Performance, Scalability and Implementation issues in Homomorphic Encryption

Annual Progress Report 2016-2017
Submitted in partial fulfilment of the
requirements for the degree
Doctor Of Philosophy

By

Tikaram Sanyashi

Roll No. 144050007

Under the supervision of

Prof. Bernard Menezes

Department of Computer Science and Engineering,
Indian Institute of Technology Bombay
Mumbai 400076 (India)

Abstract

Homomorphic encryption scheme allows computation on encrypted domain. It allows one to compute arbitrary functions over encrypted data without decryption key -i.e., given encryption $\mathrm{E}(m_1), \cdots, \mathrm{E}(m_t)$ of $m_1, \cdots, m_t$ one can efficiently compute a compact ciphertext that encrypts $f(m_1, \cdots, m_t)$ for any efficiently computable function f.

Homomorphic encryption has wide range of applications. For example, it enables query on a search engine,if a user submits a query in encrypted form then search engine computes encrypted answer without ever looking at the query in clear. It also got applications on the cloud computing and distributed system.

In 2009 Craig Gentry solved the Fully Homomorphic encryption in his PhD thesis using ideas from algebra, number theory and geometry of numbers, but the space and time complexity of his scheme is too high. In recent years, a no of approaches for Fully Homomorphic Encryption(FHE) has been proposed but space and time complexity precludes their use in practice.

Some of the other scheme are also there which are not fully Homomorphic named as Somewhat Homomorphic(SHE). The time and space complexity of these scheme are low so they are only used in practical scenarios.

In this report we have used Zhou-Wornell and Yu-Lai-Payor's Somewhat homomorphic encryption scheme for inventory management and forecasting. Latter on made a comparison between normally forecasted data and homomorphically forecasted data as preliminary result.

Contents

List of Figures

Chapter 1

Introduction

Homomorphic encryption is a from of encryption scheme that allows computation in encrypted domain giving an encrypted result, that when decrypted matches with the result of operation performed on plaintext. Homomorphic encryption allows complex mathematical operations to be performed in encrypted domain without compromising encryption. In mathematics, homomorphic means transformation of one data set into another while preserving relationship between elements of both sets. The term homomorphic is derived from the Greek word which means same or similar form. Data in homomorphic encryption scheme retains same structure. Therefore mathematical operations whether performed on encrypted or decrypted data will yield equivalent results.

Homomorphic encryption has great importance in cloud. If a user have data and he stores it on untrusted server in encrypted form, latter he wants to perform some operations, he can first download data in his local machine and perform operations or he can directly say to the server to perform required operations on the stored data on his behalf. In the former case the communication and computation cost will be too high so it is not preferable. The latter case brings homomorphic encryption scheme into picture. It is more efficient in terms of communication and computation as only the result of the operation is sent back to the user and it is of very small size. The computation speed of the sever is very high as compared to the clients computation speed so computation will take less time. Homomorphic Encryption Scheme can be fully described using the figure1 below

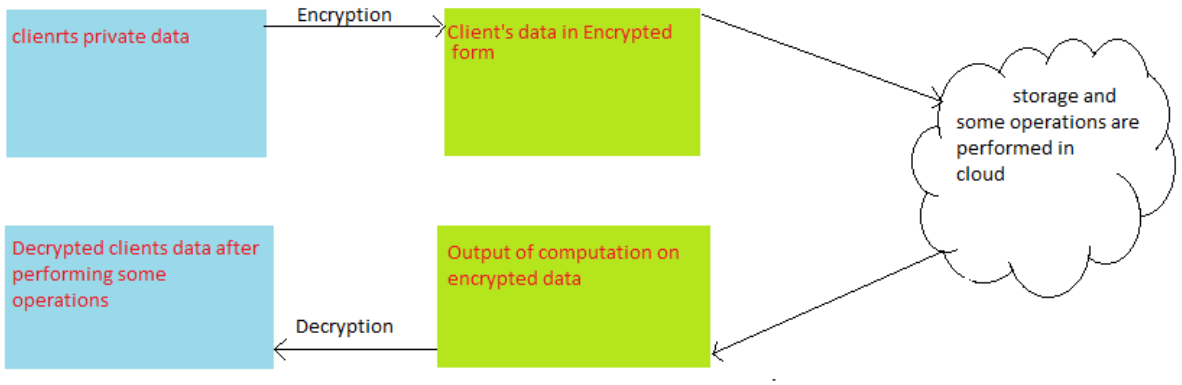

Figure 1.1: Computation with Homomorphic Encryption

Other applications of Homomorphic Encryption is querying search engine in encrypted domain without revealing to the server what search is going on. Here the computation will be performed by the search engine and the search algorithm would be known to the server only.

1.1 Brief History

The concept of Homomorphic Encryption is not new, In 1978 shortly after the invention of the RSA, Rivest,Adleman and Dertouzos came up with an idea which they named as "Privacy homomorphisms". Their paper states that, "although there are some truly inherent limitations on what can be accomplished, we shall see that it appears likely that there exist encryption functions which permit encrypted data to be operated on without preliminary decryption of the operands, for many sets of interesting operations. These special encryption functions we call 'Privacy Homomorphisms'; they form an interesting subset of arbitrary encryption schemes"[1]. Despite the effort made by Rivest,Adleman and Dertouzos, fully homomorphic encryption remained unsolved until 2009, when Craig Gentry first gave the idea of fully homomorphic encryption scheme.

A number of crypto system are homomorphic but with respect to one operation. RSA, ElGamal encryption are homomorphic with respect to multiplication.

In RSA, public key is (N,e) and private key is d, here N is a product of two large primes and de $\equiv 1$ mod $\phi(N)$. If m $\in \mathbb{Z}/n\mathbb{Z}$ is the plain text, then the cipher is c=m^e mod N. Decryption is done using c^d mod N=m. If two messages m_1 and m_2 are encrypted using public key (N,e) then the product of the resulting ciphertext is the cipher of the product of the plaintexts m_1 and m_2

$$(m_1^e \ mod \ N)(m_2^e \ mod \ N)$$
$$= (m_1 m_2)^e \ mod \ N.$$

Thus, $Decrypt(c_1.c_2) = Decrypt(c_1).Decrypt(c_2)$.

In ElGamal, private key is x $\in \{1, \cdots, \text{n-1}\}$ public key is h=$g^x \in$ G, if m_1,$m_2 \in$ G are plaintext messages, then the corresponding ciphertexts are of the form $c_i = (a_i, b_i) = (g^{r_i}, m_i h^{r_i}) \in G*G$ for i=1 and 2, where the r_i are chosen by the encryptor at random $\in \{1, \cdots, n-1\}$, then

$$\text{Decrypt}(c_1, c_2)$$
$$=\text{Decrypt}(a_1 a_2, b_1 b_2)$$
$$=((a_1 a_2)^x)^{-1} b_1 b_2$$
$$=(a_1^x)^{-}1 b_1 .(a_2^x)^{-1} b_2$$
$$=Decrypt(c_1).Decrypt(c_2)$$

There are lots of schemes which are homomorphic for example Goldwasser-Micali cryptosystem and its generalization Paillier cryptosystem are homomorphic with respect to addition of plaintext i.e.

$$\text{Decrypt}(c_1.c_2)=m_1 + m_2$$

but are not homomorphic with respect to multiplication of plaintexts.

Boneh, Goh, and Nissim gave a partially homomorphic encryption scheme that can do one multiplication and any number of addition.

1.2 Analogy

To understand the homomorphic encryption more clearly we can take an physical analogy as described in [2]. Suppose alice has an jewelry store and wants her employees to assemble raw materials (diamonds, gold etc.) into finish products, but she does not trusts her employees and is afraid of theft. So she comes up to a solution by constructing transparent glove box for which she only have key and is only capable of opening it. She puts her raw materials inside it and tells her employee to work using gloves. Using gloves, a employee can manipulate the items inside the box but is not able to take out.

In this analogy, encryption means the employee is unable to take something out of the box and not that he is unable to see it.

Now when employee is done with his work Alice opens the jewelry box and recovers the finished product using key available to her.

1.3 Definition related to Homomorphic encryption

Somewhat Homomorphic Encryption: It is homomorphic encryption which allows a limited no of operations (addition and multiplication) on the ciphertext before the noise becomes too high and decryption starts resulting wrong result.
Fully Homomorphic Encryption: It is the homomorphic encryption which allows a unlimited no of addition and multiplication operation without failing.

Conventional public-key encryption scheme ϵ consists of four algorithms viz. $KeyGen_\epsilon$,$Encrypt_\epsilon$,$Decrypt_\epsilon$ and an additional algorithm $Evaluate_\epsilon$:

1. $KeyGen_\epsilon$: It is a randomized algorithm that takes a security parameter λ as input, and outputs a secret key sk and public key pk;
2. $KeyGen_\epsilon$: It is a randomized algorithm that takes pk and a plaintext $m \in$ P as input, and outputs a ciphertext c $\in$ C.
3. $Decrypt_\epsilon$: It takes sk and c as input, and outputs the plaintext m.
4. $Evaluate_\epsilon$: It takes input as public key pk, a circuit C from permited set of circuits C_ϵ and a couple of ciphertext c_1,c_2,c_3...c_n and outputs a ciphertext c.

The computational complexity of all of these algorithms must be polynomial in λ(security parameter) and for the evaluation it should be polynomial in the circuit size C. The correctness of the schemes ε for circuits in C_ϵ, (sk, pk) generated from $KeyGen_\epsilon(\lambda)$,any plaintext p_1,p_2 ,p_3...p_n and any ciphertext c_1,c_2,c_3...c_n is defined as :

if $KeyGen_\epsilon \rightarrow$ (sk,pk)
$Encrypt_\epsilon(pk, m) \rightarrow$ c, where m $\in$ P
Then $Decrypt_\epsilon(sk, c) \rightarrow m$.

Chapter 2

Motivation

Now a days computing environment is changed, everyone has portable computing devices in form of mobile phones and access to large servers i.e. cloud. This change presents fundamental challenge of outsourcing computation, which is motivated by the asymmetry of the available computing power.

In recent computing scenarios clients are trusted and weak, while computationally strong servers are untrusted as we do not have full control over them. How to outsource the computation ? What about privacy of the outsourced computation? For example, how to outsource computing on medical data, which must be kept confidential at all times? Standard solution to all of these would be to encrypt the data: this will perfectly solves any privacy related issues. However, requirements for standard encryption schemes also do not let us achieve wanted functionality: we cannot perform any computations on the encrypted data.

The solution to all those problems is Homomorphic Encryption. Homomorphic Encryption is one that permits computation on encrypted data. Here client can encrypt his data x and send the encryption $Enc(x)$ to the server. The server can then take the ciphertext $Enc(x)$ and evaluate a function f on the underlying x obtaining the encrypted result Enc(f(x)). The client can decrypt this result achieving the wanted functionality, but the server learns nothing about the data that he computed on.This is very desirable in modern communication system. It also allows to create many other secure crypto systems for example secure voting systems, private information retrieval scheme etc. Below we are discussing one such problem that can be solved efficiently using homomorphic encryption.

