Master of Science in Computer Science
February 2017

Homomorphic Encryption: Working and Analytical Assessment

DGHV, HElib, Paillier, FHEW and HE in cloud security

Srinivas Divya Papisetty

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author(s):
Srinivas Divya Papisetty
E-mail: srpa15@student.bth.se

University advisor:
Dr. Emiliano Casalicchio
Dept. of Computer Science & Engineering

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

ABSTRACT

Context. Secrecy has kept researchers spanning over centuries engaged in the creation of data protection techniques. With the growing rate of data breach and intervention of adversaries in confidential data storage and communication, efficient data protection has found to be a challenge. Homomorphic encryption is one such data protection technique in the cryptographic domain which can perform arbitrary computations on the enciphered data without disclosing the original plaintext or message. The first working fully homomorphic encryption scheme was proposed in the year 2009 and since then there has been a tremendous increase in the development of homomorphic encryption schemes such that they can be applied to a wide range of data services that demand security. All homomorphic encryption schemes can be categorized as partially homomorphic (PHE), somewhat homomorphic (SHE), leveled homomorphic (LHE), and fully homomorphic encryption (FHE). Each encryption algorithm has its own importance and usage in different realms of security. DHGV, Paillier, HElib, and FHEW are the algorithms chosen in this study considering their wide usage and scope for further advancement in this subject area. A public-key algorithm named RSA is also chosen for comparison of the impact of HE and PKE (Public-key encryption) algorithm on the CPU and Memory. The utilization of various homomorphic schemes and concepts in the trending cloud storage systems is a prevailing field of research and can be expanded further by knowing the current state-of-the-art of homomorphic encryption. Hence, the necessity of comprehending the knowledge of homomorphic encryption schemes and their aspect in cloud security becomes vital.

Objectives. The objective of this study is to analytically assess homomorphic encryption and various homomorphic encryption schemes. A comprehensive investigation on working and performance of the selected HE schemes is another objective of this research. Also, an experiment to run publicly available libraries of DGHV, Paillier, HElib, and FHEW is one of the main objectives. In addition to these, comprehending the impact of HE and PKE on CPU and Memory is also among the objectives of the study. The role and practice of homomorphic encryption in the cloud storage system are among the secondary objectives of this research in terms of securing confidential data. These objectives are set based on the research gap identified by conducting an exhaustive literature review.

Methods. The objectives of this study are achieved by adopting the methods exhaustive literature review and experiment. Scientific databases such as IEEE Xplore, ACM Digital Library, Inspec, Springer Link etc. are used and literature is accordingly selected based on the relevance to the research topic. An exhaustive literature review is conducted and extensive bibliographic research is done to accomplish the objective of comprehending the working, applications, significance of homomorphic encryption. Apart from literature review, bibliographic research, an experiment is also conducted to run the publicly available homomorphic encryption libraries to evaluate, compare, and analyze the performance of DGHV, Paillier, HElib, and FHEW schemes. Experiment to run publicly available PKE algorithm is also conducted. Finally, the conclusion and outcome by adopting these research methods for accomplishing the objectives are theoretically presented in detail.

Results. By conducting an exhaustive literature review, the importance, working, application of homomorphic encryption and its schemes is discerned. And by conducting an experiment, the impact of HE and PKE is also discerned. Apart from this, the limitations of HE and selected HE schemes along with the distinction between public and private key cryptography is understood by finding and mapping in connection with each other. From the experiment conducted, it is examined that despite the encryption libraries being publicly available for use, the possibility of running and employing few libraries successfully is remarkably low inferring that there is much improvement needed in this cryptographic discipline.

Conclusions. From this research, it can be concluded that homomorphic encryption has a wide scope of extending towards efficiency and application in various fields concerned with data protection. It can also me concluded that the experimental assessment of state of the art of few HE schemes libraries that are available online are remarkably impractical for real-time practice. By analyzing the selected

schemes, it can be concluded few HE schemes do not support any other operations on encrypted data other than addition and multiplication due to which chances of increasing noise for each encryption is relatively high. From the experiment conducted for Paillier encryption (HE) and RSA (PKE) encryption, it is concluded that both the schemes increase linearly with an increase in the input size when CPU and Memory utilization is measured. Apart from these conclusions, it can also be inferred that not all the homomorphic encryption algorithms are IND-CCA1 and IND-CCA2 secure. From this study, it can be deduced that more empirical validation and analysis of HE algorithms is required in terms of their performance and security. In order to address these problems, much research and improvement are required as it inferred from the results of this research that Homomorphic encryption is still in its early stage of development and enormous utility can be anticipated when enhanced correctly.

Keywords: Homomorphic encryption, Homomorphic Encryption Schemes, Cloud security, Cryptography

ACKNOWLEDGEMENTS

It gives me an immense pleasure in expressing my wholehearted sense of gratitude towards my beloved guide & Professor Dr. Emiliano Casalicchio, Associate professor, Dept. of Computer Science and Engineering under whose able guidance, I could develop an interest in research & complete this study.

I would like to express my heartfelt gratitude to my parents, Mr. P. Srinivasulu and Mrs. P. Nirmala, my beloved grandmother Mrs. Jayalakshmi, my beloved sister Dr. P. S. Bhagya Shri and my Brother-in-law Dr. K. Sanath, for their eternal love, kind support, and well wishes in all the stages of my master thesis.

I would also like to express my deepest thanks to my friends Prerika Arora, Alex Bramah Lawani, Afrin Mahaboob, Harshini Sangewar, Susmitha Rao, Usha Das, Dharmvir Singh, Rahul Deekonda, Biswajeet Mohanty, and Karthik Madhavan who have helped and encouraged me to successfully accomplish my master thesis.

Finally, I am extremely thankful to all my dear relatives and friends who motivated and supported me all through the journey of my master thesis.

Above all, I thank the divine existence whose presence is always with us.

Srinivas Divya Papisetty

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

In this modern day world where technology has engulfed mankind in every aspect of life, data protection becomes as crucial as efficient data storage. Data protection generally refers to have controlled access to the data that is being secured i.e. authorizing someone before taking an action for something. There are three major tools used security that is paramount for securing information which is; authentication, access control, and encryption. Cryptography is the art of concealing information such that it is accessible only by the legitimate receiver of the information and no other intruder can access it and cryptosystems are those systems that deal with encryption and decryption of information. Unlike now, cryptography in ancient times was more linked with encryption which was mainly for message confidentiality. Conventional methods of encryption would protect their data while it is in transit, but not while the computation is in progress. This method requires that all the arithmetical and logical operations needed in the computation, which may be represented by circuits or gates, be applied to the encrypted form of the data [1]. Homomorphic encryption is one such type of encryption where the original plaintext or data is not revealed and arbitrary computations can be performed on the encrypted text.

1.1.1 What is Encryption?

Encryption has been a vital part of human activity, as far back as in Rome, Julius Caesar used the “Caesar” Cipher, also the Japanese used cryptography extensively in WWII and used the “Purple” cipher. Encryption is a security technique that helps in providing security to programs, networks, databases and electronic communication in order to prevent an intruder from making attempts to access the confidential data by preventing, modifying, fabricating or intercepting the data that the sender has sent to the receiver.

1.1.2 Traditional encryption

As we know that encrypting of a text means transforming the text into an unreadable form by using codes etc. Traditional encryption or the early methods of encrypting data usually involved only two methods for encrypting information. The two methods are transposition and substitution where transposition means jumbling of letters and substitution means substituting the existing letters with other letters or numbers etc. We can say that transposition was used first than substitution. As already mentioned, the Ceaser cipher is one of the oldest ciphers used and was made by transposing the letters. Later substitution of letters and words emerged with the help of using codes. For example, in world war II the Enigma machine developed by the German military used the technique of substitution for ciphering texts and also developed a code book for encrypting the texts. Therefore, the traditional encryption incorporates only two methods for encrypting the texts namely transposition and substitution and this type of encryption was in practice since ancient times that also helped researchers to develop advanced encryption techniques such as public-key encryption, Homomorphic encryption etc. which are discussed in following sections.

1.1.3 Symmetric and Asymmetric encryption

Symmetric-key and Asymmetric key encryption are also called as secret-key encryption and private key encryption respectively. In Symmetric-key encryption, there is only one private or secret key shared between the send and the receiver for both encryption and decryption of the plaintext. The ciphers used in most of the symmetric-key algorithms are stream ciphers and block ciphers. The process in such type of encryption usually includes key generation, substitution, and transposition. The various Symmetric-key algorithms are AES (Rijndael),

Triple-DES, Blowfish, IDEA etc. All these algorithms have different key sizes and a different set of substitutions and transposition. As the encryption and decryption are done by using a shared secret key which is not always exchanged securely so Symmetric-key encryption algorithms are more susceptible to plaintext attacks.

Asymmetric key encryption can be defined as encryption of plaintext using two key; the public and private key that are generated together. Unlike in symmetric-key encryption, the private or secret is not shared between both the sender and the receiver. Only the public key is shared among both the parties and only the person having the private key for that particular encrypted message will be able to decrypt it. This type of encryption also involves methods such as key generation, transposition, and substitution of letters. Asymmetric key algorithms are not prone to plaintext attacks because the secret key is never shared between the sender and the receiver in an unsecured way. Also, these encryptions are more powerful that Symmetric-key encryptions. RSA, Diffie-Hellman etc. are few examples of asymmetric-key algorithms that are widely known and used.

1.1.4 Homomorphic Encryption

A much stronger and secure encryption than private and public key encryption developed is Homomorphic encryption. Homomorphic encryption is a technique of encrypting the plaintext and performing computations on the encrypted text without disclosing the plaintext i.e. without decrypting it. Homomorphic Encryption can be called building blocks of modern day cryptography as it is used in many tools of cryptography. Homomorphism and Non-malleability are antipodal properties of an encryption scheme. Homomorphic encryption schemes permit anyone to transform an encryption of a message m into an encryption of f(m) for non-trivial functions f. Non-malleable encryption, on the other hand, prevents precisely this sort of thing; it requires that no adversary be able to transform an encryption of m into an encryption of any “related” message [2].

As the definition of HE says, unlike in other encryptions, Homomorphic Encryption involves computations that are performed on the enciphered texts and not on the plaintext after the decryption. Computing on encrypted data means that if a user has a function f and want to obtain f (m1, ... ,mn) for some inputs m1, ... ,mn, it is possible to instead compute on encryptions of these inputs, c1, . . . , cn, obtaining a result which decrypts to f(m1, . . . ,mn) [3]. This property of HE makes it more robust and secure than other encryption algorithms. Earlier, homomorphic encryption was defined as a form of encryption where a specific algebraic operation performed on plaintext is equivalent to another algebraic operation performed on its ciphertext. In mathematics, these operations are called mappings or functions and are operation preserving (*OP*) mappings [1]. A simple working example of homomorphic encryption of numbers and homomorphic encryption for concatenation of two words is shown in Figure 1.

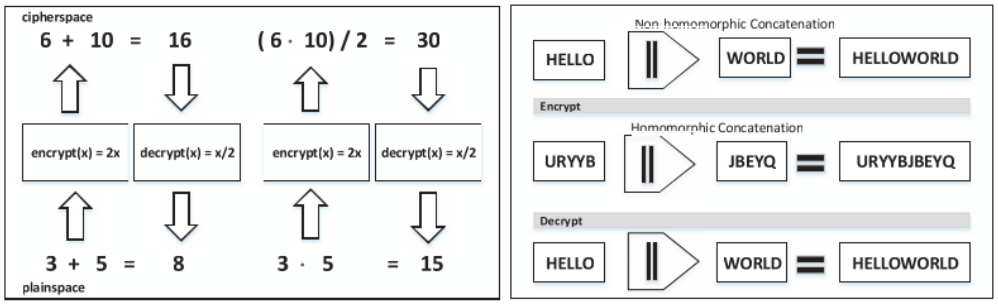

Figure 1: (a) Example of an encryption of numerical values as plaintext using homomorphic encryption; (b) Example of the concatenation of two words using homomorphic encryption [1]

Partial HE schemes support only one type of operation i.e. either addition or multiplication such Paillier which is additive and RSA which is multiplicative etc. FHE schemes without "refreshing" the noise can also be employed as partial HE schemes. These schemes usually support a large number of additions and limited levels of multiplications. HE schemes with this property are usually referred to as Somewhat Homomorphic Encryption Schemes (SWHE) [4]. Fully HE schemes are different from partial and somewhat HE schemes in a way that both, addition and multiplication can be performed and also many times without any limit on the number of times computations can be done. A normal fully HE can perform any number of additions and multiplications homomorphically whereas a leveled fully HE can homomorphically calculate the circuits only up to a certain depth.

The benefit of leveled fully HE over normal fully HE is that secure encryption of the secret key can be done with the available public-key. For FHE schemes to be practical, there are few constraints that need to be satisfied such as versatility, speed, and the ciphertext size. Existing Partial HE schemes can also be aimed to make them practically work as Fully HE scheme by modifying them in a way that the constraint to be an FHE scheme satisfies. This research highlights the significance of Homomorphic encryption and gives a detailed description of selected four HE schemes namely; DGHV, BGV, Paillier, and FHEW that are discussed in the later sections of the document.

1.2 State of the art

In order to gain an extensive knowledge about where and how to advance further in the research; one must know the present state of the art. Various institutions and universities spend millions of dollars yearly in research into cryptography and as a result of which, there is a constant advancement in the field of cryptography and security. Encryption has now entered into everyday technologies such as WhatsApp, iMessage, Telegram, Signal etc. are mobile apps where encryption to some extent is done on texts, images and also on voice calls. Though the encryption in these technologies does not guarantee complete protection of data but serves as a foundation to further broaden and enhance the current research in a much more promising method. One of the most emerging concepts of cryptography from the past few decades is Homomorphic Encryption.

As explained earlier, Homomorphic encryption is a type of encryption where the original plaintext is not revealed and arbitrary computations can be performed on the encrypted text. Fully Homomorphic encryption is long regarded as "holy grail" of cryptography [2]. Today, different types of HE algorithms are used for providing security in various areas such as Bank accounts, e-voting systems, cloud computing, consumer privacy in advertisements and much more. Various organizations have developed various types of cryptosystems; some are open source while some are proprietary. The first fully homomorphic encryption was proposed by Craig Gentry in his dissertation "A fully homomorphic encryption scheme" [5]. The development of fully homomorphic encryption is a revolutionary advance, greatly extending the scope of the computations which can be applied to process encrypted data homomorphically. Since Gentry published his idea in 2009 there has been huge interest in the area, with regard to improving the schemes, implementing them and applying them [3].

With the increase in a number of cloud services and cloud services providers, the need for protecting the data on the cloud also increases. Hybrid Clouds that include private and public clouds intensify the security concerns of cloud computing customers regarding security guarantees in heterogeneous environments consisting of multiple service providers. The security measures provided by the cloud providers to achieve the protection goals for customer data processed on their infrastructure are still in an early stage [6]. Homomorphic encryption is one such technique that also helps cloud systems to secure the data on their cloud even in

the presence of adversaries.The users send data to an online storage service, the data can be encrypted before sending it. In this case, data privacy is assured and the provider can neither use nor analyze the data. However, analyzing current implementations of cloud computing services like Software-as-a-Service or even Infrastructure-as-a-Service, data can be encrypted during the transfer phase but it must arrive at the destination as plaintext [7].

1.3 Scope

Homomorphic encryption is a very vast subject area due to which the scope of the research is restricted to a selection of only four schemes and describing the importance and working of HE with the selected schemes. A comprehensive explanation about HE and working and performance of the selected schemes is described in this research. Experiments are conducted to understand the working of the schemes with the help of available libraries. The results of the experiments are then analyzed and described theoretically. Also, the HE schemes are compared and contrasted in order to realize the significance of the subject.

1.4 Problem statement and Motivation

Data security has now become a primary concern for many organizations and institutions. Data breach is one of those security breaches that stands up as one of the most frequently stated problems with stealing, fabrication, modification, and interception of confidential data. One of the key solutions to address this security breach is by encrypting the data. Due to the rapidly growing need for data security, continuous research is being done in the field of security which leads to the emergence of new encryption algorithms and schemes. Cryptography is one such discipline that is used for securing the data from intruders by encrypting and decrypting it. In order to provide a great encryption algorithm to protect data, one must understand the fundamental theory that the security of a system lies in the strength of the cryptogram that protects it. However, over the years many cryptograms that had been deemed unbreakable have been broken and has given rise to increased research and development in the field of cryptography. Despite the importance of data security, there is still a paucity of efficient encrypting algorithms that can convince an abiding end to the threat of data intruders.

Although many studies have been carried out and different kinds of literature have been published that describe the importance and working of various HE schemes, there is always a constant battle going on between code breakers and code makers which lead to the development of strong encrypting as well as decrypting algorithms. A central objective in the theory of cryptography is to classify the relative complexity of various cryptographic tasks [8]. According to the results of the latest Google Trends search for Homomorphic encryption, the interest on HE over a period of 9 years i.e. since Gentry proposed the first fully working homomorphic encryption is ~ 45%. This means that research in this field is still being continued and many advancements in this subject area can be anticipated in the coming future. It is also perceived that in the recent years, when compared to other topics in the field of cryptography, there is only a little enhancement done in HE and there exist only a few traces of flexibility for applying HE in real time. This gap that is identified out indicates that there is a need to understand the various perceptions of HE and expand the current state of the art that can turn limitations of HE to effective cryptographic developments. This serves as motivation to carry out this research. We must understand the significance of existing HE schemes thoroughly so that it can benefit researchers to advance more in this subject area. This study also discusses the various fundamental metrics that are used to understand and evaluate the performance of HE and Public-key encryption schemes.

