Private Computation of Schulze Voting Method

A THESIS

FOR THE DEGREE OF

MASTER OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

BY

ANSHUL ANAND ICM2014501

UNDER THE SUPERVISION OF
Prof. SHEKHAR VERMA
IIIT-ALLAHABAD

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, ALLAHABAD
(A UNIVERSITY ESTABLISHED UNDER SEC.3 OF UGC ACT, 1956 VIDE NOTIFICATION NO. F.9-4/99-U.3 DATED 04.08.2000 OF THE GOVT. OF INDIA)

A CENTRE OF EXCELLENCE IN INFORMATION TECHNOLOGY ESTABLISHED BY GOVT. OF INDIA

CANDIDATE’S DECLARATION

I hereby declare that the work presented in this thesis entitled ”Private Computation of Schulze Voting Method”, submitted towards fulfillment of MASTER’S THESIS of M.Tech. (IT) with specialization in Information Technology at Indian Institute of Information Technology, Allahabad, is an authenticated record of our original work carried out under the guidance of Prof. Shekhar Verma. Due acknowledgements have been made in the text to all other material used. The project was done in full compliance with the requirements and constraints of the prescribed curriculum.

Date: / /
Place : Allahabad

ANSHUL ANAND (ICM2014501)

CERTIFICATE FROM SUPERVISOR

This is to certify that the statement made by the candidate is correct to the best of my knowledge and belief. The master's thesis titled "Private Computation of Schulze Voting Method" is a record of candidate's work carried out by him under my guidance and supervision. I do hereby recommend that it should be accepted in the fulfillment of the requirements of the MASTER'S THESIS at IIIT Allahabad.

Date: / /
Place : Allahabad

(Prof. Shekhar Verma)

Counter Signed By Dean (A)

CERTIFICATE OF APPROVAL

The forgoing thesis is hereby approved as a credible study in the field of Information Technology carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made opinion expressed or conclusion drawn therein but approve the thesis only for the purpose for which it is submitted.

Signatures of the Examiners Committee Members
(On final examination and approval of the thesis)

PLAGIARISM REPORT

The text in this thesis was found to be x percent plagiarized as reported by the xyz plagiarism checker.

Date: / /
Place : Allahabad

ANSHUL ANAND (ICM2014501)

Acknowledgements

The satisfaction that accompanies the successful completion of any project work would be impossible without the mention of the people who made it possible and whose constant guidance and encouragement crown all the efforts. This thesis was not only an endeavour but also an interesting learning experience for me and it bears the imprint of a number of people who directly or indirectly were a great source of help and constant encouragement.

I would like to express my sincere thanks to my mentor Prof. Shekhar Verma, for his continuous motivation and guidance. His valuable suggestions, comments and support were an immense help for me. I am grateful to him for taking out time from his busy schedule and being very supportive in guiding my work.

I would like to show my sincere appreciation towards Mr. Vijay Kumar Yadav, who helped me and guided me in understanding the various theoretical aspects of Fully Homomorphic Encryption and the underlying mathematics of the topic.

Special thanks to my parents and peers at IIIT-A for their efforts and cooperation. The interesting and informative discussions we had together greatly contributes to the completion of this work.

Anshul Anand

ABSTRACT

Users often outsource data to the cloud to get the result of an intended computation. Many at times the user would like to keep the data and the result of the delegated computation private due to its sensitive nature. Advancements in the fields of Machine Learning and Artificial Intelligence have made it even easier for service providers to extract private information of a person such as personal preferences, voting, etc. Hence there is a need to start computing on data privately wherever sensitive data, which might leak a piece of private information is involved. A significant breakthrough in realizing private computation has been Homomorphic Encryption which allows computation on Encrypted Data. Since the data and the final result are encrypted and can't be decrypted without the required key, the leakage of sensitive information is substantially contained. We will focus on the private computation of a real-life application involving sensitive information called Schulze Voting Method which uses computation of strongest paths in a directed weighted graph. In the following work, we will be making use of Levelled-BGV fully homomorphic encryption scheme to design and implement the private computation of the strongest path in Schulze method for voting.

Contents

List of Figures

Chapter 1

Introduction

Over the past few years it can be observed that there has been rise in the trend of delegating or outsourcing of digital to cloud platforms for computation. On the sidelines of this development, we can see an increasing concern about a user's privacy in the current digital age. Recent developments regarding the Aadhar Card, EU GDPR Regulations and Facebook Cambridge Analytical data leak, have once again brought user data privacy discussion into the limelight. Also, the developments in Machine Learning and AI fields have been able to derive relevant information about the preferences and behavior of users. Due to these developments and evolutions in algorithm extracting sensitive information from a collection of useful looking data is also now a possibility. Hence there is a dichotomy where the user or client wants to perform cloud-based or server-based computing due to computational constraints or other reasons but also keep its sensitive data and the result of the computation as private to prevent the breach of privacy or information which might have some adverse effect on him. For example - A scientific lab might want to outsource the computation of fingerprint matching but would not want to reveal the sensitive biometric information to the cloud service provider as the cloud service provider might not be a complete trusted party. There is the addition of a third dimension where apart from computation and privacy we also want absolute transparency to the whole calculation as in the case of Open Source Development where the purpose of the work and certain other information is out in the open.

Traditional encryption can't be used to perform private computation on encrypted data. HE allows us to compute on encrypted data. Hence the data sent is in the unintelligible form at the cloud's end, but it is still possible to perform computations on it. Until recently HE didn't have efficient schemes to compute addition and multiplication simultaneously. Hence they were limited on the amount of computation they could perform. It was only when Craig Gentry in his seminal work gave the first FHE scheme that work in this domain has started to pick up started to pick up. Since Gentry's work [1] new and faster FHE schemes have been coming up and very recently researchers have begun using it for computation of statistical measures on the data and more complex calculations like machine learning. So it is

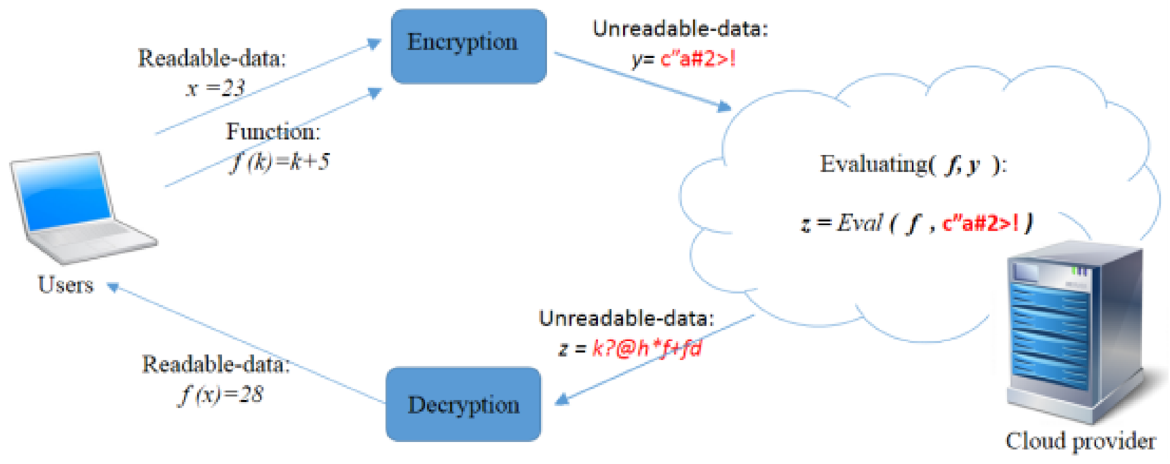


Figure 1.1: HE

possible now to compute complex operations like comparison and finding max and minimum of two numbers in an encrypted manner which forms building blocks for more significant computations. The avenues for FHE in data privacy are expanding each day as the amount of data and users opting for outsourcing computation increase.

A lot of problems and relations are being modeled as graphs especially social graphs which consist of data on smoking preferences, sexual preferences, religious inclinations, etc. which include sensitive data. The solution to graph problems like the shortest distance between the nodes etc. give the desired information out of these graphs. Hence there is a need to be able to compute the graph algorithms privately. In this work, we intend to provide algorithm and implementation for private computation of Widest Path Problem in the case of Schulze Voting Method which is a popular voting method used by several organizations including Wikimedia, Debian, Gentoo, and the KDE. In the Schulze voting method, the voting preferences of the electors are provided which might become sensitive if the cloud collides with the candidates and they might against people who have not given them a voting choice. In this work, we will be using the levelled-BGV FHE scheme to perform the private computation.