Inventory Management: Invantory refers to maintenance of stocks available for the purpose of resale or repair. Inventory management is a key component for cost of goods sold and is key drive of profit, total assets and tax liability. Buying too much stuff can lead to paying warehousing, insurance,shipping and other services related to obtaining and maintaining inventory. All these can affect bottom line. Thus finding the best way to buy and store can make the difference between profits and losses for many companies.

Now a days, due to modularization of production different parts are produce at different locations and after production they are bring together to assemble and give final shape. In this scenario it is extremely necessary to keep a centralized inventory and is confidential data of that company. As the information is used by different part producers so it is necessary to store it on cloud.

Similarly, after production goods are stored at warehouse located at different locations, so that it will be easy to dipatch to end user or retailer from there. To maintain the proper stock at warehouse it is very important to keep stocks record in cloud and is a confidential data of the company. Due to the competition in the market it should not be available to other party, if available then also they should not be able to extract any information from data. So it is utmost necessary to store data on cloud in encrypted form. If data is stored on cloud in encrypted form then we should be able to process data in encrypted form itself, so it takes us to field of homomorphic encryption.

Chapter 3

Comparison Of Different Homomorphic Schemes

3.1 Homomorphic Encryption Schemes

In this chapter we will discuss different schemes available for homomorphic encryption. A homomorphic encryption scheme is the one where we can perform operations in the encrypted domain itself. After the break through work of Craig Gentry in 2009 more homomorphic encryption schemes came to the field of homomorphic cryptography that are based on learning with error and over the integers respectively. To date there are three different families of fully homomorphic schemes available:

1. Gentry's original scheme based on the ideal lattice.
2. Brakerski and Vaikuntanathan's scheme based on Learning With Error(LWE) and Ring Learning With Error(RLWE) problem.
3. Van Dijk, Gentry, and Vaikuntanathan (DGHV) scheme over the integers. This scheme is simplest among all schemes available. Here all operations are performed on integers unlike others.

3.2 Lattice Based Schemes

Gentry's original scheme is based on ideal lattice[2]. It was first implemented by Smart and Vercauteren in 2010 [3]. They used a variant based on principle ideal lattices with constraint that the determinant of lattice is a prime no. In there paper, they have mentioned that they are not able to get bootstrappable scheme because that requires a lattice dimentation of at least $n = 2^{27}$. In addition due to the prime determinant requirement they could not generate keys for dimension n $>$ 2048.

Later on, Gentry and Halevi[4] implemented Gentry's scheme. They implemented using many clever optimization methods available including suggestion to previous work by Smart and Vercauteren. In their implementation they followed the same procedure as that of Smart and Vercauteren but they remove the condition of determinant being a prime no for key generation. In their implementation they claim that for 72 bit security, the key generation takes around 2.2 hours the public key size is 2.3 GB and a ciphertext refresh procedure takes around 30 minute on a high end workstation.

The hard problem underlying Gentry's scheme was problem of finding a "good" basis for an ideal in a number field, given a "bad" basis. Due to the quantum attack presented in paper [5] these first generation setting are not considered as secure against a quantum adversary.

3.3 Integer Based Schemes

The schemes based on Integers are comparatively simpler then lattice based schemes. Here all the operands are integers. Integer based schemes are firstly proposed by van Dijk,Gentry,Halevi and Vaikuntanathan in 2010 after that lots of enhancement has been done to the original scheme.

3.3.1 DGHV Scheme over the Integers

Homomorphic Encryption over the integers can be done by symmetric key encryption scheme and public key encryption scheme. Here we describe the procedure mentioned in[6, 7]. A symmetric key homomorphic encryption scheme is easier and computationally faster then public key homomorphic encryption scheme. It can be transformed into public key homomorphic encryption scheme by doing some extra computation.

The scheme described here is very simple, it encrypts message in bit level i.e. it encrypts bit individually. A similar scheme was used by Craig Gentry[8] to make his scheme Fully Homomorphic but because of high complexity his scheme is not feasible in practical scenario. A bit level encryption scheme makes computation in encrypted domain easy and can be used directly to add and multiply the ciphertexts. We can also use it to create any boolean function.

Symmetric key Encryption

Construction: The scheme is composed of basic operations as mentioned below:

1. **Encrypt**($\mathbf{pk}, \mathbf{m} \in \{\mathbf{0}, \mathbf{1}\}$): Encryption is done by a simple techniques. To encrypt a message bit m $\in \{0, 1\}$ set the ciphertext as an integer whose mod p has the same parity as the message bit parity viz. set ciphertext c=pq + m. where the integer q is chosen from $\{Z \cap [0, 2^{\gamma}/p)\}$ and q $\gg$ p. To make the scheme more hard to crack a random error term 2r is added to obtained ciphertext which is called as the noise term. The noise terms r is chosen from $\{Z \cap (-2^{-\rho}, 2^{\rho})\}$. where ρ=λ (bits), γ=O(λ^5) and λ is a security parameter chosen by user. However while adding the noise term we have to make sure that 2r< $\frac{p}{2}$ in absolute term.
2. **Decrypt**$(\mathbf{p}, \mathbf{c})$:To decrypt the ciphertext obtained, we do the reverse of the encryption operation performed. So to get the plaintext back we do as follows

   $$\text{m=(c mod p) mod 2.}$$

   Here the first mod operation removes the pq term from the cipher and second mod operation removes the noise term so we get back the parity of the message bit.

When the noise r is sufficiently smaller than the secret key p, this scheme becomes both additively and multiplicatively homomorphic for shallow arithmetic circuits. Moreover we can also use the Craig Gentry's techniques as defined in [8] to make it fully homomorphic.

To see how the error terms in this scheme increases with addition and multiplication we can describe it as shown below. Let m_1, and m_2 be the two plaintext messages and c_1 and c_2 be the corresponding ciphertexts then addition and multiplication of these ciphertext will be

$$\begin{gathered} c_1\text{=p}q_1 + 2r_1 + m_1. \\ c_2\text{=p}q_2 + 2r_2 + m_2. \\ c_1 + c_2\text{=}p(q_1 + q_2) + 2(r_1 + r_2) + (m_1 + m_2) \end{gathered}$$

Similarly,

$$c_1 * c_2\text{= } p(pq_1q_2 + q_1 2r_2 + q_1m_2 + 2r_1q_2 + m_1q_2) + 2(r_1m_2 + r_2m_1 + 4r_1r_2) + m_1m_2$$

It is clear from the above, when addition is performed error term increases linearly and multiplication operation increases error term quadratically. so we can perform only a limited no of addition and multiplication operation, but the no of multiple we can perform are less than the no of addition.

Public key Encryption

For many applications it is required to have public key encryption scheme rather than symmetric key scheme. The procedure defined above can be converted into public key by doing some minor changes.

Construction: Public key encryption scheme is based on algorithms as mentioned below.

1. **KeyGen(λ):** The Secret key is an η-bit odd integer where η=O(λ^2). The public key x_i consists of i=0,1,$\cdots$, τ number of encryption of zero. Where $\tau \geq \gamma + \omega(log\lambda)$ and γ=O(λ^5). Here x_i is generated using the distribution $D_{\gamma,\rho}(p)$ where $D_{\gamma,\rho}(p)$ is

   $$D_{\gamma,\rho}(p)= \{ \text{ choose q} \leftarrow Z \cap [0, 2^{\gamma}/p), r \leftarrow Z \cap (-2^{\rho}, 2^{\rho}) : \text{output x=pq+r } \}$$

   Here λ is a security parameter and is user chosen. x_0 is largest no and is equal to pq others are chosen using the above distribution.

2. **Encrypt(pk,m $\in \{0,1\}$):**. To encrypt a message bit m choose a random subset S $\subseteq \{1, 2, \cdots, \tau\}$ and random noise r from (-$2^{\rho}, 2^{\rho}$) and output c=$[m + 2r + 2\Sigma_{i \in S} x_i]_{x_0}$ as ciphertext.

3. **Decrypt(pk,c)**: To obtain a message bit m from the ciphertext c perform the operation as mentioned below

   $$\text{m=(c mod p) mod 2.}$$

Security : The scheme is semantically secure and is based on the hardness assumption of approximate-GCD problem.

3.3.2 DGHV Scheme with Quadratic Form Encryption

This scheme is enhancement of already existing DGHV scheme and is provided by Jean-Sebastien Coron [9]. Here encryption is done in Quadratic form which leads to shorter public keys as a result we only have to keep less no of pubic keys.

The encryption method works with integers x'_{ij} of the form x'_{ij}=$x_{i,0}.x_{i,1}$ mod x_0 for $1 \leq i, j \leq \beta$ where β is a new parameter and denotes the no of distinct public key required. Here only 2β integers $x_{i,b}$ where b $\in$ {0,1} need to be stored in the public key to generate $\tau = \beta^2$ integers x'_{ij} used for encryption. Here encryption is done using quadratic form in the public key elements instead of a linear form and doing this enables to reduce the public key size from τ down to $2\sqrt{\tau}$ integers of γ bits.

Here x_0 is error free term ($x_0 = q_0.p$) and is used to take mod after each operation otherwise error will grow to large. Additionally for encryption one considers a linear combination of the $x'_{i,j}$ with coefficient in [0,2^{α}] instead of bits; this also helps to reduce the public key size. Now lets discuss the actual scheme available below.

KeyGen(1^{λ}): $x_0 = q_0.p$ where q_0 is a random square free 2^{λ}-rough integer in [0,$2^{\gamma}/p$] and p $\in [2^{\eta-1}, \eta)$. Generate integers $x_{i,b}$ for $1 \leq i \leq \beta$ and $b \in \{0, 1\}$:

$$x_{i,b} = p.q_{i,b} + r_{i,b},\ 1 \leq i \leq \beta\ ,\ 0 \leq b \leq 1$$

where $q_{i,b}$ are random integers in [0,q_0), $r_{i,b}$ are integers in $(-2^{\rho}, 2^{\rho})$, sk=p and pk=$(x_0, x_{1,0}, x_{1,1}, ...x_{\beta,0}, x_{\beta,1})$.

Encryption(pk,$m \in \{0, 1\}$): To encrypt a message bit generate a random vector b=$(b_{i,j})$ of size $\tau = \beta^2$ and components in range $[0, 2^{\alpha})$ Generate a random integer r in $(-2^{\rho}, 2^{\rho})$ and output the ciphertext as

$$c = m + 2r + 2 \sum_{1 \leq i,j \leq \beta} b_{i,j}.x_{i,0}.x_{j,1}\ mod\ x_0$$

Evaluate(pk, C, c_1,c_2,c_3,...c_t): Given circuit C with t input bits and t ciphertexts c_i, apply addition and multiplication gates of the circuit C to the ciphertexts, perform all multiplication and addition over the integers and return the resulting integer mod x_0.

Decryption(sk,c): Output m= (c mod p) mod 2.

Security : The scheme is semantically secure and is based on the hardness assumption of error-free approximate-GCD problem.

3.4 Learning With Error Based Schemes

The first public-key encryption scheme based on learning with error was given by Regev[10]. Latter on Brakerski and Vaikuntanathan published the Fully Homomorphic Encryption scheme based on Learning With Error and Ring Learning With Error(RLWE)[11] [12]. In their paper they have used dimension reduction techniques and new modulus switching techinques to shorten the ciphertext length and reduce the decryption complexity.