1.5 Research aim and Objectives

The primary aim of this research was to fill the research gap that has been deduced by conducting a literature review on the topic Homomorphic Encryption. The other aims of the study were to get in-depth knowledge about the working of Homomorphic Encryption and different schemes associated with it. Another purpose of this study was to test various existing HE schemes and evaluate them based on CPU and Memory utilization. In addition to this, perceiving the impact of HE and Public-key encryption on CPU and Memory utilization serves as one of the main objectives of the research. This study systematically reviews the existing state of the art and seeks to obtain extensive knowledge about the working of various HE schemes. This research explores a way to discover the significance of HE usage and its future implementations.

With respect to the aim of the research given above, the fundamental objectives of the research were

- To gain insights about the importance of Homomorphic encryption in the current state of cryptographic studies.
- To study about the selected four Homomorphic Encryption libraries that are available for investigating deep into the HE schemes.
- To realize the distinctness between public and private key encryptions and homomorphic encryption.
- To analyze the performance of the selected four HE schemes based on various performance and security metrics.
- To conduct an experiment to run the HE libraries and critically evaluate the working of the schemes.
- To conduct an experiment to run a Public-key encryption library for comparing and analyzing working of both HE and Public-key encryption schemes.
- To identify the advantages and limitations of HE that can benefit researchers to focus on the areas for additional advancement in the topic.
- To fill the identified gap and to provide the basis for further progress in the subject area based on the present state of the art.

1.6 Research questions

This section accounts the research questions framed based on the above-mentioned research aims and objectives.

RQ 1) How does Homomorphic Encryption work and what is its importance?

Motivation: Cryptography is a very broad subject area and Homomorphic encryption is one of those topics in cryptography where there has been an only limited advancement in the past few years. A systematic understanding of how homomorphic encryption works and how important it can be when applied and incorporated to different technologies was needed. This led to the framing of RQ 1. This research question is answered by an exhaustive literature review.

RQ 2) What are the differences between Homomorphic Encryption and Public and Private key encryption in terms of functionalities and performance?

Motivation: Cryptography has entered the field of security long ago. The development of public and private key encryptions has assisted many technologies with promising security to

their extent possible. So far, however, there is not much discussion about the analysis of HE over public and private key cryptography. As homomorphic encryption is the next level of public key encryption, the significance of HE can be better perceived when contrasted with the existing encryption technologies in terms of its functionalities and performance. This led to the formulation of this research question which is answered by conducting an exhaustive literature review and extensive bibliographic research.

RQ 3) What are the differences between the selected Homomorphic Encryption schemes?

Motivation: There are many HE schemes that have been developed since past few decades. But only a few of them are widely being tested and used for further development. The four schemes are selected based on their growth, availability, and applicability. No previous studies have given sufficient consideration to the particular selected He schemes altogether which are DGHV, BGV, Paillier, and PHEW. Research on this study has been mostly restricted to the limited comparison of Pailliers, BGV, and DGHV. This gap motivated to the formulation of RQ 4. It is answered by performing exhaustive literature review and experiments.

RQ 4) What are the impact of Homomorphic encryption and Public-key encryption on CPU and memory utilization?

Motivation: To know the strength of an algorithm, it is important to evaluate its performance. Although there are studies that describe the performance of the algorithms, there is still a scarcity of knowing the strength, application, and durability of these schemes. This led to the framing of this research question and is answered by conducting experiments and Exhaustive literature review. The experiment was conducted for a HE (Homomorphic encryption) scheme named Paillier and a Public-key encryption scheme named RSA which are chosen based on the resource availability, working, and popularity in terms of usage.

1.7 Contribution and expected outcomes

The contribution to this research is given based on the research gap identified and also with respect to the research questions framed. An investigation about the importance, working, applications and limitations of Homomorphic Encryption is done. A theoretical assessment of HE schemes namely DGHV, Paillier, BGV, and FHEW along with a brief explanation of Public and Private key encryption schemes is given. An impact of HE and Public-key schemes named Paillier and RSA respectively on CPU and Memory utilization is also given. Libraries related to the selected HE algorithms are installed and run to analyze the working of the schemes. RSA public-key encryption library is also run and installed for discerning the impact on CPU and memory utilization. The deep insights into the topic gave rise to new ideas about improvement in HE and their schemes which are discussed in the future work in the later part of the document.

By answering the research questions and achieving the objectives of this research, this study intends to reflect the following anticipated outcomes:

- Extensive knowledge on working and application of Homomorphic Encryption along with distinguishing HE with public and private key encryption.
- Gaining experience with working and implementation of HE libraries to know their application in real time.
- Evaluating the performance of selected four HE algorithms with the help of performance and security metrics to realize the strength of the algorithms.
- Comparing and contrasting the selected HE schemes based on the results obtained by conducting experiments.
- Experiment a working Public-key algorithm and a HE algorithm to know their impact on CPU and Memory utilization.

1.8 Outline of the thesis

This section details about the structure of the thesis. This research document is divided into different sections and consists of six chapters. Introduction (Chapter 1) gives an introduction to the whole research briefly in different sub-sections of the chapter.

Background and Related work (Chapter 2) explains about the previous practice in cryptography and homomorphic encryption along with all the studies that have been carried out till date with respect to homomorphic encryption.

Methodology (Chapter 3) describes the methodology adopted for carrying out this research. A detailed description about how each research question is answered by using a particular research method is also explained. This section helps in understanding the scientific methodology put in place to achieve the desired results and also the motivation behind not selecting other research methods for conducting this research.

Results (Chapter 4) presents all the results obtained by carrying out the research methods in order to fulfill the aims and objectives of the research.

Analysis and Discussion (Chapter 5) discusses the complete topic of the research in a very detailed manner and analysis of the results obtained is also discussed in this section.

Conclusion and future work (Chapter 6) discusses the entire research in the form of a summary. Based on the contribution to the study, the insights gained are presented in the future work in a way that can help other researchers to advance in this subject area

2 BACKGROUND & RELATED WORK

In order to discern the beauty of Homomorphic encryption, the very first crucial step is to comprehend the past and the present of it. An exhaustive literature review and bibliographic research was done for this study that helped gain insights into the topic can benefit other researchers to analyze the role and state of HE in the present day cyber technologies. Moreover, an inclusion of previous studies that led to the development of homomorphic encryption and an inclusion of HE in cloud security and problems associated with HE helps in realizing the need for further enhancement and expansion in this subject area.

2.1 Security in computing

Security in computing field is commonly referred to protection of software as well as hardware. Security not only provides protections to the data and connections but also ensures that there are no disruptions in any services provided by the machine and communications through it to the extent possible. A threat in computing domain can be treated as an intimidation for loss of the data or compromising the security of the information. In order to provide efficient security, it is very important to provide the three properties which are availability, integrity, and confidentiality of information that helps in satisfying the security aspects of the information. Availability is defined as the availability of a system to ensure that an asset can be used by any authorized parties whereas integrity and confidentiality are defined as the ability of a system to ensure that an asset is modified only by authorized parties and the ability of a system to ensure that the asset is viewed only by authorized parties respectively. These three properties are generally mentioned as “Security Triad” [9].

The other properties for secure information include audibility, authenticity, and non-repudiation. Auditability refers to the capability of a system to keep track of all the actions that has occurred concerning an asset of the system. Authenticity is called as validation of the identity of the information sender whereas non-repudiation or accountability is defined as the capability of a system to validate that the sender has sent something and cannot convincingly deny this fact. The damage caused by compromising these aspects of security can result in interception, interruption, modification and fabrication of information that has been sent between the sender and the receiver. These harms can be occurred due to the vulnerability of the system.

Authentication, access control, and cryptography are components of security toolbox. Authentication involves two steps namely identification and authentication. Access control refers to having a controlled access on the data that is being protected and cryptography, as discussed earlier, is the practicing of hiding sensitive information from illegitimate access by attackers. Difference between identities and authentication is that identities such as name etc. are usually known and are not protected/not private whereas authentication using that identity is private which is proof of authentication: card number, name on card etc. are identities and can be known but password that protects the card, the pin etc. are used for authenticating which is proof for identifying who that person claims to be. In order to know a user’s identity, any of the following 3 mechanisms can be utilized. They are; what the user knows, is and has and a combination of these three can also be done. Authentication can be done in different ways and passwords being the oldest and one of the popular ways of authentication till date. The other ways authenticating parameters can be security questions, biometrics, using cookies, based on tokens. Security in computing is not only provided to the data but also operating systems, networks, databases, data on the web and the cloud. This shows that the need for protecting data is ubiquitous in the current cyber cobwebbed world.

The most developed and expanded component of the security toolbox is cryptography which is discussed in detail in the next section. In addition to security aspects, there exist few

policies concerning security. The various policies in computer security are a security policy, confidentiality, integrity, and also hybrid policies. These policies help in realizing what it means for a system to have confidentiality, integrity, and security. A security professional analyzes situations by finding threats and vulnerabilities to the confidentiality, integrity, and/or availability of a computing system. Often, controlling these threats and vulnerabilities involves a policy that specifies *who* (which subjects) can access *what* (which objects) *how* (by which means) [9].

2.2 Cryptography

How far back one looks for the origins of cryptology depends on how far one is willing to stretch definitions. It seems that every culture that develops writing develops cryptography soon thereafter [10]. The art of concealing information from attackers in computing systems is known as Cryptography. It can also be realized as an essential instrument for protecting data in computer and cloud systems. Cryptography is a deep mathematical subject and was principally associated with encryption of the data preceding the current modern age. Prior to the development of public and private key encryption, the methods used in cryptography were simple substitution and transposition of characters which is known as classical cryptography. Substitution implies substituting the alphabets and numbers in the text with different characters, alphabets, and numbers. Code books were also maintained by cryptographers for these created substitutions for future reference. Transposition method implies only swapping of alphabets and numbers within the same text. The ciphers used in these methods were called as substitution and transposition ciphers respectively. Anagramming is the process used for attacking transposition ciphers.

The other ciphers that come under classical cryptography are the Vigenère cipher and One-time pad. Vigenère cipher chooses a sequence of keys, represented by a string. The key letters are applied to successive plaintext characters, and when the end of the key is reached the key starts over. The length of the key is called as the period of the cipher [11]. The one-time pad is just a variant of Vigenère cipher and OTP is a very important theoretical landmark in cryptography since it is the only known mathematically unbreakable cipher till date which satisfies the notion of perfect secrecy (also known as Shannon security or unconditionally secure) [12]. However, these methods seem not much difficult to break but the evolution of advanced cryptographic ciphers and codes stood as a challenging task to many cryptanalysts. Right from the emergence of ancient cryptographic codes to the present cryptographic ciphers, there has been a continuous progress in cryptanalysis. Cryptanalysis is the study and practice of unfolding the hidden information in computer systems. Cryptanalysis is always a boon to information security unless used for deliberate espionage to cause data breach and harmful effects concerning data security and authenticity. Nevertheless, cryptanalysis is considered a boon, adversaries are constantly making it as a curse by using it for illegitimate actions on sensitive data.

The basic component of cryptography is a cryptosystem. A cryptosystem is a 5-tuple (E, D, M, K, C), where M is the set of plaintexts, K the set of keys, C is the set of ciphertexts, $E: M \times K \rightarrow C$ is the set of enciphering functions, and $D: C \times K \rightarrow M$ is the set of deciphering functions [11]. The main objective of cryptography is to keep the encrypted or the enciphered text unrevealed. Considering an example where an adversary plans to break an encrypted text, assuming that the adversary is aware of the algorithm used but does not know the cryptographic key that has been used in the process of encryption, in such a situation, the adversary generally tries to attack in three ways namely; ciphertext only attack, known plaintext attack, and chosen plaintext attack. All the three types of attacks are prevented when there is a good cryptosystem to protect from these attacks which are usually mathematical and statistical. The famous and one of the oldest cipher is the Ceaser cipher that used substitution cipher which was prone to statistical ciphertext-only attack. Encryption and decryption

algorithms, cryptographic hash functions, and pseudorandom generators are the cryptographic primitives for solving problems involving secrecy, authentication or data integrity. In many cases, a single building block is not sufficient to solve the given problem: different primitives must be combined. A series of steps must be executed to accomplish a given task. Such a well-defined series of steps is called a cryptographic protocol with a condition that a protocol only exists between two or more parties [13]. The modern day cryptography is different from classical cryptography in a way that modern day cryptography uses cryptographic keys for encryption and decryption of plaintexts. The later sections explain modern cryptography and various advancements in it with much detail.

2.2.1 Private and Public-key Encryption

The next level after classical encryption techniques is the modern encryption techniques which are private and public encryption. There is a significance difference between the classical and the modern encryption. The fundamental difference is the usage of cryptographic keys in private and public-key encryption which classical encryption techniques do not use. Private key encryption is also known as symmetric key encryption. As the name suggests, symmetric key encryption is defined as encryption and decryption that takes place within the sender and receiver with the help of one shared a secret key. The sender sends a plaintext to the receiver that is encrypted with the help of a secret key that is shared between the two parties and the receiver decrypts the encrypted plaintext with the help of the secret key. This type of encryption can be done using either stream ciphers or block ciphers. Block ciphers encrypt data in blocks of fixed sizes whereas stream ciphers are operated in continuous streams of data but one bit at a time. A symmetric-key encryption scheme consists of a map

$$E: K \times M \rightarrow C$$

such that for each $k \in K$, the map

$$E_k: M \rightarrow C,\ m \rightarrow E(k,m)$$

is invertible. The elements $m \in M$ are the plaintexts and C is the set of ciphertexts, the elements $k \in K$ is the keys. E_k is called the encryption function with respect to the key k. The inverse function $D_k := E_k^{-1}$ is called the decryption function. It is assumed that efficient algorithms to compute E_k and D_k exist. The key k is shared between the communication partners and kept secret. A basic security requirement for the encryption map E is that, without knowing the key k, it should be impossible to successfully execute the decryption function D_k [13].

The various symmetric key encryption algorithms are AES (Advance Encryption Standard), DES (Data Encryption Standard), triple-DES, RC2, RC4, IDEA (International data encryption algorithm), Blowfish, CAST5, serpent etc. Among all these AES and DES are most commonly used and remain famous till date for certain application though they have been proved breakable. Cascading encryption or multiple encryption refers to the application of encryption algorithms one or more time to already encrypted text with the usage of either previously used encryption algorithm or any other encryption algorithm. This can help in proving more security to the encrypted text and much protection from brute force attacks. Triple DES best illustrates multiple encryption. There can be different key sizes in each encryption algorithms depending on the security strength required. The DES algorithm takes 56-bit keys and 64-bit plaintext messages as inputs and outputs a 64-bit cryptogram. An encryption with DES_k consists of 16 major steps, called rounds. In each of the 16 rounds, a 48-bit round key k_i is used. The 16 round keys $k_1, \ldots k_{16}$ are computed from the 56-bit key k by using an algorithm. The cryptographic strength of the DES function depends on the design of Function f, especially on the design of the eight famous S-boxes which handle the eight substitutions [13]. Due to the constant increase in the computing power against a stationary target which is fixed key sizes in DES; an aspect of concern and consideration, AES was developed by security analysts. AES also uses a symmetric block cipher same as that of DES

but has variable key sizes. The algorithm consists of 10, 12 or 14 cycles, for a 128, 192, or 256-bit key, respectively. Each cycle (called a "round" in the algorithm) consists of four steps namely; byte substitution, shift row, mix column and add subkey. Each round performs both confusion and diffusion, as well as blending the key into the result [9].

A simple structure of AES algorithm is shown in Figure 2. Advanced Encryption Standard was accepted as a FIPS standard in November 2001 by National Institute of Standards and Technology (NIST). Since then, software and hardware implementations of AES have received considerable attention [14]. Block ciphers need some extension because in practice most of the messages have a size that is distinct from the block length. i.e. message length exceeds the block length. Modes of operation handle this problem which was first specified in conjunction with DES, but they can be applied to any block cipher. The various modes of operations are Electronic Codebook mode, Cipher-Block Chaining mode, Cipher Feedback mode, and Output Feedback mode [13].

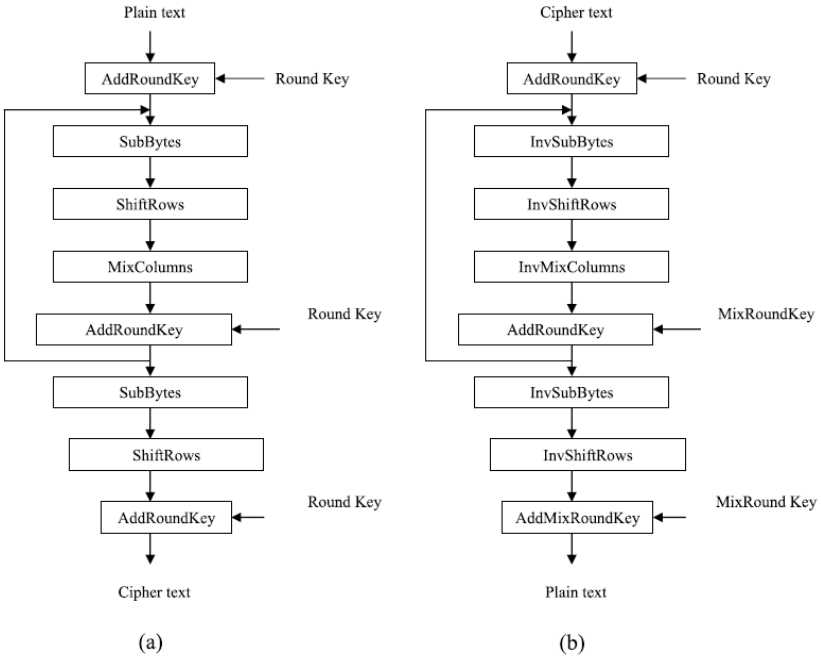

Figure 2: AES Algorithm (a) Encryption structure (b) Equivalent Decryption structure [14]

As mentioned in the previous chapter, public key cryptography deals with encryption and decryption of plaintext with the help of two keys namely public and private. The basis for public key encryption is to allow the key to be divulged but to keep the decryption technique secret. Public key cryptosystems accomplish this goal by using two keys: one to encrypt and the other to decrypt. The public key encryption is different in a way that they can send cryptographic keys even when there is no convenience of acknowledging to a secret key in private. The problem of key proliferation can be reduced by using public key approach. The user may freely publish the public key because each key does only encryption or decryption, but not both. The keys operate as inverses, meaning that one key undoes the encryption provided by the other key [9]. Although these keys are produced in mathematically related pairs, an outsider is effectively unable to use one key to derive the other. An encryption algorithm is said to be an asymmetric key encryption when it satisfies three conditions which are;

- It must be computationally easy to encipher or decipher a message given the appropriate key
- It must be computationally infeasible to derive the private key from the public key.
- It must be computationally infeasible to determine the private key from a chosen plaintext.