1.1 Motivation

We can see that graph modelling of data is finding an increased usage. Since the data involved can be sensitive we need to have private computation methods for graph algorithms. Our motivation was to provide a solution to a graph problem with a real world application. The motivation behind private computation of Schulze Voting method is that it is a preferential voting method where users have to provide their order of preference. We know that revelation of preference of voting has been known to be dangerous as candidates later intimidate the voters or go against people who have not voted for them [2]. Also if we look in literature, then we can see that currently homomorphically encryption has only been used to tally conventional election where voting is first past the post and the result is by simple addition of votes to check who has the most number of votes like in the work presented by Hirt et al. [3], the work provided Sandler et. al [4] also discusses about using homomorphic counters to find the winner of the election. But till now there has been no work showing the construction and implementation for Schulze voting method i.e a condorcet voting method as it requires computation of complex operations like comparison, finding maximum of two numbers, finding minimum of two numbers, and binary addition etc. in a private manner. Also it requires designing and constructing a secure algorithm for computation of strongest path in directed weighted graphs with encrypted weights and has been an unexplored area till now. But since, Schulze voting is being used by a lot of organizations across the globe and there might be privacy concerns of revealing voting preferences of for example a department or a team etc. hence we are motivated to provide a private computing method for Schulze Voting.

1.2 Problem definition

- To design, develop and to provide an open source implementation for the private computation of comparison, binary carry adder and max and min functions using HElib which is a software library implementing the levelled-BGV FHE scheme.
- To provide an algorithm for private computation of Schulze Voting Method's ballot tallying using FHE and to estimate the time taken by our algorithm via the implementation.
- Hence we aim to solve the privacy of voting preferences in Schulze voting method using levelled-fully homomorphic encryption.

1.3 Objectives

- To understand the various Privacy issues pertaining to the digital age.
- To understand the working of the FHE.
- To understand the fundamentals and mathematical background of cryptography.

- To learn how to use software HElib such as HElib and its parameterization.
- To propose an algorithm for private computation of the widest path problem with application to Schulze method of vote tallying.
- To estimate the time taken by the algorithm to for various numbers of candidates.

1.4 Contributions

Our approach leverages the principles of using levelled-FHE scheme to implement computational circuits so that the strongest paths in a weighted graph can be computed in a private fashion. This is different than privately computing traditional voting schemes for election as they employ first past the post system which requires simple incrementation of the vote count and we need to just check the maximum number of votes in the end. Contrary to that Schulze voting method is a condorcet voting method and hence requires computation of strongest paths in a weighted graph in order to find out the winner. In summary, the major contributions of our work are as follow:

- We first describe the private computation for deciding the winner in the Schulze Voting method which was suggested by Markus Schulze [5]. We fully design and provide an algorithm to homomorphically tally the Schulze ballot by computing the strongest path in the preferences weighted graph and evaluate the winner.
- In order to realise the algorithm we implement the encrypted evaluation of comparison, minimum and maximum operations using the HElib library.We have implemented the aforementioned primitives using HElib which employs the BGV Leveled FHE Scheme and provide an open-source implementation available at [1] to the source code to the research community.
- We evaluate our algorithm using an HElib implementation for various parameters like time and number of levels in the modulus chain and also evaluate the size of the public key and secret key used for encryption and decryption for different number of levels in the modulus chain.

1.5 Overview

The rest of the work has been set up in the following manner, in Chapter 1 we give the basic introduction to the schulze voting method, HE and what we plan to achieve.

[1]https://github.com/anshulxanand/TARE

In Chapter 2 we give an overveiew of the related work that has taken place in the domain of privacy using HE. In Chapter 3 we define our problem statement and our motivation to do the work and our objectives. In Chapter 4 we give the pre-requisite mathematical and cryptographic background required in order to understand the proposed work. In Chapter 5 and 6 we discuss HE and Levelled-BGV FHE in detail. In Chapter 7 we discuss the schulze voting method in detail with examples. In Chapter 8 we discuss our proposed approach and the results and evaluation. In Chapter 9 we discuss our software and hardware requirements for the thesis. In Chapter 10 we conclude our work and in Chapter 11 we discuss the various future directions for our work.

Chapter 2

Related Work

Post the first FHE scheme proposed by Gentry [1] there has a been a significant growth in the field of HE. Gentry's scheme [1] is based on ideal lattices. Since then, more and more researchers have started preferring Lattices for the construction of their cryptosystems. Schemes like Smart and Vercauteren's work [6] try to improve upon Gentry's scheme [1] by reducing the ciphertext size and the key size. Van Dijk et al. [7] introduced a FHE scheme over integers using only elementary modular arithmetic based on the hardness of the Approximate-GCD problem i.e, given a list of integers that are near-multiples of a hidden integer, output that hidden integer. Brakerski and Vaikuntanathan gave another scheme in [8] whose hardness is based solely on the LWE assumption and the security of the scheme is based on ”shortest vector problem” on lattices. Brakerski et al. [9] introduced a FHE scheme which doesn't require the need for bootstrapping and is a levelled scheme capable of evaluating polynomial of arbitrary sizes, this scheme is popularly known as the BGV scheme based on the author's name. The [9] scheme has the option to use both the LWE and RLWE assumptions as two modes. Lopez et. al present construction of a multikey FHE scheme based on the NTRU cryptosystem in [10]. Gentry et. al in [11] also popularly known as the GSW scheme, the author's present an FHE scheme based on the LWE assumption and introduce the ”approximate eigenvector method” for building a FHE scheme. More recently, Chillotti et. al in [12] make use of the mathematical structure of Torus to construct a fast FHE scheme.

Implementations of many of the schemes mentioned above are available in literature but only a handful are available in the public domain to the researchers to make use of. HElib [13] is one of the most popular and widely used FHE libary currently. The implementation of the library focuses on the Gentry-Halevi-Smart optimizations and also the Smart-Vercauteren ciphertext packing techniques for improving the performance. Halevi and Shoup [14] describe in a paper the algorithms employed on the implementation, as well as some considerations to have in mind depending on the hardware used. Other popular libaries are Microsoft's *SEAL* and Ducas et. al's *FHEW* library [15].

As an outcome of these developments there have been some works which make use of comparison operations or binary arithmetic on homomorphically encrypted data to perform desired evaluations in a similar manner as our work. In Homomorphic Computation of Edit Distance Cheon et al. [16] provide private computation for Edit Distance for matching short DNA sequences. In Low Depth Circuits for Efficient Homomorphic Sorting Cetin et al. [17] provide a method to sort numbers using Direct Sort Scheme. In [18] Kocabas et. al utilize HE to privately perform operations related to health like avgerage heartrate, LQTS etc. over cloud. In [19] Chatterjee et. al describe methods for translating basic operations in non-encrypted domain to encrypted domain over cloud. In our work, we shall present a novel application and implementation for Schulze voting using HElib and Levelled-BGV FHE.

Chapter 3

Background

There are a number of building blocks which are needed for our understanding of cryptosystems and their security and working aspects. Number theory and abstract algebra concepts are discussed to establish a proper mathematical background for the cryptographic scheme used. HE with its mathematical and cryptographical preliminaries has been discussed thoroughly and examples have also been discussed for clarity.

3.1 Abstract Algebra

In this section we will be discussing

3.1.1 Groups

A **group** G, denoted by $\{\mathrm{G}, \cdot\}$ is a set of elements with a binary operation, which is denoted by the operator $\cdot$. This operator is generic, and can refer not only to multiplication, but also to addition or some other mathematical operation.
A group has the following properties which hold for each of the pairs a and $b \in G$:

(Closure: If $a \in G$ and $b \in G$, then $a \cdot b \in G$.

Identity element: $\exists$ e in G such that $a \cdot e = e \cdot a = a$ for all a in G.

Associative: $a \cdot (b \cdot c) = (a \cdot b) \cdot c \; \forall \; a, b, c \in G$.

Inverse element: $\exists \; a' \in G \; \forall \; a \in G$ s.t. $a \cdot a' = a' \cdot a = e$.

An **abelian** group is such a group whch apart from following the above mentioned group properties also follows the property of commutatitivity and is a commutative group.

Commutative Property $a \cdot b = b \cdot a \; \forall \; a, b \in G$.

In the case of group operation being addition, we have 0 as identity element; $-a$ as the inverse element of a;And $\forall$ a,b $\in G$ $a-b=a+(-b)$ defines an operation of subtraction among the group elements.

3.1.2 Rings

A *ring R*, which is sometimes denoted by $\{R,+,\times\}$, is a non empty set of elements with two binary operations, usually written as *addition* and *multiplication*, s.t. $\forall$ a, b, $c \in R$, some properties always hold int form of axioms.

A ring R follows all the properties of an abelian group in the case of addition operation; that is, closure, existence of identity element, associativity, existence of inverse element and commutatitivity. The rest of the axioms that hold due to the second operator of a ring are mentioned as follows:

Closure: $\forall$ a and $b \in R$, $a \times b$ is a member of R.

Associativity: $a \times (b \times c) = (a \times b) \times c$ $\forall$ a, b, $c \in R$.

Distributivity of multiplication over addition: $a \times (b+c) = (a \times b) + (a \times c)$ $\forall$ a, b, $c \in R$.
$(a+b) \times c = (a \times c) + (b \times c)$ $\forall$ a, b, c in R.