Recently Brakerski,Gentry and Vaikuntanathan[13] introduce a new FHE scheme where noise increases only lineally with the multiplication depth unlike earlier exponential error increase of Brakerski and Vaikuntanathan scheme. Here they have shown that bootstrapping is no longer necessary to achieve FHE. This scheme is based on the Brakerski and Vaikuntanathan's scheme with new modulus switching techinque. Here ciphertext encrypted under certain modulus p is transformed into a new ciphertext of different modulus $p^{'}$ with reduced noise.

Later on Gentry, Sahai and Wagner [14] in their paper introduce a new encryption techinque based on LWE that removes the need for relineraisation and modulus switching and made bootstrapping optional. Here in order to get rid of relineraisation and make homomorphic operation easier ciphertext are choose to be matrices and plaintext are chosen as an integers.

Based on Gentry, Sahai and Wagner's encryption scheme in 2014 Zhou and Wornell proposed a new encryption scheme. In their encryption scheme they have encrypted integer vector and produce a ciphertext of integers with increase size and dimension. In the following year in 2015 their scheme is again enhanced by Yu,Lai and Payor and claim that their scheme is same secure as that of Zhou and Wornell scheme.

3.4.1 Brakerski Gentry and Vaikuntanathan Scheme

The Scheme of Brakerski, Gentry and Vaikuntanathan is based on ring-LWE scheme of Brakerski and Vaikuntanathan. In this scheme they use a novel idea of "noise" management technique which is based on related techniques of Brakerski and Vaikuntanathan [13]. In their scheme instead of having a single modulus q, they used a set of modulus $q_0, ..., q_l$ of decreasing size. When a fresh encryption of plaintext is done then modulus is q_0, and ciphertext is said to be at level zero. Then as soon as the "noise" gets too large modulus is switch to the new modulus.

parameters: The basic parameters of BGV are:

- F(x): polynomial of degree N where N is a ring dimension.
- σ : standard deviation of the noise distribution.
- p: Plaintext modulus. $p_0, ..., p_l$ are set of co-prime modulus defining the ciphertext modulus $d_t = \Pi_{i=0}^{L-t} P_i$
- A "large" modulus P co-prime to all the p_i.

Here χ^n denotes some n-dimensional Gaussian distribution on integers i.e here each entry in the n-dimensional vector is taken sampling from a Gaussian distribution $N(0, \sigma)$ and then rounding to the nearest integer.

Message Space:

- Pick a polynomial $F(X)$ of degree N the polynomial should be cyclotomic.
- pick co-prime moduli $p_0, ..., p_L$ and set q_t as $q_t = \Pi_{i=0}^{L-t} P_i$.
- $sk \leftarrow R_{p_0}$ with small coefficients say coefficients in set $\{-1, 0, 1\}$. where R_p=$(\mathbb{Z}/p\mathbb{Z}[X])/F(x)$. for some modulo p.
- $a \leftarrow R_{q_0}$.
- $e \leftarrow \chi^N$, say e is an element of R_{q_0}.
- $b \leftarrow a.sk + p.e$
- $\mu \leftarrow R_p$ and $C = R_{q_i}^2$ this is a ciphertext space at level i.

Encryption(m,pk,r):

- $(r_1, r_2, r_3) \leftarrow r$ treat each element as an element of R_{q_0}
- $c_0 \leftarrow b.r_1 + p.r_2 + m$
- $c_1 \leftarrow a.r_1 + p.r_3$
- Output ciphertext as $c = (c_0, c_1) \in C_0$

ModSwitch(c,Q,q): This operation takes ciphertext c and modulus Q and returns a ciphertext with modulus q where $q < Q$ (ciphertext at level $l^{'}$ with modulus $q_{l'}$ and $l^{'} > l$.). Given (c_0, c_1) at level l replace them with $(c_0^{'}, c_1^{'})$ at level $l^{'}$ by Performing the following

- $c_0^{'} \leftarrow (q/Q).c_0$ computed over $\mathbb{Q}[X]$
- $c_1^{'} \leftarrow (q/Q).c_1$ computed over $\mathbb{Q}[X]$
- Round $c_i^{'}$ to the nearest integer polynomial such that $c_i^{'} \equiv c_i$ (mod p).
- Output $c = (c_0, c_1)$

Here if the noise associated to the input ciphertext is around τ, then the noise associated to the output ciphertext is around

$$(q.\tau/Q) + \text{constant}$$

and the output ciphertext decrypts to the same input ciphertext.

Decrypt(c,sk): To get the plaintext back from the ciphertext following procedure is followed

- If c is at level $l \neq 0$ then apply $c \leftarrow ModSwitch(c, l, L)$ else
- $m \leftarrow (c_0 - sk.c_1(mod\ q)_L)$ mod p.

Here the decryption works only if the value $c_0 - sk.c_1(mod\ q)_L$ s identical to value

$$(p.e.r_1 + p.r_2 + m - p.sk.r_3) + m$$

This will happen only if q_L is large enough, as the terms above is a sum of the three products of polynomials in R_{q_L} with small coefficients.

ExtendKey(epk,sk,$sk^{'}$): This method takes an extended key (initially $epk = pk$) and produces another extended key by adding "Key-switch matrix" from secret key $sk^{'}$ to sk to return a new extended key epk.
The operations are performed as follows:

- For $i = 0, ..., log_p q_0$
  -$a_i \leftarrow R_q$
  -$e_i \leftarrow \chi^N$
  -$b_i \leftarrow a_i.sk + p.e_i - p^i.sk^{'}$.
- $pk_{sk' \rightarrow sk} \leftarrow \{(a_i, b_i)\}_{i=0}^{log_p q_0}$
- $epk \leftarrow epk \cup pk_{sk' \rightarrow sk}$

The values a_i, b_i are also called as "quasi-encryptions" of the elements $2^i.sk^{'}$ for $i = 0, ..., log_p q_0$.

$KeySwitch(d, epk)$**:** This algorithm takes a three component " ciphertext" $d = (d_0, d_1, d_2)$ at level l which decrypts in the following manner

$$(d_0 - sk.d_1 - sk^{'}.d_2(mod\ q_l))(mod\ p),$$

for some secret key $sk^{'}$, and turns it into standard ciphertext (c_0, c_1) which decrypts under the key sk.
Writing d_2 in base p as:

$$d_2 = \sum_{i=0}^{log_p^{q_l}} d_{2,i}.p^i$$

where $d_{2,i}$ is a polynomial with small coefficient.
Then output the ciphertext as

$$c_0 = d_0 + \sum_{i=0}^{log_p^{q_l}} d_{2,i}.b_i,$$
$$c_1 = d_1 + \sum_{i=0}^{log_p^{q_l}} d_{2,i}.a_i.$$

Where $pk_{sk' \rightarrow sk} \leftarrow \{(a_i, b_i)\}_{i=0}^{log_p q_0}$ is the key-switching matrix in epk for the key $sk^{'}$
From the above it can easily seen that if $(d_0 - sk.d_1 - sk^{'}.d_2(mod\ q_l))(mod\ p)$, evaluates to m, then $(c_0 - sk.c_1(mod\ q_l))(mod\ p)$ will also evaluate to m.

Homomorphic operation: $ADD(c, c^{'})$: Addition of the ciphertext in the same level is done componentwise viz. given $c = (c_0, c_1)$ encrypting m and $c^{'} = (c_0^{'} + c_1^{'})$ encrypting $m^{'}$ then addition will be

$$c \oplus c^{'} = (c_0 + c_0^{'}, c_1 + c_1^{'})$$

which is the same form as a valid encryption, only difference is that the randomness is now chosen from a distribution which is larger. The output level of a ciphertext from the addition operation is $max(l, l^{'})$, where l and $l^{'}$ are the respective level of the ciphertext c and $c^{'}$. Here the ciphertext are first move to the same level before adding via a ModSwitch operation.

$Mult(c, c^{'})$: If the ciphertext are at the different level then ciphertext are first move to the same level and multiplication is performed. To perform multiplication a three element ciphertext (d_0, d_1, d_2) is first obtained as follows:

$$d_0 \leftarrow c_0.c_0^{'},$$
$$d_1 \leftarrow c_1.c_0^{'} + c_0.c_1^{'},$$
$$d_2 \leftarrow -c_1.c_1{'}.$$

The obtained three element ciphertext is decrypted by performing operation

$$(d_0 - sk.d_1 - sk^2.d_2(mod\ q))(mod\ 2)$$

Here the noise associated to the three element ciphertext is roughly same as product of two constituent ciphertexts.
As long as the extended public key *epk* contains a key-switching matrix for the key switch $sk^2 \rightarrow sk$ we can then apply KeySwitch operation above to form a two element ciphertext. The obtained two element ciphertext have noise roughly equivalent to that of three element ciphertext. If noise needs to be further controlled then ModSwitch operation can be performed to move the output ciphertext from level l to level $l+1$.

Security: Security of the above scheme is based on Ring-Learning With Error problem.

3.4.2 Yet Another Somewhat Homomorphic Encryption Scheme (YASHE)

Lopez proposed a FHE scheme based on the work of Stehle and Strinfeld where a provable secure version of NTRUEncrypt is presented with security based on the standard problem of ideal lattice. YASHE [15] makes an additional assumption of uniformity of public key also called as decisional small polynomial ratio (DSPR) to allow homomorphic operations and remain semantically secure. Below is the more practical variant of the scheme.

Parameters: Given a security parameter λ, fix a positive integer d and modulus q that determines R=$\mathbb{Z}[X]/(\Phi_d(X))$, and t with $1 < t < q$, and distribution χ_{key},χ_{err} on R.

Message space Here the message space is given by

$$\text{R/tR}=(\mathbb{Z}[X]/(\Phi_d(X)))/(t(\mathbb{Z}[X]/(\Phi_d(X))))$$

KeyGen: $(d, q, t, \chi_{err}, \chi_{key}, \omega)$

- Sample $f^{'}, g \leftarrow \chi_{key}$ and let f=$[tf^{'}+1]_q$
- If f is not invertible modulo q, choose a new $f^{'}$
- Compute the inverse $f^{-1} \in R$ of f modulo q and set h=$[tgf^{-1}]_q$
- Sample $\vec{e}, \vec{s} \leftarrow \chi_{err}^{l_{w,q}}$, compute $\vec{\gamma}$=$[P_{w,q}(f) + \vec{e} + h.\vec{s}]_q \in R^{l_{w,q}}$
- Output (pk,sk,evk)=(h,f,$\vec{\gamma}$).

Encrypt(m,pk):

- For a message $m + tR$, choose $[m]_t$ as its representative.
- Sample $s, e \leftarrow \chi_{err}$.
- Output the ciphertext c of plaintext m as:

$$\text{c}=[\lfloor q/t \rfloor [m]_t + e + hs]_q \in R$$

Decryption(c,sk): The decryption of ciphertext c is performed as follows: m=$[\lfloor t/q.[skc]_q \rfloor]_t \in R$

KeySwitch($\tilde{C_{mult}}$,evk) : The key switch operation outputs the ciphertext as follows:

$$[< D_{w,q}(\tilde{C_{mult}}, evk) >]_q$$

The key switching algorithm transforms an intermediate encryption into a ciphertext that can be decrypted with f itself. The evaluation key is the one obtained as f=$[P_{w,q}(f)+\vec{e}+h.\vec{s}]_q$ where $\vec{e}, \vec{s} \leftarrow \chi_{err}^{l_{w,q}}$ are vectors of polynomials sampled from the error distribution χ_{err}. The key is a vector of quasi-encryptions of the secret key f under its corresponding public key and is required for the homomorphic multiplication operation so it is made public. So it have to satisfy circular security condition.