These conditions are to be satisfied because one key is public, and its complementary key is kept secret [11]. Key management plays a very pivotal role in private and public key encryption because it serves as the essence of modern cryptography by providing various series of actions such as key generation, key distribution, key authentication, and storage of keys. An archetype in the public key cryptography or asymmetric key cryptography is Alice and Bob. The first major breakthrough in modern cryptography came at this point when Diffie and Hellman proposed the idea of public-key encryption in the year 1976. Cryptography transformed from an artistic pursuit to a scientific one, and its affair with pure mathematics was now out in the open [15]. Diffie-Hellman was the first asymmetric encryption algorithm that facilitated transmission of cryptographic keys in presence of insecure medium also. Generally, this encryption algorithm is not employed on hardware and works better when the proper mathematical group is used in the encryption process. The different asymmetric key algorithms developed after Diffie-Hellman are RSA, Digital signature algorithm, ElGamal, ECDSA, XTR etc. A simple working of public key encryption can be understood with the following expressions. Let M=Message, E=encryption, D=Decryption, k=cryptographic key, k_{PRI}= Private key, k_{PUB}=Public key, now the expression that explains asymmetric key encryption in a simple form is

$$M= D(k_{PRI,} E(k_{PUB,}M))$$

From the above expression, it can be understood that a message or plaintext that has to be transmitted to the receiver is being encrypted first with the help of public key and then later decrypted with the corresponding private key. Few asymmetric key algorithms also facilitate the other way round which is encrypting the message with private key and then decrypting the message with the corresponding public key which can be put in a form expression as follows;

$$M= D(k_{PUB,} E(k_{PRI,}M))$$

This shows that the cryptographic keys can be used in either way. RSA (Rivest-Shamir-Adelman) is one such public key encryption algorithm that facilitates application of cryptographic keys in both ways. There exists a combination of secrecy and authentication in Public-key cryptosystems. RSA uses an exponential cipher which encrypts plaintexts using $C = A^e \pmod{p}$ where C is ciphertext and A is a block of plaintext. Exponential ciphers are less liable to frequency analysis (cipher breaking method) than block ciphers. RSA encryption algorithm can be explained as follows, two large prime numbers p and q are chosen and let $n = pq$. The totient $\varphi\varphi(n)$ of n is the number of numbers less than n with no factors in common with n. An integer $e < n$ that is relatively prime to $\varphi\varphi(n)$ is chosen. A second integer d such that $ed \bmod \varphi\varphi(n) = 1$ has to be found. The public key is (e, n), and the private key is d. Let m be a message. Then $c = m^e \bmod n$ and $m = c^d \bmod n$; which is encryption and decryption of the plaintext. The key size usually starts from 256 and above but no less than 256. In addition to confidentiality, RSA can provide data and origin authentication. If Alice enciphers her message using her private key, anyone can read it, but if anyone alters it, the (altered) ciphertext cannot be deciphered correctly [11]. ElGamal asymmetric encryption can be defined as the predecessor of DES with the key arrangement as that of Diffie-Hellman. It is generally used for key exchanges and sending digital signatures. RSA (Rivest, Shamir, Adleman) is one of the most widely used asymmetric algorithm. In RSA, usually the decryption algorithm is used to generate signatures and the encryption algorithm is used to verify them and is therefore often referred to as the “hash-then-decrypt” paradigm [13]. Among all public key algorithms, Diffie-Hellman and RSA are most widely used algorithms. The most obvious, most direct, and most difficult way to attack RSA is by factoring the modulus. Figure 3 gives an overview of RSA algorithm.

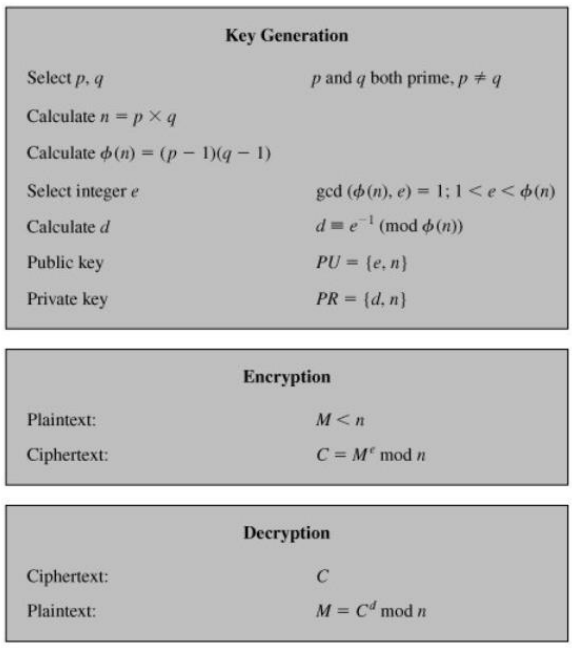

Figure 3: RSA Algorithm [16]

There are eleven non-factoring RSA attacks till today and few of them are common modulus attack, man in the middle attack, low decryption exponent, low encryption exponent attack, Ron was wrong, Whit is right attack etc. [10]. In contrast to fast substitution and transposition of symmetric algorithms, exponentiation is extremely time-consuming on a computer. Even worse, the time to encrypt increases exponentially as the exponent (key) grows longer. Thus, RSA is markedly slower than DES and AES [9]. Unlike other asymmetric algorithms, DSA is used for signing of data and not for encryption of data. It was developed by United States Government and is a FIPS (Federal Information Processing Standards) for digital signatures. Based on a chosen prime number, the signing method is carried out through a series of calculations. Digital signatures are created quickly than validating them when DSA is used whereas in the usage of RSA, digital signatures are validated quickly than creating them. As public key encryptions are usually exponential and not lightweight, they are preferred to be used in key exchange than for bulk encryption of data. When built together, symmetric key encryptions result in a stronger cryptosystem than asymmetric key encryptions. Hybrid encryption is also in practice where private and public key algorithms are used for encrypting the actual message and encrypting the generated key or vice versa respectively by utilizing the strengths of each type of encryption.

Public and private key encryptions are widely being used. Asymmetric key encryptions are used in many online transaction systems to provide a secure transaction medium. Not only it is used in many payment gateways but also used in online protocols such as SSL, DNSSEC etc. Nowadays, for safe communication over the web, modern cryptography has become a base. As from the past decade, with the invention of smartphones, the era of various applications has started. And many organizations such as Apple, Google and much more have developed applications embedded with encryption technology. The application of modern cryptography is deployed in salesforce, Adobe analytics cloud, Office 365, and many cloud storage systems such as Google Drive, Dropbox, OneDrive etc.

2.2.2 Homomorphic Encryption: Working and Types

Homomorphic encryption is a technique of performing secure arbitrary computations on the ciphertext without revealing the plaintext. Encryption schemes that support operations on encrypted data have a very wide range of applications in cryptography. This concept was introduced by Rivest et al. shortly after the discovery of public key cryptography [17]. Informally a homomorphic encryption scheme refers to an encryption function that allows one to induce a binary operation on the plaintexts while only manipulating the ciphertexts without the knowledge of the encryption key: $E(x_1)*E(x_2) = E(x_1 \otimes x_2)$ where $E(x_1)$ is encryption of plaintext x_1 and $E(x_2)$ is encryption of plaintext x_2. If the scheme supports the efficient homomorphic evaluation of any efficiently computable function, it is called a fully homomorphic encryption scheme (FHE) [18]. A homomorphic public key encryption scheme E has four algorithms: the usual KeyGen, Encrypt and Decrypt, and an additional algorithm "Evaluate". The algorithm Evaluate takes as input a public key pk, a circuit C (contains simple operations), a tuple of ciphertexts c = <c1….. ,ct> (one for every input bit of C), and outputs another ciphertext c. These 4 functions of a homomorphic encryption are shown in Figure 4.

A fully homomorphic encryption can be trivially realized from any secure encryption scheme, by an algorithm "Evaluate" that simply attaches a description of the circuit C to the ciphertext tuple and a Decrypt procedure that first decrypts all the ciphertexts and then evaluates C on the corresponding plaintext bits. Two properties of homomorphic encryption that rule out this trivial solution are circuit-privacy and compactness [19]. The other properties of a homomorphic encryption are that they are both additive and multiplicative, unlike normal encryption schemes which are either additive or multiplicative but not both.

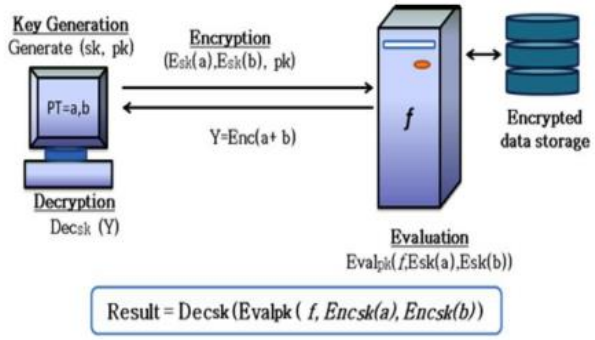

Figure 4: Homomorphic encryption functions [20]

Homomorphic encryption can be classified as partially homomorphic encryption (PHE), somewhat homomorphic encryption (SWHE), and Fully homomorphic encryption (FHE). Fully homomorphic encryption can further be classified as leveled FHE and normal FHE. A fully homomorphic encryption scheme E should have an efficient algorithm Evaluate E that, for any valid E key pair (sk, pk), any circuit C, and any ciphertexts ψi ←Encrypt E(pk,πi), outputs

ψ ←Evaluate E(pk,C,ψ1, . . . ,ψt) such that Decrypt E(sk,ψ) = C(π1, . . . ,πt)

Formally, there are different ways of defining what it means for the final ciphertext to "encrypt" f(π1, . . . ,πt) and the minimal requirement is the correctness [5] where sk is secret key and pk is a public key.

In general, when different ciphertexts are obtained by the same message that is enciphered several times by the utilization of arbitrariness in an encryption algorithm, it is referred as probabilistic encryption. It is important to notice that all FHE schemes are (by necessity) grounded in probabilistic encryption schemes which are traditionally asymmetric and usually operate at the bit level. The theoretical efficiency of an FHE scheme is measured by its per (bit level) operation computational overhead in terms of a security parameter usually denoted 'λ' (the cryptosystem being dimensioned such that the best-known attack requires an order of magnitude of 2λ operations). Although theoretically efficient, Gentry's initial construction (and most of its siblings) is impractical as its overhead is a high degree polynomial of λ [21]. As mentioned previously, there are different variants of homomorphic encryption.

A leveled fully homomorphic encryption scheme is a homomorphic scheme where the Homomorphic encryption Keygen (Key generation) gets an additional input 1^L.
(now (pk, evk, sk) ← HE. Keygen (1^κ , 1^L)) and the resulting scheme is homomorphic for all Depth-L binary arithmetic circuits where 'pk' is a private key, 'evk' is evaluation key, and 'sk' is the secret key. The bound s(κ) on the ciphertext length must remain independent of L. From then on, when we say fully homomorphic, we refer to leveled fully homomorphic encryption [22]. The difference between somewhat and leveled homomorphic schemes is a potential point of confusion. The depth of circuits which a somewhat homomorphic encryption can handle can be increased through parameter choice which generally means that the ciphertext size will increase with the depth of the circuits allowed.

For a leveled homomorphic encryption scheme, the maximum depth is an input parameter and the length of the ciphertext does not depend on it [3]. Somewhat homomorphic encryption system which computes many additions and a small number of multiplications on ciphertexts: for example, averages require no multiplications, standard deviation requires one multiplication, and predictive analysis like logistical regression requires a few multiplications (depending on the precision required) [23].To be bootstrappable, E (encryption scheme) needs to be able to evaluate not only its decryption circuit, which merely allows recryptions of the same plaintext, but also slightly augmented versions of it, so that we can perform non-trivial operations on plaintexts and make progress through a circuit [24]. Due to this structure of the homomorphic scheme E, the algorithms for addition, multiplication and decryption are almost atomic arithmetic operations of the underlying library for large integer handling out of which the most time-consuming operations are K (Keygen) and E (Encrypt) [25].

The problem of privately outsourcing computation to a cluster of machines should be considered; which typically happens when the computation needs to be performed over massive datasets, e.g., to analyze large social networks or train machine learning algorithms on large corpora. At such scales, computation is beyond the capabilities of any single machine so it is performed by large-scale clusters of workers. To address this problem, parallel homomorphic encryption (PHE) schemes is considered which are encryption schemes that support computation over encrypted data through the use of an evaluation algorithm that can be efficiently executed in parallel [26]. The most important property of FHE is unlimited chaining of algebraic operations in the cipher space, which means that an arbitrary number of additions and multiplications can be applied to encrypted operands. To achieve this, an FHE scheme must provide a mechanism to reduce the noise of cipher values, because these schemes are based on a slightly inaccurate representation of the plaintext values. Every single operation on a ciphertext causes even lower accuracy and eventually, the ciphertext can no longer be properly decrypted [25].

For a somewhat homomorphic scheme to be bootstrappable, Gentry described in [22] a transformation to squash the decryption procedure, reducing the degree of the decryption

polynomial. This technique is done by adding to the public key an additional hint about the secret key, in the form of a "sparse subset-sum" problem (SSSP). Namely, the public key is augmented with a big set of vectors, such that there exists a very sparse subset of them that adds up to the secret key. A ciphertext of the underlying scheme can be "post-processed" using this additional hint, and the post-processed ciphertext can be decrypted with a low-degree polynomial, thus obtaining a bootstrappable scheme [17]. PHE (parallel homomorphic encryption) scheme is multi-use which means that the evaluations of different functions can be done on a single ciphertext which is also referred to as the online/offline setting. It is applied mostly for the setting of outsourced computation where a weak computational device wishes to make use of the resources of a more powerful cluster. Using PHE, a client can encrypt a large database during an offline phase and then, have the workers evaluate many different functions on its data during the online phase [26].

2.2.3 Ciphertext indistinguishability

Ciphertext indistinguishability in cryptology can be defined as the characteristic of many encryptions algorithms by the virtue of their nature. An adversary or an attacker is incapable of distinguishing or differentiating the combined ciphertexts in the encrypted text when an encryption algorithm acquires ciphertext indistinguishability. Most of the public key cryptosystems which have convinced to be secure possess ciphertext indistinguishability under chosen plaintext attack (IND-CPA) which is considered to be the essential condition. IND-CPA (Indistinguishability under chosen-plaintext attack) is called the notion of security for public-key encryption (PKE). Subsequently, later indistinguishability was defined under non-adaptive chosen-ciphertext attack, where the adversary is allowed access to a decryption oracle prior to seeing the challenge ciphertext but not after. The notion now universally accepted as the "right" target is IND-CCA, indistinguishability under adaptive chosen- ciphertext attack, where the adversary is allowed access to the decryption oracle both before and after seeing the challenge ciphertext, but cannot query the challenge cipher- text itself [27].

Under the ciphertext indistinguishability rationale in security, there are three definitions namely; IND-CPA, IND-CCA1 (Indistinguishability under (non-adaptive) chosen ciphertext attack), IND-CCA2 (Indistinguishability under adaptive chosen ciphertext attack). Among the three definitions, IND-CCA2 stands as the powerful and strongest of all because each of these signifies the security of the former one in the order IND-CPA, IND-CCA1, and IND-CCA2 respectively. Although there are many SHE (Somewhat homomorphic encryption) and FHE (Fully homomorphic encryption) schemes proposed, most of the SHE and FHE scheme are not IND-CCA2 (secure against adaptive chosen ciphertext attack) based on the fact that the adversary is allowed to manipulate the challenged ciphertext and submit it to the decryption oracle in an IND-CCA2 attack. Also, with Gentry's approach, the resulted FHE scheme cannot be IND-CCA1 secure based on the fact that the private key is encrypted and the adversary is able to submit the ciphertext to the decryption oracle [28]. Non-malleability is an aspect of handling message integrity and not message confidentiality. Ciphertext indistinguishability is also being considered for random noise as "Indistinguishability from random noise". Here an adversary is unable to distinguish the random noise in the elements of encrypted datagram for making traffic analysis more onerous. In order to preserve secrecy in enciphered communications, indistinguishability is a crucial feature.

2.3 Homomorphic Encryption Schemes

Right from the emergence of Homomorphic encryption till today, there has been rapid development in this area w.r.t development of various homomorphic encryption schemes. In previous sections, classification of homomorphic encryption schemes namely, PHE (Partial Homomorphic Encryption), SHE (Somewhat Homomorphic Encryption), LHE (Levelled Homomorphic Encryption), and FHE (Fully homomorphic encryption) were discussed and all

homomorphic encryption algorithms/schemes fall under these main four categories. Another classification of homomorphic encryption schemes is whether they are additively homomorphic, multiplicatively or mixed homomorphic. Starting from the partial homomorphic encryption, the different scheme that falls under this category are Paillier, ElGamal, Unpadded RSA [20], Benaloh, and Goldwasser-Micali. Additive schemes can also be converted to multiplicatively homomorphic schemes using scheme converters. This procedure is discussed in detail in [4].