Hence in a ring R we can perform addition, subtraction $[a-b=a+(-b)]$, and multiplication amongst the elements without leaving the set. A **commutative** ring is one in which apart from the above mentioned properties the property of commutativity also holds:

Commutativity of multiplication: $(a \times b) = (b \times a)$ $\forall$ a, $b \in R$.

Commutative ring can be converted into **integral domain** if it additionally follows some more axioms. These axioms are:

Multiplicative identity: $\exists$ an element **1** in R such that
$(a \times 1) = (1 \times a) = a$ $\forall$ $a \in R$.

No zero divisors: If a, b in R and $(a \times b) = 0$, then either $a = 0$ or $b = 0$.

Polynomial Ring

Introduction to Mathematical Cryptography [20], describes a polynomial ring as follows.

The collection of all polynomials with coefficients taken from $\mathbb{Z}$ forms a ring under the usual operations of polynomial addition and multiplication. This ring is denoted by $\mathbb{Z}[x]$. Thus we write $\mathbb{Z}[x]$ = a_0 + a_1x + a_2x^2 + $\ldots$+ a_nx^n : $n \geq 0$ and a_0, $a_1, \ldots,$ a$_n \in \mathbb{Z}$.

More generally, if R is any ring, we can form a ring of polynomials whose coefficients are taken from the ring R. For example, the ring R might be $\mathbb{Z}$/q$\mathbb{Z}$ or a finite field F_p. We denote these general polynomial rings as $R[x]$.

Convolution polynomial rings

Introduction to Mathemtatical Cryptography [20] defines convolution polynomial ring as follows:

Fix a positive integer N. The ring of convolution polynomials (of rank N) is the quotient ring $R = \mathbb{Z}[x]/(x^N - 1)$. Similarly, the ring of convolution polynomials (modulo q) is the quotient ring $R_q = (\mathbb{Z}_q)[x]/(x^N - 1)$. We know that every element of R or R$_q$ has a unique representative of the form $a_0 + a_1x + a_2x^2 + \ldots + a_{N-1}x^{N-1}$ with the coefficients in $\mathbb{Z}$ or $\mathbb{Z}$/q$\mathbb{Z}$, respectively. We observe that it is easier to do computations in the rings R and R$_q$ than it is in more general polynomial quotient rings, because the polynomial $x^N - 1$ has such a simple form. The point is that when we mod out by $x^N - 1$, we are simply requiring x^N to equal 1. So any time x^N appears, we replace it by 1.
Example - Let us consider the quotient ring $\mathbb{Z}_q[x]/(x^N + 1)$
Here $q = 17$, $N = 4$ That is after addition and multiplication we will replace X^4 by -1.

$$\begin{aligned}
a(x) &= 15 + 2x + 4x^2 + 7x^3 \in \mathbb{Z}_17[x]/x^4 + 1 \\
b(x) &= 8 + 9x + 3x^2 + 4x^3 \in \mathbb{Z}_17[x]/x^4 + 1 \\
a(x) + b(x) &= 6 + 11x + 7x^2 + 11x^3 mod(17, x^4 + 1) \\
a(x).b(x) &= 120 + 151x + 95x^2 + 158x^3 + 83x^4 + 37x^5 + 28x^6 \\
&\equiv 37 + 114x + 67x^2 + 158x^3 mod(x^4 + 1) \\
&\equiv 3 + 12x + 16x^2 + 5x^3 mod(17, x^4 + 1)
\end{aligned}$$

3.1.3 Fields

A field is a set F with two composition laws $+$ and $\times$ such that

- $(F, +)$ is a commutative group;
- $(F^*, \times)$, where $F^* = F\backslash\{0\}$, is a commutative group.
- Distribution of $\times$ over $+$ holds.

Thus, a field is a non-zero commutative ring such that every non-zero element has an inverse.

3.1.4 Homomorphisms

As we have seen that due to the closure property of a group after performing operations on the group elements the result is also part of the group. We know that only one operation is allowed in a group like addition or multiplication. There are also groups which have different set of elements and they consist of different operators. A *homomorphism* consists of the construction of a function that *translates* elements from one group to another group with the same properties.

Group Homomorphism

A homomorphism as defined by Beachy and Blair [21], is as follows: Let G and H be groups with operators * and *' repsectively, and let $\phi : G \to H$ be a function. Then ϕ is said to be a **group homomorphism** if

$$\phi(a * b) = \phi(a) *' \phi(b) \tag{3.1}$$

for all a, b in G.

Let us take an example as given on

Where there are two sets: GL(n, $\mathbb{R}$), the set of all invertible $n \times n$ matrices over $\mathbb{R}$ and the second set be $\mathbb{R}$ itself where $\mathbb{R}$ is the set of real numbers. We can see that there exists a map det() where det(A) computes determinant of matrices from GL(n, $\mathbb{R}$) $\to \mathbb{R}$.

$$\begin{aligned} \phi : GL(n, \mathbb{R}) &\to \mathbb{R} \\ \phi : det(M), M &\in GL(n, \mathbb{R}) \\ \phi(A) \times \phi(B) &= \phi(AB) \\ \det(A) \times \det(B) &= det(AB) \end{aligned}$$

From this example, we see that group homomorphisms are closely related to group structures. Thus, group homomorphisms are very important in studying structural properties of groups.

We say that f is an **isomorphism** if the group homomorphism $f : G \to H$ has an inverse.

Ring Homomorphism

We also have homomorphism for rings in a similar way as for group, the basic difference is that instead of one operator we have two operations for a ring. Let us take a look at the definition as descriped by [22]:

Let R and S be rings with addition and multiplication. The map $\phi : R \to S$ is a homomorphism if:

1. ϕ is a group homomorphism on the additive groups $(R,+)$ and $(S,+)$.
2. $\phi(xy) = \phi(x)\phi(y)\ \forall x,y \in R$.

We can also note the following as is presented in [23]:

A ring homomorphism which is a bijection (one-one and onto) is called a **ring isomorphism**

If $\phi : R \to S$ is such an isomorphism, we call the rings R and S isomorphic and write $R \cong S$.

- Isomorphic rings have all their ring-theoretic properties identical. One such ring can be regarded as ”the same” as the other.
- The inverse map of the bijection f is also a ring homomorphism.

3.2 Chinese Remainder Theorem

William Stein describes the Chinese Remainder theorem in his book ”Algebraic Number Theory, a Computational Approach” [24] as follows:

The classical CRT asserts that if $n_1,\ldots,n_r$ are integers that are coprime in pairs, and $a_1,\ldots,a_r$ are integers, then there exists an integer a such that $a \equiv a_i \pmod{n_i}$ for each $i = 1,\ldots,r$. Here “coprime in pairs” means that $\gcd(n_i,n_j) = 1$ whenever $i \neq j$; it does *not* mean that $\gcd(n_1,\ldots,n_r) = 1$, though it implies this. In terms of rings, CRT asserts that the natural map

$$\mathbb{Z}/(n_1 \cdots n_r)\mathbb{Z} \to (\mathbb{Z}/n_1\mathbb{Z}) \otimes \cdots \otimes (\mathbb{Z}/n_r\mathbb{Z}) \tag{3.2}$$

that sends $a \in Z$ to its reduction modulo each n_i, is an isomorphism.

Let us take an example and see how this isomorphism by chinese remainder theorem makes computation easy for us. Let us consider the isomorphism map $f : X \to X_p \times X_q$

- $N = p \times q$, $\gcd(p,q) = 1$. Then $\mathbb{Z}_N \cong \mathbb{Z}_p \times \mathbb{Z}_q$ and $\mathbb{Z}_N^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^*$
- $f : X \to X_p \times X_q$
- $f(X) = ([X mod p],[X mod q])$
- Then f is an isomorphism $\mathbb{Z}_N \to \mathbb{Z}_p \times \mathbb{Z}_q$ as well as an isomorphism $\mathbb{Z}_N^* \times \mathbb{Z}_p^* \times \mathbb{Z}_q^*$.

$$\begin{aligned}15 &= 5 \times 3\\ p &= 5, q = 3\\ Z_{15}^{*} &= 1,2,4,7,8,11,13,14\\ 1 \leftrightarrow (1,1), 2 &\leftrightarrow (2,2),\\ 4 \leftrightarrow (4,1), 7 &\leftrightarrow (2,1),\\ 8 \leftrightarrow (3,2), 11 &\leftrightarrow (1,2),\\ 13 \leftrightarrow (3,1), 14 &\leftrightarrow (4,2)\\ [11^2 mod 15] &\leftrightarrow (1,2)^2\\ (1^2 mod 5, 2^2 mod 3) &= (1,1) \leftrightarrow 1\end{aligned}$$

In the above example $X \leftrightarrow (X_p, X_q)$ means $f(X)$ that is changing its representation $([X mod p],[X mod q])$. We also note from this example that the same operations can be performed after translating the number into a a newer representation with shorter numbers by virtue of isomorphism. One very important and popular application of chinese remainder theorem (CRT) is speeding up the decryption of RSA algorithm.

