Major Project Report
On
USING HOMOMORPHIC ENCRYPTION
TO SECURE E-VOTING

Submitted in partial fulfilment of the requirements
for the award of the degree of

Bachelor of Technology
In
Computer Science & Engineering

Guide:

Ms. Rachna Jain

Submitted by:

Abhishek Gupta 00711502712

Harsh 03011502712

Yogeesh Kapila 01911502712

Yogesh Kumar 05811502712

BHARATI VIDYAPEETH'S COLLEGE OF ENGINEERING
A 4, Paschim Vihar, New Delhi, Delhi 110063
GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY, DELHI

(2012-16)

DECLARATION

We hereby declare that the project entitled **"USING HOMOMORPHIC ENCRYPTION TO SECURE E-VOTING"** submitted for the B. Tech Degree is our original work and the project has not formed the basis for the award of any degree, associate ship, fellowship or any other similar titles.

Signature of the Students:

Abhishek Gupta

Harsh

Yogeesh Kapila

Yogesh Kumar

Place:

Date:

CERTIFICATE

This is to certify that the project entitled "USING HOMOMORPHIC ENCRYPTION TO SECURE E-VOTING" is the bonafide work carried out by **Abhishek Gupta, Yogeesh Kapila, Harsh, Yogesh Kumar** students of B. Tech (CSE), BVCOE affiliated to GGSIPU, during the years 2015-16, in partial fulfilment of the requirements for the award of the Degree of Bachelor of Computer Technology and that the project has not formed the basis for the award previously of any degree, diploma, associate ship, fellowship or any other similar title.

Ms. Rachna Jain	Dr. Bindu Garg
Signature of the Guide	Signature of the HOD
Place:	Place:
Date:	Date:

ACKNOWLEDGEMENT

Though only our names appear on the cover of this report, a great many people have contributed to its production. We owe our gratitude to all those people who have made this project possible and because of whom our research experience has been one that we will cherish forever.

We would like to take this opportunity to express our profound gratitude and deep regard to Ms. Rachna Jain, for her exemplary guidance, valuable feedback and constant encouragement throughout the duration of the study. Her valuable suggestions were of immense help throughout our project. We have been amazingly fortunate to have an advisor who gave us the freedom to explore on our own, and at the same time the guidance to recover when our steps faltered. Working under her was an extremely knowledgeable experience for us.

We are also indebted to the faculty members of the Computer Science Department, BVCOE with whom we have interacted during the course of our project.

Nevertheless, we are thankful to our families and classmates for their kind co-operation, encouragement and useful insights which helped us in the completion of this project.

Abhishek Gupta Harsh Yogeesh Kapila Yogesh Kumar

ABSTRACT

Cryptography as we know it, is among the most discussed topics in the security world. Any transaction, financial or social, any data, corporate or private is not secure in an environment such as the Cloud, where everything is connected to everything. The only way to save anything or to make a transaction securely is to make it meaningless to the rest of the world. It can be made meaningless when converted to some other form, and this some other form can only be obtained through encryption. In this paper we discuss various techniques of Homomorphic Encryption applied on Cloud Computing and the need of security over the cloud by citing relevant statistics. We then propose and implement a secure E-voting system using the Paillier Encryption Scheme.

Electronic voting (also known as e-voting) is voting using electronic means to either aid or take care of the chores of casting and counting votes. Depending on the particular implementation, e-voting may encompass a range of Internet services, from basic data transmission to full-function online voting through common connectable household devices. Similarly, the degree of automation may vary from simple chores to a complete solution that includes voter registration & authentication, vote input, local or precinct tallying, vote data encryption and transmission to servers, vote consolidation and tabulation, and election administration. A worthy e-voting system must perform most of these tasks while complying with a set of standards established by regulatory bodies, and must also be capable to deal successfully with strong requirements associated with security, accuracy, integrity, swiftness, privacy, auditability, accessibility, cost-effectiveness, scalability and ecological sustainability.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

"E-voting" is a term that is described in different information and communication technologies (ICT) platforms: Internet systems, polling booth machines, and telephone voting system [48]. Each platform has both negative and positive features. However, the Internet voting systems are most popular and successful today. Each voter can vote for candidates remotely through the Internet. Voters can easily open websites, software, or applications on their computers or smartphones to vote anytime, anywhere. However, their voting system can be attacked if it does not have any algorithm or protocol to protect it. Attackers can catch packets that voters transfer on the Internet easily; they change information and send to the voting server. In addition, they can interrupt votes which voters send over the Internet. Accordingly, researchers are thinking about methods to defend against such attacks using cryptography algorithms to encrypt data a vote before sending it to the server. This idea was first mentioned in 1981 [44]. Some algorithms are being used in voting systems such as Diffie-Hellman Encryption, RSA Encryption, Elgamal Encryption, and Paillier Encryption. This project uses the homomorphic properties of Paillier Encryption to protect a vote sent over the Internet.

The main goal of this project is to design a secure voting system using the Internet platform to communicate between the voting system and voters.

E-voting process consists of several steps, many of which can be found in the traditional voting. It is more accessible and convenient to voters than the traditional voting. The big advantage is that it does not require voters to present at the poll station. Another advantage, voters can vote from anywhere as soon as they meet the requirements to vote. However, to implement a secure voting system is difficult. Here are some requirements which a secure voting system has to adopt.

Auditability: Reliable and demonstrably authentic election records should be generated. Motivation for the electronic voting system

Authentication: Only authorized voters should be able to vote. Voters can request to change their information if necessary. A voter's information can be collected from the birth certificate or other similar document [50].

Reliability: Systems should work robustly to prevent electoral frauds or attacks from outside the system. The E-voting system should be very reliable. The result of an election must be correct and shows up to voters after the election ends.

Voter Confirmation: The voting system should send an email to the voter to confirm that his or her vote has been received by the system correctly. At the end, the result of the voting also can be sent to the voters so that they will know which candidate is a winner.

Uniqueness: No voter should be able to vote more than once. This requirement is necessary and same as the traditional voting. This feature prevents coercion or buying votes.

Accuracy: Voting systems should record the votes correctly. After the vote is recorded, the voter should be able to check if his vote was recorded or not. If he tries to re-vote or change the vote, the system should prevent that.

Integrity: A vote is just for one time decision and cannot be changed. No one should be able to determine how the voter voted, so no one can change the vote.

Verifiability: Verifying that the votes are correctly counted in the final tally should be possible. This feature is mandatory. It has to be audited and tested before the system is used.

E-voting is the most convenient to vote. It is excellent on equality, building a trust in electoral organization, adding reliability to election results, and increasing the overall efficiency of the polling process [52]. However, to build an E-voting system that can work perfectly over the Internet is a big challenge. Two major challenges that can be considered are security and supporting a large number of voters. Most E-voting solutions cannot work with large number of voters. E-voting also faces security issues because voters vote and send their votes over the Internet which is not a controlled environment [52]. In addition, voting under an electronic

voting system occurs automatically without any human supervision. Certainly, voters would like to vote by paper votes at a post office or polling station rather than through an E-voting system because they do not trust the E-voting as their ballots are transferred over the Internet [53]. For example, in 2012, the federal government allowed to use the E-voting for Canton of Zurich voters. The voters had three options to vote: ballot voting, postal voting, and Internet voting. However, only 20 percent of the votes were cast through the Internet voting [54]. To address some of these challenges, in this project, we have developed an E-voting system that can allow a large number of voters. Specifically, Paillier algorithm is used to support a large number of voters and secure a vote when it is counted. To secure a vote on the Internet, the Secure Socket server/client protocol is used to transfer information over the Internet [50]. The security properties of SSL are discussed in the following sections.

Here is a very simple example of how a homomorphic encryption scheme might work in cloud computing:

- Business XYZ has a very important data set (VIDS) that consists of the numbers 5 and 10. To encrypt the data set, Business XYZ multiplies each element in the set by 2, creating a new set whose members are 10 and 20.

- Business XYZ sends the encrypted VIDS set to the cloud for safe storage. A few months later, the government contacts Business XYZ and requests the sum of VIDS elements.

- Business XYZ is very busy, so it asks the cloud provider to perform the operation. The cloud provider, who only has access to the encrypted data set, finds the sum of 10 + 20 and returns the answer 30.

- Business XYZ decrypts the cloud provider's reply and provides the government with the decrypted answer, 15.

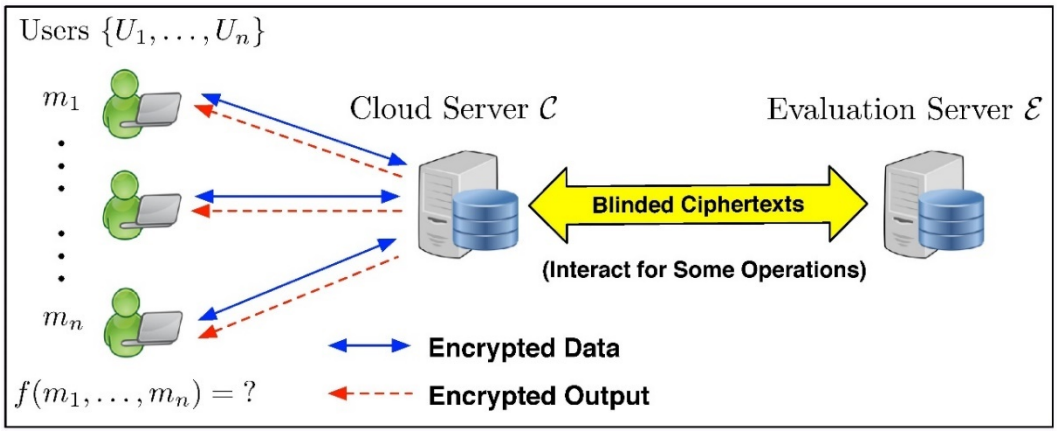


Figure 1 Homomorphic Encryption Model

Homomorphic encryption is expected to play an important part in cloud computing, allowing companies to store encrypted data in a public cloud and take advantage of the cloud provider's analytic services.

CHAPTER 2
HOMOMORPHIC ENCRYPTION

Homomorphic encryption is a form of encryption that allows for some computations to be performed on the ciphertext without decrypting the ciphertext. The result of the operations is returned as an encrypted result, which when decrypted is the same as if some operation was performed on the plaintext. Some applications for such a system are the implementation of secure voting systems and cloud computing. There are many form of partially homomorphic cryptosystems that allow for some specific operations to be performed (namely addition and multiplication), but due to some very major drawbacks of fully homomorphic encryption, fully homomorphic encryption is not very practical. Some examples of such drawbacks are processing time and implementation complexity. [1]

2.1 General Definition

HOMOMORPHISM - a transformation of one set into another that preserves in the second set the relations between elements of the first. [1]

HOMOMORPHIC ENCRYPTION - an operation performed on a set of ciphertexts such that decrypting the result of the operation is the same as the result of some operation performed on the plaintexts. [1]

2.2 Technical Definition

Definition We say that a CPA(Chosen-Plaintext Attack)-secure public key encryption scheme (G, E, D) with one bit messages is *fully homomorphic* if there exists an algorithm $NAND$ such that for every $(e, d) \leftarrow G(1^n)$, $a, b \in \{0, 1\}$, and $\hat{a} \leftarrow E_e(a)$, $\hat{b} \leftarrow E_e(b)$,

$NAND_e(\hat{a}, \hat{b}) \approx E_e(a\ NAND\ b)$

where ≈ denotes statistical in distinguishability (i.e., $n^{-\omega(1)}$ statistical distance), and $a\ NAND\ b$ denotes $\neg(a \wedge b)$.

(There are several variants of the definition, and we chose the simplest and strongest one.)

We stress that the algorithm NAND does *not* get the secret key as input. Otherwise it would be trivial: just decrypt $\hat{a}, \hat{b}$, compute $aN\ AN\ Db$ and re-encrypt. [2]

Homomorphic encryption is a form of encryption which allows specific types of computations to be carried out on ciphertext and obtain an encrypted result which when decrypted gives the result of operations performed on the plaintext. For example, one could add two encrypted numbers and then another could decrypt the result, without either of them being able to find the value of the individual numbers.

Methods that would allow operation on data without knowing the actual content can help in lot of areas. Homomorphic encryption is one such method. Today most systems operate with help of a trusted party. Users have to trust an entity, human or machine to maintain secrecy of their data. But an attack on the trusted party or vulnerability with the system can expose the users secret.

Hence the necessity of systems where even the service providers have no detailed knowledge of the users data is growing.

In the next section, some of the work done in the area of homomorphic encryption is described. Homomorphic encryption originated from the concept of privacy homomorphism, introduced by the Rivest et al. In their paper, they discussed about performing operations on the encrypted data. The RSA cryptosystem introduced by them also exhibited the property of partially homomorphic encryption, allowing multiplication of encrypted data, which when decrypted will give the product of the plaintexts. Ever since many schemes with homomorphic properties have been proposed. Another cryptosystem developed during the same period was the ElGamal Cryptosystem. Developed and named after Taher El Gamal, ElGamal cryptosystem also had some homomorphic properties. Although these two cryptosystems in

their basic form is not that popular but still serves as a great introduction to the concept and many variation of the basic scheme are used in some applications. Paillier cryptosystem [20], developed by Pascal Paillier is one of the popular cryptosystem supporting additive homomorphism. Although the scheme is partially homomorphic but its simplicity and performance makes it one of the best homomorphic scheme today.

However, a cryptosystem that supports both additive and multiplicative homomorphism has been investigated since a long time. In that context, Boneh-Goh-Nissim Cryptosystem provided promising potential for a long time. It allowed unlimited addition along with one multiplication operation.But the first fully homomorphic scheme was possible due to Craig Gentry.

Gentry's scheme involves creating a somewhat homomorphic scheme and then bootstrapping it make it fully homomorphic.Although homomorphic encryption has existed since a long time, but due to its computational and storage overhead it has received less attention. Further the possibility of a working fully homomorphic encryption in real world was one of the question. However, in recent years development of fully homomorphic encryption schemes have attracted a lot of focus into this field of cryptography. Homomorphic encryption has great potential for use in scenarios ranging from multi-party communication to secure computation in cloud systems.

2.3 Algorithms

Encrypting the data has the advantage that the data remains unintelligible to all unless it is decrypted, therefore protecting the data from malicious users and hackers. Thus data on the cloud is made secure by this. However, security comes at the cost of increased computation time and cumbersome procedure of decrypting the data always before its use, even by the original user. In order to solve this problem a technique of Homomorphic Encryption was devised.

What's so special about Homomorphic Encryption? Homomorphic Encryption is a way of encrypting the plain text in such a way such that if any operation is carried out on the cipher

text and then decrypt the result, it is the same as if the operation had been carried out on plain text. This technique of encryption has huge applications as it provides the flexibility of operating on cipher texts instead of plain texts, which in turn lets a user data be stored on some server in encrypted form and the third party can manipulate the data and provide the result (encrypted) on the user's behalf, not ever knowing what the real data is.

Homomorphic encryption is an asymmetric cryptography technique, i.e., two keys are used for encryption and decryption. The public key is made accessible to all which is used for encryption, whereas the private key is used for decryption, which is kept hidden. The basic framework is as:

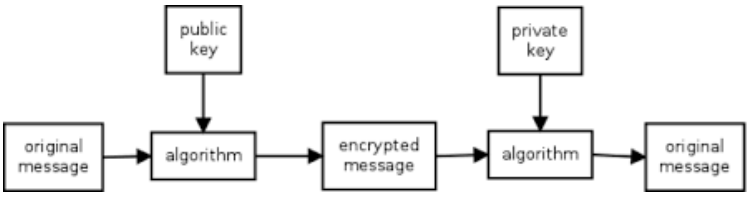


Figure 2Asymmetric Cryptography Framework

All the user data stored on Cloud servers of a third party can be even more secure with the implementation of Homomorphic Schemes. How? The answer is that for every computation the overhead of decrypting the data, performing operation and then again encrypting it for the user is saved. With Homomorphic Encryption computations can be carried out straight away on cipher texts. Secondly, an added layer of security is provided for the user data. Now the data would be hidden even from the third party Cloud server providers, hence even more secure data. Hence Homomorphic Encryption has huge applications especially in Cloud Computing.There are basically two types of Homomorphic Encryption Schemes namely:

- Partially Homomorphic Encryption Schemes.
- Fully Homomorphic Encryption Schemes.