Homomorphic Operations:

Add(c_1, c_2): Given two ciphertexts $c_1, c_2 \in R$ of two plaintext messages m_1, m_2 with inherent noise term ν_1, ν_2, then their sum modulo q, $c_{add} = [c_1 + c_2]_q$ encrypts the sum of the messages modulo t, $[m_1 + m_2]_t$. We can also write $[m_1]_t$+$[m_1]_t$=$[m_1 + m_2]_t$+tr_{add} for some $r_{add} \in R$ with $\|r_{add}\|_{\text{inf}} \leq 1$.

Mult(c_1, c_2, evk): Output the ciphertext

$$c_{mult} = KeySwitch(\tilde{C_{mult}}, evk), \text{ where } \tilde{c_{mult}} = [\lfloor \tfrac{t}{q} c_1.c_2 \rceil]_q$$

for ciphertexts $c_1, c_2 \in R$ of two plaintext messages $m_1, m_2 \in R$, Here the intermediate ciphertext $\tilde{C_{mult}}$ during the homomorphic multiplication Mult satisfies $f^2 c_{mult}$= $\Delta[m_1 m_2]_t + \tilde{\nu}_{mult}$ mod q.

Security: The security of the YASHE scheme is based on the hardness assumption of the RLWE assumption.

3.4.3 Gentry Sahai Waters(GSW) Scheme

The Gentry-Sahai-Waters(GSW) scheme[14] is an LWE-based scheme that removes the need for relinersation and modulus-switching and makes bootstrapping optional.

Parameters: The parameters of the GSW are as follows

- modulus q.
- Lattice dimension parameter n.
- Error dimension χ
- choose parameter m=$O(n \log(q))$
- l=$\lfloor log(q) \rceil$+1 and $N = (n + 1).l$

Message Space: Two decryption algorithms are present here and message space is different for both the algorithms. First one is Decrypt it enables decryption of small messages μ where $\mu \in \{0, 1\}$ and Second one is MPDecrypt, it allows decryption for any $\mu \in \mathbb{Z}_q$

The scheme Here ciphertext is an $N \times N$ matrix over $\mathbb{Z}_q$ with small entries and the secret key v an N-dimension vector with entries in the same space. We say ciphertext C encrypts a message μ if the following condition is satisfied $C.v = \mu.v + e$. Here the vector v should be approximate eigen vector of the matrix c

KeyGen():

- Sample $t \leftarrow \mathbb{Z}_q^n$
- Output $sk = s \leftarrow (1, -t_1, ..., -t_n) \in \mathbb{Z}_q^{n+1}$
- Let $v = Powersof2(s)$ where Powersof(2)=$(b_1, 2b_1, ..., 2^{l-1}b_1, ...b_k, 2b_k, ..., 2^{l-1}b_k)$.
- Generate $B \leftarrow \mathbb{Z}_q^{m \times n}$ uniformly and $e \leftarrow \chi^m$

- $b = B.t + e$ and A to be the $(n+1)$-column matrix $A = (b||B)$ where $||$ represents the concatenation of the vector b and matrix A.
- Finally, set public key as pk=A note that $A.s = e$

Encryption(μ, pk): To encrypt a message $\mu \in \mathbb{Z}_q$ sample $R \in \{0,1\}^{N \times m}$ uniformly and output ciphertext C as

$$C = Flatten(\mu.I_N + BitDecomp(R.A)) \in \mathbb{Z}_q^{N \times N}$$

where BitDecomp(a)=$(a_{1,0}, ..., a_{1,l-1}, ..., a_{k,0}, ..., a_{k,l-1})$
$BitDecomp^{-1}(a^{'})$=$(\sum 2^j.a_{1,j}, ..., \sum 2^j.a_{k,j})$ and
Flatten($a^{'}$)= $BitDecomp(BitDecomp^{-1}(a^{'}))$

Decrypt(C,sk): The decryption of the ciphertext c is achieved as follows:

- First compute $x_i \leftarrow < C_i, v >$ and then compute
- Output $\mu^{'} = \lfloor x_i/\nu_i \rceil$

MPDecrypt(C,sk): Here the decryption is done as mentioned below:

- Recover LSB(μ) from $\mu.2^{l-2} + small$
- After above step recover the next LSB from $(\mu - LSB(\mu))\mu.2^{l-3} + small$

Homomorphic Operations: Here we have two ciphertext c_1 and c_2 where $c_i.v = \mu_i.v + e_i$ and $i \in \{1, 2\}$.
ADD(c_1, c_2): Define c^+=$c_1 + c_2$ the correctness of the addition comes as follows $c^+.v = (\mu_1 + \mu_2).v + (e_1 + e_2)$ where error is still small.
MULT(c_1, c_2): Define c^*=$c_1 + c_2$ for correctness check we can define c^* as $c^*.v$=$\mu_1.\mu_2.v + \mu_2.e_1 + c_1.e_2 = \mu_1.\mu_2.v +' small'$.

Security: Security of the above scheme is based on the LWE hardness assumption.

3.4.4 Zhou and Wornell Scheme

Symmetric key Encryption:

The encryption schemes describe here is an natural extension of the PVW scheme [16] from binary vectors to integer vectors. In this section we have discussed the scheme described in paper [17]. Here the plaintext vector $x \in Z_p^m$ has length m and alphabet size p. Ciphertext vector $c \in Z_q^n$ is the cipher of x with lenth $n > m$ and alphabet $q \gg p$. Typically q should be super-polynomial in the ciphertext length n. The secret key is a matrix $S \in Z_p^m$, and it satisfies

$$Sc = qk + wx + e, \tag{3.1}$$

for some integer vector k and error vector e, w is an secret parameter and have to satisfy $w > 2|e|$ condition and is less then $\frac{2q}{2p+1}$. By the notation $|v|$, $|M|$ we mean the maximum absolute value of the entries in a vector v and matrix M.

The process of encryption of the plaintext vector x is to find a ciphertext vector c that satisfies the equation above for some integer vector k and error vector e. The encryption can be done as described below:

KeyGen: Choose some random matrix $S \in Z^{m*n}$
Encrypt(S,x): Find a vector c that satisfies equation above
Decrypt(S, c): It can be done as shown below

$$x=\lceil \frac{Sc \ mod \ q}{w} \rfloor_p$$

Here $\lceil a \rfloor_q$ means the nearest integer to a with modulus q. The decryption will be only sucessfull if the magnitude of e, i.e. $|e|$ is smaller than $\frac{w}{2}$.

public key Encryption:

The symmetric key encryption scheme described above can be extended to the public key encryption scheme, but for that we have to switch the symmetric key to public key and secret key. The key switching is done based on the algorithm described by Brakerski and Vaikuntanathan in [18]. The Algorithm has two steps, we apply it to switch the secret key $S \in Z_q^{m*n}$ to another secret key $S^{'} \in Z_q^{m*n^{'}}$ during the process of the key conversion we get a new ciphertext $c^{'}$ that still encrypts the same integer vector x. The steps of the key switching is done as follows:

Step 1: In this step we switch the secret key S to intermediate secret key S^* such that new ciphertext c^* has a much smaller magnitude than c. The main idea here is to represent each element c_i in c with binary vector. Hence it results in a new ciphertext c^* with $|c^*|$=1. Say c_i=$b_{i0} + 2b_{i1} + \cdots + 2^{l-1}b_{i(l-1)}$, then it is written as

$$\begin{bmatrix} b_{i0} \\ b_{i1} \\ \cdots \\ b_{i(l-1)} \end{bmatrix}$$

in this way writing each c_i we get c^*. Here the value of l is given by $l = \lceil \log_2 q \rceil$.
Next we construct a secret key $S^* \in Z^{m*nl}$ such that

$$S^*c^* = Sc.$$

This can be done by writing each element of S_{ij} in S by a vector $[S_{ij}, 2S_{ij}, \cdots, 2^{l-1}S_{ij}]$.

step 2: In this step, intermediate secret key $S^* \in Z^{m*nl}$ obtained in the step1 is switch to the desired secret key $S^{'} \in Z^{m*n^{'}}$. For doing this we construct an integer matrix $M \in Z^{n^{'}*nl}$ such that

$$S^{'}M = S^* + E \ (mod \ q) \tag{3.2}$$

here matrix E is an error matrix and each terms of the matrix are $< \frac{w}{2}$ for correct decryption.

If $S^{'}$ =$[I \ T]$ with an identity matrix I, then integer matrix M can be found by

$$\text{M}=\begin{bmatrix} -TA + S^* + E \\ A \end{bmatrix}$$

Here A $\in Z_q^{(n^{'}-m)*nl}$ is a random matrix.

Now the new ciphertext is

$$c^{'} = Mc^* \ (mod \ q) \tag{3.3}$$

The implementation of the above scheme is done as follows: Suppose user have stored the encrypted data in cloud now he wants to perform some operation on cloud. He will first generate the key-switching matrix M based on the two secret key S and $S^{'}$, (the secret key S and $S^{'}$ is known to him only). He sends the calculated public key matrix M to the server publicly. Server using obtained M calculates $c^{'}$ from already existing c.

Suppose user wants to perform some operation on cloud for that he will encrypt his operation using M and sends it to server, server without knowing the query, performs operation in the encrypted domain and sends back the result obtained. User then decrypts the obtained result using his secret key $S^{'}$.

Homomorphic Operations:

Zhou Wornell scheme[17] is a somewhat homomorphic scheme. It correctly decrypts when the error term is lower then W. It is very useful in data analysis, signal processing, machine learning etc. It is very efficient as it directly operates on the integer vector. This scheme supports 3 operations viz. addition, linear transformation and weighted inner product. Moreover using these three primitive operations we can calculate arbitrary polynomial efficiently and securely.

Addition:

Addition is easy to perform in this scheme. For addition there are two cases either both the ciphertext are encrypted using the same key or by different key. Below we have described both the scenarios of addition separately.

Case 1: Both the ciphertext are encrypted using same key: Let X_1 and X_2 be the two vectors and are encrypted by the same key S i.e.

$$Sc_1 = qk_1 + wX_1 + e_1$$
$$Sc_2 = qk_2 + wX_2 + e_2$$

Then there addition is given by

$$Sc^{'} = qk^{'} + w(X_1 + X_2) + (e_1 + e_2)$$

where $k^{'}$ is an integer vector with small magnitude.