3.3 Cyclotomic Polynomial

If n is a positive integer, an n^{th} **root of unity** is a complex number ζ such that $\zeta^n = 1$. Let n be a positive integer, let $\omega = e^{\frac{2\pi i}{n}}$, and let $\zeta = \omega^k$ be an n^{th} root of unity. Then ζ is a primitive n^{th} root of unity if and only if $\gcd(k,n) = 1$.

Cyclotomic Polynomial: The n^{th} cyclotomic polynomial is the monic polynomial $\Phi_n(x)$ whose roots are exactly the primitive n^{th} roots of unity; that is,

$$\Phi_n(X) = \prod_{\substack{\gcd(k,n)=1\\ 1\le k\le n}} \left(X - \zeta^k\right).$$

The first ten cyclotomic polynomials are shown as below:

$$\begin{aligned}
\Phi_1(x) &= x - 1\\
\Phi_2(x) &= x + 1\\
\Phi_3(x) &= x^2 + x + 1\\
\Phi_4(x) &= x^2 + 1\\
\Phi_5(x) &= x^4 + x^3 + x^2 + x + 1\\
\Phi_6(x) &= x^2 - x + 1\\
\Phi_7(x) &= x^6 + x^5 + x^4 + x^3 + x^2 + x + 1\\
\Phi_8(x) &= x^4 + 1\\
\Phi_9(x) &= x^6 + x^3 + 1\\
\Phi_{10}(x) &= x^4 - x^3 + x^2 - x + 1
\end{aligned}$$

Figure 3.1: First ten cyclotomic polynomials

Factorization of $\Phi_d(x)$ Modulo p:
Let p be a prime and let d be a divisor of $p-1$. Then

$$\Phi_d(x) = \prod_{\zeta \in O(d)} \left(X - \zeta^k\right).$$

Here $O(d)$ denotes the set of elements of $\mathbb{Z}_p^*$ of order d.

3.3.1 Chinese Remainder Theorem on Cyclotomic Polynomials

The cyclotomic polynomial can be factorized into distinct l irreducible polynomials and forms the natural isomorphism as shown below.

Φ_m(x) = $\prod_{i=1}^{l} F_i(x)$ mod p

Hence there can be an isomorphism from the quotient ring and we can be able to change the representation of the same quotient ring into a simplified version. $\mathbb{Z}_p[x]/\Phi_m \cong \mathbb{Z}_p[x]/F_1(x) \otimes \dots \mathbb{Z}_p[x]/F_l(x) \cong F_{p^d} \otimes \dots F_{p^d}$ (l times)
No. of slots = l
$l = \Phi(m)/\text{order}(m,p)$ where $\text{order}(m,p) = \alpha$ s.t. $p^{\alpha} = 1$ mod m

- $m = 8, p = 17$
  $\Phi_8 = x^4 + 1 = (x-2)(x-2^3)(x-2^5)(x-2^7)$ mod 17
  d = $deg(F_i(x))$ = $\phi(m)$/l
- So each slot holds a polynomial of degree 0 polynomial modulo 17
  $1 + x + 7x^2 + 12x^3 \Leftrightarrow [8, 5, 16, 9]$

- $1 + x + 7x^2 + 12x^3 \cong 8 \bmod (17, x - 2)$
  $\cong 5 \bmod (17, x - 2^3)$
  $\cong 16 \bmod (17, x - 2^5)$
  $\cong 9 \bmod (17, x - 2^7)$

Isomorphism

As we saw in the case of traditional chinese remainder, we also have an induced isomorphism in the case of chinese remainder theorem on cyclotomic polynomial. Hence the operations can be performed on the translated representation of the quotient ring.

We can see an example as shown below-

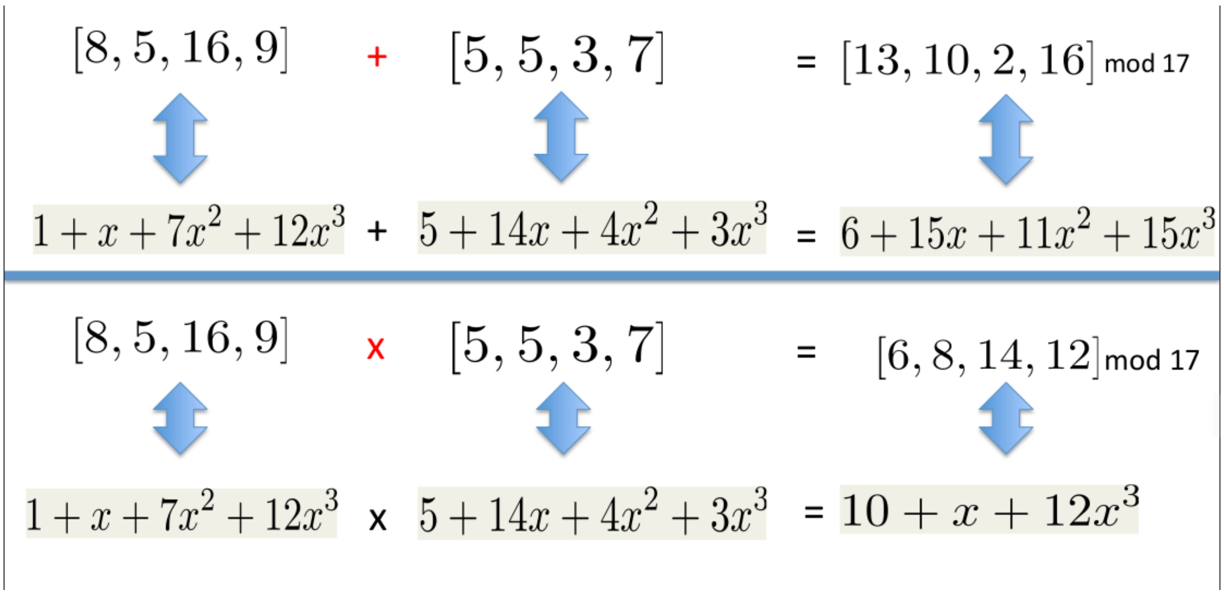

$[8,5,16,9]$ + $[5,5,3,7]$ = $[13,10,2,16]$ mod 17
$1+x+7x^2+12x^3$ + $5+14x+4x^2+3x^3$ = $6+15x+11x^2+15x^3$
$[8,5,16,9]$ x $[5,5,3,7]$ = $[6,8,14,12]$ mod 17
$1+x+7x^2+12x^3$ x $5+14x+4x^2+3x^3$ = $10+x+12x^3$

Figure 3.2: Example of isomorphism in quotient rings

3.4 Ring-LWE

3.4.1 Learning With Error

The Learning with Errors (LWE) problem was introduced in [25], in recent years it has become one of the most popular and useful concepts for creating useful cryptographic constructions. Learning with errors main claim to fame is it has been found to be as hard as worst-case lattice problems, hence all the cryptographic constructions using the learning with error problem as their basis are secure under the worst case hardness assumption of lattice problems.

LWE: The LWE problem asks to recover a secret $s \in \mathbb{Z}_q^n$ given a sequence of 'approximate' random linear equations on s. An example for the input of equations taken from [26] is:

$$\begin{aligned}
6s_1 + 7s_2 + 16s_3 + 2s_4 &\approx 3mod17\\
3s_1 + 6s_2 + 4s_3 + 5s_4 &\approx 16mod17\\
10s_1 + 4s_2 + 12s_3 + 16s_4 &\approx 12mod17\\
6s_1 + 10s_2 + 13s_3 + 1s_4 &\approx 3mod17\\
&\vdots\\
&\vdots\\
9s_1 + 5s_2 + 9s_3 + 6s_4 &\approx 9mod17\\
14s_1 + 15s_2 + 5s_3 + 2s_4 &\approx 8mod17
\end{aligned}$$

There is an additive error (say, ± 1) in the equations and the equations are approximately correct. We have to find the solution for s (The answer in this case is $s = (0, 13, 9, 11)$.)

Let us define the problem more precisely.

- Fix a size parameter $n \geq 1$, a modulus $q \geq 2$, and an 'error' probability distribution χ on $\mathbb{Z}_q$.
- Let $A_{s,\chi}$ on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ be the probability distribution obtained by choosing a vector a $\in Z_q^n$ uniformly at random.
- Choose $e \in Z_q$ according to χ, and outputting $(a, \langle a, s\ \rangle + e)$, where additions are performed in $\mathbb{Z}_q$, i.e., modulo q.
- We say that an algorithm solves LWE with modulus q and error distribution χ if, for any $s \in \mathbb{Z}_n\ q$, given an arbitrary number of independent samples from $A_{s,\chi}$ it outputs s (with high probability).