In a partially homomorphic system such as the Paillier or The RSA cryptosystem, the system allows only one operation to be performed on it such that it still retains its homomorphic property. For instance RSA is multiplicative cryptosystem whereas Paillier is additive cryptosystem. Thus each system supporting only one type of operation.

In a fully homomorphic system such as one given by Gentry, the system allows for more than one operation to be applied on it and the system still retains its homomorphic property. It is far more powerful than partially homomorphic cryptosystem.

Representing in mathematical form: [14]

Consider,

K = Cryptosystem, P = Plaintext, C = Cipher text, © = some operation and € = Encryption function

Then it can be said that K is additively homomorphic if,

$\sum$ ©: $€(x_1)$ © $€(x_2) = € (x_1 + x_2)$ …….eq.1

And it exhibits multiplicative homomorphism if,

$\sum$ ©: $€(x_1)$ © $€(x_2) = € (x_1 * x_2)$ ……..…….eq.2

These operations can be applied to any operation but for simplicity we are considering only addition and multiplication.

Now if 'K' supports either ① or ② then we can say that 'K' is a partially homomorphic system.

But if 'K' supports both ① and ② then we can say that 'K' is a fully homomorphic system.

In all public key cryptographic systems two keys are maintained. One key is publicly accessible to all i.e. the public key and that is used for encrypting the message.
One key is kept a secret and kept with the user i.e. the private key and that is used for decrypting the cipher text.
This type is also known as asymmetric cryptography or Secret Key Cryptography.

2.3.1 RSA Algorithm

It is very popular public key cryptographic technique. RSA is basically based on the factoring problem i.e. factoring the product of two big prime numbers.

The basic principle is finding three significantly large positive integers which satisfy:-

$$(m^e)^d \mod n = m \quad \text{.....eq.3}$$

Let m = plaintext, c=cipher text, e= public key, d= private key

Then encryption is as:

$$c \equiv m^e \mod n \quad \text{.....eq.4}$$

The decryption is as:

$$c^d \equiv (m^e)^d \equiv m \mod n \quad \text{.....eq.5}$$

RSA is partially homomorphic encryption scheme, supporting multiplicative operations.

If,

$C1 = m_1^e \ mod \ n$ $C2 = m_2^e \ mod \ n$ Then on multiplying the two ciphers we get,

$$(m_1^e m_2^e) \ mod \ n = > (m_1 m_2)^e \ mod \ n \quad \text{.....eq.6}$$

It uses a key size of 1024 to 4094 bit.

However, one of the drawbacks to the RSA encryption algorithm is that it leaks a singular plaintext bit in every cipher text. This bit is known as the Jacobi symbol of the plaintext, and is either "1" or "−1." Due to this an attacker can guess what the original plaintext was. This drawback was countered by the Goldwasser – Micali scheme (discussed later). [56]

2.3.2 Rabin Cryptosystem

This cryptosystem is an asymmetric cryptographic technique similar to the RSA cryptosystem. It is also based upon the problem of factorization. The problem it relies on has been proved as hard as 'Integer Factorization'; the same can't be said regarding RSA also, hence it is more difficult than the RSA cryptosystem.

It is commonly referred as a four-to-one function encryption scheme.

Consider,

p and q =large prime numbers (private key), n = p.q (public key), m =plaintext, c = cipher text, P = plaintext space i.e. {0,…..,n-1} such that m € P.

Then Encryption is as:

$$c = m^2 \bmod n \quad \text{.....eq.7}$$

Decryption is as:

First compute: $m_p = \sqrt{c} \bmod p$ and $m_q = \sqrt{c} \bmod q$,…..……eq.8

Then using the Chinese remainder theorem we can calculate the value of m.

This is called a four-to-one function because for a given cipher text there can be up to four plaintext decryptions. Hence extra complex computations are required to figure out the real plaintext.

Let for example:

p = 7 and q = 11 then n = 77, m = 20

Hence, $c = m^2 \bmod n = 400 \bmod 77 = 15$.

Upon Decryption we get $m \in \{13, 20, 57, 64\}$.

Hence, it is called a four-to-one function. [57]

2.3.3 Schmidt-Samoa Cryptosystem

This is also an asymmetric encryption technique which is based on the integer factorization problem, similar to Rabin Cryptosystem.

However, it has the advantage of not producing any ambiguity in the decryption process, like in Rabin scheme, at the cost of encryption speed.

Consider,

p and q =large prime numbers, $n = p^2q$ (public key), m =plaintext, c = cipher text and $d = N^{-1}\ mod\ lcm(p-1, q-1)$ (private key).

ENCRYPTION: DECRYPTION:

$c = m^N \mod N.$ $m = c^d \mod pq,$

Which can be computed

Chinese Remainder Theorem.

It has an advantage over RSA as it is based on much harder 'Integer Factorization' however, encryption is slow due to exponent function. [58]

2.3.4 Goldwasser-Micali Cryptosystem

This technique is an asymmetric key encryption algorithm developed in 1982. It is unique in the sense that it is the first probabilistic encryption algorithm designed which was proven to be secure under ideal cryptographic assumptions. The probabilistic nature of the algorithm ensures randomness, i.e. it generates a different cipher text each time it is used to encrypt a plaintext. It encrypts the plaintext bit by bit.

Its security depends on the problem of quadratic residuosity modulo which can be stated as:

For some: $x_p = x\ mod\ p$ & $x_q = x\ mod\ q$ then if,

$x_p^{(p-1)/2} \xrightarrow{yields} 1\ (mod\ p)$ & $x_q^{(q-1)/2} \xrightarrow{yields} 1\ (mod\ q)$, hence x is a quadratic residue mod$N$.

This cryptosystem exhibits additive homomorphic property.

Let m= message containing bits$\{m_1, \dots, m_n\}$, c= cipher text containing bits $\{c_1, \dots, c_n\}$ and $b_i = random\ value\ satisfying\ \gcd(b_i, N) = 1$

Encryption	Decryption
$C_i = b_i^2 . a^{m_i} \ (mod\ N)$	It is done by prime factorizing (p, q) and determining whether c_i is quadratic residue or not. If so $m_i = 1\ else\ m_i = 0$.

Homomorphic property

$enc\ (m_1) = c_1$ & $enc\ (m_2) = c_2$, then $c_1 c_2 mod\ (N) = enc(\ m_1 \oplus m_2)$…….....eq.9

However it is not a very efficient cryptosystem as the cipher text computed for a given plaintext is man times larger than the plaintext. [59]

2.3.5 Blum–Goldwasser Cryptosystem

The BG Cryptosystem is an asymmetric key encryption algorithm. It is a probabilistic and secure cryptosystem with fixed size cipher text expansion.

It is similar to the GM scheme with some improvements such as, this system is based on the problem of integer factorization i.e. factoring a composite value $N = P.Q\ \ where\ p\ \&\ q\ are\ large\ primes.$

This system is more efficient than the GM scheme as:

1. It is efficient in terms of storage as it produces constant size cipher text irrespective of the plaintext length.
2. Efficient in terms of computation.
3. Security depends only on integer factorization without requiring nay assumptions.

Consider,

p and q =large prime numbers, n = p.q (public key), m =plaintext, c = cipher text, private key = factorization (p,q) , r = random no.

Encryption is as:

1. Encode the message as a string of bits.
2. Use BBS pseudo random no generator to generate a random set of bits. (Key stream)
3. Compute Cipher text as :

$$\vec{c} = \vec{m} \oplus \vec{b}$$ for each bit where $\oplus$ = XOR function

Decryption follows just the reverse process using prime factorization and BBS generator computing key stream and XORing with cipher string bits to find m.

The BG scheme is secure based on the hardness of predicting the key stream bits given only the final BBS state y and the public key N.

However, this scheme is vulnerable to adaptive chosen cipher text attack. [60]

2.3.6 ElGamal Algorithm

It is a public key cryptographic technique which has the Diffie-Hellman key exchange algorithm as its base. This algorithm can be defined over any cyclic group C. Its security depends upon the difficulty of a certain problem in C related to computing discrete logarithms.

ElGamal Cryptosystem performs multiplicative homomorphic encryption property:

Let x1 and x2 be plaintexts. Then,

$$\begin{aligned} e_k(x_1, r_1)\, e_k(x_2, r_2) &= (\alpha^{r1} \bmod p,\ x_1 \beta^{r1} \bmod p)\,(\alpha^{r2} \bmod p,\ x_2 \beta^{r2} \bmod p) \\ &= (\alpha^{r1+r2} \bmod p,\ (x_1 x_2)\, \beta^{r1+r2} \bmod p \\ &= e_k(x_1 x_2,\ r_1 r_2) \end{aligned}$$

If we put the plaintext in the exponent, we get:

$$e_k(x, r) = (\alpha^r \bmod p,\ \alpha^x \beta^r \bmod p)$$

Then the homomorphism is additive:

$$\begin{aligned} e_k(x_1, r_1)\, e_k(x_2, r_2) &= (\alpha^{r1} \bmod p,\ \alpha^{x1} \beta^{r1} \bmod p)\,(\alpha^{r2} \bmod p,\ \alpha^{x2} \beta^{r2} \bmod p) \\ &= (\alpha^{r1+r2} \bmod p,\ \alpha^{x1+x2}\, \beta^{r1+r2}) \bmod p \\ &= e_k(x_{1+}x_2,\ r_{2+}r_2) \end{aligned}$$

This technique finds its use in hybrid cryptosystems. A hybrid cryptosystem is one in which the message is encrypted by use of a symmetric cryptosystem technique and the key used for symmetric encryption is then encrypted using the ElGamal encryption technique. [61]

2.3.7 Benaloh Cryptosystem

The Benaloh Cryptosystem is basically an extension of the Goldwasser-Micali Cryptosystem. In GM encryption Scheme the encryption took place bit by bit. This scheme overcomes that limitation by allowing blocks of data being encrypted at a single pass. The security of the scheme relies on the problem of '*Higher Residuosity*'.

Let public key be modulus m and the base g with block size as c then,

x = message, r = random no.

ENCRYPTION:

Enc(x) => $\mathcal{E}(x) = g^x r^c \bmod m$

DECRYPTION:

1) Compute : $a = c^{\phi/r} \mod n$ where ϕ = (p-1)(q-1)
2) Find $m = \log_x(a) \equiv x^m \equiv a \mod n.$

This scheme exhibits additive homomorphic property. [40]

$$\mathcal{E}(x_1)\cdot\mathcal{E}(x_2) \bmod m = (g^{x_1} r_1^c)(g^{x_2} r_2^c) \bmod m = g^{x_1+x_2}(r_1 r_2)^c = \mathcal{E}(x_1+x_2 \bmod m) \quad \text{...eq.10}$$

2.3.8 Okamoto-Uchiyama cryptosystem

It is a public key cryptosystem. This scheme is based upon the multiplicative groups of integers modulo n. It is similar to most of the public key cryptosystem techniques but it differs from them in the form of n.

Here n is of the form: $[p^2 q]$

Consider,

p and q =large prime numbers, $n = p^2q$, m =plaintext, c = cipher text, $g^p \neq 1 \mod p^2$ and $h = g^n \mod n$, r= random no.

ENCRYPTION:

$$C = g^m h^r \mod n$$

DECRYPTION:

if $L(x) = \frac{x-1}{p}$ then,

$$m = \frac{L\,(C^{p-1} \mod p^2)}{L\,(g^{p-1} \mod p^2)} \mod p$$

This scheme is malleable. [11].

Many of the Homomorphic Encryption schemes are malleable.

Malleability means that an adversary can convert one of the cipher texts to another cipher text. That is, given an encryption of a plaintext m, it is possible to generate another cipher text which decrypts to f(m), for a known function f, without necessarily knowing or learning m.

The adversary or the attacker can, at best, only tamper the encrypted message but cannot read the encrypted message either before or after tampering.

2.3.9 Paillier Algorithm

This cryptosystem is a probabilistic public key cryptographic technique. The cryptosystem is based upon computing the nth residue classes which is considered as a difficult problem.

Let m = plaintext, c=cipher text, e= public key, d= private key, r= random no.

Encryption is as:

$$c = g^m \cdot r^n \mod n^2 \quad \text{.....eq.11}$$

Homomorphism is as: $C1 = g^{m1}.r^n \mod n^2$ $C2 = g^{m2}.r^n \mod n^2$ On multiplying we get,

$$C1.C2 = (g^{m1}g^{m2})r^n \mod n^2 = > (g^{(m1+m2)})r^n \mod n^2$$

2.3.10 Boneh, Goh and Nissim Cryptosystem

This cryptosystem was the first one which could be called a fully homomorphic system, but it allowed only a single multiplication operation along with the addition. Thus this was called a 'somewhat homomorphic'. The additive property is same as of the ElGamal variant.

This scheme uses elliptic curves of Composite order for its implementation. Unlike other homomorphic schemes the group order is not prime.

We now describe the system:

- $\mathsf{Gen}()$: Choose large primes q, r and set $n = qr$. Find a supersingular elliptic curve $E/\mathbb{F}_p$ with a point P of order n as described above, and let $\mathbb{G} = \langle P \rangle$. Choose $Q' \xleftarrow{R} \mathbb{G} \setminus \{\infty\}$ and set $Q = [r]Q'$; then Q has order q.[1] Let $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mu_n \subset \mathbb{F}_{p^2}$ be the modified Weil pairing (constructed from the Weil pairing using a distortion map).
  Output the public key $\mathsf{pk} = (E, \hat{e}, n, P, Q)$ and the secret key $\mathsf{sk} = q$.
- $\mathsf{Enc}(\mathsf{pk}, m)$: Choose $t \xleftarrow{R} [1, n]$ and output $C = [m]P + [t]Q$.
- $\mathsf{Dec}(\mathsf{sk}, C)$: Compute $\tilde{P} = [q]P$ and $\tilde{C} = [q]C$, and output $m' = \log_{\tilde{P}} \tilde{C}$.

Decryption is correct since if $C = [m]P + [t]Q$, then $\tilde{C} = [mq]P + [qt]Q = [mq]P = [m]\tilde{P}$. Note that for efficient decryption we require the message space to be small as in the ElGamal variant.

We now describe how to add encrypted messages and perform one multiplication.

- $\mathsf{Add}(\mathsf{pk}, C_1, C_2)$: Choose $t' \xleftarrow{R} [1, n]$ and output $C' = C_1 + C_2 + [t']Q \in \mathbb{G}$.
- $\mathsf{Mult}(\mathsf{pk}, C_1, C_2)$: Choose $u \xleftarrow{R} [1, n]$ and output $D = \hat{e}(C_1, C_2) \cdot e(Q, Q)^u \in \mu_n$.

It is easy to see that if $C_1 = [m_1]P + [t_1]Q$ is an encryption of m_1 and $C_2 = [m_2]P + [t_2]Q$ is an encryption of m_2, then $\mathsf{Add}(\mathsf{pk}, C_1, C_2)$ is a (rerandomized) encryption of $m_1 + m_2$. With the same setup, we have

$$\begin{aligned} \mathsf{Mult}(\mathsf{pk}, C_1, C_2) &= \hat{e}([m_1]P + [t_1]Q, [m_2]P + [t_2]Q) \cdot \hat{e}(Q, Q)^u \\ &= \hat{e}(P, P)^{m_1 m_2} \hat{e}(P, Q)^{m_1 t_2 + t_1 m_2} \hat{e}(Q, Q)^{t_1 t_2 + u}. \end{aligned}$$

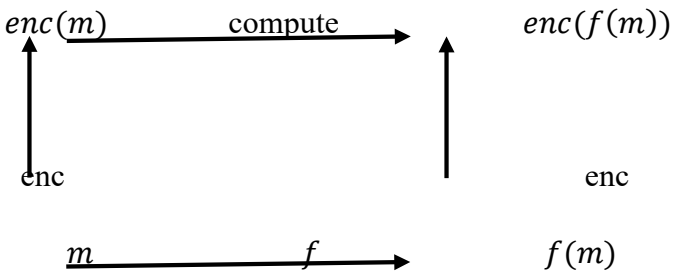
By the nondegeneracy of the pairing, $g = \hat{e}(P, P)$ is an element of $\mathbb{F}_{p^2}$ of order n and $h = \hat{e}(Q, Q)$ is an element of $\mathbb{F}_{p^2}$ of order q. Furthermore, $\hat{e}(P, Q)^q = \hat{e}(P, [q]Q) = 1$, so $\hat{e}(P, Q)$ has order q as well and is equal to h^a for some a. Thus we have

$$D = \mathsf{Mult}(\mathsf{pk}, C_1, C_2) = g^{m_1 m_2} h^{u'}$$

This scheme can be used to evaluate quadratic functions (with some bounds). It provides an optimal system for supersingular curves in terms of efficiency and security.

