, pp. 554–571, Springer, 2014.

[55] S. Halevi and V. Shoup, Bootstrapping for HElib, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 641–670, Springer, 2015.

[56] R. A. Hallman, M. H. Diallo, M. A. August, and C. T. Graves, Homomorphic encryption for secure computation on big data, 2018.

[57] C. Harvey, Cloud computing services, `https://www.datamation.com/cloud-computing/what-is-cloud-service.html`, April 2017.

[58] J. Hermans and M. Chung, KPMG's 2010 cloud computing survey, Technical report, Technical report, KPMG, 2010.

[59] J. Hoffstein, J. Pipher, and J. H. Silverman, NTRU: A ring-based public key cryptosystem, in *International Algorithmic Number Theory Symposium*, pp. 267–288, Springer, 1998.

[60] J. Hoffstein, J. C. Pipher, J. H. Silverman, and J. H. Silverman, *An introduction to mathematical cryptography*, volume 1, Springer, 2008.

[61] D. Hubbard, M. Sutton, et al., Top threats to cloud computing v1. 0, Cloud Security Alliance, 2010.

[62] J. D. i Ferrer, A new privacy homomorphism and applications, Information Processing Letters, 60(5), pp. 277–282, 1996.

[63] Y. Ishai and A. Paskin, Evaluating branching programs on encrypted data, in *TCC*, volume 4392, pp. 575–594, Springer, 2007.

[64] A. Kawachi, K. Tanaka, and K. Xagawa, Multi-bit cryptosystems based on lattice problems, in *International Workshop on Public Key Cryptography*, pp. 315–329, Springer, 2007.

[65] B. Kepes, Understanding the cloud computing stack: Saas, Paas, Iaas, Diversity Limited, pp. 1–17, 2011.

[66] R. Ko and R. Choo, *The Cloud Security Ecosystem: Technical, Legal, Business and Management Issues*, Syngress, 2015.

[67] K. Krishnan, *Data warehousing in the age of big data*, Newnes, 2013.

[68] J. M. Kukucka, *An investigation of the theory and applications of homomorphic cryptography*, Ph.D. thesis, Rensselaer Polytechnic Institute, 2013.

[69] A. K. Lenstra, H. W. Lenstra, and L. Lovász, Factoring polynomials with rational coefficients, Mathematische Annalen, 261(4), pp. 515–534, 1982.

[70] T. Lepoint and M. Naehrig, A comparison of the homomorphic encryption schemes FV and YASHE, in *International Conference on Cryptology in Africa*, pp. 318–335, Springer, 2014.

[71] B. Liu and H. Wu, Efficient architecture and implementation for NTRUEncrypt system, in *Circuits and Systems (MWSCAS), 2015 IEEE 58th International Midwest Symposium on*, pp. 1–4, IEEE, 2015.

[72] D. Liu, Practical fully homomorphic encryption without noise reduction., IACR Cryptology ePrint Archive, 2015, p. 468, 2015.

[73] F. Lombardi and R. Di Pietro, Secure virtualization for cloud computing, Journal of Network and Computer Applications, 34(4), pp. 1113–1122, 2011.

[74] A. López-Alt, E. Tromer, and V. Vaikuntanathan, On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption, in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pp. 1219–1234, ACM, 2012.

[75] V. Lyubashevsky, C. Peikert, and O. Regev, On ideal lattices and learning with errors over rings, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 1–23, Springer, 2010.

[76] T. Mather, S. Kumaraswamy, and S. Latif, *Cloud security and privacy: an enterprise perspective on risks and compliance*, " O'Reilly Media, Inc.", 2009.

[77] C. A. Melchor, P. Gaborit, and J. Herranz, Additively homomorphic encryption with d-operand multiplications., in *CRYPTO*, volume 6223, pp. 138–154, Springer, 2010.

[78] P. Mell, Big data technology and implications for security research, in *Proceedings of the 2012 ACM Workshop on Building analysis datasets and gathering experience returns for security*, pp. 15–16, ACM, 2012.

[79] P. Mell, T. Grance, et al., The NIST definition of cloud computing, 2011.

[80] D. Micciancio and S. Goldwasser, *Complexity of lattice problems: a cryptographic perspective*, volume 671, Springer Science & Business Media, 2002.

[81] M. Mikuš, Experiments with the plaintext space in Gentry's somewhat homomorphic scheme, Tatra Mountains Mathematical Publications, 53(1), pp. 147–154, 2012.

[82] H. Minkowski, *Geometrie der zahlen*, volume 40, Ripol Classic, 1910.

[83] A. Monaco, A view inside the cloud, theinstitute. ieee. org, 2012.