The error distribution χ is chosen to be a normal distribution rounded to the nearest integer (and reduced modulo q) of standard deviation αq where $\alpha > 0$ is typically taken to be 1/poly(n).

Ring-LWE

Let us start by giving a slightly more convenient, yet equivalent, definition of the ring-LWE problem. Let n be a power of two, and let q be a prime modulus satisfying $q = 1$ mod $2n$. Define R_q as the ring $\mathbb{Z}_q$[x]/($x^n + 1$) containing all polynomials over the field $\mathbb{Z}_q$ in which x^n is identified with -1.

In ring-LWE we are given samples of the form $(a, b = as + e) \in R_q \times R_q$ where $s \in R_q$ is a fixed secret, $a \in R_q$ is chosen uniformly, and e is an error term chosen

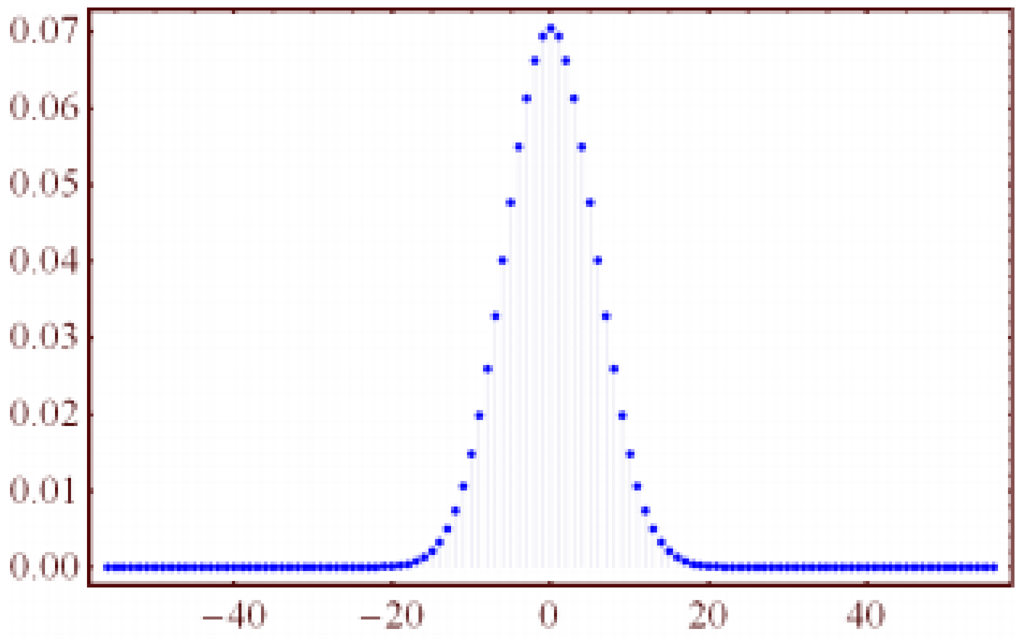


Figure 3.3: The error distribution with $q = 113$ and $\alpha = 0.05$

independently from some error distribution over R_q. The most natural choice of error distribution is to choose the coordinates of the polynomial e to be i.i.d. normal, i.e., a spherical Gaussian distribution. Our goal is to recover the secret s from these samples (for all s, with high probability).

Chapter 4

Homomorphic Encryption

Homomorphic is a word of greek etymology and it means ”of the same form or shape”. Homomorphism is used in a lot of different areas, it is used in abstract algebra which forms the foundation of its use in modern cryptography.

The concept of HE allows us to perform private computations. We are able to perform operations like addition and multiplication on encrypted data without decrypting the data or sharing the secret key used for encryption.

In the traditional non-HE, in order to perform any sort of operation on the ciphertext the entity performing computations must have the secret key in order to decrypt the ciphertext, perform the operations and recrypt it again. But if the entity in question is an untrusted one then its prudent to share the secret key with it.

However HE removes the need to share the secret key, specially with untrusted third parties. And the computations can be performed without decrypting the ciphertext and recrypting it again.

Given the property of HE to compute on encrypted data without having to decrypt it, HE has lots of potential application as mentioned by Lange [27].

These applications can be in the areas of: private information retrieval, e-voting, e-cash, (encrypted databases), and cloud computing.

4.1 Introduction

According to Pötzelsberger [28], *homomorphic cryptosystem* has an encryption function which forms an homomorphism and preserves the group operations performed on the ciphertext.

Addition or multiplication are usually the group operations chosen for group operations in this kind of cryptosystem. A HE is additive if:

$$\mathcal{H}(x+y) = \mathcal{H}(x) \otimes \mathcal{H}(y), \tag{4.1}$$

Where

- $\mathcal{H}$ denotes an encryption function.

- $\otimes$ denotes an operation depending on the used cipher
- x and y are plaintext messages.

A HE is multiplicative if:

$$\mathcal{H}(x \cdot y) = \mathcal{H}(x) \otimes \mathcal{H}(y), \tag{4.2}$$

where again $\mathcal{H}$ denotes an encryption function, $\otimes$ denotes the operation depending on the used cipher and x and y are plaintext messages.

Hence we can see that the desired operation is preserved over the plaintexts x and y without the need to decrypt the message and perform it.

4.1.1 Types of Homomorphic Encryption

HE schemes can be classified into two groups based on their definition, these groups are partially homomorphic and FHE.

- **Partially homomorphic encryption** are HE schemes defined over a group. They support only one operation either addition or multiplication. Example of additive homomorphic schemes are pallier cryptosystem [29] and Goldwasser-Micali [30] and for multiplicative homomorphic schemes is the RSA cryptosystem [31].
- **Fully homomorphic encryption** these schemes support two operations both multpilication and addition and are defined over rings. The *somewhat homomorphic encryption* schemes are similar to the fully HE schemes, however, the number of times an operation can be performed is limited and they also support two operations both multiplication and addition.

Pötzelsberger [28] points out that, somewhat HEs form the building blocks and even though they perform operations a limited number of times they provide much more efficiency and shorter ciphertexts than fully homomorphic cryptosystems.

4.1.2 Explaining Homomorphic Encryption

The concept of HE is explained in a simple way Craig Gentry [1] by using a simple jewelry making analogy. Imagine that Alice has a jewelry store and wants her workers to assemble raw materials (gold, silver, precious gems etc.) into final products like bangles, rings, necklaces etc. Since these raw materials are expensive so Alice want to keep them safe and ensure that the workers don't run away with the raw materials. So Alice restricts the access to the raw materials given to the workers. She does so by putting the materials inside a locked glove compartments attached to transparent boxes.

So the workers can work with the materials inside the box using the gloves attached to them.

Though workers are not able to take any material out of the boxes since they are locked and the workers don't have the keys but the workers can add additional things like soldering iron etc. to help make the final product.

Alice can then recover the products from the box using her key, once the products are finished. This analogy has a flaw and is not perfect as the workers can see what is inside the box, but it is still a usable one as the workers are not able to take the materials with them.

Let us consider a toy example as proposed by Hayes [32].

It is shown as follows: consider that the plaintext consists of integers;

- Let the plaintext be x
- C = Enc(x) = 2 $\times$ x
- Dec(C) = $C/2$

We will see that using this simple cryptosystem we can perform addition and modified multiplication on the encrypted data. Let us encrypt to plaintexts x and y and perform addition on them and then decrypt them to see the result. C_x + C_y = $2x+2y = 2(x+y)$. Hence it has the same effect as adding the original two plaintexts x and y in encrypted form as upon decryption the 2 cancels out and we have $(x+y)$

We have to modify the multiplication computation for the cryptosystem in order for it to work. We define the product of the ciphertexts as $(\frac{1}{2})C_xC_y$, while the plaintexts are multiplied by the regular formula xy. Let us consider $x = 5$ and $y = 7$. Upon performing $x \times y$, we will get the result of multiplication as 35. The encryption of both $x = 5$ and $y = 7$ will follow the rule mentioned above.

$$C_x = 5 \times 2 = 10, \tag{4.3}$$

$$C_y = 7 \times 2 = 14. \tag{4.4}$$

Upon performing the multiplication we get $(\frac{1}{2})10 \times 14 = 70$, for decryting this ciphertext we divide it by 2 and we get: $\frac{70}{2} = 35$. Hence we can see that we have got the correct result of multiplication.

We can see that the above cryptosystem satisfies all the criteria to be called *fully homomorphic*, as it supports both the operations addition and multiplication. But we can see that clearly, this cryptosystem is not a secure one as the anyone can decrypt the ciphertext. This is also the reason why we dont rely on obscurity of the cryptosystem to justify its security and security of the cryptosystem is solely based on the possession of the secret.