2.3.11 Gentry Scheme

For all types of calculation on the data stored in the cloud, the optimal approach would be of fully Homomorphic encryption which is able to execute all types of operations on encrypted data without the need of decryption. In 2009 Craig Gentry of IBM proposed the first encryption system "fully homomorphic" that computes any number of additions and multiplications and thus calculate any type of function on encrypted data.



The first fully homomorphic system was given by Gentry in 2009 and it was a lattice based system. The Gentry scheme makes use of ideal lattices for denoting the cipher text and the keys.

The private key of the system, V; which is in lattice form; is generated randomly by the system along with an additional matrix W such that they satisfy the following [14]:

$$V \times W \equiv c \bmod f(x) \qquad c = Constant \qquad [5]$$

B is the public key which is the Hermite Normal of V which can be represented as integers.

An application of Fully Homomorphic Encryption Scheme could be submitting an encrypted query to any search engine, which would process the encrypted query and display results in an encrypted from, which can be decrypted by the user. By way of this, even the search engine or the Cloud service provider wouldn't know what the query was and what the result was, hence enforcing data privacy and security.

The application of fully Homomorphic encryption is an important one in Cloud Computing security, more generally, we could outsource the calculations on confidential data to the Cloud server, keeping the secret key that can decrypt the result of calculation. [10] [30]

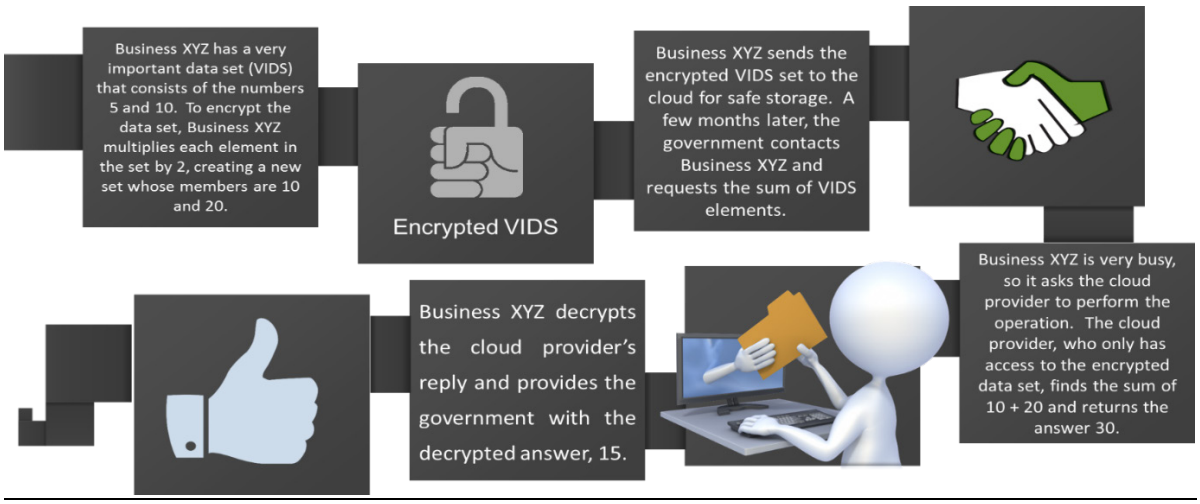


Figure 3This shows exploitation/usage of Homomorphic Encryption in Cloud Computing.

Note: For simplicity only addition is operation is shown, in case of a fully homomorphic scheme this can be any set of operations.

CHAPTER 3
E-VOTING

Electronic voting systems are basically using machines to vote. Some electronic voting systems require voters to go to their local polling station to vote through machines that have been designed and designated for voting only. It is a kind of convenience when machines can cast votes, save them, and transfer to another machine, which tallies all the votes together [43]. However, voters still have to spend their time to go to a local polling station to vote. With rapid development of technology and the Internet, researchers are developing modern E-voting systems that allow voters vote through their computer from anywhere. In this project, voters should have their email address so that the voting organization can send the voting software and ID to them. They can install the software on their and process their vote accordingly.

Electronic voting systems allow people to vote for candidates anywhere using computers, which are connected to the Internet [49]. This method has multiple advantages in comparison to traditional methods such as paper ballot. Voters do not need to present at ballot box. It provides ease of accessibility and comfort of use. Another advantage of Internet voting is wide usability that can be applied in all forms of municipal, public, state elections, and referendums [49]. Using an electronic voting system, voters also can vote rapidly and exactly because they don't need to go to Ballot station and vote through their computers. However, it is relatively insecure to deploy an electronic voting system because it requires transferring of data and information over the Internet. Many security measures should be taken to protect ballots and the system from possible attacks.

Some requirements that users must consider to design e-voting system have been discussed in the previous sections. Let us now understand the cryptology involved.

3.1 Cryptography

Cryptographic techniques are used for secure storage, communication, and handling of data and information [53]. It is beyond the scope of this report to discuss all these techniques. In this

report, we primarily focus on cryptographic voting schemes that are relevant to the project. Cryptographic techniques have been used to voting systems since 1981[44]. These techniques provide a level of assurance of accuracy and secrecy to a voting system. Largely cryptosystems can be divided to two systems: symmetric key cryptosystems and asymmetric key cryptosystem [43].

A symmetric key system uses the same secret key to encrypt and decrypt the message. For example, if A want to send a message m to B, he will use a secret key (K!,!) already shared between A and B to encrypt the message m and send to B. Then, B also uses the same secret key (K!,!) to decrypt the encrypted message and get the original message. On the other hand, asymmetric key cryptosystem uses two different keys to encrypt and decrypt the message. These keys, one of them called the public key and the other the private key, are different. A uses the public key, which can be shared with anyone who wants to send a message to B, to encrypt a message and then send it to B. Next, B uses the private key that he or she does not share with anyone to decrypt the message. Both schemes have different strengths in terms of security. For instance, asymmetric cryptosystems provide privacy and reliability but it does not work with authentication. However, an asymmetric cryptosystem can provide authentication but it is more complex than a symmetric system. A number of cryptographic schemes are used to design a voting system. Most of these schemes can count the votes securely. Mostly frequently used algorithms are RSA (Rivest, Shamir, and Adleman) encryption, ElGamal encryption, and Paillier encryption [44] [42] [43].

USING PAILLIER HOMOMORPHIC PROPERTIES IN E-VOTING SYSTEM

The encryption algorithm E is homomorphic if given $E(m1)$ and E(m2), one can obtain E(m1 +/× m2) without decrypting m1 and m2. A straightforward way, the users want to calculate the product of message m1 and message m2. All they have are the encrypted message E(m1) and E(m2) but they do not want to decrypt E(m1) and E(m2), and then calculate the product (m1 +/× m2). To calculate it, they will calculate T = E(m1) +/× E(m2). Finally, they just decrypt T to get (m1 +/× m2). This property can calculate (m1 +/× m2) but the user will not know what is message m1 and message m2. This is a major feature that is used to tally ballots in an E-

voting system. Pallier algorithm is one of algorithms which have this homomorphic property. Addition of the encrypted ballots will be in the encrypted tally [48] [43]. When an administrator decrypts the Paillier result, he will get the final result of a poll but he will not know which voter voted for which candidate.

Here is how to use Paillier encryption's additive homomorphic property for tallying votes [42] [44]:

Let us call the number of voters is N_v and the number of candidates is N_c. The base used to encrypt messages, is greater than the number of voters ($b > N_v$).

Next, the vote messages for candidates will be seen as: 1st candidate is b^0, 2nd candidate is b^1, 3rd candidate is b^2, etc. N_c-th candidate is $b^{(N_c-1)}$.

Then the maximum possible number representing a single vote m_{max} can be expressed as:

$$m_{max} = \sum_{i=1}^{Nc} b^{i-1}$$

The maximum possible tally of all votes can be:

$$T_{max} = N_v \times m_{max}$$

Then, there are three major steps that coders have to consider namely Key Generation, Encryption and Decryption.

3.1.1 Step 1: Key generation

Public Key

To be able to encrypt T_{max}, RSA modulus n must hold the following:

$n \geq T_{max} + 1$ Where $n = pq$

where p and q are large primes and gcd$(pq, (p-1)(q-1)) = 1$.

A random integer g is selected where $g \in Z^*{}_{n^2}$ and gcd $(\frac{g^{\lambda} mod\ (n^2-1)}{n}, n) = 1$

The election authorities should choose the prime numbers p and q considering the number of voters and candidates as described above.

Private Key

$\lambda = lcm(p-1, q-1)$ with $\lambda(n)$ being the Carmichael function.

Modular multiplicative inverse: $\mu = (L(g^{\lambda} mod\ n^2))^{-1}\ mod\ n$

where function L is defined as $L(u) = \frac{u-1}{n}$

3.1.2 Step 2: Encryption

$E(m_i) = c_i = g^{m_i} \times r_i^n\ mod\ n^2$ where $r \in Z_n^*$

Tallying

At the end of the election, authorities would have at most N_v of encrypted votes. Then authorities can calculate the encrypted tally, which is the product of all encrypted votes modulo n^2.

$T = \prod_{i=1}^{N_v} c_i\ mod\ n^2$ T-Tally

3.1.3 Step 3: Decryption

As described in homomorphic properties of Paillier encryption:

$$m = L(g^{\lambda} \bmod n^2) \times \mu \; mod \; n$$

$$D(T) = \sum_{i=1}^{N_v} m_i \; mod \; n \qquad D(T)\text{-Decryption of } T$$

As a result of this decryption function, one gets simple tallying of all votes. To determine how many votes cast for each candidate we can use the "Division remainder" method with number of the voters as base. The following example illustrates the use of Paillier's cryptosystem in voting.

Example

This example demonstrates the use of Paillier algorithm for tallying a small number of votes, similar to the examples in [42] and [43]. Let assume that there are 9 voters A1, A2, A3, . . ., A9 and 6 candidates B1, B2, B3, . . ., B6 so that $N_! = 9$ and $N_! = 6$ Let us select $b > N!$, say $b = 10$. Voters are supposed to select only one candidate. The vote messages to be encrypted are shown in the following table [42] [43].

Vote Messages to be Encrypted							
Voter Name	**B1** $(10^!)$	**B2** $(10^!)$	**B3** $(10^!)$	**B4** $(10^!)$	**B5** $(10^!)$	**B6** $(10^!)$	**Vote messages (m)**
A1		V					$m = 10^! = 10$
A2			V				$m = 10^! = 100$
A3	V						$m = 10^! = 1$
A4				V			$m = 10^! = 1000$
A5				V			$m = 10^! = 1000$
A6					V		$m = 10^! = 10000$
A7						V	$m = 10^! = 100000$
A8	V						$m = 10^! = 1$
A9				V			$m = 10^! = 1000$
Total	2	1	1	3	1	1	

The maximum vote message can be: $m_{max} = 10^5 = 100000$

So the maximum possible tally can be: $T_{max} = N_v \times m_{max} = 9 \times 10000 = 90000$

90000.

Let us perform the following three steps of the Pallier algorithm.

Step 1: Key generation

- Choose two primes randomly p and $q > \sqrt{90000}$, $p = 293$, $q = 433$ that gcd $(pq, (p\text{-}1) \times (q\text{-}10)) = 1$
- Calculate n = p×q = 293×433 = 126869, $n^2 = 16095743161$ and $\lambda = lcm(p-1, q-1)$ = lcm(292, 432) = 73×432 = 31536
- Choose Paillier generator g randomly where $g \in Z^*_{n^2}$ and gcd $(\frac{g^\lambda mod\ (n^2-1)}{n}, n) = 1$, so g = 2
- Calculate $\mu = (L(g^\lambda mod\ n^2))^{-1}\ mod\ n = 105161$

Step 2: Encrypt each message m as shown in the following table

$E(m_i) = c_i = g^{m_i} \times r_i^n\ mod\ n^2 = 2^{m_i} \times r_i^{126869}\ mod\ 16095743161$

Encrypted Vote C_i			
Voter Name	**Vote message to be encrypted**	**Random r_i**	**Encrypted Vote C_i**
A1	m= $10^!$ = 10	26181	1476346097
A2	m= $10^!$ = 100	11593	2441495758
A3	m= $10^!$ = 1	47971	4580939420

A4	m= 10! = 1000	15791	10051435966

A5	m= $10^!$ = 1000	28737	1698861485
A6	m= $10^!$ = 10000	39024	7276056190
A7	m= $10^!$ = 100000	21014	8664547807
A8	m= $10^!$ = 1	45258	6400165985
A9	m= $10^!$ = 1000	42283	11929667045

After tally server received all encrypted messages from voters, the server will multiply all encrypted messages following Paillier algorithm as shown below.

$$\text{Tally (T)} = \prod_{i=1}^{Nv} c_i \; mod \; n^2 = (1476346097 \times 2441495758 \times 4580939420 \times 10051435966 \times 1698861485 \times 7276056190 \times 8664547807 \times 6400165985 \times 11929667045) \; mod \; 1609574361 = 13722328518$$

Then the tally server decrypts Tally T to get tally message M.

$$M = \mathrm{L}(c^{\lambda} \; mod \; n^2) \times \mu \; mod \; n =$$

$$\left(\frac{(13722328518^{31536} \; mod \; 1609574361)-1}{126869}\right) \times 105161 \; mod \; 126869 = 113112$$

This decrypted tally already has the base 10, so the tally server does not need to convert it. Finally, the result of this voting is decided. Since candidate $B4$ has 3 votes, which is the largest number of votes received by a candidate, so he or she is the winner of this election.

CHAPTER 4
Implementation on Cloud

The application of fully Homomorphic encryption is an important brick in Cloud Computing Security; more generally, outsourcing of the calculations on confidential data to the Cloud server is possible, keeping the secret key that can decrypt the result of calculation. In our implementation, we analyze the performance of existing homomorphic encryption cryptosystems, and are working on a virtual platform as a Cloud server, a VPN network that links the Cloud with the customer, and then simulating different scenarios. For example, a Database-Server communicating with Client using FHE Cryptosystem is as shown in the figure below. [41]

4.1 Connecting using SSH

4.1.1 SSH Description

SSH was developed to provide a more secure alternative to Telnet. However, Telnet poses some very serious security issues. Most notably, passwords are sent in plaint text over Telnet. In a cloud computing environment, this is extremely risky and exposes computers and networks to a variety of major threats. With SSH, two networked computers are able to connect in a highly secure manner, even if they share an insecure connection.

Although SSH is most commonly associated with UNIX®-like operating systems, it can be used on Windows® PCs as well. In fact, now that cloud computing is becoming so ubiquitous, it is being used to connect remotely to Windows PCs on an even more regular basis.

Currently, SSH is the simplest, most elegant way to connect remotely to PCs in order to execute a variety of commands. When connectivity issues arise, the SSH protocol can be used to diagnose problems and implement solutions.

4.1.2 Uses of SSH

The SSH protocol is a handy tool for any IT professional, especially in the age of cloud computing. With SSH, it is easy to connect quickly and securely to a remote computer

through a command line interface. Once a connection is established, commands can be executed remotely.

SSH is most often used to resolve connectivity problems. However, it can also be used for tunneling, X11 connections, and forwarding TCP ports. Unlike UNIX-like operating systems, Windows does not include SSH protocol by default. However, it can be added by downloading an SSH client.

IT professionals can diagnose and repair connectivity problems through SSH connections. Files can also be transferred, and SSH is popularly used for this purpose as well. File transfers are usually conducted through SSH file transfer, or SFTP. They can also be conducted through the secure copy protocol, or SCP. As with executing commands, these files transfers are conducted in a highly safe and secure manner.

When an SSH connection is made, public key cryptography ensures the safety and integrity of the data that is subsequently transmitted.

4.1.3 Using SSH

SSH (*Secure SHell*) is a network protocol that allows you to connect to a remote computer (like your Cloud Server) via command-line interface.

SSH uses a cryptography system both for the authentication and for the work session and this is why it is preferred compared to other protocols and has in fact become the standard in remote administration of Unix / Linux systems.

4.1.4 Connect to the Linux Cloud Server

To connect to the Linux Cloud Server from a computer with **Linux Operating System** simply use the SSH command

```
ssh 192.168.123.253 -l root
```

(*where the IP is the IP of the Cloud Server; by pressing the enter key you will be requested the password of the Cloud Server*)

Once the authentication has been made it will be possible to perform all the SSH commands to run your Linux Cloud Server.