[84] G. L. Mullen and P. J.-S. Shiue, *Finite fields: theory, applications, and algorithms*, volume 168, American Mathematical Soc., 1994.

[85] D. Naccache and J. Stern, A new public key cryptosystem based on higher residues, in *Proceedings of the 5th ACM conference on Computer and communications security*, pp. 59–66, ACM, 1998.

[86] M. Naehrig, K. Lauter, and V. Vaikuntanathan, Can homomorphic encryption be practical?, in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pp. 113–124, ACM, 2011.

[87] K. Nuida and K. Kurosawa, (batch) fully homomorphic encryption over integers for non-binary message spaces, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 537–555, Springer, 2015.

[88] N. Ogura, G. Yamamoto, T. Kobayashi, and S. Uchiyama, An improvement of key generation algorithm for Gentry's homomorphic encryption scheme, in *International Workshop on Security*, pp. 70–83, Springer, 2010.

[89] T. Okamoto and S. Uchiyama, A new public-key cryptosystem as secure as factoring, Advances in Cryptology—EUROCRYPT'98, pp. 308–318, 1998.

[90] P. Paillier et al., Public-key cryptosystems based on composite degree residuosity classes, in *Eurocrypt*, volume 99, pp. 223–238, Springer, 1999.

[91] C. Peikert et al., A decade of lattice cryptography, Foundations and Trends® in Theoretical Computer Science, 10(4), pp. 283–424, 2016.

[92] H. Perl, M. Brenner, and M. Smith, An implementation of the fully homomorphic smart-vercauteren crypto-system, in *Proceedings of Conference on Computer and Communications Security*, pp. 837–840, 2015.

[93] P. S. Pisa, M. Abdalla, and O. C. M. B. Duarte, Somewhat homomorphic encryption scheme for arithmetic operations on large integers, in *Global Information Infrastructure and Networking Symposium (GIIS), 2012*, pp. 1–8, IEEE, 2012.

[94] R. A. Popa, N. Zeldovich, and H. Balakrishnan, CryptDB: A practical encrypted relational BMS, 2011.

[95] Porticor, `http://www.porticor.com`, 2014.

[96] Priti, Hybrid cloud hosting and its market value, `https://www.mytechlogy.com/IT-blogs/11965/hybrid-cloud-hosting-and-its-market-value/`, May 2016.

[97] Y. G. Ramaiah and G. V. Kumari, Efficient public key homomorphic encryption over integer plaintexts, in *Information Security and Intelligence Control (ISIC), 2012 International Conference on*, pp. 123–128, IEEE, 2012.

[98] Y. G. Ramaiah and G. V. Kumari, Towards practical homomorphic encryption with efficient public key generation, International Journal on Network Security, 3(4), p. 10, 2012.

[99] O. Regev, On lattices, learning with errors, random linear codes, and cryptography, Journal of the ACM (JACM), 56(6), p. 34, 2009.

[100] O. Regev, The learning with errors problem, Invited survey in CCC, p. 15, 2010.

[101] K. Ren, C. Wang, and Q. Wang, Security challenges for the public cloud, IEEE Internet Computing, 16(1), pp. 69–73, 2012.

[102] J. W. Rittinghouse and J. F. Ransome, *Cloud computing: implementation, management, and security*, CRC press, 2016.

[103] R. L. Rivest, L. Adleman, and M. L. Dertouzos, On data banks and privacy homomorphisms, Foundations of secure computation, 4(11), pp. 169–180, 1978.

[104] R. L. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM, 21(2), pp. 120–126, 1978.

[105] K. Rohloff, The PALISADE lattice cryptography library, `https://git.njit.edu/palisade/PALISADE`, 2017.

[106] K. Rohloff and D. B. Cousins, A scalable implementation of fully homomorphic encryption built on NTRU, in *International Conference on Financial Cryptography and Data Security*, pp. 221–234, Springer, 2014.

[107] T. Sander, A. Young, and M. Yung, Non-interactive cryptocomputing for NC1, in *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pp. 554–, IEEE Computer Society, Washington, DC, USA, 1999, ISBN 0-7695-0409-4.

[108] P. Scholl and N. P. Smart, Improved key generation for Gentry's fully homomorphic encryption scheme, in *IMA International Conference on Cryptography and Coding*, pp. 10–22, Springer, 2011.

[109] J. Sen, Homomorphic encryption: theory & applications, arXiv preprint arXiv:1305.5886, 2013.

[110] I. Sharma, Fully homomorphic encryption scheme with symmetric keys, arXiv preprint arXiv:1310.2452, 2013.