Let us see an example of a cryptosystem with inherent homomorphic Stuntz [33]. Let us consider the classical cipher ROT13 or the caesar's cipher which is quiet popular but unsecure in modern context. ROT13 cipher("rotate by 13 places") is a simple substitution cipher which is a special case of the Ceaser cipher in which the shift is always 13. For decryption, it does the opposite: we rotate each alphabet by 13 places. We can clearly see that it is possible to concatenate two different pieces of ciphertexts. Hence, ROT13 is partially homomorphic with respect to the concatenation operation. When we decrypt the concatenated pieces of ciphertext we recover the concatenation of the plaintext messages. Therefore, it is partially homomorphic with respect to concatenation.

Let us consider a pair of plaintext messages $P_1 =$ HELLO and $P_2 =$ WORLD. The concatenation of these two messages gives us "HELLOWORLD". The ciphertexts if If both plaintexts were encrypted using ROT13 would be $C_1 =$ URYYB and $C_2 =$ JBEYQ. The ciphertext upon concatenation give us "URYYBJBEYQ". Upon decrypting the ciphertext the message obtained is "HELLOWORLD", which is exactly the same message obtained when the pair of plaintexts were concatenated originally. The above example is better explained using the diagram as shown in Fig. 4.1.

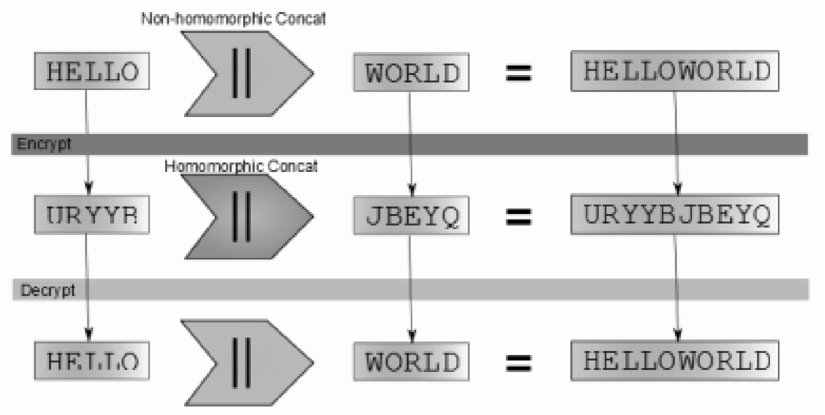


Figure 4.1: Example of the homomorphic property in ROT13. Based on Stuntz [33].

4.1.3 The First FHE Scheme

According to the article published by NSA research staff in 2014, NSA [34], the term FHE has been mentioned in scientific literature for the past 30 years but it has been actualized only recently in the year of 2009 by the seminal work of Craig Gentry. This work by Gentry has opened the doors for plethora of future work in a domain which was previously unexplored.

Gentry [1] defines "fully" homomorphic encryption as the HE scheme in which there is no limit on the type of operation to be performed. To explain this further let us consider,

- Consider the ciphertexts $c_1,\dots,c_t$ corresponding to the messages $m_1,\dots,m_t$
- f is any efficiently computable function i.e. anyone can compute ciphertexts or set of ciphertexts that encrypt $f(m_1,\dots,m_t)$.
- This permits all kinds of computations on the encrypted data without leaking any information about the plaintexts $m_1,\dots,m_t$ or the value of $f(m_1,\dots,m_t)$.

For constructing a FHE scheme, Gentry starts first by construction a *somewhat homomorphic encryption scheme* (SWHE). A SWHE is capable of evaluating "low-degree" polynomials homomorphically.

To realize the FHE scheme, Gentry proposed the **bootstrapping theorem**. Using this theorem allows us to transform a somewhat homomorphic encryption scheme into a *leveled* FHE one.

A major problem in the construction of the FHE scheme is controlling the noise in the ciphertext. As we keep on performing operations such as addition or multiplication on the ciphertexts we can find gradual distortion in them. This gradual distortion is known as noise.

As the number of operations being performed on the ciphertext increases the noise also increases, this renders the ciphertext undecipherable and the decryption obtained is a wrong one.

Each homomorphic addition doubles the noise, and each multiplication squares it as pointed out by Hayes [32]. Hence in order to have an uncorrupted ciphertext at the end of the computation we must ensure proper noise management.

A simple an trivial solution to this noise management problem would be to decrypt the message and re-encrypt it every time the noise reaches its critical level. This way the noise is reset to its original value. But this solution leads us to the original problem solved by HE schemes i.e. the entity performing the computations doesn't have access to the secret key used for decryption.

To overcome the problem of noise management Gentry came up with the solution known as Bootstrapping. In a FHE scheme the *evaluate* function can compute any operation on the ciphertext till the point where the noise is below the critical threshold. Hence we use the very evaluate function for the *decrypt* function, which takes as input the noise-heavy ciphertext that has gone through several operations, and the *encrypted secret key*. So now when we have represented the decryption using the evaluate function we will not be getting a plaintext but we shall be getting a new ciphertext (as evaluate always gives us an encrypted value). Hence after the evaluate function has been run the new ciphertext obtained has much less noise than the previous noise heavy ciphertext. This process of recrypting the ciphertext can be done again and again as it practically doesn't have a limit.

But bootstrapping comes with an overhead, it is computationally intensive to perform. As pointed out by the research staff from the NSA [34], if the process were to be used by Google to search the web homomorphically, the normal computing time required would be multiplied by about a trillion. This is the reason why bootstrapping is not a suitable solution for implementation and there is still ongoing research for better noise reduction.

Chapter 5

Levelled-BGV Fully Homomorphic Encryption

The Brakerski-Gentry-Vaikuntanathan (BGV) scheme [9] is a levelled-FHE. It does not rely on bootstrapping, but rather employs it as an optimization.

The BGV scheme follows on the footsteps of Gentry's FHE scheme by employing lattice-based cryptography with certain other improvements.

It is described as *leveled* because the parameters of the scheme are polynomially dependent on the depth of the circuits that the scheme is capable of evaluating.

The BGV scheme has significantly improved the performance compared to earlier proposed FHE schemes which were based on bootstrapping. Also, the security assumption employed by BGV scheme is weaker compared to earlier ones. Ring-learning with error (RLWE) forms the basis for the security of the BGV scheme and it guarantees 2^{λ} security against known attacks.

5.1 Basic Scheme

The basic scheme for the BGV FHE is given below.

- Choose integers q and d and set $R = (\mathbb{Z}_q)[x]/(x^d + 1)$, $R_q = R/qR$. Furthermore χ is a distribution on R_q and $N = (2n+1)\log_2 q$
- KeyGen: sample $s' \in {R_q}^N$ and set $sk = s = (1, s')$. Generate uniformly at random $B \in {R_q}^N$ and error vector $e \in \chi^N$. Set b = Bs' + 2e and A = [b, -B].
  Output:$(sk, pk) = (s, A)$
- Enc(m, pk): for message $m \in R_2$, set $m = (m, 0)$. Sample $r \in {R_2}^N$ and set $c = m + A^T r = (c0, c1)$.
  Remark: Observe that $A.s = 2e$.
  Output: (c_0, c_1)
- Dec(c, sk): compute $z = [[\langle(\mathrm{c}_0, \mathrm{c}_1),(1, \mathrm{s'})\rangle]_q]_2$.
  Output: z

- **Correctness** - $[[\langle(c_0, c_1),(1, s')\rangle]_q]_2.$ = $[[(m^T + r^T A).s]_q]_2$ = $[[m + 2r^T e]_q]_2$ = $[m + 2r^T e]_2 = m$

5.2 Noise Management and Modulus Switching

Similar to Gentry's first FHE scheme [1] we see that upon performing computations on the ciphertext the noise gradually starts increasing and the ciphertext becomes more distorted.

Specifically, performing one multiplication roughly squares the noise whereas addition between two ciphertexts doubles the noise roughly.

As long as the noise in the ciphertext is below a certain threshold level the ciphertext gives the correct decryption and returns the correct value of the plaintext.

The BGV scheme makes use of a keeps the noise in check by using a noise reduction technique which does not use bootstrapping but rather reduces noise after performing each HE performed on the ciphertext.

This technique which helps in noise reduction is called the **modulus switching** technique, and it was first described in the work of Brakerski and Vaikuntanathan [8].
The lemma that describes the original modulus-switching technique says that [8]

> An evaluator who does not know the secret key s, but instead only knows a bound on its length, can transform a ciphertext c modulo q into a different ciphertext modulo p while preserving correctness, expressed as $[\langle c', s\rangle]_p \equiv [\langle c, s\rangle]_q \pmod 2$.

The interesting property found here is that the ciphertext actually decreases given the condition that s is short and p is sufficiently smaller than q. The "noise" here means that
$|[\langle c', s\rangle]_p| < |[\langle c, s\rangle]_q|$. Hence we can see that the magnitude of the noise decreases without the knowledge of the secret key, and without depending on bootstrapping. The BGV scheme takes the aforementioned technique, and further improves noise management by making use of a *ladder of gradually decreasing moduli*.