4.2 Challenges on Cloud

The double layer of encryption causes the system runs too slowly for practical use. We are working on optimizing the same for specific applications such as searching databases for econfidentiality is not feasible and it requires considerable (~10 yrs) of usage exposure. A team from MIT's Computer Science and Artificial Intelligence Laboratory, who worked in conjunction with the University of Toronto and Microsoft Research, sought to combine multiple schemes to solve these challenges. The system starts with homomorphic encryption, with a decryption algorithm embedded in a garbled circuit which is itself protected by attribute-based encryption this ensures the process stays encrypted.

CHAPTER 5
DESIGN

The sequence diagram for the Homomorphically Encrypted E-Voting system is shown here. The four entities involved are The Client, The Server Program, Database and The Administrator. The messages are shown by arrows and replies by broken arrows.

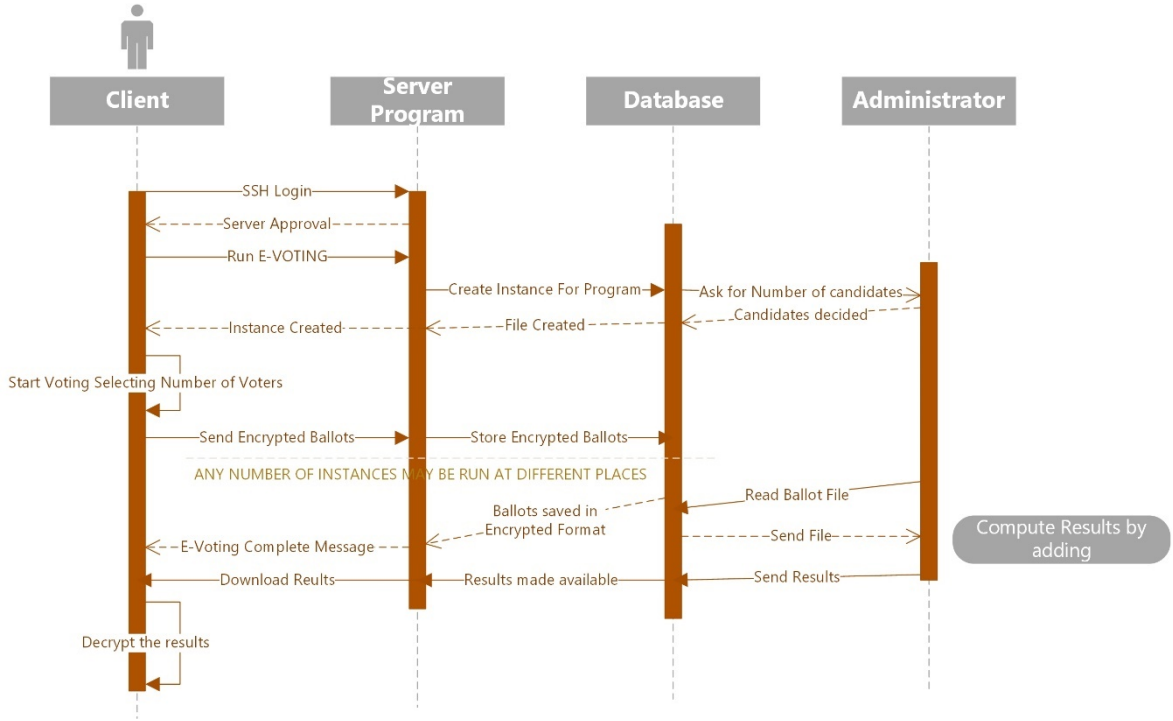


Figure 4 Sequence Diagram

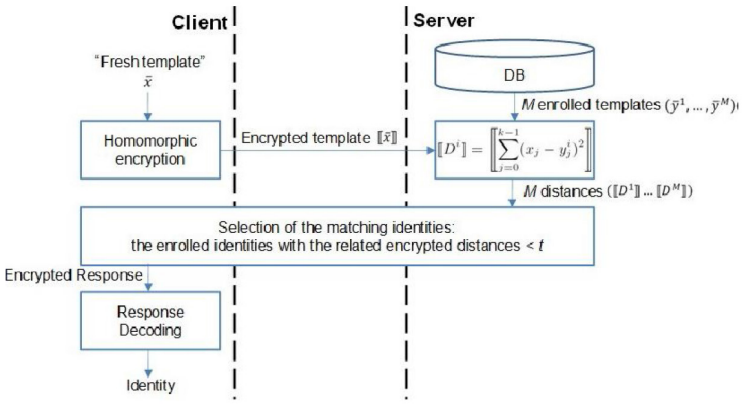


Figure 5 A Database-Sever & Client implementing Homomorphic Encryption

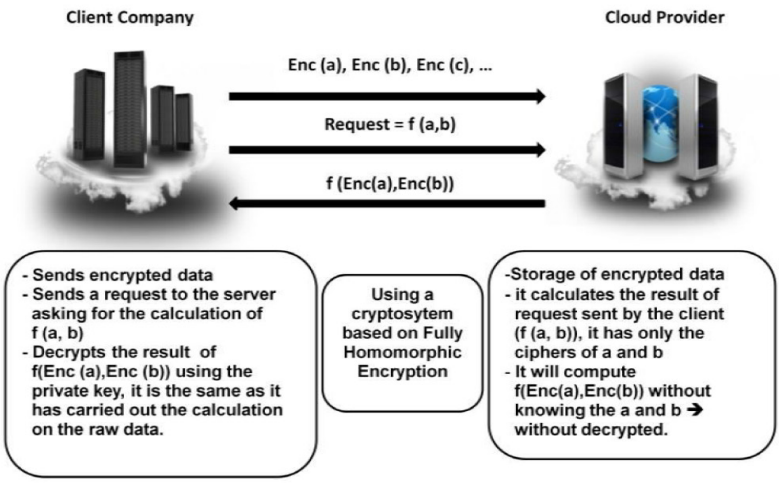


Figure 6 The Cloud Computing scenario

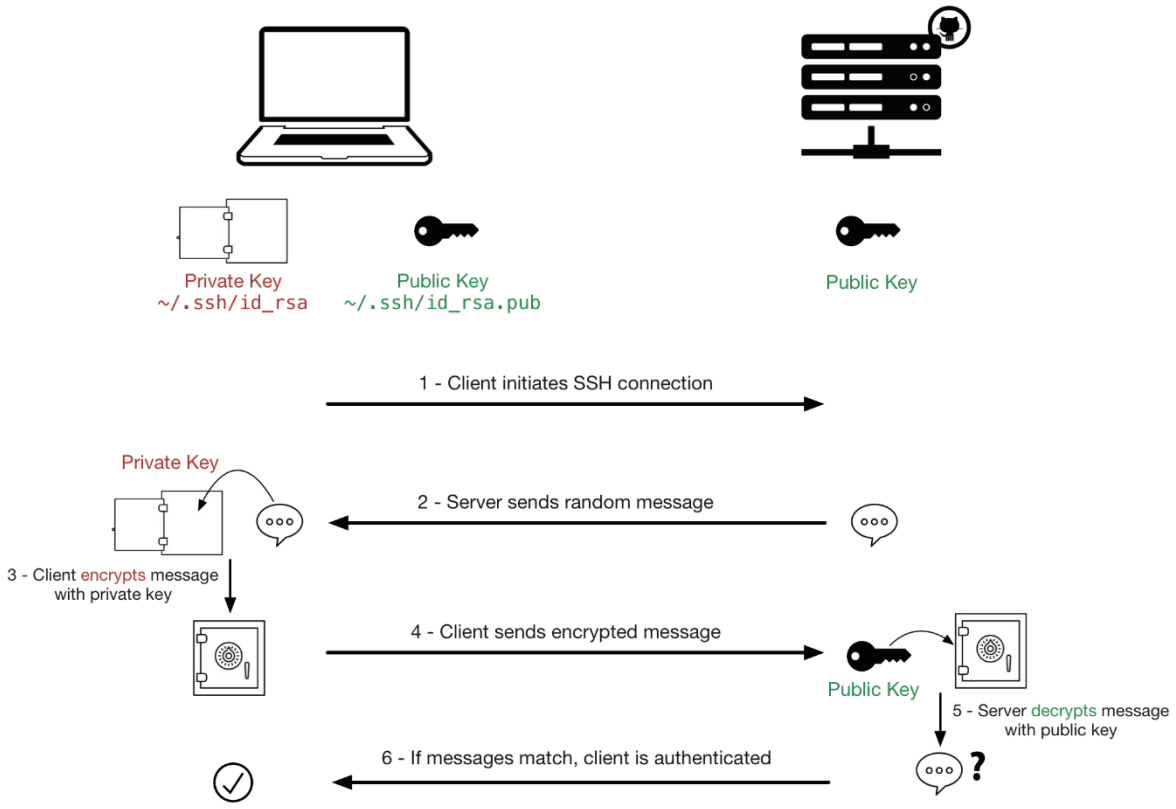


Figure 7 SSH Secured Connection Scenario

SSH provides extra security in the form of encrypted private tunnel on a public network to protect it from hackers. Even if the hackers manage to break the security they will only find the encrypted values that were a part of the Homomorphic System. So this also provides two-layered security.

CHAPTER 6
RESULT AND ANALYSIS

Now the voting algorithm can be strengthened using the Paillier Cryptosystem Technique.

All the casted votes could first be encrypted and stored on the Cloud server and then they could be added homomorphically to produce the encrypted result.

At the end we can get the tally of votes by which we can extract the total number of votes for each candidate y using division and the remainder theorem. The decryption and further computation of the result would then finally give the winner of the election process.

The result of the Voting Process after all computations is as follows:

Votes for Candidate #1 = 5 (WIN- NER)

Votes for Candidate #2 = 3

Votes for Candidate #3 = 2

Votes for Candidate #4 = 2

Votes for Candidate #5 = 2

The below mentioned example would help get a clear picture.

EXAMPLE

Consider there are 14 voters and 5 candidates (C1, C2, C3, C4, C5) standing for election.

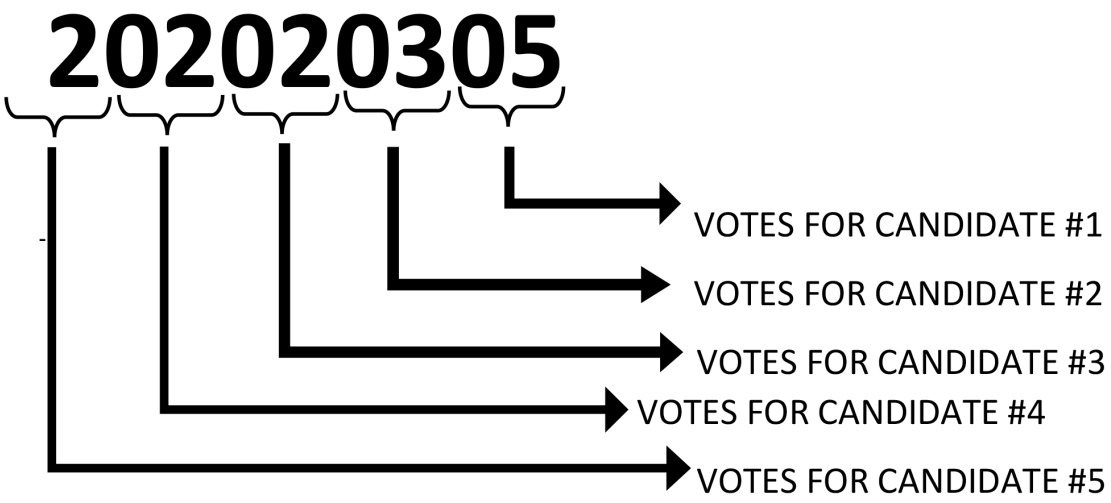
VOTER NUMBER	C1	C2	C3	C4	C5	COMPUTATION x
V1	✓					$x = 10^{2*(1-1)} = 1$
V2		✓				$x = 10^{2*(2-1)} = 100$
V3	✓					$x = 1$
V4		✓				$x = 100$
V5			✓			$x = 10^{2*(3-1)} = 10000$
V6					✓	$x = 10^{2*(5-1)} = 100000000$
V7					✓	$x = 100000000$
V8				✓		$x = 10^{2*(4-1)} = 1000000$
V9			✓			$x = 10000$
V10		✓				$x = 100$
V11	✓					$x = 1$
V12	✓					$x = 1$
V13				✓		$x = 1000000$
V14	✓					$x = 1$
TOTAL	**5**	**3**	**2**	**2**	**2**	

✓ : Vote cast to

After casting all the votes all the votes are added.

So, $X = 202020305$ (after addition of all x values).

From this X we can compute votes for each of the five candidates as:



The following examples proves the aunthencity of the voting process. The values that appear in the above example may seem as simple values but in the actual process those will be in an encrypted form.

TIME ANALYSIS OF SOME HOMOMORPHIC ALGORITHMS

Table 6-1 Specifications of virtual machines

Specification	**VirtualBox**	**VMPlayer**	**QEMU/KV M**
No. of CPUs	2 VCPU	2VCPU	2 VCPU
RAM	2GB	2GB	2GB
Host OS	Fedora	Fedora	Fedora
Guest OS	Opensuse	Opensuse	Opensuse
Network configure	Attached to : Bridged adapter	Attached to : Bridged adapter	Attached to : NAT
Network Bandwidth	10/100 Mbps	10/100 Mbps	10/100 Mbps
Harddisk	15GB	15GB	15GB
Database	Mysql	Mysql	Mysql
IDE	NetBeans- 7.4	NetBeans- 7.4	Netbeans- 7.4
Language	Java	Java	Java

Table 6-2 Paillier Algorithm

VH/HM	**One lakh 10-Digits Plain data storing onto Remote Database Timings(ns)**	**One lakh 10-digit Plain Data Retrieving from Remote Database Timings(ns**	**Average two 10-digits Encrypted data storing timings on remote Database(ns)**	**Two one lakh Encrypted data retrieving timings from remote database(ns)**
HM	79602938359	244758194	1007085	5352871322
VirtualBox	11033023546	33218317	1116383	5560466914
VMPlayer	92647282758	29774894	1103970	5670430487
QEMU/KV	M118758064	28011704	1370803	5596187770

Table 6-3 ElGalmal Algorithm

VM/HM	**Average two one lakh ten digit numbers encrypted data storing timings(ns)**	**Retrieving two one lakh encrypted data timings (ns)**
HM	778212.2146	376793541
Virtual Box	1113011.3910	401117791
VMPlayer	934501.4437	378696177
QEMU/KVM	1205078.1696	393303481

Table 6-4 RSA Algorithm

VM/HM	Average two one lakh ten digit Numbers Encrypted data storing timings(ns)	Retrieving two one lakh ten digit encrypted data timings(ns)
HM	811439.2649	563870998
Virtual Box	1119615.4912 1	572365618
VMPlayer	933226.9286	556169480
QEMU/KVM	1229763.2439	567895853

Table 6-5 Benolah Algorithm

VM/ HM	Average two one lakh ten digit numbers encrypted data storing timings(ns)	Retrieving two one lakh ten digit encrypted data timings (ns)
HM	1061823.9314	5749744411
Virtual Box	1265576.2116	5349154763
VMPlayer	1104598.3080	5716719636
QEMU/KVM	1563666.8861	5975955788

Table 6-6 Average timings for two one lakh ten digits plain data addition and multiplication

Platform	Average Addition of two one lakh ten digit numbers timing(ns)	Average multiplica- tion of two one lakh ten digit numbers timing(ns)
Host	424.2612	516.3624
VirtualBox	513.0177	451.9619
VMplayer	299.5329	248.4567
QEMU/KVM	567.0812	372.9772

Table 6-7 Overhead of four Algorithms homomorphic property over plain data in numbers of times

Platform	Algorithm	Over head Addition of Encrypted dataover plain data for 10-digit numbers	Over Head Multiplication of encrypted data over plain data for 10-digit numbers
Host	Paillier	17 times	278 times
Host	Elgamal	NA	1.72 times

Host	RSA	NA	1.79 times
Host	Benaloh	5.6 times	NA
Virtual Box	Paillier	15 times	399 times
Virtual Box	Elgamal	NA	1.6 times
Virtual Box	RSA	NA	1.5 times
Virtual Box	Benaloh	5.36 times	NA
VMPlayer	Paillier	22.4 times	518 times
VMPlayer	Elgamal	NA	11.7 times
VMPlayer	RSA	NA	3.48 times
VMPlayer	Benaloh	5.48 times	NA
KVM	Paillier	11.99 times	346 times
KVM	Elgamal	NA	8.9 times
KVM	RSA	NA	1.5 times
KVM	Benaloh	3.5 times	NA

CHAPTER 7
LIMITATIONS

While the benefits to homomorphic encryption are great, they do not come without considerable limita. One of the biggest drawbacks is the complexity of the systems. In partially homomorphic cryptosystems, there is not much overhead involved in performing the computations, at least for those presented. However, fully homomorphic encryption requires a lattice-based cryptosystem that is significantly more complex. Implementation of such a cryptosystem even for basic operations requires significantly more complicated computations and massive ciphertext sizes. When using recommended security parameters, ciphertexts produced are on the order of 128MB and a public key of 128PB. Ciphertexts and public keys of this size are simply not practical. Even when minimizing security parameters to the point of homomorphism no longer being possible, the key size is still on the order of several GB, with encryption of a single bit still requiring up to 30 minutes. [1]

Another potential drawback of homomorphic cryptosystems is that in some cases, they are vulnerable to malware. Consider the case of a secure voting system that implements additively homomorphic encryption. If one of the voting booths were to be infected with malware, it is within the realm of possibility that the malware could manipulate votes before they are submitted. Similar flaws could be applied to nearly any other system that exhibits homomorphism, such as a homomorphic cryptosystem used for banking in which transactions are modified to withdraw or deposit a different amount than the user intended. [1]

It has not achieved wide review and acceptance

While there is promising work on homomorphic encryption, there is no clear consensus on the best method or implementation approach. Typically a new cryptographic method will not get a full review from the cryptographic community until there is some consensus, and not until a standards body takes up the new method in a formal review process. There are a large number of potentially good encryption methods that have been thoroughly reviewed by the professional cryptographic community but which have not achieved the status of an approved standard.Homomorphic encryption has not yet been through this process and it is too early to trust any current proposals or implementations. [28]

It is not a standard

Standards are important in the encryption world. Standard encryption algorithms receive the full scrutiny of the professional cryptographic community and we all benefit from this. Weaknesses are discovered much faster, weak implementations are identified, and we all have much more confidence in encryption based on standards. The Advanced Encryption Standard (AES) has stood the test of time since its adoption by NIST in 2001.