The feasibility of homomorphic schemes is best when deployed properly according to the characteristics of each scheme. In order to achieve this, it is essential that the working, properties and the usage of schemes are known. Homomorphic encryption schemes that allow simple computations on encrypted data have been known for a long time. For example, the encryption systems of Goldwasser and Micali El Gamal, Cohen and Fischer, and Paillier support either adding or multiplying encrypted ciphertexts, but not both operations at the same time. Boneh, Goh and Nissim were the first to construct a scheme capable of performing both operations at the same time; their scheme handles an arbitrary number of additions but just one multiplication [23]. FV and YASHE are examples of leveled homomorphic encryption [29] whereas DGHV [19], BGV [30], and FHEW [31] are examples of fully homomorphic encryption scheme.

The HE encryption schemes selected for this research are DGHV, FHEW, Paillier, and HElib. These are selected on the basis of availability of literature, libraries, and other resources. Also on the basis of the popularity of the schemes in terms of usability, working, and feasibility. Explanation about these four HE schemes is given in the following sections.

2.3.1 DGHV Scheme

In 2010, the first fully homomorphic encryption scheme over integers was described by Van Djik, Gentry, Halevi, Vaikunthanathan [19] and so the scheme is known as DGHV. This scheme differs from Gentry's scheme in a way that it is operated on integers instead of ideal lattices. The scheme is carried out in three steps as follows;

- **KeyGen**: The key is an odd integer, chosen from some interval $p \in [2^{\eta-1}, 2^{\eta})$.
- **Encrypt (p, m):** To encrypt a bit $m \in \{0, 1\}$, set the ciphertext as an integer whose residue mod p has the same parity as the plaintext. Namely, set $c = pq + 2r + m$, where the integers q, r is chosen at random in some other prescribed intervals, such that 2r is smaller than p/2 in absolute value.
- **Decrypt (p, c):** Output (c mod p)mod 2 [19].

Where p= secret key, m=plaintext, q=large random number, and r=noise. From the above steps, it can be identified that noise r is smaller than the secret key m. The ciphertext is obtained with some amount of noise parameter which is twice the original plaintext and it is usually less the p which results in a limited number of addition and multiplication operations, this is called as somewhat homomorphic encryption. Now, for DGHV to be fully homomorphic the amount of noise in the ciphertext must be reduced and this process is called as refresh. The next process is bootstrapping where decryption circuit is evaluated by encrypting the secret key bits. And finally to decrease the depth of the decryption algorithm, squashing is done. The size of the ciphertext in DGHV is very large so as to prevent from attacks based on lattice reduction algorithm. For flat circuits, this scheme can work as both additively and multiplicatively homomorphic but using squashing or bootstrapping methods as explained by Gentry [24], this can be morphed as a fully homomorphic encryption scheme.

The size of the public key is larger than secret key because all the ciphertexts are included in the public key. To obtain a public key encryption scheme, a public set of ciphertext

$$x_i = q_i * p + 2 * r_i$$

which are all different "encryptions of Zero" is generated. In an improved attack against the Approximate GCD Problem on which the DGHV scheme is based, one pseudo-random of Xi of the same size is generated and computes a small correction d_i such that $x_i = X_i - d_i$ which is small modulo p. These small corrections are stored in public key, with the seed to the PRNG. Application of this compression technique reduces the size of the public key from 802 MB to 10.1MB [32]. Since the development of the very first DGHV scheme in 2010, there have been many improvements done to the scheme and published. The main reason behind improving the scheme is to increase efficiency. The DGHV scheme focuses on encryption and processing of bits, even though it is able to process small numbers, depending on the chosen primary key size. Every operation can be mapped to binary addition and multiplication. DGHV ensures circuit privacy, which consists of hiding previous operations performed over current ciphertext. The circuit privacy is not relevant in traditional cryptography because the ciphertext has only been applied to the encryption algorithm [33].

2.3.2 Paillier Scheme

Paillier cryptosystem implements Paillier scheme which is a partially homomorphic encryption scheme. It is usually additive homomorphic scheme which means we can compute the sum of plaintext m_1 and m_2 as m_1+m_2 when provided with secret key and encryption of m_1 and m_2. It is a probabilistic asymmetric encryption scheme possessing homomorphic properties for both addition and multiplication. This scheme is efficient both in encrypting and decrypting as it takes many bits at a time for one operation with a constant expansion factor. The generalized Paillier scheme when compared to the original scheme, reduces the expansion factor from 2 to nearly 1. Paillier cryptosystem can be used to solve certain discrete logarithms and also provides semantic security against chosen-plaintext attacks. The Paillier scheme is a partially homomorphic encryption scheme which initially could perform only additions on ciphertext but later improvements in the scheme proposes that multiplication can also be performed on the ciphertexts. The paillier scheme is the most known, and maybe the most efficient partially homomorphic encryption scheme. Paillier interestingly comes back to a standard RSA modulus N=pq [34]. There are three main steps involved in this scheme namely Key generation, Encryption, and Decryption. The underlying steps in each of these steps are explained as follows which is reference from and explained in much detailed in [34];

Steps for Key Generation

1. Choose two large prime numbers p and q randomly and independently of each other.
2. Compute the modulus n; the product of two primes $n = p.q$ and (λ is Carmichael's function)
3. Select random integer g where $g \in \mathbf{Z}^*_{\mathbf{n}^2}$ and g's order is a non-zero multiple of n.

(since $g = (1+n)$ works and is easily calculated)

4. Ensure n divides the order of g by checking the existence of the following modular multiplicative inverse:

$$u = L(g^{\lambda} \bmod n^2)^1 \bmod n,$$ where function L is defined as

where L is Lagrange function and $L(u) = u - 1 \backslash n$ *for* $u = 1 \bmod n$

- The public (encryption) key is (n, g).
- The private (decryption) key is (λ) .

Steps for Encryption

1. Plaintext is m where $m < n$.
2. Find a random r
3. Let cipher text $c = g^m . r^n \bmod n^2$

Steps for Decryption

1. The cipher text $c < n^2$.
2. Retrieve plaintext

$$m = L(c\lambda \ mod n^2) \ / \ L \ (g\lambda \ mod n^2 \ ) mod \ n$$

Or in same expression

$$m = L \ (c\lambda \ mod \ n^2 \ ). \ u \ \text{mod} \ n$$

From the above equation it can be inferred that decryption is done by first removing the random part r^n, then exponent $m \ mod \ n^2$ is retrieved [34]. The Figure 5 and Figure 6 shows the encryption and decryption in Paillier cryptosystem. Given the public key (g, n), the Paillier encryption of m ∈ [0, n−1] is c = E(m, r) where r is chosen at random in [0,n−1]. Encryption is therefore probabilistic and features homomorphic properties as multiplying $E(m_1, r_1)$ by $E(m_2, r_2)$ modulo n_2 provides an encryption of m_1+m_2 mod n. This property makes this factoring-based cryptosystem particularly attractive for many cryptographic applications. Paillier encryption is semantically secure against chosen-plaintext attacks under the DCR (Decisional composite residuosity) assumption and was shown to hide O(logn) plaintext bits under a slightly stronger assumption [35].

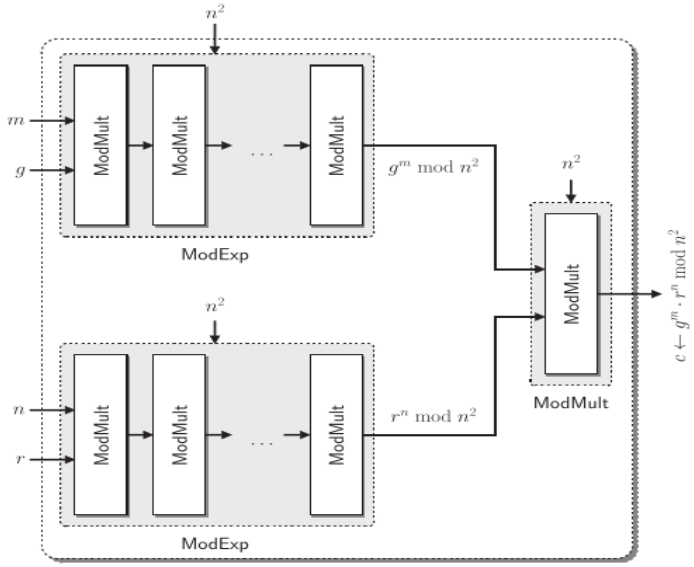

Figure 5: Architecture of Paillier encryption procedure of Paillier homomorphic cryptosystem [36]

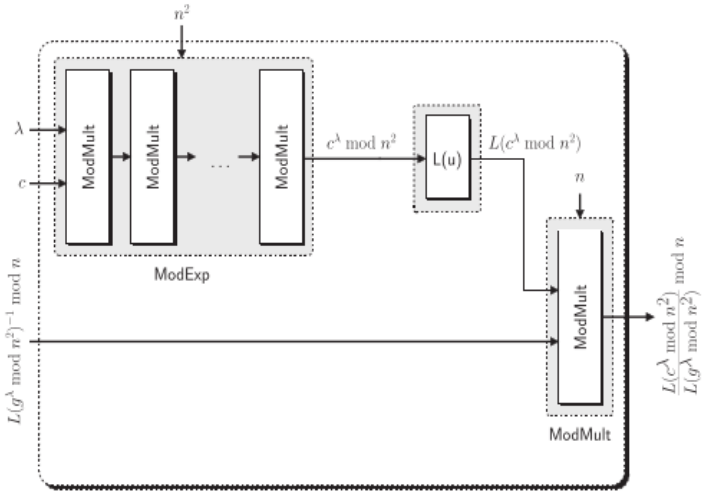

Figure 6: Architecture of Paillier decryption procedure of Paillier homomorphic cryptosystem [36]

The notations and formulas to be known for understanding the scheme are as follows:

- Z* {n^2}
- gcd (L(g^lambda mod n^2), n) = 1
- u = (L(g^lambda mod n^2))^(-1) mod n
- lambda = lcm(p-1, q-1) = (p-1)*(q-1)/gcd(p-1, q-1)[2]
- g – Private random BigInteger in Z* {n^2}
- p, q – Two large primes
- n – Product of p and q
- c – ciphertext
- m – plaintext

The various functions available in the library to perform Paillier homomorphic encryption operations are:

- Paillier(int bitLengthVal, int certainity) – This function takes bitLengthVal and certainity as integer parameters where bitLengthVal is number of bits of modulus and certainity is the probability that the prime number represented by new bigInteger will exceed (1-2^(-certainity)).
- Paillier()- A public method that is used for constructing an instance with 512 bits of modulus and at least 1-2^(-64) certainty of primes generation.
- KeyGeneration(int bitLengthVal, int certainity)- This is a public method used for setting the public and private key. It takes bitlengthVal and certainity as integer parameters.
- Encryption(BigInteger m, BigInteger r)- a public method that takes BigInteger m and r as plaintext and random plaintext respectively. The encryption of plaintext m, returns a ciphertext c= g^m * r^n mod n^2 where random input r is automatically generated to help with encryption.
- Decryption(BigInteger c)- This public method takes ciphertext c as a parameter to decrypt c so as to return plaintext m by using the formula m = L(c^lambda mod n^2) * u mod n.

- main()- Here the operations on encrypted texts em_1,em_2 of plaintexts m_1,m_2 respectively are done. The input is the plaintext as integers. To test the homomorphic properties, addition and multiplication are performed on em_1 and em_2. And if decryption of the sum and multiplication of encrypted texts should return the same sum and multiplication of plaintexts. Decryption of sum can be obtained by $D(E(m_1)*E(m_2) \bmod n^2) = (m_1 + m_2) \bmod n$ and decryption of product can be obtained by $D(E(m_1)^{m_2} \bmod n^2) = (m_1*m_2) \bmod n$.

The paillier scheme is a research result regarding trapdoor discrete logarithms - based cryptosystems. The IND-CPA security of Paillier relies on computing composite residuosity problem. Paillier cryptosystem allows encrypting many bits during a single operation with a constant expansion factor of 2. Moreover, Paillier allows efficient decryption [7].

2.3.3 HElib library for BGV scheme

HElib [37] is an open-source C++ library that implements the BGV scheme, focusing on effective use of ciphertext packing and the GHS (another variant of BGV) optimizations. It includes an implementation of the BGV scheme itself with all its basic homomorphic operation, and also some higher-level procedures implementing the GHS data-movement procedures and simple linear algebra [38]. Brakerski, Gentry, and Vaikuntanathan proposed a new FHE scheme (BGV) based on LWE problems. Instead of recryption, this new scheme uses other light weighted method to refresh the ciphertexts. These methods cannot thoroughly refresh the ciphertexts as the recryption does, however they can limit the growth of the noise so that the scheme can evaluate much deeper circuits [4]. Initially, an improvised FHE scheme based on LWE was proposed that altered for the AES- block cipher efficiency without bootstrapping and then the BGV scheme was proposed which is a somewhat homomorphic encryption scheme. The central feature of BGV-type cryptosystems is that the current secret key and modulus evolve as we apply operations to ciphertexts [39]. BGV is an asymmetric encryption scheme that encrypts bits. Like most (somewhat) FHE schemes, it is based on lattices. There are two versions of the cryptosystem: one dealing with integer vectors (the security of which is linked with the hardness of the decisional LWE (Learning With Errors) problem) and the other one with integer polynomials (the security of which is linked with the hardness of the decisional R-LWE (Ring-Learning With Errors) problem [21].

The implementation of BGV in HElib has a native plaintext space of the BGV cryptosystem is R_p^r for a prime power p^r. The scheme is parametrized by a sequence of decreasing moduli $q_L >> q_{L-1} >> \cdots >> q_0$, and an "$i^{th}$ level ciphertext" in the scheme is a vector $c_t \in (R_{qi})^2$ [30]. Secret keys are elements $s \in R$ with "small" coefficients (chosen in $\{0, \pm 1\}$ in HElib), and we view 's' as the second element of the 2-vector $sk = (1, s) \in R^2$. A level-i ciphertext $ct = (c_0,c_1)$ encrypts a plaintext element $m \in R_p^r$ with respect to $sk = (1, s)$ if we have

$$[sk, ct]_{qi} = [c_0 + s \cdot c_1]_{qi} = m + p^r \cdot e \text{ (in R) for some "small" error term, } p^r \cdot \|ek\| << q_i$$

where sk is secret key, m is message, ek is encryption key, z is intergerm and R is rinf R. In Helib, each q_i is a product of small (machine-word sized) primes [30]. Elements of the ring R_{qi} are typically represented in DoubleCRT format: as a vector a polynomial modulo each small prime t, each of which itself is represented by its evaluation at the primitive m^{th} roots of unity in Z_t. In DoubleCRT format, elements of R_{qi} may be added and multiplied in linear time and is detailed deeply in [30]. The lower-level of HElib implements an "assembly language" which is executed on a "hardware platform" given by the underlying HE scheme. The "platform" defines a set of operations that can be applied homomorphically and the cost of these operations; providing efficient implementation of simple routing and linear-algebra procedures over that "platform." Since the homomorphic operations define element- wise

operations on the vector of plaintext values, the "platform" defines for us a SIMD environment [38].

2.3.4 FHEW Scheme

FHEW is a fully homomorphic encryption scheme and is available as an open-source software. In this scheme, Homomorphic evaluation of arbitrary Boolean circuits on encrypted data is supported by encryption and decryption of single bit messages. The central theme of the scheme is to perform homomorphic NAND operation of messages '0' and '1' and decrypt the NAND result. It allows to homomorphically compute the exclusive-or of two bits. The way this operation is extended to a logical NAND computation is by moving (during bootstrapping) from arithmetic modulo 2 to arithmetic modulo 4.
The major steps can are as follows;

- The two plaintexts m_1 and m_2 are taken. Given two encryptions $E(m_1)$ and $E(m_2)$ one can compute a noisier version of $E(m_1 + m_2)$.
- Adding $E(m_1)$ and $E(m_2)$ results in the encryption. E(m) of m = 2 (if $m_1 \barwedge m_2 = 0$) or $m \in \{0,1\}$ (if $m_1 \barwedge m_2 = 1$). Moving from this ciphertext to the encryption of $m_1 \barwedge m_2$ is then achieved by a simple affine transformation [40].
- The encrypted texts of b_1, b_2, b_3, b_4 are c_1, c_2, c_3, and c_4 respectively using the secret key sk.
- The encrypted "(c1 NAND c2) NAND (c2 NAND c3)" is computed homomorphically.
- The main idea to perform this encrypted NAND computation is to assume that the input ciphertexts are available in a slightly different form i.e. Boolean values, unlike integers or texts. [31]. As an extension of usage of FHEW, the parallelism of GPU is taken advantage and bootstrapping procedure of FHEW is explored. This procedure is clearly mentioned in [41].

2.4 Homomorphic Encryption in practice

This Section discusses the various areas of technology where Homomorphic encryption is used. As there has been an accelerated development since Gentry's proposal of fully homomorphic encryption scheme [5] there applicability of oh HE in different areas technologies that require security also accelerated. The following are the different applications where homomorphic encryption is used.

HE is used in a private cloud medical records storage system (Patient Controlled Encryption) was proposed, in which all data for a patient's medical record is encrypted by the healthcare providers before being uploaded to the patient's record in the cloud storage system. The access to the records and sharing of the secret keys with particular key providers is controlled by the patient (features include a hierarchical structure of the record, ability to search the encrypted data, and various choices for how to handle key distribution) [23]. As we know that by employing homomorphic encryption into technologies, we are able to compute operation on the encrypted data. In this example also, the cloud can perform an operation on the encrypted data like updating, setting alerts etc. based on the data the patient receives.

Paillier cryptosystem is widely used in electronic voting where the basic idea is that each voter broadcasts an encryption of his vote (by sending it to a bulletin board) together with a proof that the vote is valid. All the valid votes are then combined to produce an encryption of the result, using the homomorphic property of the encryption scheme. Finally, a set of

trustees (who share the secret key of the scheme in a threshold fashion) can decrypt and publish [42].