Case 2: Both the ciphertext are encrypted using different keys: Let X_1 and X_2 be two vectors and are encrypted by the key S_1 and S_2 i.e. by different keys. So to perform addition operation we need to switch one key to the other key i.e. we switch S_1 to S=S_2 as described earlier and perform addition operation. Similarly if S_1 and S_2 are correlated then for the purpose of guaranteeing security, one secret key S is created and both the keys S_1 and S_2 are switch to the new secret key S. Let $c_1^{'}$ and $c_2^{'}$ denotes the ciphertexts obtained after the key switching, and they satisfy

$$Sc_i^{'} = qk_i{'} + wX_i + e_i^{'}$$

with $| k_i^{'} |$ and $| e_i^{'} |$ small for i=1,2.
Now the ciphertext are simply added to get the addition of the plaintext as follows :

$$Sc^{'} = qk^{'} + w(X_1 + X_2) + e^{'}$$

with $| k^{'} | \leq | k_1^{'} | + | k_2^{'} |$ and $e^{'} = e^{'} + e^{'}$.

Case 3: Addition with constant vector: If X_2 is a constant vector and we want to add this vector with our vector X_1 i.e. we want to perform X_1 + a then we can simply add it to the vector X_1. The ciphertext will be c=$c_1 + w[a^T, 0, \cdots, 0]^T$ mod q. It is possible to perform since our secret key is of the form [I T].

In addition operation of vectors the error term increases linearly.

Linear Transformation

Linear transformation satisfies two properties as given below

$$\mathrm{T}(v_1 + v_2)=\mathrm{T}v_1 + \mathrm{T}v_2$$
$$\mathrm{T}(\alpha \mathrm{v})=\alpha \mathrm{T}(\mathrm{v})$$

where v_1 and v_2 are two vectors and α is a scalar.

Linear transformation in Zhou's scheme is performed as shown below

$$GSc_1 = qGk_1 + wGx_1 + Ge_1$$

Here the vector x_1 is transformed to the new vector Gx_1, where G is a matrix and x_1 is a vector. Usually $\mid G \mid \ll$ q, in that case $c^{'} = c_1$ is treated as ciphertext of Gx_1 with secret key GS=$S^{'}$. Here the noise is multiplied by G in each operations performed.

For a smaller integer a, scalar transformation is calculated as ax_1 and its corresponding ciphertext will be

$$\mathrm{c}=\mathrm{a}c_1 \bmod \mathrm{q}$$

Here the noise is multiplied by a after each operation performed.

Weighted inner products

A group of weighted inner product $x_1^T H_j x_2$ is computed by applying the techniques presented in [18]. To find the weighted inner product we need to find the value of

$$\mathrm{P}= \mathrm{vec}(S_1^T H_j S_2).\ \mathrm{vec}(c_1 c_2^T),$$

The value of P can be give as

$$\mathrm{P}=(S_1 c_1)^T H_j (S_2 c_2) = \mathrm{vec}(S_1^T H_j S_2).\ \mathrm{vec}(c_1 c_2^T),$$

Here H represents the weight matrix. But the left hand side of the above equation evaluates to

$$(S_1 c_1)^T H_j (S_2 c_2) = q(qk_1^T H_j K_2 + wk_1^T H_j X_2 + wX_1^T H_j X_2 + wX_1^T H_j k_2) + w^2 (X_1^T H_j x_2) + qk_1 T H_j e_2 + qe_1^T H_j k_2 + wX_1^T H_j e_2 + we_1^T H_j x_2 + e_1^T H_j e_2$$

So the Secret key is $\mathrm{vec}(S_1^T H_j S_2)$ and the ciphertext is $c^{'} = \lceil \frac{vec(c_1 c_2^T)}{w} \rceil$
Performing the multiplication of the secret key and the plaintext matrix it evaluates to $x_1^T H_j x_2$ as required.

As seen from the above operations each time weighted inner product is done the noise squares so we can perform weighted inner product operations only a limited no of times.

3.4.5 Yu,Lai and Payor Scheme

Symmetric key Encryption:

Yu-Lai-Payor encryption scheme [19] describe here is an slight modification of Zhou-Wornell scheme [16]. The plaintext vector x $\in Z_p^m$ has length m and alphabet size p. Ciphertext vector c $\in Z_q^n$ is the cipher of x with length $n > m$ and alphabet $q \gg p$. Typically q should be super-polynomial in the ciphertext length n. The secret key is a matrix $S \in Z_p^m$, and it satisfies

$$Sc = wx + e, \tag{3.4}$$

Here e is a error vector, w is an secret parameter and have to satisfy $w > 2|e|$ condition and is less then $\frac{2q}{2p+1}$. By the notation $|v|$, $|M|$ we mean the maximum absolute value of the entries in a vector v and matrix M.

The process of encryption of plaintext vector x is to find a ciphertext vector c that satisfies the equation above error vector e. The encryption can be done as described below:

KeyGen: Choose some random matrix S $\in Z^{m*n}$
Encrypt(S,x): Find a vector c that satisfies equation above
Decrypt(S, c): It can be done as shown below

$$\text{x}=\lceil \frac{Sc}{w} \rfloor_p$$

Here $\lceil a \rfloor_q$ means the nearest integer to a with modulus q. The decryption will be only sucessfull if the magnitude of e, i.e. $|e|$ is smaller than $\frac{w}{2}$.

public key Encryption:

The symmetric key encryption scheme described above can be extended to the public key encryption scheme, but for that we have to switch the symmetric key to public key and secret key. The key switching is done based on the algorithm described by Brakerski and Vaikuntanathan in [18]. The Algorithm has two steps, we apply it to switch the secret key $S \in Z_q^{m*n}$ to another secret key $S^{'} \in Z_q^{m*n^{'}}$, during the process of the key conversion we get a new ciphertext $c^{'}$ that still encrypts the same integer vector x. The steps of the key switching can be given as follows:

Step 1: In this step we switch the secret key S to an intermediate secret key S^* such that new ciphertext c^* has much smaller magnitude than c. The main idea is to represent each element c_i in c with binary vector. Hence it results in a new ciphertext c^* with $|c^*|$=1. Say c_i=$b_{i0} + 2b_{i1} + \cdots + 2^{l-1}b_{i(l-1)}$, then c_i is written in bit representation as:

$$\begin{bmatrix} b_{i(l-1)} \\ \vdots \\ b_{i1} \\ b_{i0} \end{bmatrix}$$

in this way writing each c_i we get c^*. Here the value of l is given by $l = \lceil \log_2 q \rceil$.
Next we construct a secret key $S^* \in Z^{m*nl}$ such that

$$S^*c^* = Sc.$$

This can be done by writing each element of S_{ij} in S by a vector $[2^{l-1}S_{ij}, \cdots, 2S_{ij}, S_{ij}]$.

step 2: In this step, intermediate secret key $S^* \in Z^{m*nl}$ obtained above is switch to the desired secret key $S^{'} \in Z^{m*n^{'}}$. For that we construct an integer matrix $M \in Z^{n^{'}*nl}$ such that

$$S^{'}M = S^* + E \tag{3.5}$$

here matrix E is an error matrix and each terms of the matrix are $< \frac{w}{2}$ for correct decryption.

If $S^{'}$ =[I T] with an identity matrix I, then integer matrix M can be find out by

$$\text{M}=\begin{bmatrix} -TA + S^{*} + E \\ A \end{bmatrix}$$

Here $A \in Z_q^{(n^{'}-m)*nl}$ is a random matrix and $E = \mathbb{Z}^{m \times nl}$ is a random noise matrix.

Now the new ciphertext is

$$c^{'} = Mc^{*} \tag{3.6}$$

from above it is clear that

$$S^{'}c^{'} = S^{*}c^{*} + Ec^{*}$$

where $Ec^{*} = e^{'}$ is the new error with $|e^{'}|$ kept small, because $|c^{*}| = 1$ by Construction and E can be randomly generated with $|E|$ small.

Homomorphic Operations:

Yu, Lai and Payor scheme[17] is a somewhat homomorphic scheme. It correctly decrypts to the level when the error term is lower then W. It is very useful in data analysis, signal processing, machine learning etc. It is very efficient as it directly operates on the integer vector. This scheme supports 3 operations viz. addition, linear transformation and weighted inner product. Moreover using these three primitive operations we can calculate arbitrary polynomial efficiently and securely.

Addition:

Addition is easy to perform in this scheme. For addition there are two cases either both the ciphertext are encrypted using the same key or by different key. Below we have described both the scenarios of addition separately.

Case 1: Both the ciphertext are encrypted using same key: Let X_1 and X_2 be the two vectors and are encrypted by the same key S i.e.

$$Sc_1 = wX_1 + e_1$$
$$Sc_2 = wX_2 + e_2$$

Then there addition is given by

$$Sc^{'} = w(X_1 + X_2) + (e_1 + e_2)$$

Case 2: Both the ciphertext are encrypted using different keys: Let X_1 and X_2 be two vectors and are encrypted by the key S_1 and S_2 i.e. by different keys. So to perform addition operation we need to switch one key to the other key i.e. we switch S_1 to S=S_2 as described earlier and perform addition operation. Similarly if S_1 and S_2 are correlated then for the purpose of guaranteeing security, one secret key S is created and both the keys S_1 and S_2 are switch to the new secret key S. Let $c_1^{'}$ and $c_2^{'}$ denotes the ciphertexts obtained after the key switching, and they satisfy

$$Sc_i^{'} = wX_i + e_i^{'}$$

with $|e_i^{'}|$ small for i=1,2.
Now the ciphertext are simply added to get the addition of the plaintext as follows :

$$Sc^{'} = w(X_1 + X_2) + e^{'}$$

with $e^{'} = e^{'} + e^{'}$.

Case 3: Addition with constant vector: If X_2 is a constant vector and we want to add this vector with our vector X_1 i.e. we want to perform X_1 + a then we can simply add it to the vector X_1. The ciphertext will be c=$c_1 + w[a^T, 0, \cdots, 0]^T$. It is possible to perform since our secret key is of the form [I T].

In addition of vectors the error term increases linearly.

Linear Transformation

Linear transformation satisfies two properties as given below

$$\mathrm{T}(v_1 + v_2)=\mathrm{T}v_1 + \mathrm{T}v_2$$
$$\mathrm{T}(\alpha \mathrm{v})=\alpha \mathrm{T}(\mathrm{v})$$

where v_1 and v_2 are two vectors and α is a scalar.

Linear transformation in Zhou's scheme is performed as shown below

$$GSc_1 = wGx_1 + Ge_1$$

Here the vector x_1 is transformed to the new vector Gx_1, where G is a matrix and x_1 is a vector. Usually $\mid G \mid \ll$ q, in that case $c^{'} = c_1$ is treated as ciphertext of Gx_1 with secret key GS=$S^{'}$. Here the noise is multiplied by G in each operations performed.

For a smaller integer a, scalar transformation is calculated as ax_1 and its corresponding ciphertext will be

$$\mathrm{c}=\mathrm{a}c_1$$

Here the noise is multiplied by a after each operation performed.