Homomorphic encryption has not yet achieved the status of an accepted and published standard.

Note: Mathematical proofs do not a standard make. They are required as a part of the standards review and adoption process, but mathematical proofs alone do not rise to a level of an accepted standard. Claims to the contrary are false. [28]

It cannot be certified by a standards body

Since homomorphic encryption is not a standard, there is no independent standards body process to validate a vendor's implementation. This is important - in an early study by NIST of encryption solutions submitted for validation, nearly 37% of the solutions contained errors in the implementation and failed validation. The failure rate for implementations of homomorphic encryption are likely as high and unknowable. All serious vendors of encryption technology have validated their AES implementations to FIPS 197 standard through the NIST AES validation process. No such similar standards validation process exists for homographic encryption. [28]

CONCLUSION AND FUTURE SCOPE

This project, based on SSH client/server and Paillier encryption, implements a secure E-voting system. This combination warrants that the vote can be transferred over the Internet securely and counted correctly. This E-voting system ensures the end-to-end verification of the whole voting process. It is the quickest, cheapest, and the most efficient way to vote. Thus, if an election supports this E-voting system, voters can use it to vote without worrying about their votes. Moreover, they can save their time by not going to a polling station to vote. All in all, the project meets important secure requirements of an E-voting system. Some other future work as mentioned can be done in further research to complete the professional voting system.

This E-voting system works correctly, but it is still simple. It can implement some secure methods that can convince voters using the E-voting system instead of using traditional voting. However, to have a professional E-voting system, the following challenges should be further investigated in future work.

First, based on this material, a developer can write a plugin that voters can install in their web browsers. They can use web browsers to vote. The plugin has to ensure the use of the Paillier algorithm for tallying and SSH client/server for communication between a client and a server.

Second, applications to vote from smartphones can be built as well. There are two platforms needed to be considered to develop them: Android and iOS. Accordingly, voters will need to download the voting application and install to their smartphones, then they will be able to vote. It is much more convenient because at the present, most people use smartphones. However, cryptography algorithms and the SSH client/server protocol should be considered for such implementation. All methods should be supported to develop a professional E-voting system for such platforms.

REFERENCES

[1] Liam Morris. Analysis of Partially and Fully Homomorphic Encryption

[2] Boaz Barak. Lecture 19 — Homomorphic Encryption 1: Definition and Application to Private Information Retrieval, Zero knowledge.

[3] Rebecca Meissen. A Mathematical Approach to Fully Homomorphic Encryption

[4] http://homepages.math.uic.edu/

[5] http://cris.joongbu.ac.kr/publication/evoting_implementation-APIEMS2004.pdf Implementation issues in a secure e-voting schemes, Riza Aditya, Byoungcheon Lee, Colin Boyd and Ed Dawson.

[6] http://en.wikipedia.org/wiki/Public-key_cryptography Public-key cryptography

[7] [7] I. Damgard, M. Jurik, J. Nielson, A Generalization of Paillier's Public-Key System with Applications to Electronic Voting, Aarhus University, Dept. of Computer Science.

[8] http://en.wikipedia.org/wiki/Paillier_cryptosystem , Paillier Crytosystem from Wikipedia, the free encyclopedia.

[9] http://www.brics.dk/RS/00/45/BRICS-RS-00-45.pdf Ivan Damgard and Mads J. Jurik, A Generalization, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System, PKC 2001.

[9] B. Wilson, C. E. Chow, Paillier Threshold Cryptography Web Service User's Guide, University of Colorado – Colorado Springs Master's Project, 2006.

[10] Gentry, C., Halevi, S. Implementing Gentry's Fully-Homomorphic Encryption Scheme. 2011.

[11] Lo Ducas and Daniele Micciancio. FHEW: Bootstrapping Homomorphic Encryption in less than a second.

[12] Zvika Brakerski, Craig Gentry, Vinod Vaikuntanathany. Fully Homomorphic Encryption without Bootstrapping.

[13] https://mpclounge.wordpress.com/2013/04/29/new-fully-homomorphic-encryption-library/

[14] https://www.comodo.com/resources/small-business/digital-certificates2.php

[15] Rothblum, R.: Homomorphic encryption: from private-key to public-key. Electronic Colloquium on Computational Complexity (ECCC) 17, 146 (2010), http://eccc.hpi-web.de/report/2010/146

[16] http://shaih.github.io/HElib/

[17] http://link.springer.com/chapter/10.1007%2F978-3-662-44371-2_31

[18] https://pwnhome.wordpress.com/category/helib/

[19] http://shaih.github.io/HElib/files.html

[20] https://gmplib.org/

[21] https://www.cygwin.com/ml/cygwin/2004-08/msg01165.html

[22] http://cpp.knowcoding.com/view/163247-installing-gmp-library.html

[23] http://www.shoup.net/ntl/

[24] http://src.gnu-darwin.org/ports/math/ntl/work/ntl-5.4/doc/tour-unix.html

[25] http://www.cryptopp.com/docs/ref/

[26] http://www.cryptopp.com/wiki/Main_Page

[27] http://web.townsendsecurity.com/bid/72771/Homomorphic-Encryption-is-Cool-and-You-Should-NOT-Use-It

[28] Gentry, C. Fully homomorphic encryption using ideal lattices. 2009.

[29] Gentry, C., Halevi, S. Implementing Gentry's Fully-Homomorphic Encryption Scheme. 2011.

[30] Paillier, P. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. 1999.

[31] Smart, N., Vercauteren, F. Fully Homomorphic Encryption with Relatively Small Key and Ci- phertext Sizes. 2009.

[32] Snook, M. Integer-Based Fully Homomorphic Encryption. 2011.

[33] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In CRYPTO, pages 10–18, 1984.

[34] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, pages 169–178. ACM, 2009.

[35] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-hop homomorphic encryption and rerandomizable yao circuits. In Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, pages 155–172. Springer, 2010.

[36] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. Journal of Computer and System Sciences, 28(2):270–299, 1984.

[37] Oded Goldreich. Foundations of Cryptography. Volume I: Basic Tools. Cambridge University Press, 2001.

[38] Oded Goldreich. Foundations of Cryptography: Volume 2: Basic Applications. Cam- bridge University Press, 2004.

[39] Benaloh, Josh (1994). Dense Probabilistic En- cryption. (PS). Workshop on Selected Areas of Cryptography. pp. 120128.

[40] A Fully Homomorphic Encryption Implementation on Cloud Computing, Shashank Bajpai and Padmija Srivastava, Cloud Computing Research Team, Center for Development of Advanced Computing [C-DAC], Hyderabad, Ministry of Communications and Information Technology, Government of India.

[41] Choinyambuu, S. (2010, February, 1). HS09_Homomorphic_Tallying_with_Paillier.

[42] Fontaine, C., & Galand, F. (2007). A Survey of Homomorphic Encryption for Nonspecialists. EURASIP Journal On Information Security, (1), 013801.

[43] Ryan, P. Y. A. (2007). The computer ate my vote. Retrieved from http://www.dagstuhl.de/Materials/Files/07/07091/07091.RyanPeter.Paper.pdf

[44] Oracle Help Center. (2010). To use keytool to create a sever certificate. Retrieved from http://docs.oracle.com/cd/E19798-01/821-1841/gjrgy/index.html

[45] Mac Developer Library (2004). This manual page is part of Xcode tools version 5.0. Retrieved from https://developer.apple.com/library/mac/documentation/Darwin/Reference/Manpages/ man1/keytool.1.html

[46] Academia de Studii Economice a Moldovei (ND). Muiti-threaded client/server applications. Retrieved from http://www.ase.md/~aursu/ClientServerThreads.html

[47] Andreas, S. (2009). E-voting glossary. Retrieved from http://security.hsr.ch/msevote/glossary.html

[48] Vejacka, M. (2013). Evaluation of Internet Voting Systems based on Requirements Satisfaction. International Review Of Social Sciences & Humanities, 6(1), 4-52.

[49] Gilbert, J. E., Dunbar, J., Ottley, A., & Smotherman, J. (2013). Anomaly detection in the electronic voting systems. Information Design Journal (IDJ), 20(3), 194-206. doi:10. 1075/idj.20.3.01gil

[50] Jardi-Cedo, R., Pujol-Ahullo, J., Castella-Roca, J., & Viejo, A. (2012). Study on poll-site voting and verification systems. Computers & Security, 31989-1010. doi:10.1016/j.cose.2012.08.001

[51] Deitel, P., & Deitel, H. (2010). Java how to program (Eight ed.). New Jersey: Pearson

[52] Schneier, B. (1996). Applied cryptography : protocols, algorithms, and source code in C / Bruce Schneier. New York : Wiley, c1996.

[53] The Open Web Application Security Project (2010). Using the java secure socket extensions. Retrieved from https://www.owasp.org/index.php/Using_the_Java_Secure_Socket_Extensions

[54] Cisco (ND). SSL: Foundation for web security. Retrieved from http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_1-1/ssl.html

[55] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. Communications of the ACM, 21(2): 120-126, 1978. Computer Science, pages 223-238. Springer, 1999

[56] Rabin, Michael. Digitalized Signatures and Public-Key Functions as Intractable as Fac- torization. MIT Laboratory for Computer Sci- ence, January 1979.

[57] A New Rabin-type Trapdoor Permutation Equivalent to Factoring and Its Applications.

[58] S. Goldwasser, S. Micali, Probabilistic encryp- tion and how to play mental poker keeping secret all partial information, in Proceedings of 14th Symposium on Theory of Computing, 1982, pp.365377

[59] M. Blum, S. Goldwasser, ”An Efficient Prob- abilistic Public Key Encryption Scheme which Hides All Partial Information”, Proceedings of Advances in Cryptology - CRYPTO ’84, pp. 289 299, Springer Verlag, 1985.

[60] Osman Ugus and al. Performance of Addi- tive Homomorphic EC-ElGamal Encryption for TinyPEDS, 6th Fachgesprch Drahtlose Sensornetze, pp. 55–58, July 2007.