A robust secure data-prevention (RSD) method uses homomorphic encryption and proxy re-encryption method. The RSD method supports all of the following: key generation and sharing, data outsourcing, data accessing, user revocation and user rejoining. This model can overcome the problem of user revocation and rejoining and hence making authorization simple when initiating the handoff process [43]. Another scenario where HE is used is a situation in which both the data and the function to be computed on the data is private and proprietary. Considering an example, the data about corporations, their stock price or their performance or inventory is often relevant to making investment decisions. Data may even be streamed on a continuous basis reflecting the most up-to-date information necessary for making decisions for trading purposes. Functions which do computations on this data may be proprietary, based on new predictive models for stock price performance and these models may be the product of costly research done by financial analysts, so a company may want to keep these models private to preserve their advantage and their investment [23].

With the help of fully homomorphic encryption, few functions such as uploading encrypted data by the customer to the cloud is evaluated privately because the cloud computed privately based on the encrypted input it receives and later the output is returned to the customer by the cloud. Another application of HE is the construction of single server PIR protocol, which is an important application of homomorphic encryption that uses sub-linear communication, from any additively HE scheme. In cryptography, PIR permits users to get certain data from a database stored on the network or the Internet without revealing what he actually needs. The receiver wants the i^{th} value from the database, but the sender must learn nothing about i [7]. Homomorphic encryption in data mining allows a data miner to compute frequencies of values or tuples of values in the customers' data in a privacy-preserving manner. Technically, the problem can be reduced to a problem of securely summing private inputs. The requirements for an efficient protocol in this domain are that each customer only sends one flow of communication to the miner and that no customer needs to communicate with other customers [44]. In typical PPDM (Privacy-preserving data mining) applications, privacy is a twofold problem: preserving privacy in the user-to-data holder and data holder-to-data holder cases, as depicted in Figure 7. While the bottom part of Figure 7 exhibits the framework for considering user-to-data holder, that is, individuals' privacy concerns, the top part shows data holder-to-data holder scenario considering corporate privacy in case of distributed data among data holders. The studies focusing on individual privacy framework concentrate on how user profiles can be collected and processed while ensuring privacy [36].

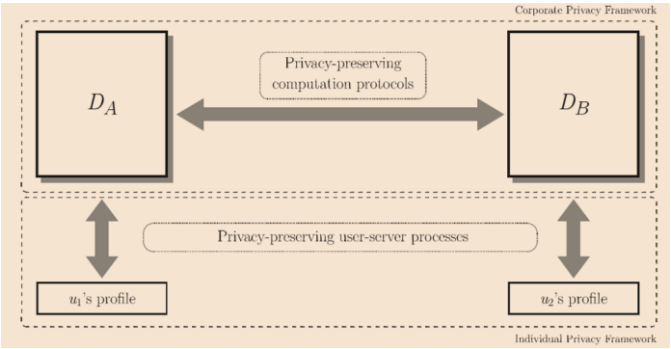

Figure 7: Privacy framework for privacy-preserving data mining applications [36]

Cryptography-based protection of the transmission signals over the communication links is one of the effective approaches to make the networked control system secure. The application of homomorphic encryption in signal transmission by encrypting the controller functions to enhance the cyber-security. A linear controller is encrypted using homomorphic

encryptions of RSA and ElGamal schemes. The most remarkable property of the concept applying homomorphic encryption in signal transmission is that it can encrypt the controller's parameters and the signals inside the controller device, such that the controller need not keep any private keys for calculating the control input, which means that the decryption process is not required inside the controller [45]. HE has novel implications such as being able to carry out operations on database queries in the form of ciphertext and returning the result to the user so that no information about the query is revealed at the server's end [46]. In the financial industry, there is a potential application scenario in which both the data and the function to be computed on the data is private and proprietary. Functions which do computations on this data may be proprietary, based on new predictive models for stock price performance and these models may be the product of costly research done by financial analysts, so a company may want to keep these models private to preserve their advantage and their investment [44].

Also, HE is used in advertisement industry also. For example, when the data is made accessible to some cosmetic company, all the contextual data is encrypted and then uploaded to the server; the advertiser uploads encrypted ads to the server; then the cloud can operate and compute on this data, and the consumer can decrypt the received ad. As long as the cloud service provider does not collude with the advertisers, and semantically secure FHE encryption is employed, the cloud and the advertisers don't learn anything about the consumer's data [23]. Homomorphic encryption is also used in GPUs in which efficient modular multiplication is crucial for the decryption primitive. The other primitives only use modular reduction at the very end of the computations only once. Many cryptographic software implementations employ the Montgomery multiplication algorithm; Montgomery multiplication replaces costly divisions with additional multiplications. Unfortunately, the interleaved versions of the Montgomery multiplication algorithm generate long carry chains with little instruction-level parallelism. For the same reason, it is hard to implement the interleaved Montgomery multiplication algorithm on parallel computing friendly GPUs [18].

There are many areas where Homomorphic encryption is used, nut among all the services, cloud storage system has gained much demand with respect to incorporating HE techniques in the cloud. This is discussed exclusively in the next Section due to its importance and applicability of cloud storage system.

2.5 Homomorphic Encryption in cloud security

It is known that the major services provided by cloud computing are PaaS (Platform as a service) and SaaS (Software as a service), there are four deployment models where cloud computing is developed namely; private cloud, public cloud, community cloud, and hybrid cloud. At present, only data injection to the cloud environment is possible which served enough for accessing and processing the data out of the cyber-space. Nevertheless, this brings about few issues with respect to consistency and correctness of the enciphered data. A noticeable area of application of HE would be cloud computing because based on the individual privacy demands, either small or medium computation can be performed. A sample working of homomorphic encryption in cloud computing is depicted in Figure 9.

Moreover, in cloud computing, all services are delivered through web applications and data that has been outsourced is no longer owned by the users. Shifting all the data and computing resources to the cloud can have implications on privacy and security [47]. An accidental delivery of confidential data is an upsetting problem for most of the cloud storage systems. This vulnerability in cloud computing is called Data Leakage or cloud drain or data loss. This is a very serious vulnerability in cloud computing. If the data store on the cloud is not safe many will be unwilling to use the mobile cloud. Data leakage may occur at the client

side when initiating the handoff process. It may also occur when data in transit i.e. data leakage may occur when a hacker gets access to data via observing the data exchange between client and server [43]. As we know, one of the major problems in Cloud Computing is the fact that the security and confidentiality of a remote resource cannot be technically validated by the customer. As a con- sequence, the security aspect is still subject to contracts and trust. In contrast to the execution of programs, the transmission of data can be well protected by different cryptographic techniques. Asymmetric methods can satisfy a number of requirements in the field of encrypted communication, such as confidentiality, integrity, and authenticity and also the classic multi-party (MP) computation describe a scenario of two parties who want to execute a function with secret input data on each side and a common output [48]. There are many Cloud service providers such as IBM, Google, and Amazon who use the virtualization on their Cloud platform and on the same server that can coexist a virtualized storage and treatment space that belong to concurrent enterprises. For this, the aspect of security and confidentiality must intervene to protect the data from each of the enterprises [49].

Cloud computing is a type of shared computing infrastructures for which Data confidentiality is a key challenge. In order to address this data confidentiality problem, a practical solution named MrCrypt has been proposed which uses homomorphic encryption to ensure data confidentiality by automatically transforming the programs. MrCrypt performs a static analysis of Java programs to identify the most efficient homomorphic encryption scheme supporting the necessary operations on each column of input data, and it then automatically rewrites the program to execute on encrypted data, and instead of encoding each column using a fully homomorphic encryption scheme, one can encrypt it using a more efficient scheme that supports only the necessary operation. Using this method, the problem of securing cloud computations to that of identifying the subset of primitive operations is performed on each column of the input [50]. The architecture of MrCrypt is depicted in Figure 8 for a better understanding of this concept.

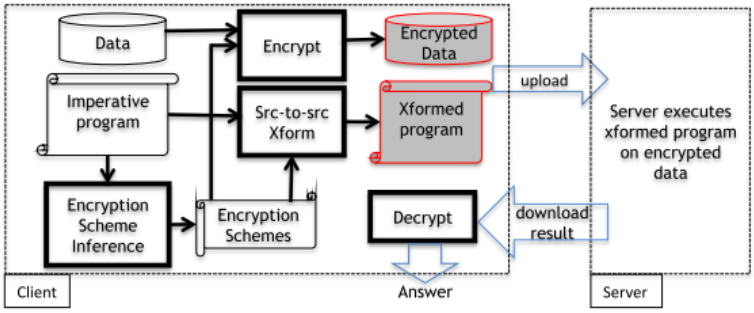

Figure 8: MrCrypt architecture [50]

Another application of HE in the cloud is a multi-key fully homomorphic encryption scheme is the one that supports an unbounded number of homomorphic operations for an unbounded number of parties. In this setting, the parties first send their encrypted input to the cloud, which performs the homomorphic operation and sends the output to the parties. The parties then execute a multi-party computation protocol for joint decryption [51].

Since the data is stored and managed in the cloud, security and privacy settings depend on the IT management provided by the cloud. The CSP (Cloud service provider) typically works with many third party vendors so there is no guarantee how these vendors may safeguard the data which implies that it is not desirable to trust services which use third party vendors [47].

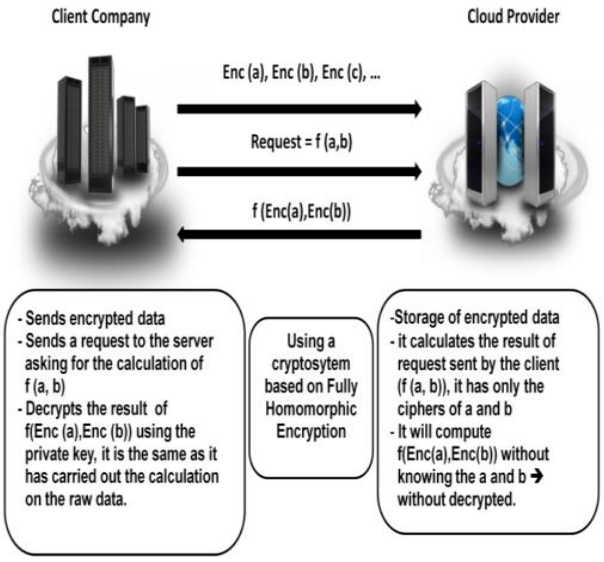

Figure 9: Example homomorphic encryption scheme when applied to the cloud computing [49]

3 METHODOLOGY

This section describes the procedure and the methods used to carry out this research. A detailed explanation this research from the beginning to end is described in the following sections along with the motivation for choosing the research methods to conduct this research, the overall research design, and the data collection methods. The research design is a basic scheme established in a logical step-by-step to bring research questions to research answers. The research methods chosen to accomplish this study are literature review and experiment selected based on the suitability criteria to achieve the aims of the research. The research questions were answered by adopting these research methods. This research is more qualitative than quantitative because of the hindrances caused in the experimentation process i.e. intended quantitative research is performed partially as all experiments could not be run successfully due to few complications faced while performing experiments which are analyzed and deduced as a part of results for this research.

3.1 Motivation

As discussed in Section 1.4, the state-of-the-art of homomorphic encryption is very primitive and hence there is a pressing need to realize the importance and current level of homomorphic encryption in real time application for better application and usage of the technology in the best way possible. An exhaustive Literature review and experiment are the research methods selected for to achieve the objectives of this research. In order to obtain robust results to the extent possible, experiment method is chosen. Homomorphic encryption is a recent and developing topic, it is essential to gain the fundamental knowledge about the topic by studying extensively about it through various scientific papers and books. In order to accomplish one of the aims of this research which are to analyze the performance of various publicly available HE schemes homomorphic encryption, experiment method fits in best to fulfill the requirements of expected outcomes.

3.2 Literature review

A literature review is one of the most commonly used research methods to gain initial insights into the subject area. A meaningful literature review is much more than a collection of summaries of papers or an elaborated annotated bibliography of multiple research manuscripts. It is defined as “the use of ideas in the literature to justify the particular approach to the topic, the selection of methods, and demonstration that this research contributes something new”. It is also noted that for the literature review, “quality means appropriate breadth and depth, rigor and consistency, clarity and brevity, and effective analysis and synthesis” [52]. In general, a literature review has three main levels namely Input, Processing, and Output. The three stages are shown in Figure 10. The research design represents the first step in organizing and planning the research process once the research idea and research hypothesis have been clearly outlined. In order to obtain the most accurate results possible, a very good research plan has to be established. The study of the research knowledge reached by other investigators orients one to decipher the research question in the most critical manner. For a research to be successfully conducted, the research design would have to be clear, appropriately conceived, and based on logical concepts to advance in the best possible way the research concept being developed [53].

Stage 1: Input

The inputs stage is the initial step before entering the processing phase where the input refers to all the literature i.e. the scientific papers, research articles, books, conference papers,

research manuscripts etc. As there is deemed access for the research to a lot of online scientific research databases all the input scientific literature has been fetched based on their availability online. The literature has been searched by suing keywords such as 'Homomorphic encryption*', 'homomorphic AND security', 'Cloud computing AND homomorphic encryption', 'Homomorphic OR encryption in cloud', 'DGHV and homomorphic encryption', 'BGV or HElib', 'FHEW and homomorphic encryption', and 'Paillier AND cryptography'. The literature search is extended and improved by altering the search string depending on the requirement of the literature fetched and needed. The required literature has been extracted from IEEE Xplore, Springer link, ACM Digital Library, Inspec, Inderscience, Safari Tech books online, and Science Direct. These inputs are searched and selected based on the relevance to the topic. As part of the research objective, for conducting the experiment of Homomorphic encryption schemes, publicly available libraries for the DGHV, Paillier, FHEW, and HElib are chosen from Github which is an open-source code-search library. As homomorphic encryption is a recent topic, many databases and online resources were searched in order to fetch relevant material required for this research.

Stage 2: Processing

The second stage consists of six steps to process, which are a) Know the literature, b) Comprehend the literature, c) Apply, d) Analyze, e) Synthesize, and f) Evaluate. The literature in Stage 1 is now processed before going to the final stage that is output. This stage helped in formulating the research questions and setting up research objectives and goals. Exhaustive literature review and a bibliographic search resulted in extraction of appropriate material needed for this research. Cryptography being a vast subject area, and homomorphic encryption being a growing topic in cryptography, literature search resulted in literature corresponding to different areas of cryptography. As many scientific literature were retrieved, the relevant literature was selected by reading the abstract and further reading the full text if its relevance to the topic was more. The literature pertaining to each research question were synthesized and evaluated. This second stage of the whole literature process is iterative based on the requirement of the literature for the research.

Stage 3: Output

The literature retrieved in the previous step resulted in a solid theoretical foundation for this study. It helped in answering RQ1, RQ2, and a part of RQ3. The literature found is used and fit into the research with respect to the scope of this study for fulfilling the objectives of this research that needed a theoretical base.

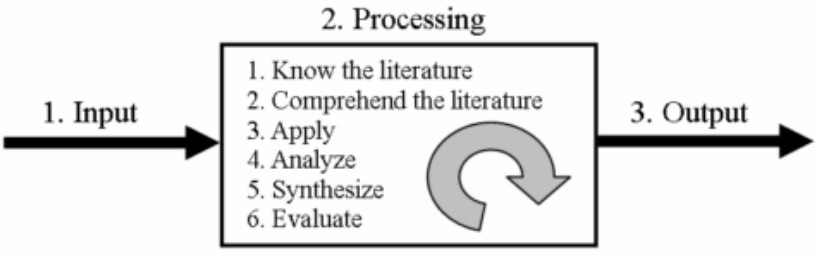

Figure 10: Literature review process [52]

As stated in [52], effective literature review should include the following characteristics: a) methodologically analyze and synthesize quality literature, b) provide a firm foundation to a research topic, c) provide a firm foundation for the selection of research methodology, and d) demonstrate that the proposed research contributes something new to the overall body of knowledge or advances the research field's knowledge base.

3.3 Experimentation

This section discusses the method used for answering RQ3 and RQ4. The publicly available online code-search library Github is used for obtaining the source code for the homomorphic encryption schemes DGHV, Paillier, HElib, and FHEW. The RSA encryption library which is an asymmetric key algorithm is also retrieved from Github online repository. The experiments were conducted according to the availability and working of the libraries as there were very few publicly available working implementations of HE algorithms.

3.3.1 Testing environment and experiment execution

All schemes were experimented in Oracle virtual machine VirtualBox 5.1.10. As each scheme required a different platform for its execution, the Virtual machine is chosen such that the respective platforms can be created in order to run the libraries and also monitoring the resources utilized by the schemes when run can be measured conveniently.

3.3.1.1 RSA

RSA asymmetric key encryption algorithm, whose description is discussed in Section 2.2.1, is tested on Ubuntu with a help of Oracle Virtual Box 5.1.10. The library is retrieved from Github [54] and implements RSA public-key algorithm written in Java. This library is imported to the VM and run. The library facilitates the encryption, decryption, key generation required for the algorithm. As the library is written in Java, the algorithm is simple to run using the terminal. To know the impact of the algorithm on CPU and Memory utilization, the source code is modified in such a way that it takes different sizes of input for encryption and then decrypts it. The scheme is run for multiple times in order to get appropriate results. The scheme is iterated with various input sizes of 20KB, 40KB, 60KB, 80KB, AND 100KB so as to mimic real time encryption. The input files contain randomly generated numbers that are fetched each time the program executes. CPU and Memory utilization by RSA is measured every 1 second and is stored in a file for further analysis. The monitoring of the resource used by RSA when run is measured with the help of a shell script written to measure CPU and Memory utilization.