Weighted inner products

A group of weighted inner product $x_1^T H_j x_2$ is computed by applying the techniques presented in [18]. To find the weighted inner product we need to find the value of

$$\mathrm{P}=\ \mathrm{vec}(S_1^T H_j S_2).\ \mathrm{vec}(c_1 c_2^T),$$

The value of P can be give as

$$\mathrm{P}=(S_1 c_1)^T H_j (S_2 c_2) = \mathrm{vec}(S_1^T H_j S_2).\ \mathrm{vec}(c_1 c_2^T),$$

Here H represents the weight matrix. But the left hand side of the above equation evaluates to

$$(S_1 c_1)^T H_j (S_2 c_2) = w^2 (X_1^T H_j x_2) + w X_1^T H_j e_2 + w e_1^T H_j x_2 + e_1^T H_j e_2$$

So the Secret key is vec($S_1^T H_j S_2$) and the ciphertext is $c^{'}=\lceil \frac{vec(c_1 c_2^T)}{w} \rceil$
Performing the multiplication of the secret key and the plaintext matrix it evaluates to $x_1^T H_j x_2$ as required.

As seen from the above operations each time weighted inner product is done the noise squares so we can perform weighted inner product operations only a limited no of times.

Chapter 4

Preliminary Work

4.1 Forecasting Time series data

Time series data are the quantitative observed data that are evenly spaced in time and measured successively. Examples of time series data includes continuous monitoring of a person's heart rate, hourly reading of air temperature, daily closing price of a company stock, monthly rainfall data, yearly sales figures etc. To perform operation on time series data usually data should be atleast 50 or more data points in a series. If the time series exhibits seasonality, there should be at-least 4 to 5 cycles of observations in order to fit a seasonal model of data.

Time series data can be analyzed to understand the underlying structure and function that produce the observations. Understanding the mechanisms of a time series will help us to get mathematical model that explains the data in such a way that prediction, monitoring, or controlling can be done. Examples include forecasting, which is widely used in economics and business. Monitoring of ambient conditions, or of an input or an output, is common in science and industry.

In general it is assumed that time series data exhibits at least one systematic pattern. The most common patterns are trends and seasonality. Trends can be linear or quadratic. Seasonality is a trend that repeats itself systematically over time.

In todays scenario, it is necessary to have stock shared between retailer and manufacturer. After each period of time manufactures can forecast data and can deliver according amount to the retailer in this way manufacturer and retailer can perform optimal. Storing stock in unencrypted form in cloud can be dangerous so retailer must store stock detail in encrypted form. To perform forecast in encrypted domain manufacturer should be able to do it homomorphically.

In the below sections we are going to discuss different Forecasting methods available. We have foretasted time series data both normally and homomorphically and latter on made a compairsion between both. To forecast time series data homomorphically we have used Zhou's scheme. As forecasting constant usually lies between zero and one so we can not use the Zhou's scheme directly. so we have used the below procedure to handle fractions.

4.1.1 Dealing with Fractions

As we know Zhou's scheme handles integers, so to use it we have to modify our data some way so that we can use it. We will follow the below conventions to represent floating point numbers as integers:

- All floating point numbers greater than 1, we will round them to nearest integer.
- Proportions between 0 and 1, can be represented as integers by multiplying them with a large integer and then rounding off.

Here our main concern is the floating numbers between 0 and 1.
Suppose we have one such fraction β such that $0 < \beta < 1$. Here we will take a large integer C and represent β as $round(C\beta)$ where $1 < C < p$ (plaintext space) and $1 < C < q$ (ciphertext space)

Note that this conventions only supports for liner function f. If the function is non-linear then it will calculate non-liner power of C along with β. This can give erroneous results in the encryption scheme as the powers C can quickly exceed q, the alphabet size of cipher.

4.1.2 Simple Exponential Forecasting

The simplest of the exponential forecasting is called as "Simple Exponential forecasting". This method is suitable for forecasting if forecasting data do not have any trend and seasonality pattern. The exponential equation used for forecasting in simple exponential smoothing is

$$y_t = \alpha.x_t + (1-\alpha)y_{t-1}$$

for t=1,2,...,T and $0 \leq \alpha \leq 1$ is the forecasting parameter. The initial parameter for forecast is taken as y_0 value.

Suppose we have encrypted data $c_i \ \forall x_i$ and we want to forecast the encrypted data then also we can not apply the above recursive formula directly. This is because we are making higher power of α. Thus we need to represent the above equation in linear form. Expanding the above equation we get

$$y_t = \alpha \sum_{j=0}^{i-1} (i-\alpha)^j x_{i-j} + (1-\alpha)^i x_0$$

Thus we can represent y as a linear transformation of vector x i.e. $y = Gx$ where G is given by

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ (1-\alpha) & \alpha & 0 & 0 & \cdots & 0 \\ (1-\alpha)^2 & \alpha(1-\alpha) & \alpha & 0 & \cdots & 0 \\ (1-\alpha)^3 & \alpha(1-\alpha)^2 & \alpha(1-\alpha) & \alpha & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \end{bmatrix}$$

Since all the values of the matrix G above lies between zero and one so we have to multiply each element of the matrix by an appropriate constant and make it an integer.

New transformation matrix $G^{'}$=round(KG) is an integer matrix which can be directly used for homomorphic linear transformation as described in the Zhous's scheme.

At the end when we decrypt the result, we divide the vector by K and round it off as follows

$$L = Gx \approx round(Decrypt_{q,w}(c, G^{'}S)/K)$$

Where c is the ciphertext of x. Note that division operation by K is performed by the user after decryption. In the next section we will use the same process for transformation.

It is observer that whenever error is introduced in the forecasting case it is solely due to the rounding off. If $K|x| < p$ and $K|S| < q$ then no error is introduced by it. We can achieve this by choosing large enough p and q value.

The below figure is drawn for monthly milk production data(Jan 62 to Dec 75). From the figure it can be seen that graph between original data with Normal forecasting and Homomorphic forecasting is almost same.

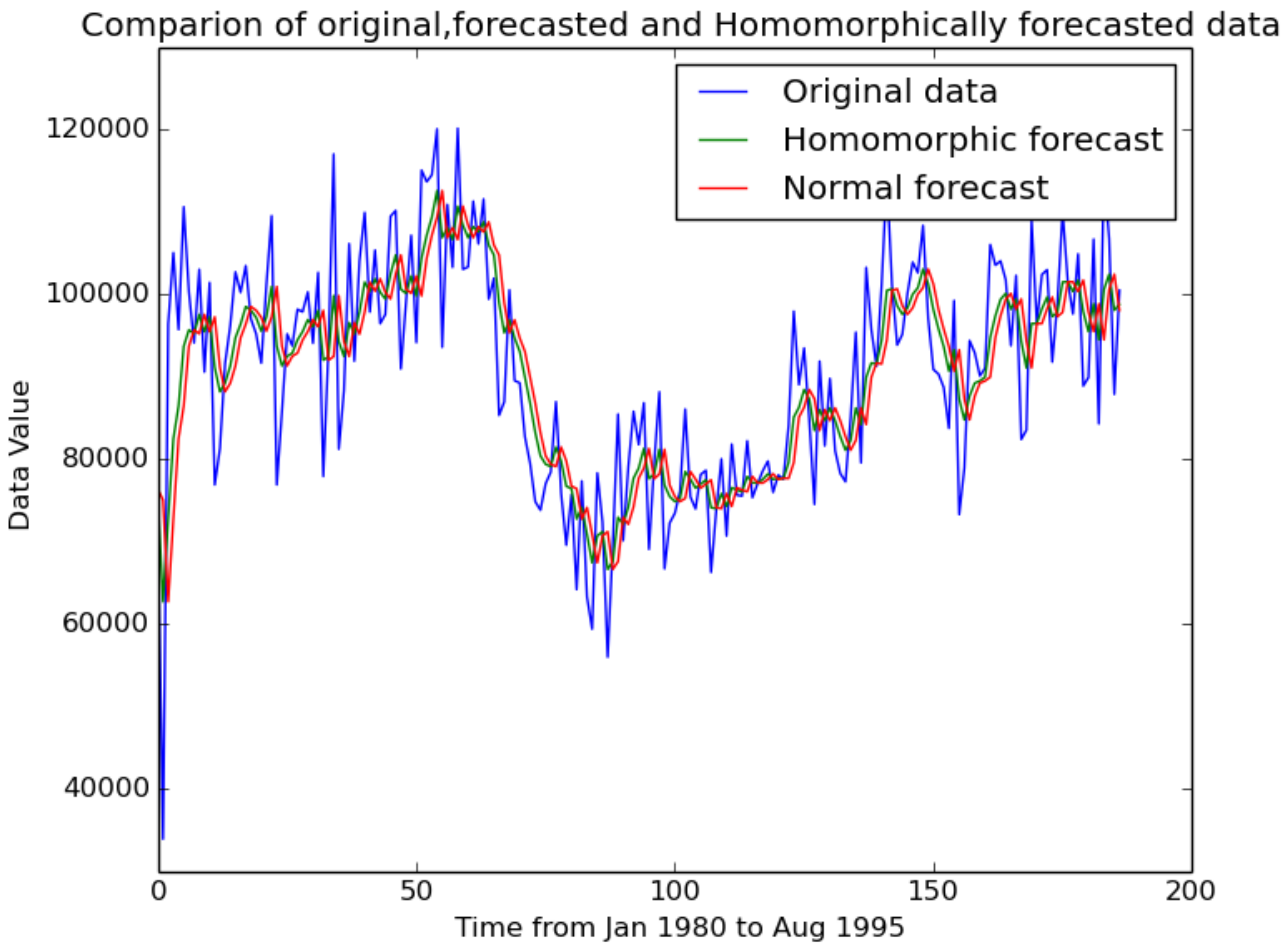

Figure 4.1: Simple Exponential Forecasting

4.1.3 Double Exponential Forecasting

If trend is present in the data set then simple exponential smoothing do not perform well. In such scenario double exponential smoothing performs well and is used for forecasting. Double exponential smoothing is a recursive application of an exponential filter twice thus it is termed as double exponential smoothing. The basic idea of double exponential smoothing is to introduce a term to take into account the possibility of a series exhibits some form of trend. This method is also known as Holt's linear trend model. The recursive equation of Holt's linear trend model is given as follows:

$$y_t = \alpha.x_t + (1-\alpha)y_{t-1}$$
$$b_t = \beta.(y_t - y_{t-1}) + (1-\beta)b_{t-1}$$

for t=1,2,...,T. and $0 \leq \alpha, \beta \leq 1$ is the forecasting parameter.
The initial parameter for forecast is taken as $y_1 = x_1$ and $b_1 = x_1 - x_0$.
The m period ahead forecast is given by F_{t+m}. Where F_{t+m} is given by

$$F_{t+m} = y_t + mb_t$$

As mentioned in the single exponential smoothing these equations cannot be use directly for forecasting. To use them for forecasting firstly we have to convert them into linear form. From the above two equations it is clear that they are interdependent to each other, so we can not directly expand them to get a nice linear form. So here we have used a slight variations of the equations as mentioned below.

We use the following approximation

$$y_t^{'} = \alpha.x_t + (1-\alpha)y_{t-1}^{'}$$
$$b_t = \beta.(y_t^{'} - y_{t-1}^{'}) + (1-\beta)b_{t-1}$$
$$y_t = y_t^{'} + (1-\alpha)b_{t-1}$$

Now we can find the $y^{'}$,b and y from the above equations. To find $y_t^{'}$ we can use the same procedure as used in single exponential smoothing. To find b we can use $b = G_1 G x$ where G_1 is given by

$$G_1 = \begin{bmatrix} -\beta & \beta & 0 & 0 & 0 & \cdots & 0 \\ -\beta(1-\beta) & -\beta^2 & \beta & 0 & 0 & \cdots & 0 \\ -\beta(1-\beta)^2 & -\beta^2(1-\beta) & -\beta^2 & \beta & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \end{bmatrix}$$

After calculating vector $\vec{b}$ add $(1-\beta)^{termno} * b_0$ to each term.