APPENDIX A. IMPLEMENTATION

A1. E-VOTING

```
package evoting;

import java.io.BufferedWriter;
import java.math.*;
import java.util.*;
import java.util.Scanner;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class Evoting {

    public static final String ANSI_RESET = "\u001B[0m";
    public static final String ANSI_RED = "\u001B[31m";
    public static final String ANSI_GREEN = "\u001B[32m";

    /**
     * p and q are two large primes.
     * lambda = lcm(p-1, q-1) = (p-1)*(q-1)/gcd(p-1, q-1).
     */
    private BigInteger p,  q,  lambda;
    /**
     * n = p*q, where p and q are two large primes.
     */
    public BigInteger n;
    /**
     * nsquare = n*n
     */
    public BigInteger nsquare;
    /**
     * a random integer in Z*_{n^2} where gcd (L(g^lambda
mod n^2), n) = 1.
     */
    private BigInteger g;
    /**
     * number of bits of modulus
     */
```

```
    private int bitLength;


    public Evoting(int bitLengthVal, int certainty) {
        KeyGeneration(bitLengthVal, certainty);
    }

    public Evoting() {
        KeyGeneration(512, 64);
    }
    public void KeyGeneration(int bitLengthVal, int
certainty) {
        bitLength = bitLengthVal;
        /*Constructs two randomly generated positive
BigIntegers that are probably prime, with the specified
bitLength and certainty.*/
   /*    p = new BigInteger("7");
        q = new BigInteger("11");
     */
        /*p = new BigInteger(bitLength / 2, certainty, new
Random());
        q = new BigInteger(bitLength / 2, certainty, new
Random());
        */

        p = new BigInteger("117151688919191892 53");
//589074029825796408442014453970943300899511859929133352199601856682267598780 13
        q = new BigInteger("97392376180600168 49");
//790544485606218242211924475033370246961294333401400933696359644102851692490 41

        n = p.multiply(q);
        nsquare = n.multiply(n);

        g = new BigInteger("2");
        lambda =
p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE)).divide(

p.subtract(BigInteger.ONE).gcd(q.subtract(BigInteger.ONE)));
        /* check whether g is good.*/
        if (g.modPow(lambda,
nsquare).subtract(BigInteger.ONE).divide(n).gcd(n).intValue(
) != 1) {
```

```
            System.out.println("g is not good. Choose g
again.");
            System.exit(1);
        }
    }

    /**
     * Encrypts plaintext m. ciphertext c = g^m * r^n mod
n^2. This function explicitly requires random input r to
help with encryption.
     * @param m plaintext as a BigInteger
     * @param r random plaintext to help with encryption
     * @return ciphertext as a BigInteger
     */
    public BigInteger Encryption(BigInteger m, BigInteger r)
{
        return g.modPow(m, nsquare).multiply(r.modPow(n,
nsquare)).mod(nsquare);
    }
    /**
     * Encrypts plaintext m. ciphertext c = g^m * r^n mod
n^2. This function automatically generates random input r
(to help with encryption).
     * @param m plaintext as a BigInteger
     * @return ciphertext as a BigInteger
     */
    public BigInteger Encryption(BigInteger m) {
        //BigInteger r = new BigInteger("13");
        BigInteger r = new BigInteger(bitLength, new
Random());

        //System.out.println("R=  " + r);
        return g.modPow(m, nsquare).multiply(r.modPow(n,
nsquare)).mod(nsquare);

    }

    /**
     * Decrypts ciphertext c. plaintext m = L(c^lambda mod
n^2) * u mod n, where u = (L(g^lambda mod n^2))^(-1) mod n.
     * @param c ciphertext as a BigInteger
     * @return plaintext as a BigInteger
     */
    public BigInteger Decryption(BigInteger c) {
        BigInteger u = g.modPow(lambda,
nsquare).subtract(BigInteger.ONE).divide(n).modInverse(n);
```

```
        return c.modPow(lambda,
nsquare).subtract(BigInteger.ONE).divide(n).multiply(u).mod(
n);
    }

    /**
     * main function
     * @param str intput string
     */
    public static void main(String[] str) {
        /* instantiating an object of Evoting cryptosystem*/
        Evoting paillier = new Evoting();
        /* instantiating two plaintext msgs*/
        Scanner sc= new Scanner(System.in);
        BigInteger[] em = new BigInteger[20];
                ; //array of bigintger

         BigInteger m2 ;
         //System.out.println(paillier.p);
        //System.out.println(paillier.q);

System.out.println(ANSI_RED+"*************************
CRYPTOGRAPHIC ELECTRONIC VOTING SYSTEM
*************************");
        System.out.println("");
        System.out.println();
        System.out.println(ANSI_RED+"\t\tINSTRUCTIONS\n");
        System.out.println(ANSI_RED+"1. Choose No. of
candidates and No. of Voters  for election");
        System.out.println(ANSI_RED+"2. Default value for
Candidates is 2 and candidates cannot be more than 8");
        System.out.println(ANSI_RED+"4. Enter candidate
number for whom you want to vote");
        System.out.println(ANSI_RED+"5. Do not disclose your
vote to anybody");
        System.out.println(ANSI_RED+"6. Contact the
admininstrator for any query"+ANSI_RESET);
        System.out.println("");
        System.out.println("");
        System.out.println("");
        System.out.println();
        System.out.println();

        System.out.print(ANSI_GREEN+"Select No. of
candidates standing for election:  "+ANSI_RESET);
        int nocand=sc.nextInt();
```

```
        if(nocand >8 || nocand < 2)
        {
            nocand=2;
            System.out.println(ANSI_RED+"Incompatible value
entered , default value has been chosen. "+ANSI_RESET);
        }

        System.out.print(ANSI_GREEN+"\n\nSelect No. of
Voters for election:  "+ANSI_RESET);
        int voters=sc.nextInt();

        System.out.println("\n\n");
        System.out.println("");
        //string path = ;
        try{
            File f = new File("Encrypted Votes.txt");
            f.createNewFile();
        }
        catch(Exception e)
        {}

        for(int i=1;i<=voters;i++)
        {
        System.out.println("\n\n----------------------------
------------------------------------------------------------
-----------------------------------");
        System.out.println("voter "+ i+ ":");
        System.out.print("Choose candidate (1-"+ nocand+") :
");
        String a=sc.next();
                // c=10^(a-1);

        //BigInteger m1 = new BigInteger(a);
        int m1 = Integer.parseInt(a);

        if(m1<1 || m1>nocand)
        {
            System.out.println(ANSI_RED+"\n\nWRONG VALUE
ENTERED, KINDLY CHECK THE INPUT AGAIN\n\n"+ANSI_RESET);
            i-=1;
            continue;
        }

        m2 = new BigInteger ("10");

        m1=m1-1;
```

```
            System.out.print(m1+"  ");
            m2=m2.pow(2*m1);
            System.out.println(m2);
            em[i] = paillier.Encryption(m2);
            try{
                Thread.sleep(1200);
            }
            catch(Exception e){}
            System.out.println(ANSI_GREEN+em[i]+ANSI_RESET);
            System.out.println();
            System.out.println("");

            try(FileWriter fw = new FileWriter("Encrypted
Votes.txt", true);
           BufferedWriter bw = new BufferedWriter(fw);
           PrintWriter out = new PrintWriter(bw))
        {
          out.println(em[i]);
          out.println();
        //more code
        }      catch (IOException e) {
        //exception handling left as an exercise for the reader
          }

            }
            System.out.println("----------------------------------
--------------------------------------------------------------
--------------------------------");

            /* test homomorphic properties -> D(E(m1)*E(m2) mod
n^2) = (m1 + m2) mod n */
            System.out.println();
            try{
                Thread.sleep(500);
            }
            catch(Exception e){}
            System.out.println("PROCEDURE FOR VOTING HAS
FINISHED\nTHANKS FOR VOTING!!\n");
         /*   System.out.println("ADDING ALL VOTES AND PREPARING
RESULT....");
            System.out.println("\n");
            System.out.println("");
            System.out.println("\n");
            BigInteger product_em1em2 = new BigInteger("1");
            for(int j=1;j<=voters;j++)
```

```
                { product_em1em2 =
product_em1em2.multiply(em[j]);
             }
                  product_em1em2 =
product_em1em2.mod(paillier.nsquare);
        //BigInteger sum_m1m2 = m1.add(m2).mod(paillier.n);

        System.out.println("Encrypted Sum of the votes is:
\n");
        System.out.println(ANSI_RED + product_em1em2 +
ANSI_RESET);
        System.out.println("\n");

        System.out.println("decrypted sum: " +
paillier.Decryption(product_em1em2).toString());
        String x1 =
paillier.Decryption(product_em1em2).toString();
        BigInteger x2,x3;
        x2= paillier.Decryption(product_em1em2);
         int max,maxp,mm;
         int[] d= new int[x1.length()];
        max=maxp=mm=0;

         //System.out.println(x1.length());
        System.out.println("\n\nRESULT\n");
        BigInteger m4 = new BigInteger ("100");
        BigInteger m5 = new BigInteger("0");
         for(int k=1;k <= x1.length();k++)
         {
            x3=x2.remainder(m4);
            maxp= x3.intValue();
            d[k-1]=maxp;
                        if(max < maxp )
            { max=maxp;
              mm=k;
            }
                       //System.out.println(x3);
            //x3=x2%10;
            x2=x2.divide(m4);

                       //  System.out.println(x2);

            System.out.println("No. of votes for candidate
"+ k +" :-  "+ x3);

            if(x2.equals(m5))
```

```
                break;
            }

            System.out.println();
            int c=0;
            for(int k=0;k<x1.length();k++)
            {
                if(d[k]==max)
                c+=1;
            }
            if(c==1) {
                System.out.println("Winner of election is
candidate no: "+ mm +" having a total no. of votes:  "+
max);
            }
            else {
                System.out.println(ANSI_RED+"There's a draw
between candidates, hence no clear winner!"+ANSI_RESET);
            }*/
    }
}
```

A2. VOTECOUNT & DECRYPTION

```
package evoting;

import java.io.BufferedReader;
import java.math.*;
import java.util.*;
import java.util.Scanner;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.logging.Level;
import java.util.logging.Logger;


public class Votecount {

    public static final String ANSI_RESET = "\u001B[0m";
    public static final String ANSI_RED = "\u001B[31m";
    public static final String ANSI_GREEN = "\u001B[32m";
```

```
    /**
     * p and q are two large primes.
     * lambda = lcm(p-1, q-1) = (p-1)*(q-1)/gcd(p-1, q-1).
     */
    private BigInteger p,  q,  lambda;
    /**
     * n = p*q, where p and q are two large primes.
     */
    public BigInteger n;
    /**
     * nsquare = n*n
     */
    public BigInteger nsquare;
    /**
     * a random integer in Z*_{n^2} where gcd (L(g^lambda
mod n^2), n) = 1.
     */
    private BigInteger g;
    /**
     * number of bits of modulus
     */
    private int bitLength;


    public Votecount(int bitLengthVal, int certainty) {
        KeyGeneration(bitLengthVal, certainty);
    }


    public Votecount() {
        KeyGeneration(512, 64);
    }
    public void KeyGeneration(int bitLengthVal, int
certainty) {
        bitLength = bitLengthVal;
        /*Constructs two randomly generated positive
BigIntegers that are probably prime, with the specified
bitLength and certainty.*/
   /*    p = new BigInteger("7");
        q = new BigInteger("11");
     */
        /*p = new BigInteger(bitLength / 2, certainty, new
Random());
        q = new BigInteger(bitLength / 2, certainty, new
Random());
```

```
        */

        p = new BigInteger("117151688919191892 53");  //
589074029825796408442014453970943300899511859929133352199601
85668226759878013
        q = new BigInteger("97392376180600168 49");   //
790544485606218242211924475033370246961294333401400933696359
64410285169249041

        n = p.multiply(q);
        nsquare = n.multiply(n);

        g = new BigInteger("2");
        lambda =
p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ON
E)).divide(

p.subtract(BigInteger.ONE).gcd(q.subtract(BigInteger.ONE)));
        /* check whether g is good.*/
        if (g.modPow(lambda,
nsquare).subtract(BigInteger.ONE).divide(n).gcd(n).intValue(
) != 1) {
            System.out.println("g is not good. Choose g
again.");
            System.exit(1);
        }
    }

    /**
     * Encrypts plaintext m. ciphertext c = g^m * r^n mod
n^2. This function explicitly requires random input r to
help with encryption.
     * @param m plaintext as a BigInteger
     * @param r random plaintext to help with encryption
     * @return ciphertext as a BigInteger
     */
    public BigInteger Encryption(BigInteger m, BigInteger r)
{
        return g.modPow(m, nsquare).multiply(r.modPow(n,
nsquare)).mod(nsquare);
    }

    /**
     * Encrypts plaintext m. ciphertext c = g^m * r^n mod
n^2. This function automatically generates random input r
(to help with encryption).
```

```
     * @param m plaintext as a BigInteger
     * @return ciphertext as a BigInteger
     */
    public BigInteger Encryption(BigInteger m) {
        //BigInteger r = new BigInteger("13");
        BigInteger r = new BigInteger(bitLength, new
Random());

        //System.out.println("R=  " + r);
        return g.modPow(m, nsquare).multiply(r.modPow(n,
nsquare)).mod(nsquare);

    }

    /**
     * Decrypts ciphertext c. plaintext m = L(c^lambda mod
n^2) * u mod n, where u = (L(g^lambda mod n^2))^(-1) mod n.
     * @param c ciphertext as a BigInteger
     * @return plaintext as a BigInteger
     */
    public BigInteger Decryption(BigInteger c) {
        BigInteger u = g.modPow(lambda,
nsquare).subtract(BigInteger.ONE).divide(n).modInverse(n);
        return c.modPow(lambda,
nsquare).subtract(BigInteger.ONE).divide(n).multiply(u).mod(
n);
    }

    /**
     * main function
     * @param str intput string
     */
    public static void main(String[] str) {
        /* instantiating an object of Evoting cryptosystem*/
        Votecount paillier = new Votecount();
        /* instantiating two plaintext msgs*/
        Scanner sc= new Scanner(System.in);
        BigInteger[] em = new BigInteger[20];
                ; //array of bigintger

         BigInteger m2 ;

       //System.out.println(paillier.p);
        //System.out.println(paillier.q);
```

```
System.out.println(ANSI_RED+"************************
CRYPTOGRAPHIC ELECTRONIC VOTING SYSTEM
************************");
        System.out.println("");
        System.out.println();
        System.out.println(ANSI_RED+"\t\t\tRESULT OF VOTING
PROCESS\n"+ANSI_RESET);
        System.out.println("THE VOTES THAT WERE
CASTED:\n\n");
       /*
        System.out.print(ANSI_GREEN+"Select No. of
candidates standing for election:  "+ANSI_RESET);
        int nocand=sc.nextInt();
        if(nocand >8 || nocand < 2)
        {
            nocand=2;
            System.out.println(ANSI_RED+"Incompatible value
entered , default value has been chosen. "+ANSI_RESET);
        }

        System.out.print(ANSI_GREEN+"\n\nSelect No. of
Voters for election:  "+ANSI_RESET);
        int voters=sc.nextInt();

        System.out.println("\n\n");
        System.out.println("");

        //string path = ;
        try{
            File f = new File("Encrypted Votes.txt");
            f.createNewFile();
        }
        catch(Exception e)
        {}



        for(int i=1;i<=voters;i++)
        {
        System.out.println("\n\n----------------------------
------------------------------------------------------------
-----------------------------------");
        System.out.println("voter "+ i+ ":");
        System.out.print("Choose candidate (1-"+ nocand+") :
");
```

```
        String a=sc.next();
                // c=10^(a-1);

        //BigInteger m1 = new BigInteger(a);
        int m1 = Integer.parseInt(a);

        if(m1<1 || m1>nocand)
        {
            System.out.println(ANSI_RED+"\n\nWRONG VALUE
ENTERED, KINDLY CHECK THE INPUT AGAIN\n\n"+ANSI_RESET);
            i-=1;
            continue;
        }

        m2 = new BigInteger ("10");

        m1=m1-1;
        System.out.print(m1+"  ");
        m2=m2.pow(2*m1);
        System.out.println(m2);
        em[i] = paillier.Encryption(m2);
        try{
            Thread.sleep(1200);
        }
        catch(Exception e){}
        System.out.println(ANSI_GREEN+em[i]+ANSI_RESET);
        System.out.println();
        System.out.println("");

        try(FileWriter fw = new FileWriter("Encrypted
Votes.txt", true);
       BufferedWriter bw = new BufferedWriter(fw);
       PrintWriter out = new PrintWriter(bw))
    {
      out.println(em[i]);
      out.println();
    //more code
    }      catch (IOException e) {
    //exception handling left as an exercise for the reader
      }




        }
```

```
        System.out.println("-------------------------------
-----------------------------------------------------------
-------------------------------");

        /* test homomorphic properties -> D(E(m1)*E(m2) mod
n^2) = (m1 + m2) mod n */
       /* System.out.println();
        try{
            Thread.sleep(500);
        }
        catch(Exception e){}
        System.out.println("PROCEDURE FOR VOTING HAS
FINISHED\nTHANKS FOR VOTING!!\n");
        /*System.out.println("ADDING ALL VOTES AND PREPARING
RESULT....");
        System.out.println("\n");
        System.out.println("");
        System.out.println("\n");
        */


        FileInputStream fis = null;
        BufferedReader reader = null;
       int x=0;
       BigInteger temp;
        try {
            fis = new
FileInputStream("C:\\Users\\Yogi\\Documents\\NetBeansProject
s\\EVoting\\Encrypted Votes.txt");
            reader = new BufferedReader(new
InputStreamReader(fis));

          // System.out.println("Reading File line by line
using BufferedReader");

            String line = reader.readLine();
            while(line != null){


               temp = new BigInteger(line);
               em[x] = temp;

System.out.println(ANSI_GREEN+em[x]+ANSI_RESET+"\n");
                //System.out.println(line+"\n");
                line = reader.readLine();
                line = reader.readLine();
```

```
                x++;
            }
            //System.out.println("x= "+x);
        } catch (Exception e){}
        finally {
            try {
                reader.close();
                fis.close();
            } catch (Exception ex) {}

        }

        System.out.println("--------------------------------
----------------------------------------------------------
----------------------------------------------------------
-----------");
        BigInteger product_em1em2 = new BigInteger("1");
        for(int j=0;j<x;j++)
                { product_em1em2 =
product_em1em2.multiply(em[j]);
             }

         product_em1em2 =
product_em1em2.mod(paillier.nsquare);
        //BigInteger sum_m1m2 = m1.add(m2).mod(paillier.n);

        sc.nextLine();
        System.out.println("Encrypted Sum of the votes is:
\n");
        System.out.println(ANSI_RED + product_em1em2 +
ANSI_RESET);
        System.out.println("\n");


       sc.nextLine();
        System.out.println("decrypted sum: 0" +
paillier.Decryption(product_em1em2).toString());
        String x1 =
paillier.Decryption(product_em1em2).toString();
        BigInteger x2,x3;
        x2= paillier.Decryption(product_em1em2);
         int max,maxp,mm;
         int[] d= new int[x1.length()];
        max=maxp=mm=0;

         //System.out.println(x1.length());
```

```
            System.out.println("\n\nRESULT\n");
            BigInteger m4 = new BigInteger ("100");
            BigInteger m5 = new BigInteger("0");
             for(int k=1;k <= x1.length();k++)
             {
                x3=x2.remainder(m4);
                maxp= x3.intValue();
                d[k-1]=maxp;

                if(max < maxp )
                { max=maxp;
                  mm=k;
                }

                //System.out.println(x3);
                //x3=x2%10;
                x2=x2.divide(m4);
                //  System.out.println(x2);

                System.out.println("No. of votes for candidate
"+ k +" :-  "+ x3);

                if(x2.equals(m5))
                    break;
            }

            System.out.println();
            int c=0;
            for(int k=0;k<x1.length();k++)
            {
                if(d[k]==max)
                c+=1;

            }
            if(c==1) {
                System.out.println("Winner of election is
candidate no: "+ mm +" having a total no. of votes:  "+
max);
            }
            else {
                System.out.println(ANSI_RED+"There's a draw
between candidates, hence no clear winner!"+ANSI_RESET);
            }
    }
}
```

APPENDIX B. SCREENSHOTS

Below are some screenshots for the implementation:

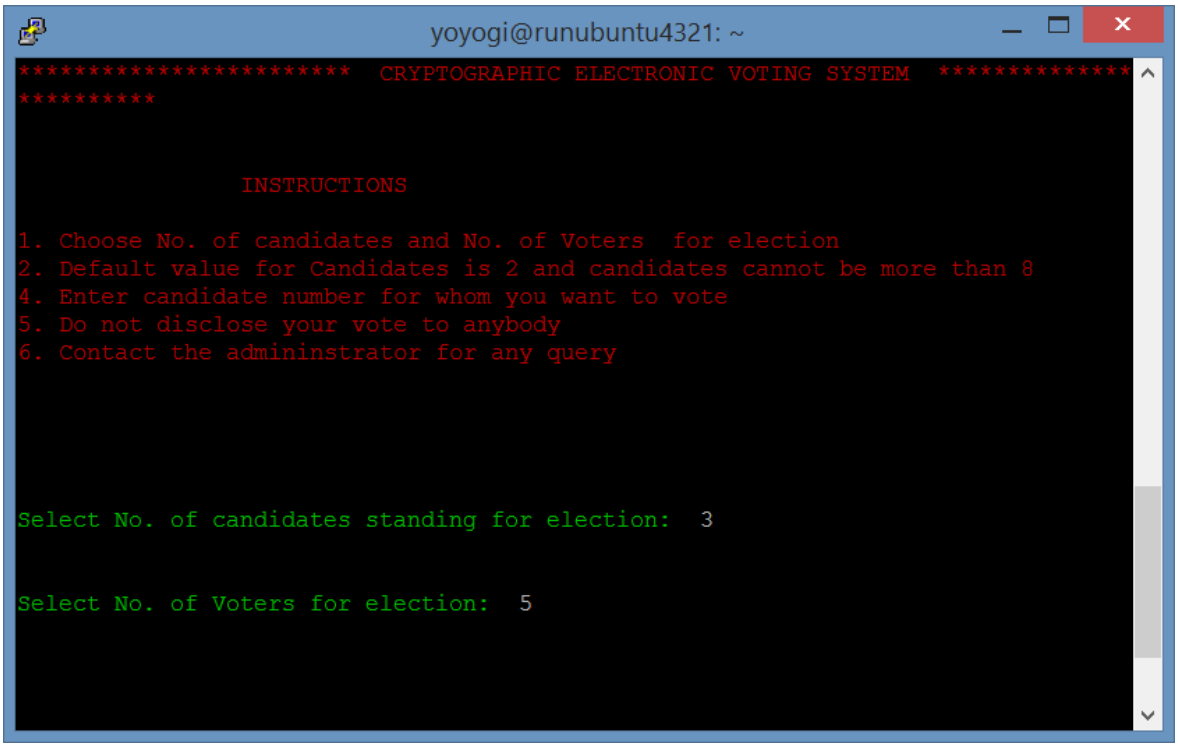


Figure 8 Instruction Screen

Figure 9 Voting Screen 1

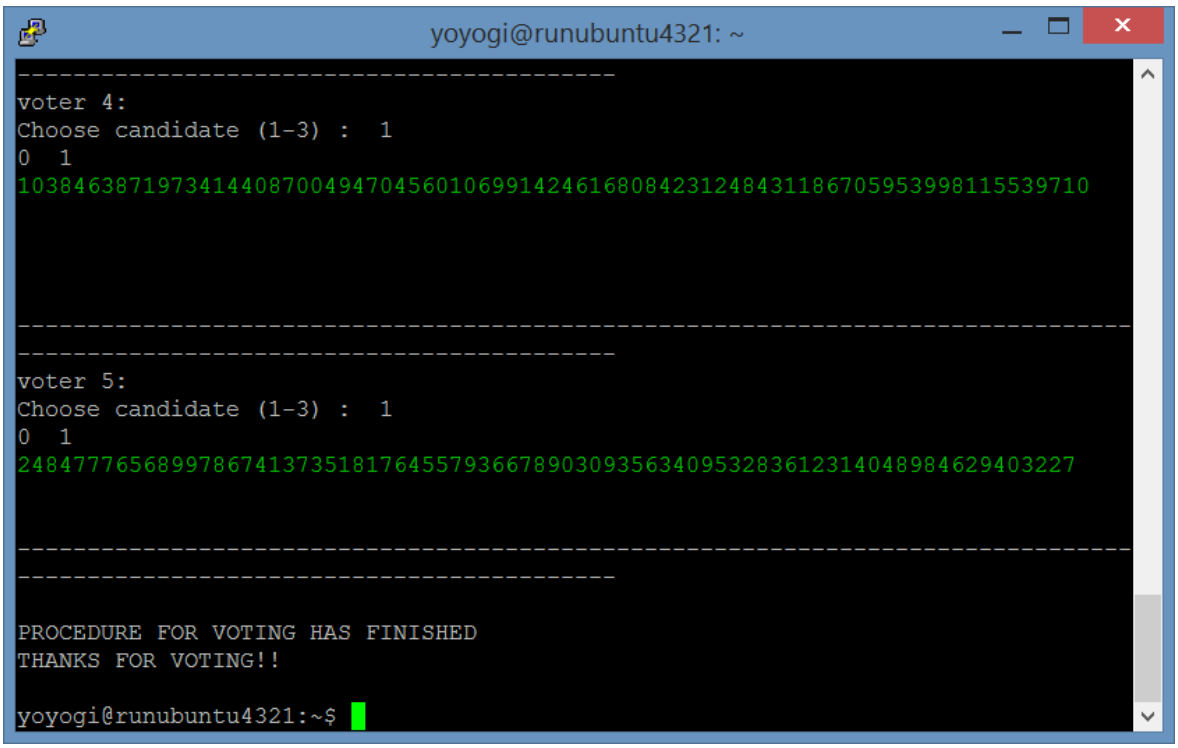


Figure 10 Voting Screen 2

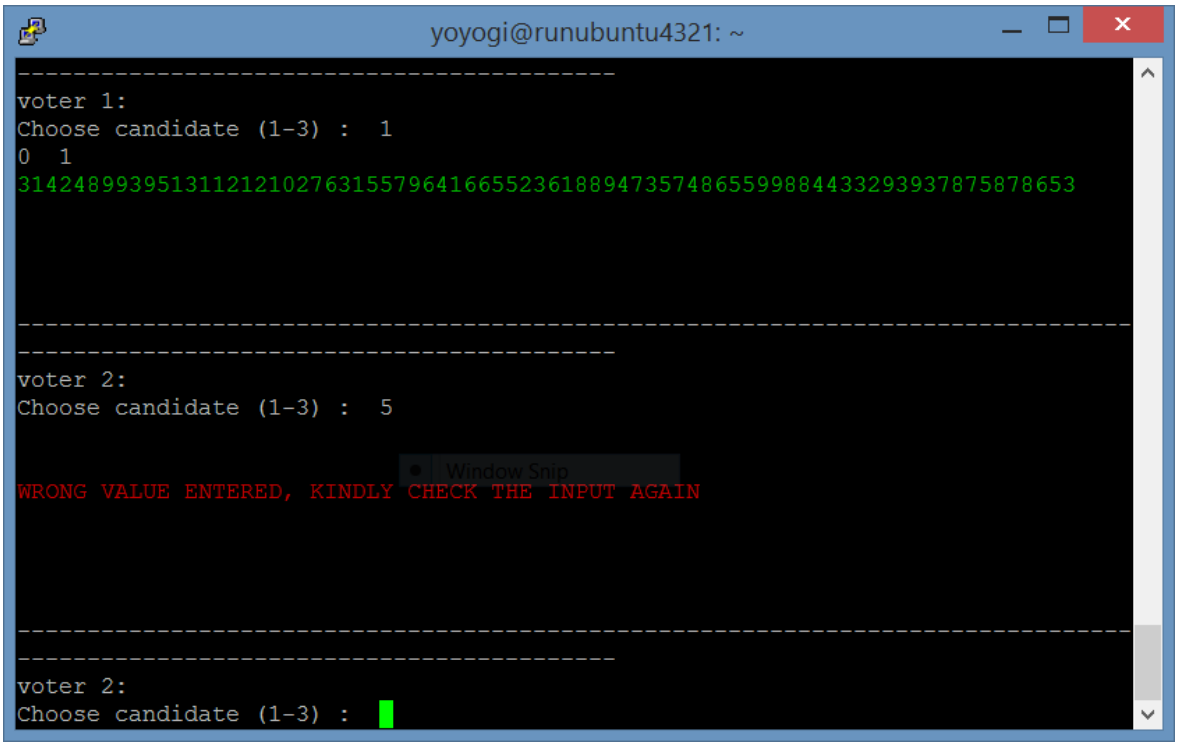


Figure 11 Error Valued Input

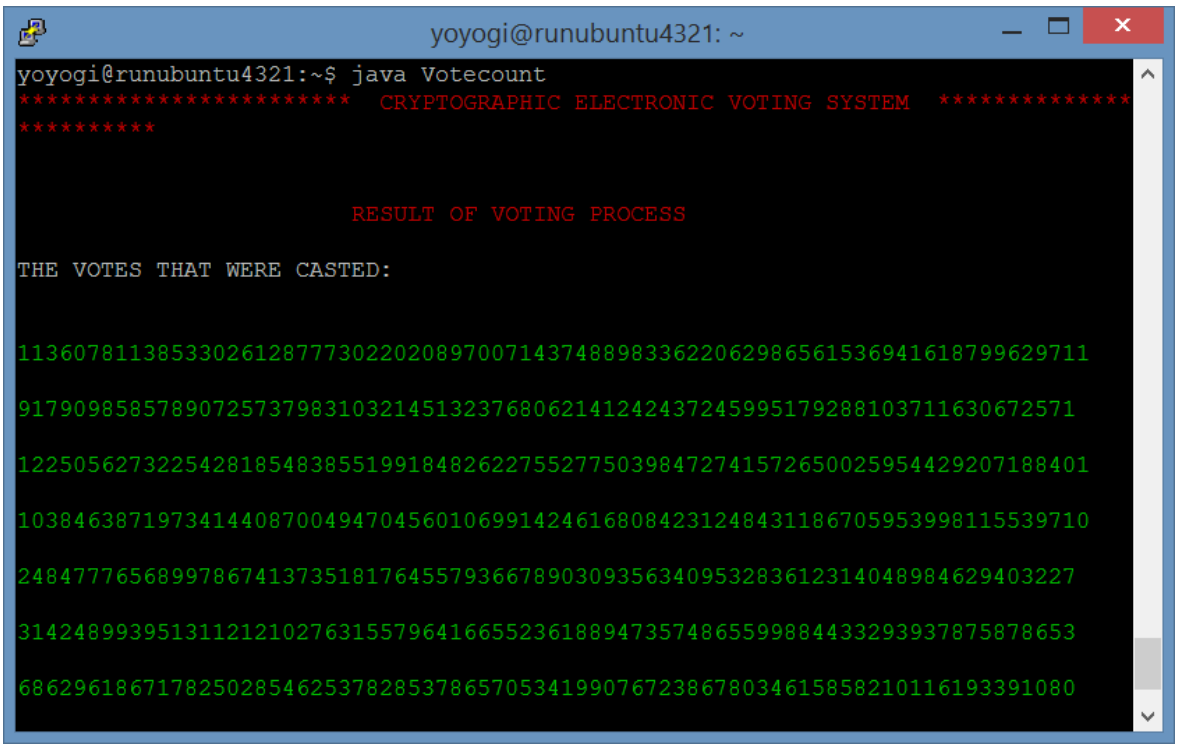


Figure 12 Encrypted list of Casted Votes

Figure 13 Results calculated and decrypted initially – Round 1

Figure 14 Results after voting for Round - 2

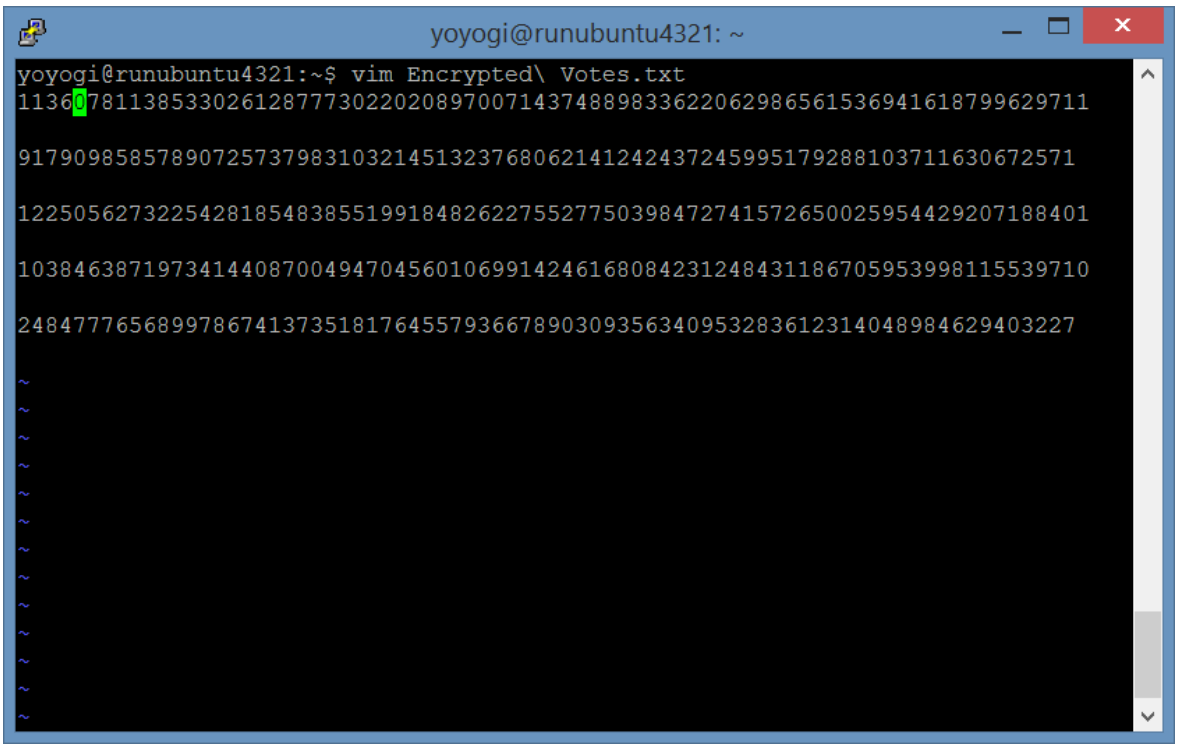


Figure 15 The file containing votes after Round 1

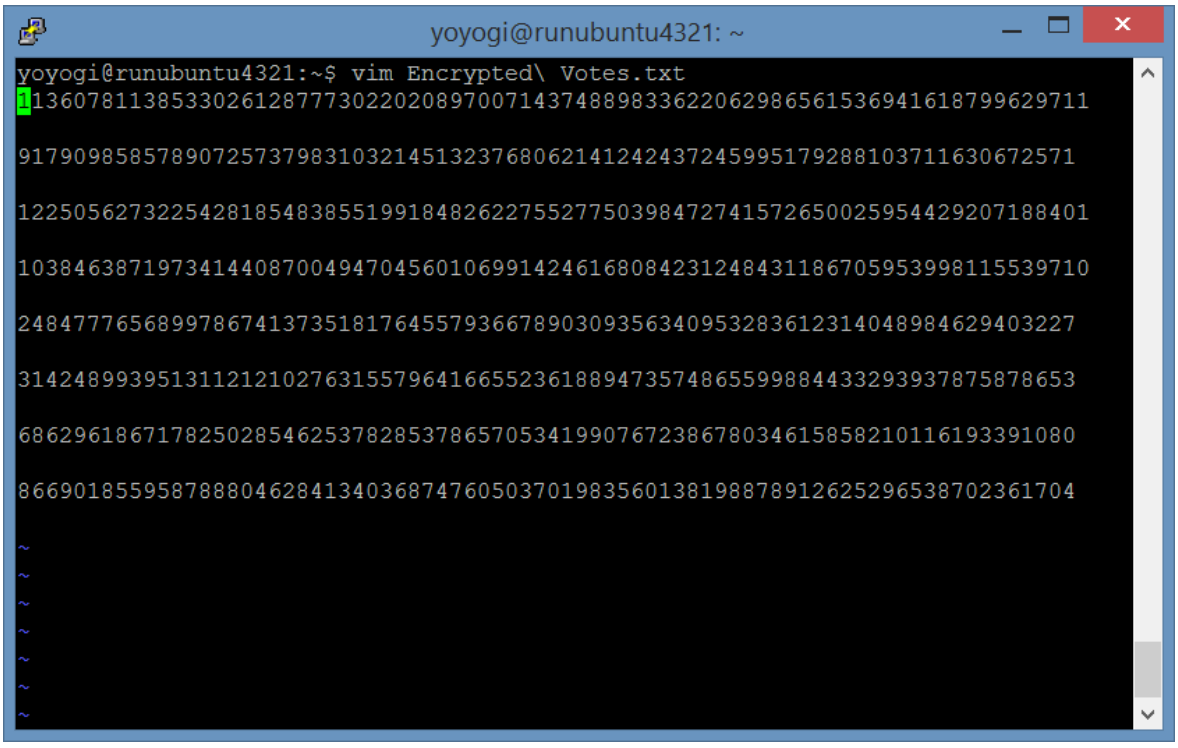


Figure 16 The file containing votes after round 2

APPENDIX C. Time Analysis of encryption schemes w.r.t. different systems.