3.3.1.2 DGHV

The DGHV scheme, whose description is discussed in Section 2.3.1 is tested on a virtual machine i.e. Oracle VM VirtualBox 5.1.10. The library that implements DGHV scheme is obtained from Github [55] which is an open-source online repository for source-code. The DGHV library is imported to facilitate encryption, decryption, key generation, addition, multiplication and ciphertext refresh procedures. These are implemented using Sage 4.7.2 mathematical library. Sage is employed within the VirtualBox with requirements of 1GB ram and minimum 7GB free disk space. The following steps help in explaining the environmental setup and implementation of DGHV scheme.

The library contains various files for running and executing the DGHV scheme. The file **dghv.sage** should be loaded from the library and run. Other files like **scalprod.spyx** and **utils.sage** should be imported in this file. The files **directscale.c** and **scalprod.spyx** are used for faster implementation of ciphertext expand. In order to demonstrate encryption, decryption, key generation, add, mul etc. testPkRecrypt() functions are used. These functions allow four set of parameters namely toy, small, medium and large. The functions for the scheme are encrypt(), recrypt(), decrypt(), add(), mult() etc. which are described in [55]. The inputs for these include plain text, theta, n, secret key, ciphertext etc. as described in section 2.3.1. For

preventing the occurrence of malloc error, the encrypt () function in static methods which is processed in two passes.

Due to the issues in the Sage library, the main file “dghv.sage” could not be loaded leading to interruption to the experimentation. Also, errors appeared while loading the other file “scalprod.spyx” despite the installation of all the dependencies and files required for the execution of DGHV scheme. The available source code for implementation of this scheme is best of all the few publicly available implementations but still failed to properly implement the scheme when attempted run in Sage on different platforms such as Mac OS, Ubuntu, and Windows. There were also few issues with the Sage such as not able to detect the command because the keyboard settings set by the Sage do not match with the normal English-US or English-UK keyboard. IT gives an unexpected error of invalid command even though few other keyboard formats were tried.

3.3.1.3 Paillier

Paillier scheme is one of the most commonly used Partial homomorphic encryption algorithm that is deployed in many applications such as electronic voting and electronic cash. There were quite a many Paillier cryptosystem implementations in different languages such as Java, C++, Python, JavaScript, and Matlab. Because of much familiarity and feasibility with Java language, Paillier implementation in Java has been selected for execution. The Java implementation of this scheme is retrieved from Github online source-code repository [56]. Ubuntu platform is selected for executing the Java implementation of the Paillier cryptosystem. As Java being one of the most commonly employed programming languages, it was not much difficult to understand the implementation of the scheme. The result obtained by running the Paillier code is discussed in Section 4 along with graphs showing the resources utilized during the program execution. This input for this scheme is integers. The source code is modified in such a way that the input is of various sizes to represent real-time encryption to the extent possible. Input files created are of 20KB, 40KB, 60KB, 80KB, and 100KB containing randomly generated pairs of integers as per the algorithm requirements and characteristics.

A much more detailed explanation about all the functions, parameters and expressions can be found in [56] containing the various important functions in the Paillier cryptosystems that help in understanding the working of the scheme. Along with the implementation of Paillier scheme in different languages, there were different variants of Paillier cryptosystem available online for research usage, but the most basic variant of the scheme is chosen for better realization of the scheme usage considering the scope of the feasibility of the cryptosystem.

3.3.1.4 HElib

HElib stands for homomorphic encryption library. It is a generic, open-source framework for the testing of homomorphic encryption schemes that also supports multi-threading. The open-source implementation of HElib is fetched from Github online repository [37]. HElib is a tool can generate a diverse set of circuits using various gate types. It uses Python script that can generate circuit descriptions and inputs based on configurable circuit parameters. Circuit descriptions are output in ASCII-format and stored as text [57]. Currently, the implementation of the Brakerski-Gentry-Vaikuntanathan (BGV) scheme is only available in HElib. The library is written in C++ and provides low-level routines like “add, mul, set” etc. for the scheme. To build the HElib library, two additional libraries namely NTL and GMP have to be installed. This can be done in Linux or in virtual box for Windows OS. The library is provided with a Readme file and Install text file to provide the users with installation guidelines. NTL is a

mathematical library for performing number theory. It is a portable C++ library that can compiled in thread safe and exception safe modes. For building HElib, NTL of version 9.4.0 or higher must be downloaded. GMP is a multiple precision arithmetic library by GNU for performing arithmetic operations without limitations. It has set of functions and interfaces for that functions. It usually operates on Unix based platforms along with Windows 32-bit and 64-bit platforms.

Installation and Usage:

Initially GMP and NTL libraries have to be installed. Both the libraries work better in Unix based platforms so the virtual box is installed for Ubuntu in windows systems. The libraries are uncompressed and installed one after the other. In the terminal, the directory should be changed to gmp. After changing the directory, we need to configure the library with the command "**./configure".** Then the **make** command is used to make the files in the library and finally installing by using the command "**sudo make install."** These steps install the library in the local folder of users i.e. /usr/local.

NTL library also is installed by uncompressing the downloaded zip file and changing the directory in the terminal to the library source of the uncompressed file. To configure the library the command "**./configure NTL_GMP_LIP=on"** is used followed by the **make** command and the **sudo make install** command. This will install the NTL library in /usr/local folder. Now HElib can be built by changing the directory to src (source) of HElib and then using the commands **make** and **make check** where "make" compiles and builds the "**fhe.a"** library and "make check" compiles and runs a set of test programs.

The HElib library contains of three layers namely math, crypto, and data movement layer. It uses NTL library for mathematical operations. The basic operations provided in this scheme are key generation, encryption, decryption and few homomorphic encryption routines like addition, multiplication, "ciphertext maintenance" etc. In the file Test_general.cpp a mix of operations are used on four ciphertexts. The functions like add(), mul(), rotate() and sub() makes uses of ciphertexts c1, c2, c3, c4, secret key, encrypted array etc. accordingly as parameters.

The library executes the already existing programs and routines and does not facilitate modification of the existing programs for compilation with different inputs to analyze the performance. Moreover, addition of any extra programs to the library for compilation of basic programs was also not supported by the library gave raise to errors. With respect to flexibility of performing additional program executions in this library, HElib is still at a development stage and needs much more improvement.

3.3.1.5 FHEW

FHEW is an open-source software under terms of General Public License. It makes use of FFTW (Fastest Fourier Transformation in the West) and is based on Fully homomorphic encryption scheme. homomorphic evaluation of arbitrary Boolean circuits on encrypted data using a public (evaluation) key is supported by encryption and decryption of single bit messages. The implementation library is taken from an online source-code repository called Github [58]. For installing the FHEW, FFTW 3 library is required which is written in C. It should be downloaded and run in a C++ compiler. In the terminal, the directory is changed to the FFTW folder and is installed using different commands. Initially, we need to configure the command "**./configure"** and then use the command "**make"** which will provide a set of test programs to run and also a libfhew.a library. The header files are installed with the command "**make install"** and the library can be tested by running the program called "**fhewTest".**

The library provides operations such key generation, encryption, decryption, NAND operation etc. The files “sec.key” and “ev.key” are used for storing secret key and evaluation key respectively. Here homomorphic NAND is performed on the ciphertext of messages o and 1 stored in files “a.ct” and “b.ct” respectively. The resulting ciphertext is stored in “c.ct” which has to be 1. The readme file in the library explains clearly about the usage and commands to be run for the scheme. FHE implementation which is represented by the recent HElib, reports a bootstrapping/refreshing procedure with running times around 6 minutes. The main improvement in this scheme with respect to its previous work is in terms of granularity and simplicity. It is effectively shown that macroscopic delays are not a necessary requirement of bootstrapped FHE computations, and bootstrapping itself can be achieved at much higher speeds than previously thought possible [40].

The function setup() is used to setup the library interface. The functions encrypt(), decrypt(),KeyGen(), homoNAND() etc. are used for encrypting the plaintexts, decrypting the ciphertext, generating the secret key, performing homomorphic NAND operation respectively. The input messages are 0 and 1 while the output should be the NAND of both i.e. 1. The scheme involves five main steps namely, key generation, encryption of file ‘a’, encryption of file ‘b’, performing NAND operation of encrypted files ‘a’ and ‘b’, and decrypting the file ‘c’ where the result of NAND operation of file ‘an’ and ‘b’ is stored. The library has many specifications to be satisfied by the platform and also the supporting libraries that need to be installed. Even though all the conditions and dependencies were installed, the library could successfully run only the first step of al whole scheme which is the key generation when running on different platforms including Mac OS. The other steps, when attempted to run, error with label “command not found” was noticed which exhibited that the library is not so well built for successful implementation. Because of these issues, further execution of extra intended programs also could not be executed resulting in impracticality nature of the library.

3.3.2 Metrics

The metrics chosen for evaluating the schemes are CPU and Memory utilizations. These are chosen because of the unsuccessful experimentation due to the issues faced when attempted to run the libraries as discussed in the previous section. As all the schemes are different with respect to encryption, decryption, key size, key generation, ciphertext size etc. CPU and Memory utilization serves as a general metric that can be applied to all. Only two schemes i.e. RSA (Public-key encryption) and Paillier (Homomorphic encryption) were successfully run and measured using these metrics, other schemes viz. DGHV, HElib, and FHEW were not measured as they were not successfully run because of few complications faced.

3.4 Exclusion of alternative methods

The research methods chosen for this study are exhaustive literature review and experiment. Other research methods such as surveys, questionnaires, and case study etc. which are based on opinion and observation respectively are not selected. Typically, surveys gather data at a particular point in time with the intention of describing the nature of existing conditions and used to scan a wide field of issues, populations, programmers etc. which also needs careful sampling [59]. This method is not chosen because selecting a sample for this methods is challenging as people working with HE in real time and people being aware of this particular cryptographic domain are less and also difficult to find. A questionnaire is an instrument for collecting survey information being able to be administered without the presence of the researcher [59]. As questionnaire is usually a later step after survey, it is not selected also because it is not suitable for this research to achieve the research aims.

A case study provides a unique example of real people in real situations and is a specific instance that is frequently designed to illustrate a more general principle [59]. This method is

not chosen as the objectives of the research do not deal with real people in a real situation and this study is more about analyzing and assessing the existing libraries to achieve the aims of the research.

As stated in Section 1.4, the average interest on Homomorphic encryption over a period of 9 years is ~45% as per the latest Google Trends result. This means that researchers working in this subject area are comparatively low than other cryptographic domains as the research is being continued and effective new concepts in HE domain proposed is relatively less. Due to this, the risk of acquiring inaccurate and improper results based on the sample selected is more and is avoided to the extent possible by choosing exhaustive literature review and experiment as research methods. A comprehensive understanding of HE and its schemes was required to carry out the research, and as the algorithms were more inclined to some mathematics finding people to contribute for surveys or questionnaires was challenging which is one of the reasons behind not selecting those research methods. Hence, in order to prevent risks of replicated and falsified results as a consequence of selecting other research methods, literature review and experiment methods are selected which are most suitable to fulfill the research objectives.

3.5 Data collection

In order to achieve the objectives of the research, the data collection methods employed are described in this section. The data for this research extracted particularly from an exhaustive literature review. Also by conducting experiments, qualitative data has been collected and limited quantitative data is gathered due to partially successful experiments. Data is collected in abundance in a qualitative study, and the analysis has as its main focus to reduce and organize data to provide a chain of evidence for the conclusions [60].

The literature obtained by searching various scientific databases, as discussed in Section 3.2, using different search keywords and strings served as a strong foundation for answering the research questions RQ1, RQ2, and partially RQ3. Successful running of all HE schemes was required in order to answer RQ3 completely. RQ4 predominantly needed successful experiments in order to answer them. Paillier and RSA encryption algorithms were successfully run and the results of the experiment are presented using graphs in Section 4. The data obtained after measuring the resources (CPU and Memory) were saved to file as a log of the resources used for every second. This was done by writing a Linux script for monitoring resources utilized by a particular process. Although the experiments were partially successful, theoretical results and data are obtained based on the amount of work performed on experiments with the libraries as discussed in Section 3.3. As mentioned earlier, because of the unsuccessful running of few libraries, all experiments could not be conducted properly as intended in order to fetch desired results for accomplishing the objectives of the research.

4 RESULTS AND ANALYSIS

The methods used to answer the research questions of this study have been discussed in the previous section and the results gained by opting those research methods are presented and discussed in this section along with investigating the obtained results. From the conducted exhaustive literature review and experiments, the results obtained are briefed as follows: Extensive knowledge on working of homomorphic encryption and its application is gained. In addition to this, the working of homomorphic encryption schemes; DGHV, Paillier, HElib, and FHEW is perceived by running the open source libraries of these schemes. Public-key encryption algorithm RSA and Paillier are successfully run and measured CPU and Memory utilized by the schemes during their execution. The robustness and complexity of the schemes are also identified. The exhaustive bibliographic research resulted in the derivation of theoretical performance measures for the schemes. The scheme E is homomorphic for a class C of circuits if it is correct for all circuits $C \in C$, whereas E is fully homomorphic if it is correct for all Boolean circuits and a SHE possesses only a few homomorphic properties such as it can perform only a limited number of operations, and the ciphertexts compactness requirement might be violated [61].

A generalized metric for any encryption algorithm is CPU and Memory utilization. In terms of technical metrics, the performance metrics for homomorphic encryption schemes and public-key encryption schemes are key size, the amount of noise produced after encryption, time is taken to encrypt, decrypt and generate the key, size of the ciphertext, and the time taken to break the encryption which is usually done by cryptanalysis. It is also perceived that PHE, LHE, and SHE can be converted to FHE with a series of steps such has bootstrapping and refreshing etc. to become fully homomorphic. In order to decrease the key size, there are various key compression techniques developed for reducing the noise in the encryption which raised due to the large size of the key. By exploiting the redundancy of value representation in the Paillier cryptosystem, reversibility of data embedding can be achieved to further exploit the redundancy of data representation in the Paillier cryptosystem, value expansion can be iteratively conducted on the cipher value as discussed in[62]. In Pailliers, one disadvantage is observed; sharing private key parts is that the storage and transmission of the parts of keys requires an amount of storage and bandwidth resources equivalent to the number of used key parts between the delegator and the involved workers [63].

The noise plays a very pivotal role in homomorphic encryption because the noise increases with each encryption performed and the encrypted text can be decrypted correctly only when there is very less noise. As ciphertext size is also a performance factor, performing more number of refresh to the ciphertext, there can be unlimited number of homomorphic encryption that can be done and hence it can result in a FHE. In the squashing technique, the encrypter starts decrypting i.e. to reduce the decryption complexity without affecting the "evaluative capacity" of the scheme at all, the encrypter is bale to start decryption, thereby easing the burden on the decrypter [5].

Table 1 shows a general comparison of FHE and PHE schemes in terms of severity with respect to speed, ciphertext size, and flexibility. From Table 1, it can be inferred that for partially homomorphic encryption schemes flexibility is the main problem whereas, for FHE schemes, speed and ciphertext are more of a concern than flexibility.

Table 1: Partial VS Fully homomorphic encryption schemes

S.No	Type	Speed	Ciphertext size	Flexibility
1	Partial HE	Fast	Small	Low
2	Fully HE	Slow	Large	High

Regarding performance, ciphertexts in Gentry's scheme remain compact in so far as their lengths do not depend at all on the complexity of the function that is evaluated over the encrypted data. The computational time only depends linearly on the number of operations performed. However, the scheme is impractical for many applications, because ciphertext size and computation time increase sharply as one increases the security level. To obtain 2k security against known attacks, the computation time and ciphertext size are high-degree polynomials in k [64]. Table 2 shows the generalized cost incurred for various homomorphic operations for homomorphic encryptions schemes where constant-add or constant-multiply refers to multiple times of performing add and multiplication respectively. By performing an exhaustive bibliographic research, it is perceived that the existence of very efficient MPC (Multi-party) protocols for circuits containing a large number of additions, and few multiplications, suggests that multiplications might be way more expensive than additions. However, there exist encryption schemes which are multiplicatively homomorphic, for example, ElGamal, in such cryptosystems, but additions cannot be performed unless a fully homomorphic scheme is used and multiplications come essentially for free [51]. The encryption can provide safety for the sensitive data stored on the cloud servers, and reduces identity theft. However, these issues limit the re-distribution of new keys to authorized users during the initiation of the handoff process [43].

Table 2: Homomorphic encryption and their cost

S. No	Operation	Noise	Time
1	Add	Low	Low
2	Constant add	Low	low
3	Multiply	High	High
4	Constant multiply	Medium	Low
5	Rotation	Low	High

The other experimental part of the research was unsuccessful because of the problems incurred while trying to run the libraries of the schemes. On attempt to use HE schemes in real time, the libraries did not work as intended which resulted in unsuccessful experimentation and are discussed as below.

DGHV

The fully homomorphic version of DGHV is possible only after DGHV being SHE first. In [61], some security bounds associated with DGHV is explained and from that, it can be inferred that the dependent dimension for the secret key is the threshold. The problem incurred with DGHV scheme is that the Sage library did not allow loading/importing of the file "dghv.sage" into the sage console for execution of the scheme which may be because of lack of appropriate documentation of the library. The issue is also continued with the file named "scalprod.spyx" This issue occurred was attempted to solve by;

- Running the sage library on different platforms including Mac OS, Ubuntu, and Windows.
- b) Changing the input keyboard format to various languages in order to make the library accept the command according to its functions in it software built for accepting input from the keyboard i.e. to resolve keyboard interrupt error.

- Carefully checking all dependencies and requirements needed to run the schemes. The severity of this problem is high because of which the experiment could not be successful.
- Attempting to solve the issue based on the solution provided in relevant websites. Few of the websites searched for a solution are as follows:
    a) https://ask.sagemath.org/question/10745/load-sage-file-into-sage-with-virtualbox-windows-8/
    b) https://ask.sagemath.org/question/24531/running-a-sage-file/
    c) https://github.com/sagemath/sagenb/issues/169

Despite the solutions provided by various websites and trying all the possible solutions, the execution of the library was not successful.