5.3 HElib

This BGV scheme has been implemented as a software library in C++, called HElib [13]. The implementation of the library focuses on the Gentry-Halevi-Smart optimizations and also the Smart-Vercauteren ciphertext packing techniques for improving the performance. Halevi and Shoup [14] describe in a paper the algorithms employed on the implementation, as well as some considerations to have in mind depending on the hardware used.

Chapter 6

Schulze Method of Voting

The Schulze method developed by Markus Schulze [5] is a voting system that decides a single winner amongst all the candidates using votes which give the preference order of a voter. A sorted list of winners can also be obtained by the schulze method i.e. the rank obtained by each of the candidates at the end of the voting. The Schulze method is also popular by other names, like ”cloneproof Schwartz sequential dropping” (CSSD), ”Schwartz sequential dropping” (SSD), ”path voting”, ”beatpath method”, ”beatpath winner” and ”path winner”.

In the Schulze method the winning candidate beats all the other candidates by a majority if we compare each candidate in a pairwise manner. In turn there is also a guarantee that a candidate which beats allof it other competitors when compared pairwise to them in a pairwise manner, then the candidate will win. By virtue of the above properties the Schulze method is defined as a Condorcet method. It is worthy to note that this guarantee is not provided to us by other preferential voting systems like Instant-runoff and Borda.

We get an ordered ranking from the Schulze voting method, hence if there are *P* number of vacant positions to be filled then we can directly use the outcome of the Schulze voting method to fill the positions by choosing the top *P* positions.

Many different ways have been proposed to implement the Schulze method, path method and the Schwartz method are the most important ones.

This voting system is being used by a lot of organizations including Ubuntu, Pirate Party, Debian, Software in the Public Interest, Gentoo, and many others.

6.1 Description of the method

6.1.1 Ballot

In the Schulze method each of the voters casts a vote where each vote has a list of the voters preference where he/she ranks the voters in order of preference, this way of input is same for all ranked single winner electoral systems.

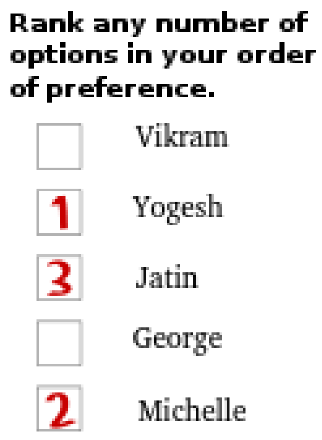


Figure 6.1: An example of Schulze preferential ballot

In Fig.6.1 we can see the way in which the voter has to allot his preference to the candidates. Each vote casted has a list of all the candidate's name and the voter puts his rank of preference besides the candidate's name, for example for his most preferred candidate the voter puts a "1" and similary "2" for the second most favoured one and so on and so forth.
Each voter may optionally:

- May show indifference between two or more candidates by giving the same rank or preference to them.
- Make use of non-consecutive number to express his preference as it is immaterial in our case. If the number is higher then we know that the candidate is less preferred than a candidate with a lesser number.
- If a voter doesn't rank any candidate by giving him a number then it is considered that the voter is indifferent amongst the unranked candidates but prefers all the ranked candidates over unranked ones.

6.1.2 Computation

Let $\delta[P,Q]$ be the number of voters who rank candidate P higher than Q i.e. they prefer P more.

A path from candidate P to candidate Q is a sequence of candidates $\kappa(1), \cdots,$ $\kappa(n)$ and it has the following properties:

- $\kappa(1) = P$ and $\kappa(n) = Q$.
- For $i = 1, \cdots, (n-1)$: $\delta[\kappa(i), \kappa(i+1)] > \delta[\kappa(i+1), \kappa(i)]$

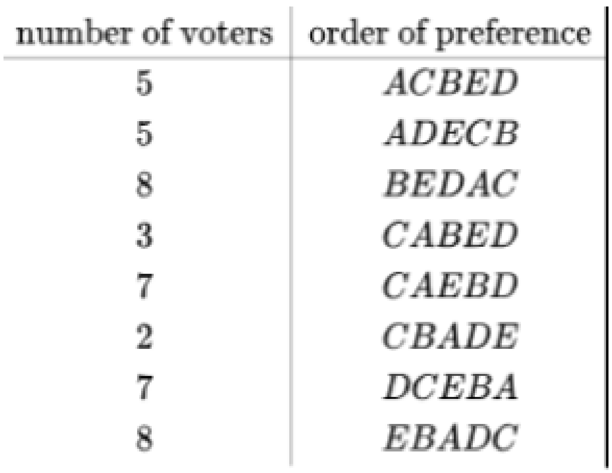
Hence we can see that each of the other candidates beats this candidate in a pairwise comparison.

In a path from candidate P to Q the smallest number of votes in the sequence of comparisons:
For $i = 1, \cdots, (n-1)$: $\delta[\kappa(i), \kappa(i+1)] \geq p$.
is called the strength ρ of the path.

- If there is atleast a path between the candidates P and Q then the strength of the strongest path $\rho[P,Q]$ is the maximum strength of the path connecting them. $\rho[P,Q] = 0$ if there is no path between the candidates P and Q at all.
- If $\rho[P,Q] > \rho[Q,P]$ then we say that candidate P beats the candidate Q.
- If $\rho[P,Q] \geq \rho[Q,P]$ for every other candidate Q, then P is a potential winner.
- We can show that $\rho[P,Q] > \rho[Q,P]$ and $\rho[Q,R] > \rho[R,Q]$ together imply $\rho[P,R] > \rho[R,P]$. Therefore, it is guaranteed
  - that there is a transitive relation while comparing the strengths of two candidates.
  - that there is always at least one candidate P with $\rho[P,Q] \geq \rho[Q,P]$ $\forall$ $Q \in C$ where C is the candidate space.

6.1.3 Example

Let us consider the case of $|V| = 45$ and $|C| = 5$ where the $|V|$ dentoes the number of voters and $|C|$ dentoes the number of candidates. The candidates are denoted by A $\cdots$ E. The pairwise preferences of the voters is listed as below in Fig. 6.2 as given by Schulze in [35]



number of voters	order of preference
5	*ACBED*
5	*ADECB*
8	*BEDAC*
3	*CABED*
7	*CAEBD*
2	*CBADE*
7	*DCEBA*
8	*EBADC*

Figure 6.2: Pairwise preference of voters

First the pairwise preference of voters is computed. Let us compare the two candidates C and D, there are $5+7+3+2 = 17$ voters who prefer C to D, and

	δ[*,A]	δ[*,B]	δ[*,C]	δ[*,D]	δ[*,E]
δ[*,A]		20	26	30	22
δ[*,B]	25		16	33	18
δ[*,C]	19	29		17	24
δ[*,D]	15	12	28		14
δ[*,E]	23	27	21	31	

Figure 6.3: Matrix for preference table

$5+8+7+8 = 28$ voters who prefer D to C. So $\delta[C,D] = 17$ and $\delta[D,C] = 28$. The complete matrix for pairwise comparisons among the candidates is given below in Figure. 6.3:

Now in order to get to the winner we must calculate the strongest paths. We first visualize a directed graph based on the Fig. 6.3. The weight of the direct edge from a candidate R to S is denotes $\delta[R,S]$. To provide an uncluttered and clean diagram, we show only those edges from R to S when $\delta[R,S] > \delta[S,R]$ $\forall$ R and S in C.

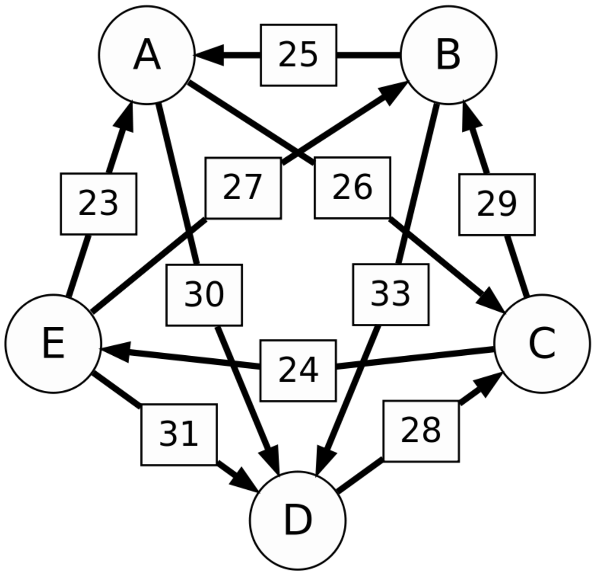


Figure 6.4

Let us calculate the strongest paths between the two candidates A and D, then we can see that $\rho[A,D] = 30$: i.e. the direct path between A and D is indeed the strongest path from A to D and has the strength 30 as shown in Fig. 6.5.