[111] S. Singh, Y.-S. Jeong, and J. H. Park, A survey on cloud computing security: Issues, threats, and solutions, Journal of Network and Computer Applications, 75, pp. 200–222, 2016.

[112] N. P. Smart and F. Vercauteren, Fully homomorphic encryption with relatively small key and ciphertext sizes., in *Public Key Cryptography*, volume 6056, pp. 420–443, Springer, 2010.

[113] D. Stehlé and R. Steinfeld, Faster fully homomorphic encryption, Advances in Cryptology-ASIACRYPT 2010, pp. 377–394, 2010.

[114] D. Stehlé and R. Steinfeld, Making NTRU as secure as worst-case problems over ideal lattices., in *Eurocrypt*, volume 6632, pp. 27–47, Springer, 2011.

[115] B. Sumitra, C. Pethuru, and M. Misbahuddin, A survey of cloud authentication attacks and solution approaches, International journal of innovative research in computer and communication engineering, 2(10), 2014.

[116] Z. Tari, X. Yi, U. S. Premarathne, P. Bertok, and I. Khalil, Security and privacy in cloud computing: vision, trends, and challenges, IEEE Cloud Computing, 2(2), pp. 30–38, 2015.

[117] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, Fully homomorphic encryption over the integers, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 24–43, Springer, 2010.

[118] A. T. Velte, T. J. Velte, R. C. Elsenpeter, and R. C. Elsenpeter, *Cloud computing: a practical approach*, McGraw-Hill New York, 2010.

[119] F. Wang, K. Wang, and B. Li, LWE-based fhe with better parameters, in *International Workshop on Security*, pp. 175–192, Springer, 2015.

[120] Y. Wang, Notes on two fully homomorphic encryption schemes without bootstrapping., IACR Cryptology ePrint Archive, 2015, p. 519, 2015.

[121] D. J. Wu, Fully homomorphic encryption: Cryptography's holy grail, XRDS: Crossroads, The ACM Magazine for Students, 21(3), pp. 24–29, 2015.

[122] M. Yagisawa, Fully homomorphic encryption without bootstrapping., IACR Cryptology ePrint Archive, 2015, p. 474, 2015.

[123] H.-M. Yang, Q. Xia, X.-f. Wang, and D.-h. Tang, A new somewhat homomorphic encryption scheme over integers, in *Computer Distributed Control and Intelligent Environmental Monitoring (CDCIEM), 2012 International Conference on*, pp. 61–64, IEEE, 2012.

[124] A. C. Yao, Protocols for secure computations, in *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pp. 160–164, IEEE, 1982.

[125] X. Yi, R. Paulet, and E. Bertino, *Homomorphic encryption and applications*, volume 3, Springer, 2014.

[126] X. Zhang, C. Xu, C. Jin, R. Xie, and J. Zhao, Efficient fully homomorphic encryption from RLWE with an extension to a threshold encryption scheme, Future Generation Computer Systems, 36, pp. 180–186, 2014.

[127] T. Zhou, X. Yang, W. Zhang, and L. Wu, Efficient fully homomorphic encryption with circularly secure key switching process, International Journal of High Performance Computing and Networking, 9(5-6), pp. 417–422, 2016.