HElib

The problem faced in running the HElib library is that "invalid command" error was identified when tried to modify the existing programs and run them. Additionally, commands for executing the programs did not work when tried to add a new program to the existing library and run that newly added file which interrupted the further steps that were ought to be executed. The interruptions were attempted to solve by compiling the modified programs of the already existed in the library by using various commands of execution on different platforms. Moreover, all the dependency issues with respect to the HElib library were carefully checked and fixed and the severity level of the issue was moderately high. Thus it can be inferred that other than just running the existing library for BGV scheme i.e. HElib library, it is not possible to add any extra programs for execution or to modify the existing programs in the library.

FHEW

The problem faced while executing PHEW library is the occurrence of "Command not found error". Out of the major five steps of the algorithm as discussed in Section 2.3.4, it was not possible to execute the encryption, decryption, and NAND operation steps which may be because of the poorly built library i.e. not able to accept the relevant commands. In order to fix the encountered issues, changes were in the commands used and dependency checks according to the library were done by considering the solutions available on the web that had been posted by users who had already experienced with this library. Despite trying the solutions, the problem still pertained. The severity of solving the issue of this problem is also high as that of DGHV.

Paillier

The issue with Paillier was that it did not support division and other operations, except addition and multiplication. Other expressions other than simple integer addition and multiplication in terms of just addition and multiplication were tried but the solution was not forthcoming.

4.1.1 Paillier and RSA results and analysis

As mentioned earlier, Paillier and RSA schemes were successfully run and measured. The results obtained by conducting an experiment for Paillier and RSA are plotted in graphs for better understanding through graphical representation. Five inputs size of 20KB, 40KB, 60KB, 80KB, and 100KB are taken which are chosen to represent real-time encryption. The Fig. 11, Fig. 12, and Fig. 13 shows the average execution time, average CPU utilization, and average Memory utilization respectively for Paillier and RSA schemes. The memory of the OS (Ubuntu) in the VM is 1024MB which is 1GB.

Figure 11: Avg. Execution time of Paillier and RSA

From the above graph, it can be inferred that the execution time of both the schemes Paillier and RSA follow some trend by increasing linearly with an increase in input size i.e. from 20KB to 100KB. The time is taken for HE i.e. Paillier is more when compared to Public-key encryption RSA which is more than double of the time taken by RSA.

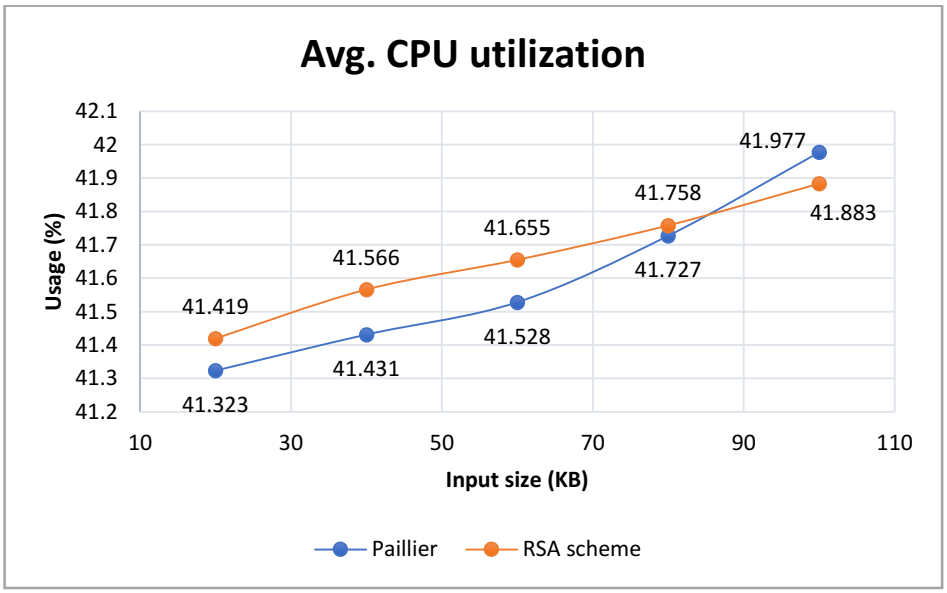

Figure 12: Avg. CPU utilization of Paillier and RSA

From the above graph of Avg. CPU utilization of Paillier and RSA, it can be inferred that as the input size increases, the CPU utilization also slightly increases linearly for both the schemes. The range of CPU utilization percentage for both schemes is same i.e. 41% - 42%. The percentage of CPU utilized for the same size of inputs by both the schemes is approximately equal.

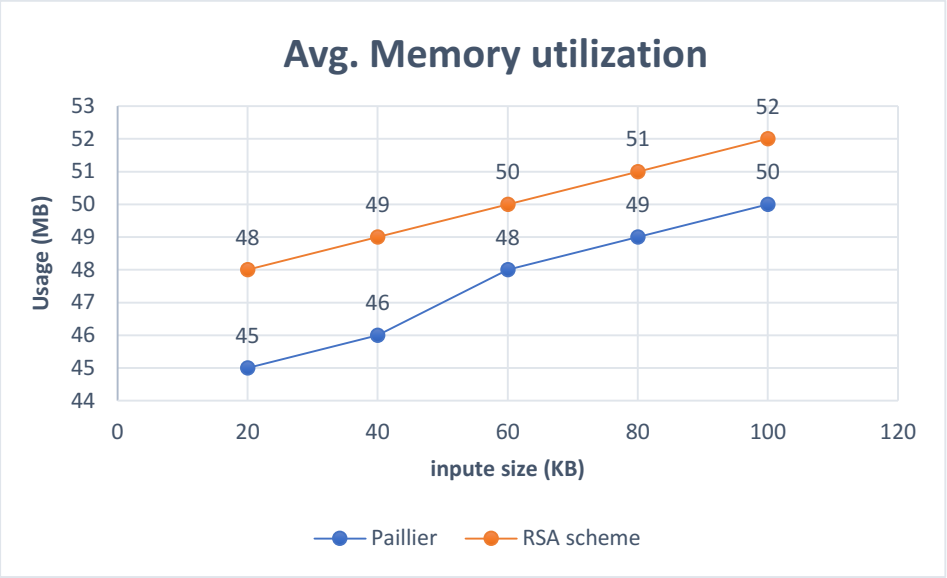

Figure 13: Avg. Memory utilization of Paillier and RSA

From the above graph, the memory utilized by Paillier and RSA during their execution increases linearly with increase in the input size. For RSA it can be stated that memory utilization is directly proportional to input size but for Paillier direct proportionality does not exist as the increase in memory utilized is not same for each increase in the input size i.e. there is a sudden increase from 46 to 48 which is a difference of 2MB for an input of 60KB which is not observed in other inputs. The amount of memory used for each input size for both the schemes is relatively similar.

Fig.14, Fig. 15, Fig. 16, and Fig. 17 shows the Box Plots for CPU and Memory utilizations respectively for Paillier and RSA schemes. A Box plot is a five-number summary, which can be used to display the center and variation of the outcome values in the dataset through graphs. The five-number summary includes a minimum value, maximum value, and quartiles in the given dataset [65]. In the Fig.14, Fig. 15, Fig. 16, and Fig. 17 the green dots represent the average values of CPU utilization and Memory utilization respectively for Paillier and RSA. The experiments are run for ~10 times to get accurate values.

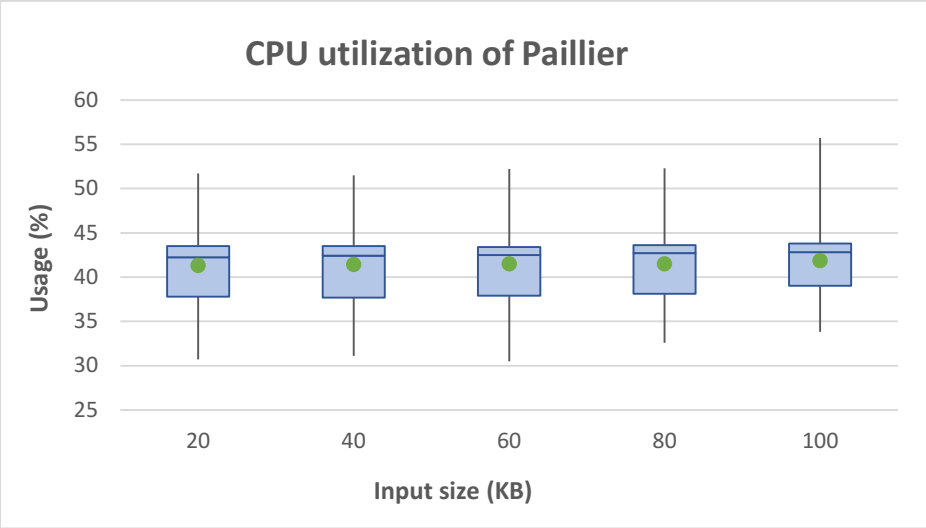

Figure 14: Box Plot showing CPU utilization of Paillier

From the above boxplot, the minimum and maximum CPU utilization of Paillier is 30.5% and 55.7%. The average CPU utilization by all the series (20, 40, 60, 80, and 100) is ~41.5%.

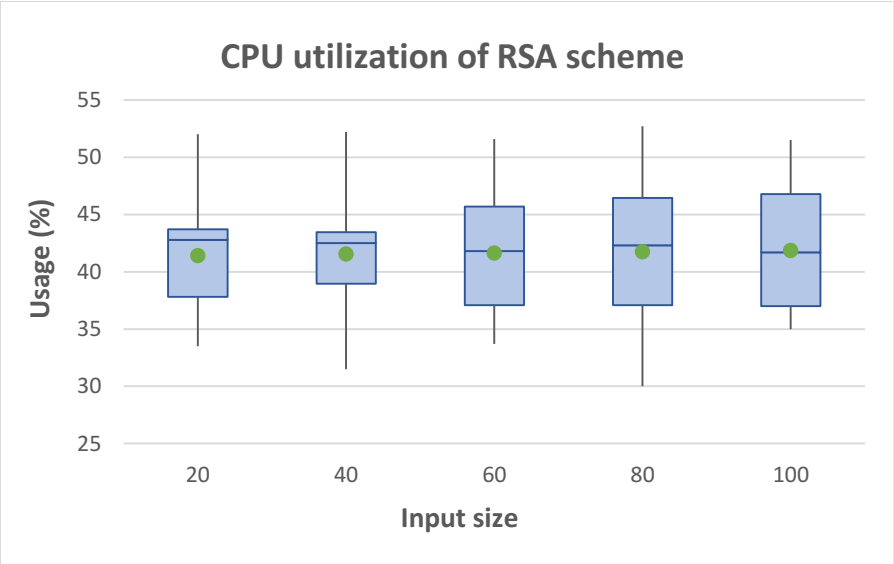

Figure 15: Box Plot showing CPU utilization of RSA

The minimum and maximum CPU utilization of RSA are 30% and 52.7% respectively. The experiment is run multiple time ~10 times and then CPU for only RSA is measured. The average CPU utilization for all the series (20, 40, 60, 80, and 100) in the box plot is ~41.5%.

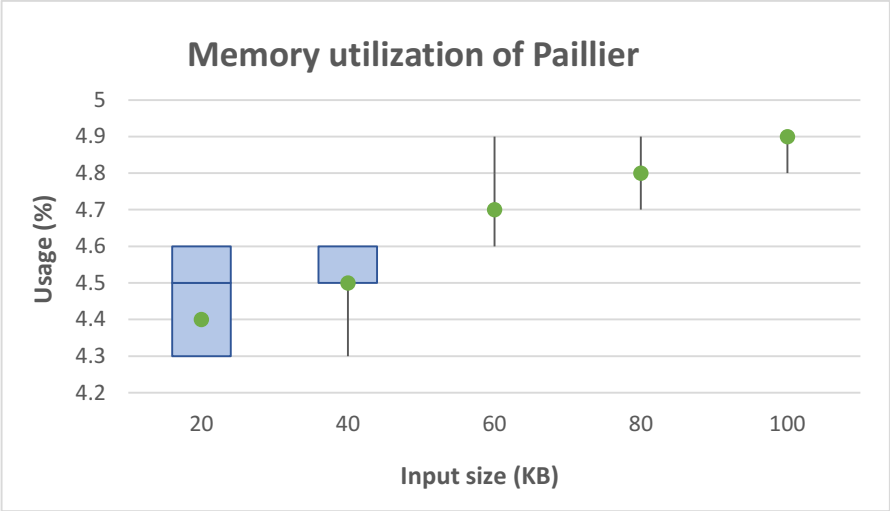

Figure 16: Box Plot showing Memory utilization of Paillier

From the above boxplot, the minimum and maximum Memory utilized by Paillier are 4.3% (~44MB) and 4.9% (~50MB) respectively. The average memory utilized by all the series (20, 40, 60, 80, and 100) is ~4.6% which is ~48MB of memory. It is also observed that there is not much significant difference in the memory utilized due to which there are few elements of the boxplot are not seen.

Figure 17: Box Plot showing Memory utilization of RSA

From the above boxplot, the minimum and maximum memory utilized by RSA are 4.6% (~47MB) and 5.1% (~52MB). The average memory utilized by all series (20, 40, 60, 80, and 100) is ~4.9% which is ~50MB of memory. From the boxplot, it is observed that there not much significant difference in the memory utilized by RSA due to which few elements of the boxplot are not seen.

5 DISCUSSION AND LIMITATIONS

The research conducted primarily helped in analytically analyzing the working of homomorphic encryption and homomorphic encryption schemes. The impact of HE algorithm and Public-key algorithm on CPU and Memory is also perceived. In addition, the study also helped in realizing the various real-time areas of application of homomorphic encryption and its increasing demand in the cloud storage systems. This section discusses the answers to the research question formulated various threats to validity followed by the limitations of the research. From the results obtained from the experiment of Paillier and RSA, which are presented in the previous section, it is analyzed that with an increase in the input, there is a linear increase in the execution time, CPU utilization, and memory utilization for both the schemes. The graphs in the previous section depicting the results of the experiment clearly explain the results graphically.

5.1 Discussion on answers to the research questions

RQ1 ***How does Homomorphic Encryption work and what is its importance?***

This research question is answered with the help of conducted literature review and by doing an exhaustive literature review and extensive bibliographic research. The working of homomorphic encryption is well explained in Section 2.2.2 and the importance of it can be realized through Section 2.4 and Section 2.5. There exist various explanation variations of homomorphic encryption in different literature, so the fundamental working of the basic homomorphic encryption is explained along with the variants of homomorphic encryption. The different uses of various types of homomorphic encryption are realized based on its application in a real-time scenario.

RQ2 ***What are the differences between Homomorphic Encryption and Public and Private key encryption in terms of functionalities and performance?***

Homomorphic encryption can be realized as the advance level after public and private key encryption because of the aspect that it can compute arbitrary computations on the enciphered text. This research question is answered by the help of literature review conducted. The literature study formed as a strong theoretical base that helped answer this research question. The sections 2.2.1 and 2.2.2 best answer this research question and is inferred that much of the difference in public key cryptography and homomorphic encryption lies in performing operations on encrypted data unlike in public key encryption.

RQ3 ***What are the differences between the selected Homomorphic Encryption schemes?***

This research question is answered by conducting both literature review and experimentation. The Sections 2.3 and 3.3 explains this well. Only qualitative differences among the schemes could be found out as the experiments did not run as intended. Out of the four selected HE schemes which are DGHV, Paillier, HElib, and FHEW, only Paillier cryptosystem is successfully run. It can also be noticed the implementation of the homomorphic encryption libraries for different schemes is different and with different levels of complexity and routines. Also, the size of the ciphertext, ciphertext key, execution circuit are different for all schemes.

RQ4 ***What is the impact of Homomorphic encryption and Public-key encryption on CPU and memory utilization?***

The CPU and Memory utilized by HE and Public-key encryption are measured to know the impact of these encryption schemes on CPU and Memory. Due to partially successful experiments, there is a lack of appropriate evaluation of all HE schemes based on CPU and

Memory utilization. Only Paillier encryption scheme and RSA (public-key encryption) are run successfully and measured CPU and Memory utilizations. The answer to this can be understood through Section 4 where the graphs representing the results of experiments (Paillier and RSA) are shown. The other selected HE schemes are only theoretical evaluated and not practically evaluated due to the unsuccessful running of the libraries.

5.2 Threats to validity

5.2.1 Internal validity

Internal validity is of concern when causal relations are examined. When the researcher is investigating whether one factor affects an investigated factor there is a risk that the investigated factor is also affected by a third factor [60]. As the researcher is a beginner, there is a convincible liability of poor data investigation leading to inaccurate results and conclusions. Also, the experiments are conducted based on the available resources. This threat has been noticed because of the errors faced in the experimentation process. By repeating the experiments on various platforms, this threat has been mitigated to some extent. Moreover, in order to handle the risk of poor data analysis, the countermeasure taken is consulting the supervisor and good foundation of literature searched helped in validating the elicited conclusions.

5.2.2 External validity

This aspect of validity is concerned with to what extent it is possible to generalize the findings, and to what extent the findings are of interest to other people outside the investigated case [60]. Because the experiment is conducted based on the available environments, the results of the experiments can be generalized to other platforms resulting in risk of generalizability to some extent. This was mitigated to some extent by confirming the quality and usage of the open-source libraries. Generalizability is addressed by referring adequate information related to homomorphic encryption and schemes and setting up connections between related work and the current work.

5.2.3 Reliability

Reliability is an aspect that is concerned with to what extent the data and the analysis are dependent on the specific researchers which are if another researcher, later on, conducted the same study, the result should be the same [60]. The study conducted is somewhat repeatable which means that the conducted study is reliable to some extent. As the researcher is a novice, there is the possibility of getting successful experimentation results when the same is conducted by other researchers.