The below figure is drawn for the same data as that of single exponential smoothing. From the figure it can be seen that graph between original data with Normal forecasting and Homomorphic forecasting is almost same.

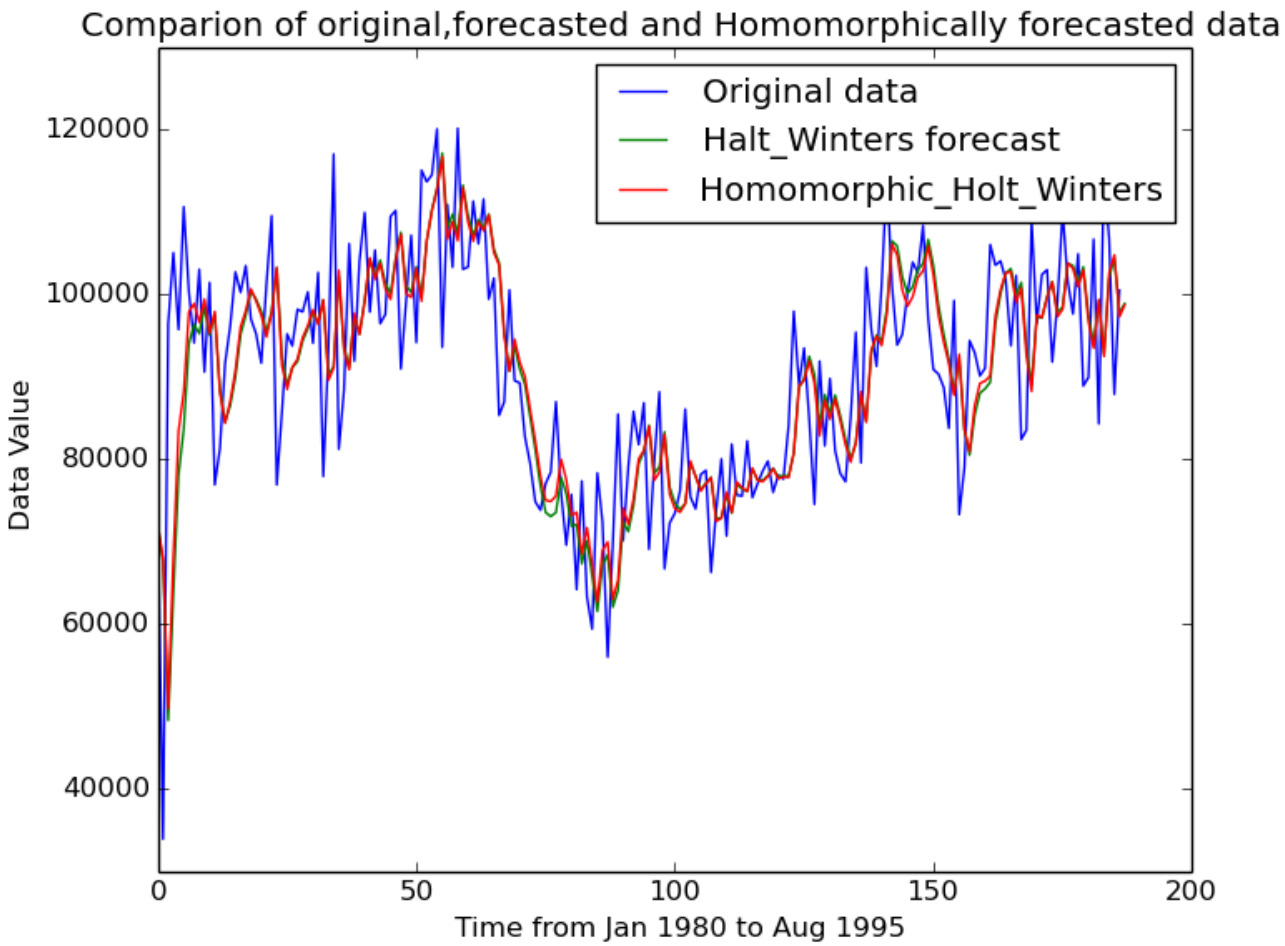

Figure 4.2: Double Exponential Forecasting

4.1.4 Triple Exponential Forecasting

Triple exponential smoothing takes into account seasonality changes as well as trend. In general seasonality is define as the data set that repeats the same pattern after every L periods like harmonic function. The term season is used to represent period of time before behavior begins to repeat itself. There are two types of seasonality : Addition and Multiplication seasonality. Triple exponential smoothing is also refereed as Holt-Winters model.

Additive Seasonality: The recursive equation of Holt-Winters additive seasonality model is given as follows:

$$\begin{aligned} y_t &= \alpha.(x_t - c_{t-L}) + (1-\alpha)(y_{t-1} + b_{t-1}) \\ b_t &= \beta.(y_t - y_{t-1}) + (1-\beta)b_{t-1} \\ c_t &= \gamma.(x_t - y_{t-1} - b_{t-1}) + (1-\gamma)c_{t-L} \end{aligned}$$

for t=1,2,...,T. and $0 \leq \alpha, \beta, \gamma \leq 1$ are forecasting parameter. The m period ahead forecast is given by F_{t+m}. Where F_{t+m} is

$$F_{t+m} = y_t + mb_t + c_{t-L+1+(m-1) \ mod \ L}$$

The initial parameter for forecast is taken as

$$\begin{aligned} y_0 &= x_0 \\ b_0 &= \tfrac{1}{L}(\tfrac{x_{L+1}-x_1}{L} + \tfrac{x_{L+2}-x_2}{L} + \cdots + \tfrac{x_{L+1}-x_1}{L}). \text{ and} \\ c_i &= \tfrac{\sum_{i=1}^{L} x_{L(j-1)+i}}{A_j} \quad \forall_i = 1, 2, \cdots, L \end{aligned}$$

Where

$$A_j = \frac{\sum_{i=1}^{L} x_{L(j-1)+i}}{A_j} \quad \forall_j = 1, 2, \cdots, N$$

In the above formula A_j represents the average value of x in the jth cycle and N represents the no of cycle present in data.

As mentioned above these equations cannot be used directly for forecasting. To use them for forecasting, firstly we have to convert them into linear form. From the above three equations it is clear that they are interdependent to each other, so we can not directly expand them to get a nice linear form. So here we have used a slight variation of these equations as mentioned below.

We use the following approximation

$$\begin{aligned} y_t^{'} &= \alpha.x_t + (1-\alpha).y_{t-1}^{'} \\ b_t &= \beta.(y_t^{'} - y_{t-1}^{'}) + (1-\beta).b_{t-1} \\ c_t^{'} &= \gamma.x_t + (1-\gamma).c_{t-L}^{'} \\ c_t &= c_t^{'} - \gamma.(y_{t-1}^{'} + b_{t-1}) \\ y_t &= y_t^{'} + (1-\alpha).b_{t-1} - \alpha.c_{t-L} \end{aligned}$$

Now we can find $y^{'}$, b, $c^{'}$, c and y from the above equations. To find $y^{'}$ and b we can use the same procedure as used in double exponential smoothing. To find $c^{'}$ we can use $b = G_2.x$ where G_2 is given by

$$G_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ (1-\gamma) & 0 & 0 & \gamma & 0 & 0 & 0 & \cdots & 0 \\ 0 & (1-\gamma) & 0 & 0 & \gamma & 0 & 0 & \cdots & 0 \\ 0 & 0 & (1-\gamma) & 0 & 0 & \gamma & 0 & \cdots & 0 \\ (1-\gamma)^2 & 0 & 0 & \gamma(1-\gamma) & 0 & 0 & \gamma & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \end{bmatrix}$$

The below figure is drawn for the same data as that of the above.

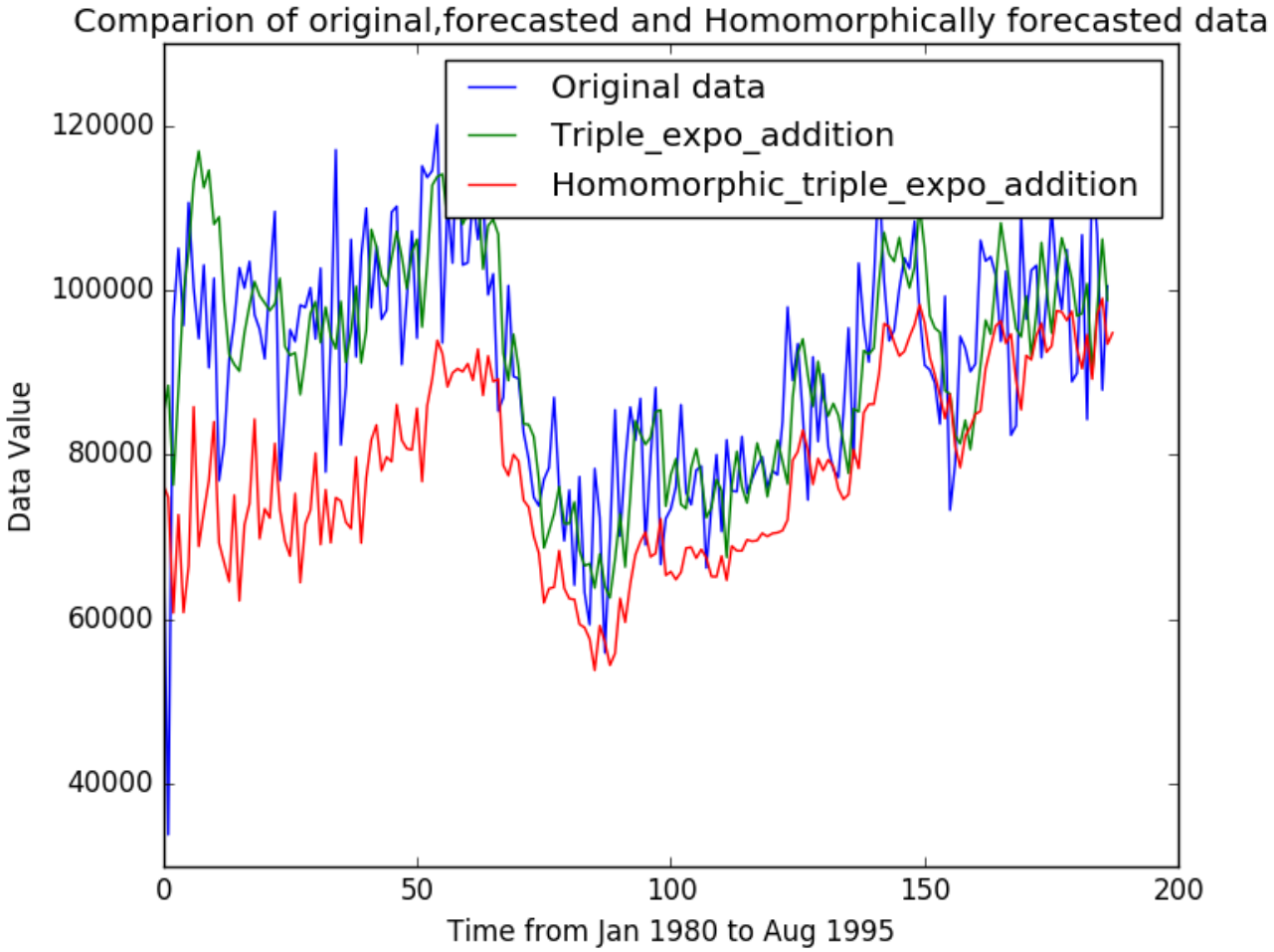

Figure 4.3: Additive Triple Exponential Forecasting

Multiplicative Seasonality: The recursive equation for Holt-Winters multiplicative seasonality model is given as follows:

$$\begin{aligned} y_t &= \alpha.(\tfrac{x_t}{c_{t-L}}) + (1-\alpha)(y_{t-1} + b_{t-1}) \\ b_t &= \beta.(y_t - y_{t-1}) + (1-\beta)b_{t-1} \\ c_t &= \gamma.(\tfrac{x_t}{y_{t-1}+b_{t-1}}) + (1-\gamma)c_{t-L} \end{aligned}$$

for t=1,2,...,T. and $0 \leq \alpha, \beta, \gamma \leq 1$ are forecasting parameter.
The m period ahead forecast is given by F_{t+m}. Where F_{t+m} is

$$F_{t+m} = (y_t + mb_t)c_{t-L+1+(m-1) \ mod \ L}$$

The initial parameter for forecast is taken as

$$\begin{aligned} y_0 &= x_0 \\ b_0 = \tfrac{1}{L}(\tfrac{x_{L+1}-x_1}{L} + \tfrac{x_{L+2}-x_2}{L} + \cdots + \tfrac{x_{L+1}-x_1}{L}). \text{ and} \\ c_i &= \frac{\sum_{i=1}^{L} x_{L(j-1)+i}}{A_j} \quad \forall_i = 1, 2, \cdots, L \end{aligned}$$

Where

$$A_j = \frac{\sum_{i=1}^{L} x_{L(j-1)+i}}{A_j} \quad \forall_j = 1, 2, \cdots, N$$

In the above formula A_j represents the average value of x in the jth cycle and N represents the no of cycle present in data.

As mentioned above these equations cannot be used directly for forecasting. To use them for forecasting, firstly we have to convert them into linear form. From the above three equations it is clear that they are interdependent to each other, so we can not directly expand them to get a nice linear form. So here we have used a slight variation of these equations as mentioned below.

We use the following approximation

$$\begin{aligned} y_t^{'} &= \alpha.x_t + (1-\alpha).y_{t-1}^{'} \\ b_t &= \beta.(y_t^{'} - y_{t-1}^{'}) + (1-\beta).b_{t-1} \\ c_t &= \gamma(\tfrac{y_t^{'}}{y_{t-1}^{'}+b_{t-1}}) + (1-\gamma).c_{t-L} \\ y_t &= \alpha.(\tfrac{x_t}{c_{t-L}}) + (1-\alpha)(y_{t-1} + b_{t-1}) \end{aligned}$$

Now we can find $y^{'}$, b, $c^{'}$, c and y from the above equations. To find $y^{'}$ and b we have used here Yu-Lai-Payor's Scheme because using Zhou's scheme we can not divide two encrypted data. From the modified equations above to find $c^{'}$ we can use $b = G_2.x$ where G_2 is given by

$$G_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ (1-\gamma) & 0 & 0 & \gamma & 0 & 0 & 0 & \cdots & 0 \\ 0 & (1-\gamma) & 0 & 0 & \gamma & 0 & 0 & \cdots & 0 \\ 0 & 0 & (1-\gamma) & 0 & 0 & \gamma & 0 & \cdots & 0 \\ (1-\gamma)^2 & 0 & 0 & \gamma(1-\gamma) & 0 & 0 & \gamma & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \end{bmatrix}$$

The below figure is drawn for the same data as that of the above.

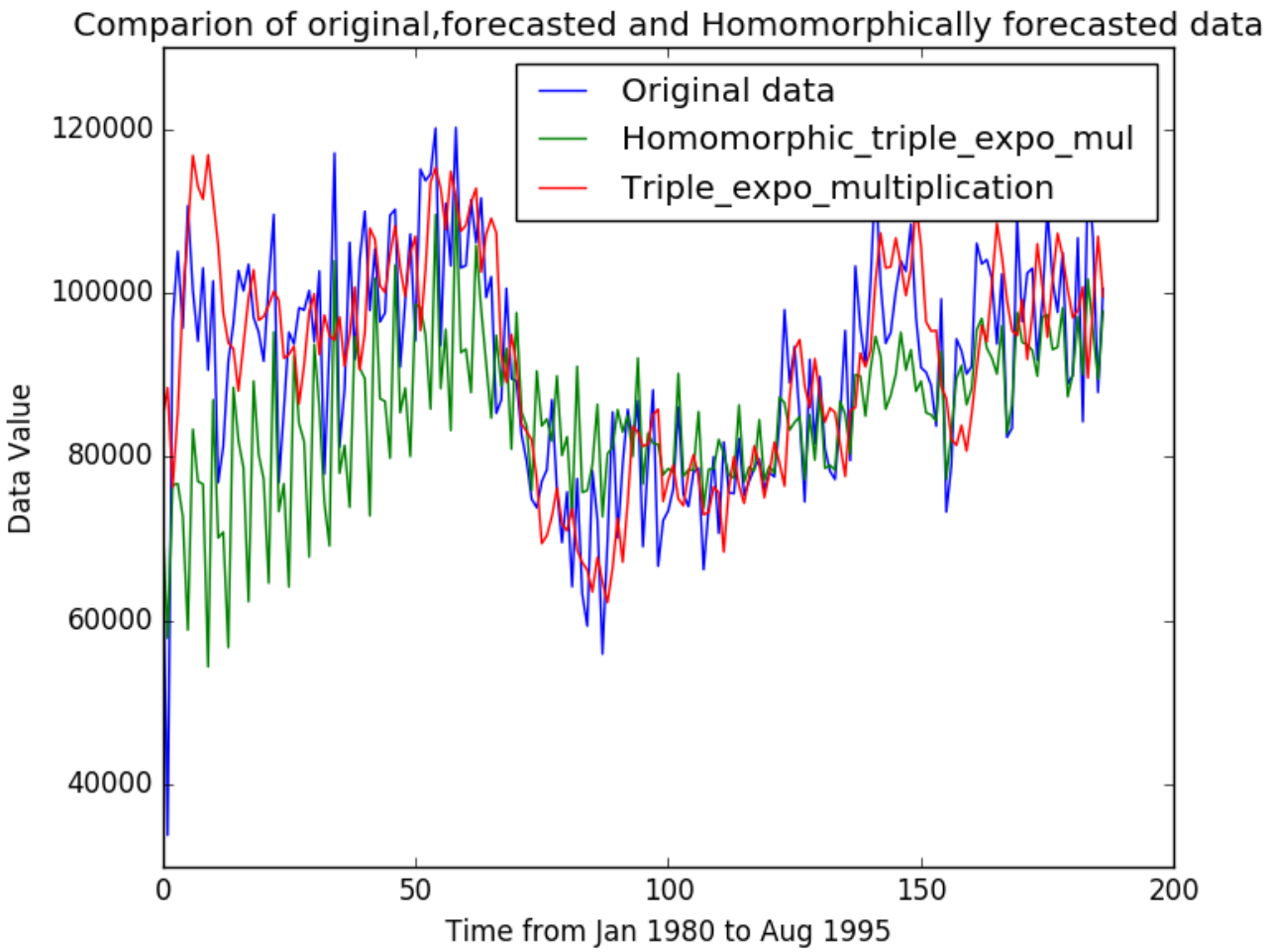

Figure 4.4: Multiplicative Triple Exponential Forecasting

Chapter 5

Conclusion and Future work

5.1 Conclusion

There are different approaches available for homomorphic operation as we have show in related work section. All of them discussed only two basic operations addition and multiplication, none of them discussed division operation. Yu-Lai-Payor's scheme is the one which can be used to perform division operation but have an error of ± 1 always.

Though we can use Yu-Lai-Payor's scheme for division operation but the result obtained is always in unencrypted form because of the fact that $\frac{wx_1+e_1}{wx_2+e_2} \approx \frac{x_1}{x_2}$. So for certain kind of confidential data we can not use this scheme too.

In our implementation we have used Yu-Lai-Payor's scheme for division operation. Using Yu-Lai-Payor's scheme gives as an division result in unencrypted form. So in future, if some new scheme will come for division operation then we can replace Yu-Lai-Payor's scheme for division operation and make homomorphically forecasting of multiplicative triple exponential forecasting more confidential.

5.2 Future work

Till now there are different schemes available for homomorphic encryption and performance of different schemes are different for different kind of data. Some of them may perform well for some particular data but may perform bad on other data. In future we will try to implement different schemes available and make a comparison between them. Some of them may not be feasible to implement, we will study that as well.

Using parallel computing also we can speed up algorithms. So we will try to implement different schemes available using parallel computing and study their performance in both cases. For some of the schemes it may be possible to enhance them we will study that as well.

There are different areas where homomorphic encryptions may be quite a important to use but not used there because of some constraints we will also try to study these areas.

References

[1] R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

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

[3] N.P. Smart and F. Vercauteren. Fully homomorphic simd operations. Cryptology ePrint Archive, Report 2011/133, 2011.

[4] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. Cryptology ePrint Archive, Report 2010/520, 2010.

[5] Michael Groves Peter Campbell and Dan Shepherd. Soliloquy: A cautionary tale. pre-sentation at etsi workshop, 2014.

[6] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Proceedings of the 29th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'10, pages 24–43, Berlin, Heidelberg, 2010. Springer-Verlag.

[7] Jean-Sebastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. Cryptology ePrint Archive, Report 2011/441, 2011.

[8] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[9] Jean-Sebastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. Cryptology ePrint Archive, Report 2011/441, 2011.

[10] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. pages 84–93, 2005.

[11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Proceedings of the 31st Annual Conference on Advances in Cryptology*, CRYPTO'11, pages 505–524, Berlin, Heidelberg, 2011. Springer-Verlag.

[12] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 97–106, Washington, DC, USA, 2011. IEEE Computer Society.

[13] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM.

[14] Craig Gentry, Amit Sahai, and Brent Waters. *Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based*, pages 75–92. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[15] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. *Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme*, pages 45–64. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[16] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2008.

[17] Hongchao Zhou and Gregory W. Wornell. Efficient homomorphic encryption on integer vectors and its applications. In *2014 Information Theory and Applications Workshop, ITA 2014, San Diego, CA, USA, February 9-14, 2014*, pages 1–9, 2014.

[18] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard). *SIAM J. Comput.*, 43(2):831–871, 2014.

[19] James Payor Angel Yu, Wai Lok Lai. Efficient integer vector homomorphic encryption.