For tables results are based on database hosted on remote machine. Graphs shows the comparison of overhead with encrypted data over plain data for addition and multiplication operations for four algorithms.

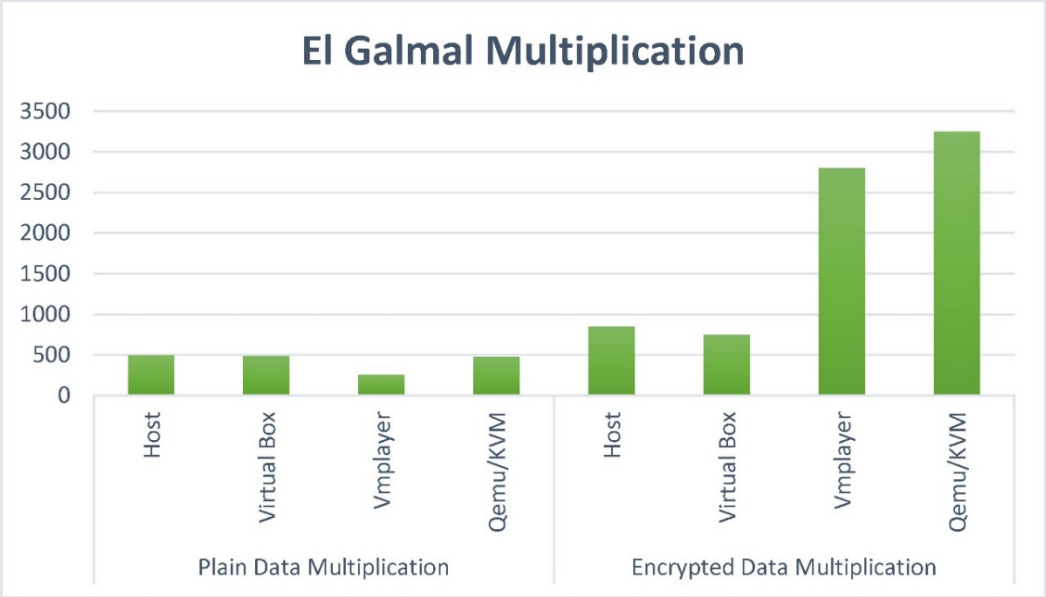


Figure 17 Comparison of overhead with encrypted data over plain data for multiplication operation for Elgamal Algorithm (y- axis in nanoseconds)

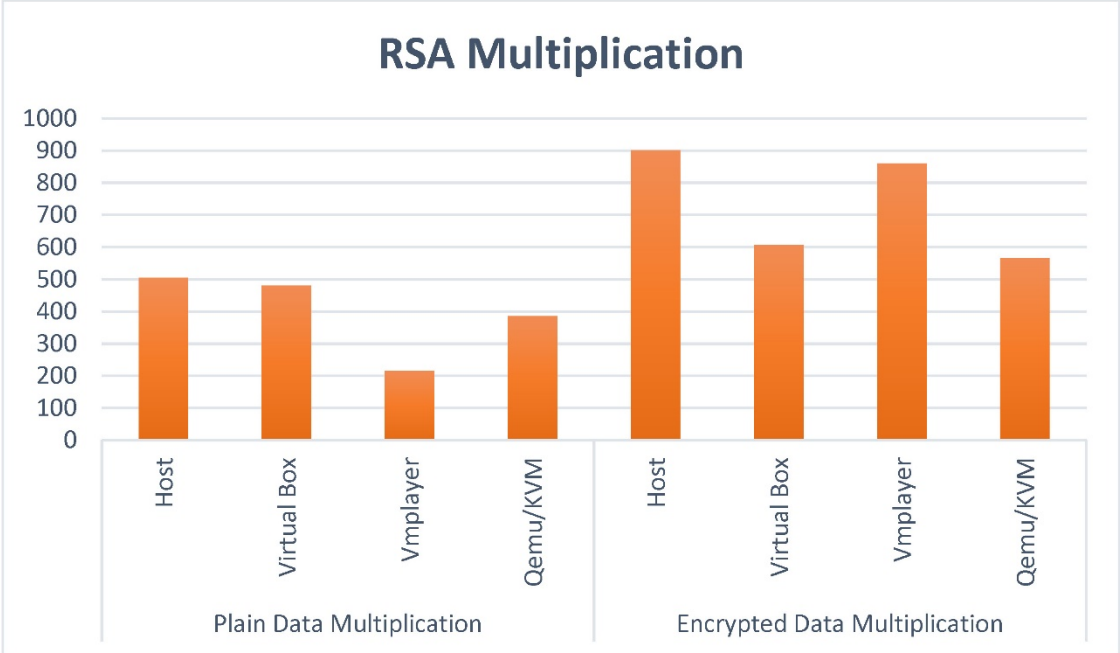


Figure 18 Comparison of overhead with encrypted data over plain data for multiplication operation for RSA

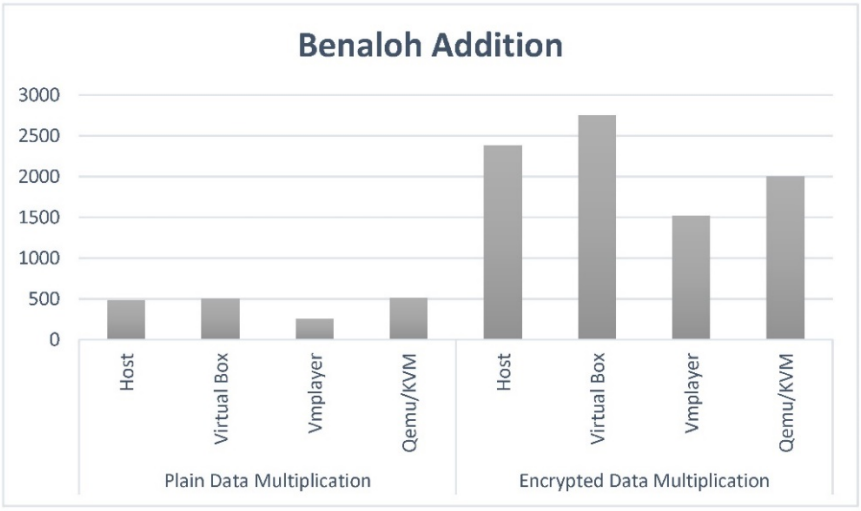


Figure 19 Comparison of overhead with encrypted data over plain data for addition operation for Benaloh Algorithm (y- axis in nanoseconds)

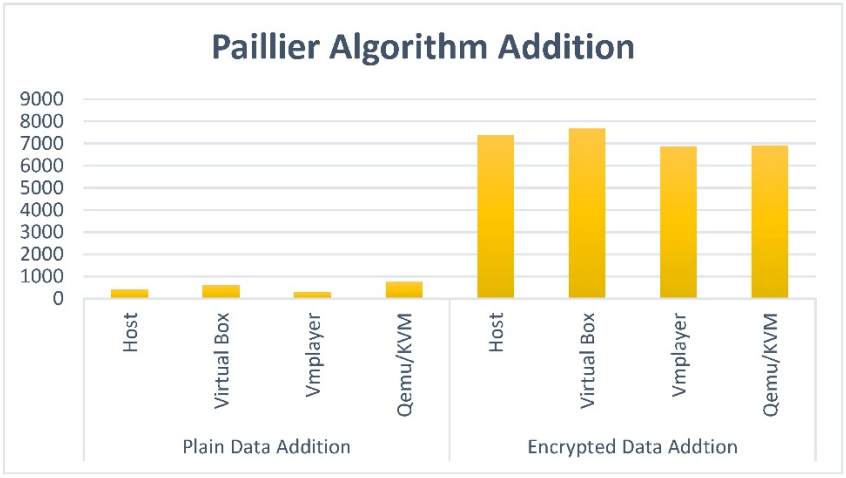


Figure 20 Comparison of overhead with encrypted data over plain data for addition operation for Paillier Algorithm (y- axis in nanoseconds)

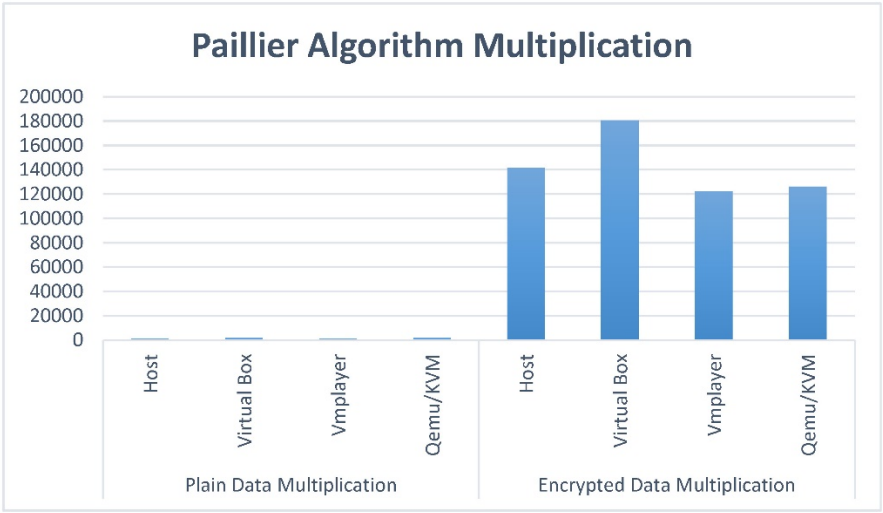


Figure 21 Comparison of overhead with encrypted data over plain data for multiplication operation for Paillier Algorithm (y- axis in nanoseconds)

APPENDIX D. SOFTWARE AND HARDWARE REQUIREMENT

D1. Software Libraries

D1.1 HELib

HElib is a software library that implements homomorphic encryption (HE). Currently available is an implementation of the Brakerski-Gentry-Vaikuntanathan (BGV) scheme, along with many optimizations to make homomorphic evaluation runs faster, focusing mostly on effective use of the Smart-Vercauteren ciphertext packing techniques and the Gentry-Halevi-Smart optimizations. [17] [18]

At its present state, this library is mostly meant for researchers working on HE and its uses. Also currently it is fairly low-level, and is best thought of as "assembly language for HE". That is, it provides low-level routines (set, add, multiply, shift, etc.), with as much access to optimizations as we can give. Hopefully in time we will be able to provide higher-level routines.

This library is written in C++ and uses the NTL mathematical library, over GMP. It is distributed under the terms of the GNU General Public License (GPL).

D1.2 Extreme Optimization Numerical Libraries for .NET

General features of the Extreme Optimization Numerical Libraries for .NET:

- Easy to use even for the mathematically not-so-inclined
- Great performance through optimized implementation of the best algorithms.
- Powerful enough to satisfy the most demanding power user.
- Intuitive object model. The objects in the Extreme Optimization Numerical Libraries for .NET and the relationships between them match our every-day concepts.
- Cross-platform. Works out-of-the-box on 32 and 64 bit platforms, .NET versions 1.1, 2.0, 3.0, 3.5.

The **Extreme Optimization Numerical Libraries for .NET** is a solid foundation for your numerical computing needs on the .NET platform. It implements a broad set of algorithms,

covering a wide range of numerical techniques, including: linear algebra, numerical integration and differentiation, solving equations, complex numbers.

The classes in the Mathematics Library of the **Extreme Optimization Numerical Library for .NET** and the relationships between them match our every-day concepts.

The Mathematics Library contains classes for a wide range of mathematical techniques. Vector and matrix classes are described in the Vector and Matrix Library User's Guide.

The classes are organized in a consistent namespace hierarchy as follows:

Namespace	Description
Extreme.Mathematics	Contains fundamental classes and base classes that define commonly-used mathematical data types, exception types, and delegates.
Extreme.Mathematics.Calculus	Contains classes for the numerical integration and differentiation of functions.
Extreme.Mathematics.Curves	Contains classes for working with points, lines and curves, including polynomials and Chebyshev approximations. Techniques include curve fitting and interpolation.
Extreme.Mathematics.Generic	Contains classes for working with generic arithmetic, including generic linear algebra.
Extreme.Mathematics.EquationSolvers	Contains classes that implement various root finding algorithms.
Extreme.Mathematics.LinearAlgebra	Contains classes for working with vectors and matrices and matrix decompositions, and for solving systems of simultaneous linear equations and least squares problems.
Extreme.Mathematics.Optimization	Contains classes and methods for optimization of functions in one or more

	dimensions, including linear programming.
Extreme.Mathematics.SignalProcessing	Contains classes and methods for one and two-dimensional discrete Fourier transforms and related functions.

D1.3 CryptOPP++ Library

. Currently the library contains the following algorithms: [26] [27]

ALGORITHM TYPE	NAME
authenticated encryption schemes	GCM, CCM, EAX
high speed stream ciphers	Panama, Sosemanuk, Salsa20, XSalsa20
AES and AES candidates	AES (Rijndael), RC6, MARS, Twofish, Serpent, CAST-256
other block ciphers	IDEA, Triple-DES (DES-EDE2 and DES-EDE3), Camellia, SEED, RC5, Blowfish, TEA, XTEA, Skipjack, SHACAL-2
block cipher modes of operation	ECB, CBC, CBC ciphertext stealing (CTS), CFB, OFB, counter mode (CTR)
message authentication codes	VMAC, HMAC, GMAC (GCM), CMAC, CBC-MAC, DMAC, Two-Track-MAC
hash functions	SHA-1, SHA-2 (SHA-224, SHA-256, SHA-384, and SHA-512), SHA-3, Tiger, WHIRLPOOL, RIPEMD-128, RIPEMD-256, RIPEMD-160, RIPEMD-320
public-key cryptography	RSA, DSA, ElGamal, Nyberg-Rueppel (NR), Rabin-Williams (RW), LUC, LUCELG, DLIES (variants of DHAES), ESIGN
padding schemes for public-key systems	PKCS#1 v2.0, OAEP, PSS, PSSR, IEEE P1363 EMSA2 and EMSA5
key agreement schemes	Diffie-Hellman (DH), Unified Diffie-Hellman (DH2), Menezes-Qu-Vanstone (MQV), LUCDIF, XTR-DH

elliptic curve cryptography	ECDSA, ECNR, ECIES, ECDH, ECMQV
insecure or obsolescent algorithms retained for backwards compatibility and historical value	MD2, MD4, MD5, Panama Hash, DES, ARC4, SEAL 3.0, WAKE-OFB, DESX (DES-XEX3), RC2, SAFER, 3-WAY, GOST, SHARK, CAST-128, Square

APPENDIX E. CERTIFICATION

This certificate was awarded at G.D. Goenka University for the Inter-College Project Presentation. We secured the first position.

G D GOENKA UNIVERSITY
FROM FOUNDATION TO FLIGHT

SCHOOL OF ENGINEERING

SPEED 2016
(An Inter University Engineering Project Competition)

STUDENTS' PROJECT EXHIBITION ON ENGINEERING DESIGN
(An Inter-University Engineering Projects Competition)
19TH FEBRUARY, 2016
Organized by: School of Engineering, G D Goenka University in association with

and Skills Gap Solutions

CERTIFICATE
of BEST PROJECT AWARD

Presented to Mr./ Ms. Yogeesh Kapila *of*
Bharti Vidyapeeth *for*
project ID SP16A157 *entitled* Homomorphic encryption on strings
and integers in a cloud based environment.
in SPEED – 2016 held at G D Goenka University, Gurgaon on February 19, 2016.

Dr. Sandeep Nagar
Chairperson, SPEED-2016

Prof. (Dr.) Deependra K. Jha
Dean, School of Engineering