5.3 Limitations

Bounding to the intricacy of the problem area of this research, there is possibility that the scope of this research is misconstrued. Firstly, the scope of this research is limited to experimenting only four HE schemes and as although widely used in various subject areas and applications being developed gradually, the availability of publicly available resources implementing HE is very limited. The experiments conducted could not fetch desired results due to various factors concerning with the implementation of libraries and dependencies. In addition to these, the other HE schemes other than Paillier are not evaluated quantitatively based on CPU and Memory utilization.

These limitations make the researcher to be confined to a limited scope of accomplishing the objectives of the research and gaining the benefits of the results of the research.

6 CONCLUSION AND FUTURE WORK

This section describes the closure of the complete research along with the future work that can be done on this topic. The past, present, and future of homomorphic encryption are also briefed in this section.

6.1 Summary and conclusion

Encryption is paving its path in all areas of technology and also mobile applications such as WhatsApp, iCloud, Dropbox, iMessage etc. The increased penetration of social media and smart devices in our lives certainly out our privacy at risk [15]. Homomorphic encryption is the process of performing computation on the enciphered data without revealing the original data. The dominant aim of the research was to discern the working and application of Homomorphic encryption and HE schemes. The research began with finding a gap in the selected research topic by performing an extensive literature review. The research question formulated for carrying out this research are mentioned in Section 1.6 which helped in determining the possible outcomes of the study. The objectives of the research were set based on the research questions formulated. The objectives and expected outcome of the research as discussed in Section 1.5 and 1.7 are fulfilled with respect to the results obtained by carrying out the research methods chosen. The flow of the research for grasping the relevant knowledge about HE started with perceiving the emergence of cryptography, traditional cryptography, and ended with Homomorphic encryption and its advances. The ancient cryptography was concerned primarily with encryption using simple transposition and substitution methods. Public and private key cryptography was developed later which refer to encrypting and decrypting of plaintext using two keys (public and secret) and single key respectively. In order to perceive the working and importance of homomorphic encryption, appropriate knowledge about the developments in cryptography before the development of homomorphic encryption is gained.

By conducting this study, it can be concluded that Homomorphic encryption; despite being in the developing stage, is employed in many fields where security of confidential data plays a pivotal role. The different variants of Homomorphic encryption schemes namely, SHE (Somewhat homomorphic encryption), PHE (Partial homomorphic encryption), LHE (Levelled homomorphic encryption), and FHE (fully homomorphic encryption). The selected HE schemes for this research are DGHV, Paillier, HElib, and FHEW. These are selected based on the literature available, availability of source code libraries, popularity in terms of scope, usability, and feasibility. Paillier cryptosystem implements Paillier homomorphic encryption scheme which is a PHE (partial homomorphic encryption) scheme, and all the other schemes are FHE. The performance and application of homomorphic encryption in the cloud and other areas are very broad such as e-voting, Cloud data protection, Genomic analysis, enhanced GPU processing in steganography, multiparty computations, protecting data in health care systems, and also used by famous organizations such as Google, Microsoft, IBM etc. Modern tools like functional encryption, homomorphic encryption, efficient two-party computation, searchable encryption, and, most recently, obfuscation, can do incredible things bordering on the realm of impossibility; allowing us to compute and reason over encrypted information without ever revealing it [15]. The applications of HE is vast and various other applications are discussed in Section 2.4. Experiments were conducted as a part of the research to achieve the objectives of the research. Each scheme required different library for that implemented it, and the source-code was fetched from a publicly available repository called GitHub. The libraries for the schemes to run were installed on Oracle VirtualBox and then run.

Due to the few problems with the concerned libraries, it was not possible to run all the schemes as intended in order to fulfill one of the aims of the research. Only Paillier and RSA (Public-key encryption) were run successfully and measured CPU and Memory utilized by the

schemes during their execution. By obtaining negative experimentation as result, it is perceived that the publicly available libraries for Homomorphic encryption is still primitive and is not much suitable for practical implementation. The results of Paillier and RSA are represented graphically in Section 4. They are run with the input of different sizes viz. 20KB, 40KB, 60KB, 80KB and 100KB is observed that there is an increase in execution time, CPU utilization, and Memory utilization with an increase in input size for both the schemes. Also, the range of CPU and Memory utilization for both the scheme is similar with similar inputs. The performance of the other selected schemes is described theoretically due to partially successful experimentation. Also, it can be noted that there were hardly any open-source libraries for homomorphic encryption schemes that affirmed the proper working of the libraries which resulted in selecting the best libraries among the available.

As an implication to practitioners, it can be said that this thesis is carried out based on the available and accessible resources and cannot be explicitly labeled as a recommendation considering the reliability and repeatability of the research.

6.1.1 Limitations of HE

The various limitations of HE and HE schemes are presented below which can assist the researchers working in this subject area in focusing on the backdrops and hence developing efficient HE algorithms. One of the major problems in HE scheme is large key size is found that all the schemes have large key sizes and result in noise because of multiple encryptions are done the key need to be squashed and compressed. In addition, existing FHE schemes are computationally expensive, which require a lot of computing resources to implement such schemes. This inhibits the mobile devices from computing in an efficient manner [47].

The main bottleneck in Homomorphic encryption is caused by the fact that all current FHE solutions are based on “noisy” encryption schemes (based on lattices or similar problems) where homomorphic operations increase the noise amount and lower the quality of ciphertexts. As more homomorphic operations are performed, the noise can easily grow to a level where the ciphertexts are no longer decryptable, and operating on them produces meaningless results [40]. Maintaining confidentiality and integrity of sensitive data is very essential. The threat of potential violations to the confidentiality and integrity of customer data is a key barrier to the adoption of cloud computing based on third- party infrastructure providers [50]. The libraries that are available open-source serve as a limitation in this research for selected HE schemes because of lack of proper building of the library and lack of robustness concerning the schemes. Also, the flexibility of partial homomorphic encryption schemes is low which makes it restricted to only those applications that do not need much flexibility. The present homomorphic encryption algorithms support only basic operations such as addition and multiplication and do not support operations such as division etc. which makes it limited to only these operations.

6.2 Future work

As a future work, this study can be extended to experiment other HE schemes that are available publicly for research and also investigating the schemes with various inputs followed by execution in any cloud service will fetch good analysis and evaluation of the schemes. Further enhancement in homomorphic encryption domain can be done by facilitating division and other operation in HE schemes to be performed on encrypted data. In order to evade key recovery attacks and side-channel leakage in SHE and FHE, the future implementations of FHE and SHE schemes can be designed with IND-CCA 1 security. Reduction of noise, large key size, large ciphertext size can result in efficient or operation limited homomorphic encryption algorithms for real time application so considering these issues, future effective

implementations can be proposed. Thus, the application of HE can be extended for efficient data retrieval and for preserving the privacy of the in large databases, cloud storage systems, and mobile applications making it play a pivotal role in all aspects of cyber security.

REFERENCES

[1] M. Ogburn, C. Turner, and P. Dahal, "Homomorphic encryption," *Procedia Comput. Sci.*, vol. 20, pp. 502–509, 2013.

[2] V. Vaikuntanathan, "Computing blindfolded: New developments in fully homomorphic encryption," *Proc. - Annu. IEEE Symp. Found. Comput. Sci. FOCS*, pp. 5–16, 2011.

[3] F. Armknecht, C. Boyd, C. Carr, K. Gjosteen, A. Jaschke, C. A. Reuter, and M. Strand, "A Guide to Fully Homomorphic Encryption," pp. 1–34, 2015.

[4] Y. Hu, "Improving the Efficiency of Homomorphic Encryption Schemes," p. 103, 2013.

[5] C. Gentry, "a Fully Homomorphic Encryption Scheme," *PhD Thesis*, no. September, pp. 1–209, 2009.

[6] A. Bouti and J. Keller, "Securing cloud-based computations against malicious providers," *ACM SIGOPS Oper. Syst. Rev.*, vol. 46, no. 2, p. 38, 2012.

[7] D. S. Maimut, A. Patrascu, and E. Simion, "Homomorphic Encryption Schemes and Applications for a Secure Digital World," *J. Mobile, Embed. Distrib. Syst.*, vol. 4, no. 4, pp. 224–232, 2012.

[8] O. Farràs and C. Padró, "Theory of Cryptography," *Tcc*, vol. 5978, pp. 165–182, 2010.

[9] Charles P. Pfleeger; Shari Lawrence Pfleeger; Jonathan Margulies, *Security in computing*, Fifth. Prentice Hall, 2015.

[10] Craig Bauer, *Secret History: The Story of Cryptology*. Chapman and Hall/CRC, 2013.

[11] Matt Bishop, *Introduction to Computer Security*. Addison-Wesley Professional, 2004.

[12] N. Nagaraj, "One-Time Pad as a nonlinear dynamical system," *Commun. Nonlinear Sci. Numer. Simul.*, vol. 17, no. 11, pp. 4029–4036, 2012.

[13] D. H. K. H, *Introduction to Cryptography*, Third Edit., vol. 86. Springer Berlin Heidelberg, 2015.

[14] R. R. Rachh, P. V. A. Mohan, and B. S. Anami, "Efficient implementations for AES encryption and decryption," *Circuits, Syst. Signal Process.*, vol. 31, no. 5, pp. 1765–1785, 2012.

[15] N. World, "Alice and Bob in the New World," vol. 21, no. 3, pp. 8–9, 2015.

[16] W. Stallings, *Cryptography and Network Security (4th Edition)*. Prentice Hall, 2005.

[17] C. Gentry, "Implementing Gentry ' s Fully-Homomorphic Encryption Scheme Preliminary Report," *Organization*, pp. 1–30, 2010.

[18] W. Wang, Y. Hu, L. Chen, X. Huang, and B. Sunar, "Exploring the feasibility of fully homomorphic encryption," *IEEE Trans. Comput.*, vol. 64, no. 3, pp. 698–706, 2015.

[19] M. Van Dijk and C. Gentry, "Fully homomorphic encryption over the integers," *Adv. Cryptology– ...*, pp. 1–28, 2010.

[20] P. V Parmar, S. B. Padhar, S. N. Patel, N. I. Bhatt, and R. H. Jhaveri, "Survey of Various Homomorphic Encryption algorithms and Schemes," *Int. J. Comput. Appl.*, vol. 91, no. 8, pp. 975–8887, 2014.

[21] S. Fau, R. Sirdey, C. Fontaine, C. Aguilar-Melchor, and G. Gogniat, "Towards practical program execution over Fully Homomorphic Encryption schemes," *Proc. - 2013 8th Int. Conf. P2P, Parallel, Grid, Cloud Internet Comput. 3PGCIC 2013*, pp. 284–290, 2013.

[22] 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," *Futur. Gener. Comput. Syst.*, vol. 36, pp. 180–186, 2014.

[23] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?," *Proc. 3rd ACM Work. Cloud Comput. Secur. Work. - CCSW '11*, pp. 113–124, 2011.

[24] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," pp. 169–178, 2009.

[25] M. Brenner, H. Perl, and M. Smith, "Practical Applications of Homomorphic Encryption," *Int. Conf. Secur. Cryptogr. (SECRYPT 2012)*, pp. 1–10, 2012.

[26] S. Kamara and M. Raykova, “Parallel Homomorphic Encryption,” *Financ. Cryptogr. Data Secur.*, no. volume 7862 of the series Lecture Notes in Computer Sciences, pp. 213–225, 2013.
[27] M. Bellare, D. Hofheinz, and E. Kiltz, “Subtleties in the Definition of IND-CCA: When and How Should Challenge Decryption Be Disallowed?,” *J. Cryptol.*, vol. 28, no. 1, pp. 29–48, 2013.
[28] D. F. Aranha and A. Menezes, “Progress in cryptology ??? LATINCRYPT 2014: Third International conference on cryptology and information security in Latin America Florian??polis, Brazil, september 17-19, 2014 revised selected papers,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8895, pp. 239–258, 2015.
[29] T. Lepoint and M. Naehrig, “A comparison of the homomorphic encryption schemes FV and YASHE,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8469 LNCS, pp. 318–335, 2014.
[30] S. Halevi and V. Shoup, “Bootstrapping for HElib,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9056, pp. 641–670, 2015.
[31] L. Ducas and D. Micciancio, “FHEW: Bootstrapping homomorphic encryption in less than a second,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9056, pp. 617–640, 2015.
[32] D. Naccache and M. Tibouchi, “Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers.”
[33] P. S. Pisa, M. Abdalla, and O. C. M. B. Duarte, “Somewhat homomorphic encryption scheme for arithmetic operations on large integers,” *2012 Glob. Inf. Infrastruct. Netw. Symp. GIIS 2012*, 2012.
[34] T. Sridokmai and S. Prakancharoen, “The homomorphic other property of Paillier cryptosystem,” *2015 Int. Conf. Sci. Technol.*, pp. 356–359, 2015.
[35] P. Paillier, “Paillier Encryption,” pp. 453–492, 1999.
[36] Kamesh and N. Sakthi Priya, “Security enhancement of authenticated RFID generation,” *Int. J. Appl. Eng. Res.*, vol. 9, no. 22, pp. 5968–5974, 2014.
[37] Shaih, “HElib,” 2014. [Online]. Available: https://github.com/shaih/HElib.
[38] S. Halevi and V. Shoup, “Algorithms in HElib,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8616 LNCS, no. PART 1, pp. 554–571, 2014.
[39] C. Gentry and N. P. Smart, *Aes_Gentry*. 2015.
[40] E. Bash, “No Title No Title,” *PhD Propos.*, vol. 1, 2015.
[41] M. S. Lee, Y. Lee, J. H. Cheon, and Y. Paek, “Accelerating bootstrapping in FHEW using GPUs,” *Proc. Int. Conf. Appl. Syst. Archit. Process.*, vol. 2015–Septe, pp. 128–135, 2015.
[42] I. Damgård and M. J. Jurik, “A Generalisation, a Simplification and some Applications of Paillier’s Probabilistic Public-Key System,” *PKC 2001 Public Key Cryptogr.*, no. December, pp. 119–136, 2001.
[43] Q. B. Hani and J. P. Dichter, “Data leakage preventation using homomorphic encryptionin cloud computing,” *2016 IEEE Long Isl. Syst. Appl. Technol. Conf. LISAT 2016*, 2016.
[44] P. Gerhard and B. Eng, “KV Web Security : Applications of Homomorphic Encryption,” pp. 1–16, 2013.
[45] K. Kogiso and T. Fujita, “Cyber-security enhancement of networked control systems using homomorphic encryption,” *Proc. 54th IEEE Conf. Decis. Control*, no. Cdc, pp. 6836–6843, 2015.
[46] S. Palamakumbura and H. Usefi, “Homomorphic Evaluation of Database Queries,” 2016.
[47] M. R. Baharon, Q. Shi, and D. Llewellyn-Jones, “A New Lightweight Homomorphic Encryption Scheme for Mobile Cloud Computing,” *2015 IEEE Int. Conf. Comput. Inf. Technol. Ubiquitous Comput. Commun. Dependable, Auton. Secur. Comput. Pervasive*

Intell. Comput., pp. 618–625, 2015.
[48] M. Brenner, J. Wiebelitz, G. Von Voigt, and M. Smith, "Secret program execution in the cloud applying homomorphic encryption," *IEEE Int. Conf. Digit. Ecosyst. Technol.*, vol. 5, no. June, pp. 114–119, 2011.
[49] M. Tebaa, S. El Hajji, a El Ghazi, S. El Hajji, and A. El Ghazi, "Homomorphic Encryption Applied to the Cloud Computing Security," *Proc. World Congr. Eng.*, vol. 1, pp. 4–6, 2012.
[50] S. D. Tetali, M. Lesani, R. Majumdar, and T. Millstein, "MrCrypt," *Proc. 2013 ACM SIGPLAN Int. Conf. Object oriented Program. Syst. Lang. Appl. - OOPSLA '13*, vol. 48, no. 10, pp. 271–286, 2013.
[51] M. Robshaw, J. K. Eds, and D. Hutchison, *Advances in Cryptology – CRYPTO 2016*. 2016.
[52] Y. Levy and T. J. Ellis, "A systems approach to conduct an effective literature review in support of information systems research," *Informing Sci.*, vol. 9, pp. 181–211, 2006.
[53] L. H. Toledo-Pereyra, "Research Design," *J. Investig. Surg.*, vol. 25, no. 5, pp. 279–280, 2012.
[54] anuj patel, "RSA-Algorithm," 2014. [Online]. Available: https://github.com/anujpatel/RSA-Algorithm.
[55] Coron, "FHE," 2012. [Online]. Available: https://github.com/coron/fhe.
[56] Kunerd, "jPaillier," 2015. [Online]. Available: https://github.com/kunerd/jpaillier.
[57] M. Varia, S. Yakoubov, and Y. Yang, "HEtest: A homomorphic encryption testing framework," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8976, pp. 213–230, 2015.
[58] Iducas, "FHEW," 2014. [Online]. Available: https://github.com/lducas/FHEW.
[59] L. Morrison, Keith; Manion and L. Cohen, *Research Methods in Education*, 7th ed. Taylor and Francis, 2013.
[60] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empir. Softw. Eng.*, vol. 14, no. 2, pp. 131–164, 2009.
[61] F. Marinelli, R. Aragona, C. Marcolla, and M. Sala, "Some security bounds for the DGHV scheme Homomorphic encryption scheme," no. June, 2014.
[62] H.-T. Wu, Y. Cheung, and J. Huang, "Reversible Data Hiding in Paillier Cryptosystem," *J. Vis. Commun. Image Represent.*, vol. 40, pp. 765–771, 2016.
[63] A. Bouti and J. Keller, "Towards practical homomorphic encryption in cloud computing," *Proc. - IEEE 4th Symp. Netw. Cloud Comput. Appl. NCCA 2015*, pp. 67–74, 2015.
[64] X. Yi, R. Paulet, and E. Bertino, "Homomorphic Encryption and Applications," pp. 27–47, 2014.
[65] J. Isotalo, "Basics of Statistics," pp. 0–82.