APPENDIX A

Program source code

```
1  include "FHE.h"
2
3  int main(int argc, char   argv)
4
5      /    BEGIN INITIALIZATION    /
6      long m = 0;      // Specific modulus
7      long p = 1021;  // Plaintext base [default=2], should be a prime number
8      long r = 1;      // Lifting [default=1]
9      long L = 16;     //  levels in the modulus chain, we assume the default is
           heuristic
10     long c = 3;      //  columns in key switching matrix, assuming default is 2
11     long w = 64;     // Hamming weight of private key
12     long d = 0;      // The field extension s degree, assuming default is 1
13     long k = 128;    // Security parameter [default=80]
14     long s = 0;      // Minimum number of slots [default=0]
15
16
17     m = FindM(k, L, c, p, d, s, 0); // Find a value for m given the specified
           values
18
19
20
21     FHEcontext context(m, p, r);     // Initialize context
22     buildModChain(context, L, c);    // Modify the context, adding primes to the
           modulus chain
23
24
25
26     ZZX G =  context.alMod.getFactorsOverZZ()[0];    // Creates the polynomial used
           to encrypt the data
27
28
29     std::cout    "Generating keys... "    std::flush;
30     FHESecKey secretKey(context);    // Build the structure of a private key
31     const FHEPubKey  publicKey = secretKey; //  FHESecKey is a subclass of
           FHEPubKey
32     secretKey.GenSecKey(w); // Generating a private key
33     //std::cout    "secretKey= "    secretKey    std::endl;
34     std::cout    "DONE!"    std::endl;
35     /    END INITIALIZATION    /
36
37     Ctxt ctx1(publicKey);    // Initialize the first ciphertext (ctx1) using
           publicKey
38     Ctxt ctx2(publicKey);    // Initialize the first ciphertext (ctx2) using
           publicKey
39     long N1;
40     long N2;
41     std::cout    "Please enter the first value: "    std::flush;
42      cin    N1;
```

```
43      std::cout    "Please enter the second value: "   std::flush;
44       cin    N2;
45
46      std::cout    "Creating the ciphertext of "   N1  " ... "  std::flush;
47      publicKey.Encrypt(ctx1, toZZX(N1));    // Encrypt the value 1
48      //std::cout    ctx1    std::endl;
49      std::cout    "DONE!"    std::endl;
50
51
52      std::cout    "Creating the ciphertext of "   N2  " ... "  std::flush;
53      publicKey.Encrypt(ctx2, toZZX(N2));    // Encrypt the value 2
54      //std::cout    ctx2    std::endl;
55      std::cout    "DONE!"    std::endl;
56
57
58      Ctxt ctSum = ctx1;  // Create a ciphertext to hold the Sum and initialize it
            with (ctx1)
59      Ctxt ctProd = ctx1; // Create a ciphertext to hold the Prod and initialize it
            with (ctx1)
60
61      std::cout    "Performing the Homomorphic Addition over Ciphertexts... "   std::
            flush;
62      ctSum += ctx2;  // Perform (ctx1) + (ctx2)
63      //std::cout    ctSum    std::endl;
64      std::cout    "DONE!"    std::endl;
65
66
67      std::cout    "Performing the Homomorphic Multiplication over Ciphertexts... "
            std::flush;
68      ctProd  = ctx2; // Perform (ctx1)   (ctx2)
69      //std::cout    ctProd    std::endl;
70      std::cout    "DONE!"    std::endl;
71
72
73      ZZX ptSum;  // Create a plaintext to hold the plaintext of the Sum
74      ZZX ptProd; // Create a plaintext to hold the plaintext of the Prod
75
76      std::cout    "Decrypting the addition result..."   std::endl;
77      secretKey.Decrypt(ptSum, ctSum);    // Decrypt the ciphertext ctSum into the
            plaintext ptSum using secretKey
78      std::cout    N1    " + "    N2    " = "   ptSum[0]    std::endl;
79
80      std::cout    "Decrypting the multiplication result..."   std::endl;
81      secretKey.Decrypt(ptProd, ctProd);  // Decrypt the ciphertext ctProd into the
            plaintext ptProd using secretKey
82      std::cout    N1    "   "    N2    " = "   ptProd[0]    std::endl;
83
84
85      return 0;
86
```

APPENDIX B

HElib Installation

Building HElib requires two libraries, namely, GMP and NTL.

GMP is pre-installed in many Linux distributions. In case it is not installed, please follow the instructions below.

1. Downloading GMP at `http://www.gmplib.org`
2. Unzip the compressed file and navigate to the directory gmp-XXX
3. Execute the commands:

```
./configure
make
sudo make install
```

4. By that, GMP should be installed into the directory /usr/local

After that, we install NTL v9.4.0 or higher:

1. Downloading NTL at `http://www.shoup.net/ntl/download.html`
2. Unzip the compressed file and navigate to the directory ntl-XXX/src
3. Execute the commans:

```
./configure NTL_GMP_LIP=on
make
sudo make install
```

4. By that, NTL should be installed into the directory /usr/local

As soon as you got the libraries installed, we can build HElib, first, navigate to HElib src directory.

Currently, HElib has a basic building system. There is no configure script, only a Makefile. The main task of the Makefile is building fhe.a, which a static library. After

that, applications programs can be built based on it (NOTE: your application should be in HElib src directory or you can set the include parameters to include the library).

It is recommended to have a look at the Makefile to consider adapting the default parameters of CFLAGS & CC. However, usually the defaults work with the majority of the systems.

In HElib src directory, execute the command:

```
make
```

this compiles then builds the library fhe.a. Following that, execute

```
make check
```

This in turn compiles & calls a set of programs for testing.

To build programs based on the library HElib, you can create a file called `myprog.cpp`, and write your program in it, then execute

```
make myprog_x
```

this compiles `myprog.cpp` and links fhe.a with the required libraries. In addition, the executable `myprog_x` will be created.

For more examples of programs that use HElib, you can check `Test_*.cpp`.

`https://github.com/shaih/HElib/blob/master/INSTALL.txt`