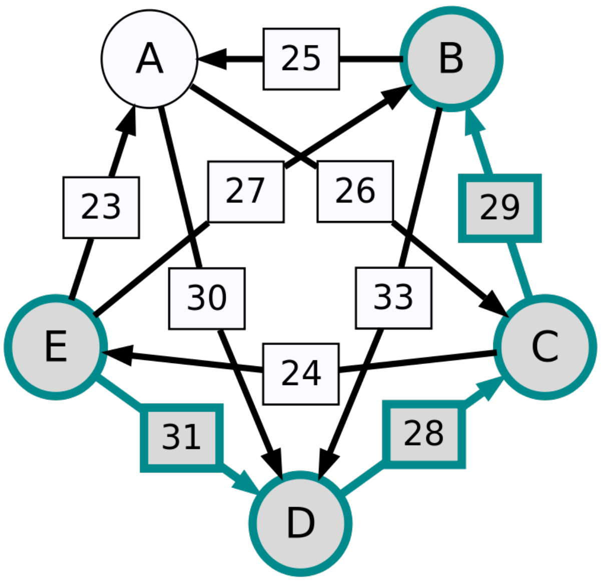


Figure 6.6

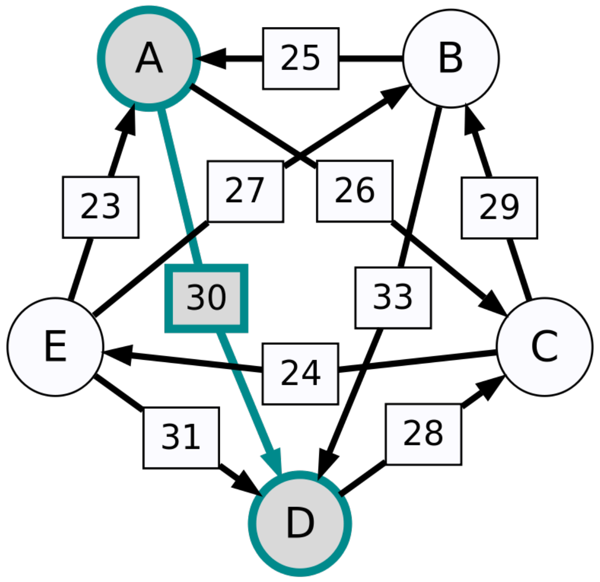
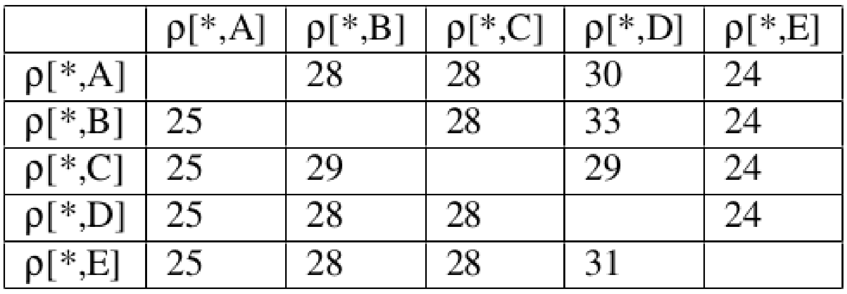


Figure 6.5: Direct path between A and D

Now let us compute the strongest path between the candidates E and B, then we can see that the direct path (E,B) of strength 27 is not the strongest path from E to B, rather the strongest path is the indirect path (E,D,C,B) which has strength $\min(31,28,29) = 28$ as shown in Fig. 6.6. Hence the value of $\rho[E,B]$

Similarly when we look at all the pairwise paths we get the following strengths of the strongest path as shown in the Fig. 6.7

We can now look at the output of the Schulze method in the following way. Let us compare the candidates A and B, we can clearly see that $\rho[A,B] > \rho[B,A]$ i.e. in a pairwise comparison the candidate A beats B. Another example is that $\rho[B,D] > \rho[D,B]$, so candidate B beats the candidate D. Hence if keep on doing this we can



	ρ[*,A]	ρ[*,B]	ρ[*,C]	ρ[*,D]	ρ[*,E]
ρ[*,A]		28	28	30	24
ρ[*,B]	25		28	33	24
ρ[*,C]	25	29		29	24
ρ[*,D]	25	28	28		24
ρ[*,E]	25	28	28	31	

Figure 6.7: Matrix for the strongest path

see that we get the ordering $E > A > C > B > D$, and E wins. Clearly since E beats every other candidate it is the winner.

6.2 Implementation

Now when we have seen the examples of strongest the only thing is to how we can compute them, calculating the strongest path is a fairly known problem and can be computed using a modification of the Floyd-Warshall Algorithm.

Input: $\rho[u, v]$, the number of voters who prefer candidate u to candidate v and $|C|$ is the total number of candidates.
Output: $\rho[u, v]$, the value of the strongest path from the candidate u to candidate v.

Algorithm 1 Strongest path in schulze voting

```
for i = 1 to |C| do
    for j = 1 to |C| do
        if i ≠ j then
            if (δ[i,j] > δ[j,i]) then
                ρ[i,j] = δ[i,j]
            else
                ρ[i,j] = 0
            end if
        end if
    end for
end for

for i = 1 to |C| do
    for j = 1 to |C| do
        if i ≠ j then
            for i = 1 to |C| do
                if i ≠ j and j ≠ k then
                    ρ[j,k] = maximum( ρ[j,k], minimum( ρ[j,i], ρ[i,k] ) )
                end if
            end for
        end if
    end for
end for
```

Figure 6.8: Algorithm for calculating the strongest path in schulze method

Chapter 7

Proposed Approach and Outcome

We shall be employing FHE, specifically the BGV scheme to privately compute the Schulze Voting Method. Specifically we will be focussing on the computation of strongest path in a private manner as given by the Algorithm **??**. We shall be providing an algorithm to compute the Algorithm **??** in a private manner. Till now we have seen that HE has been applied in e-voting to provide verifiability but the tallying of the ballots involves simple addition or incrementation of votes. Due to Condorcet nature of the Schulze voting, the procedure to get the winner involves different operations like Comparisons and Min and Max computations. This is primarily what differentiates our work from previous applications of HE in e-voting.

7.1 Assumptions and Scope

In the proposed work we make the following assumptions and also limit our scope to particular aspects only -

- We don't consider the way in which the election will take place, for simplicity we will assume the traditional client and cloud architecture where the preference matrix will be outsourced in an encrypted manner to the service provider which is not completely trusted. The computing entity will privately compute the algorithm to find the strongest path and return back the result to the client.
- Other aspects of e-voting like secret sharing schemes for additional security have also been left as part of separate work and not discussed.

7.2 Methodology

The following is the process flow for the private computation of strongest paths in schulze voting-

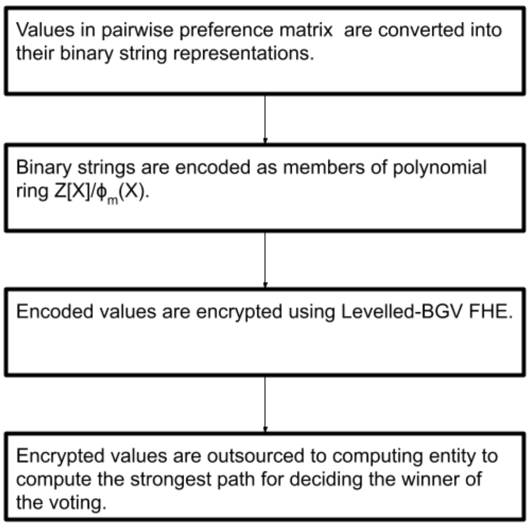

Values in pairwise preference matrix are converted into their binary string representations.
Binary strings are encoded as members of polynomial ring Z[X]/ϕ_m(X).
Encoded values are encrypted using Levelled-BGV FHE.
Encrypted values are outsourced to computing entity to compute the strongest path for deciding the winner of the voting.

Figure 7.1: Process flow for private computation of Schulze voting.

We should be able to compute the strongest path for the schulze method in a private manner.

7.2.1 Comparison

We can see that line 4 in the Algorithm **??** has a comparison operation, hence we must be able to get the values of comparison between two numbers albeit in an encrypted manner.

We use the circuit for comparison as discussed in [36] and its working is explained as below.

Let us consider two integers α and β that we want to compare represented in binary form as as $\alpha = \overline{\alpha_{n-1}\alpha_{n-2}\cdots\alpha_0}$ and $\beta = \overline{\beta_{n-1}\beta_{n-2}\cdots\beta_0}$. In order to compare two numbers we first compare the MSBs if the MSB of α is greater than β then α is greater than β. If the MSBs are equal we compare the next bit and so on.

Let us take a concrete example to understand the same-

Example

Suppose we have two 4 bit numbers - $\alpha = (\alpha_3, \alpha_2, \alpha_1, \alpha_0)$ and $\beta = (\beta_3, \beta_2, \beta_1, \beta_0)$ and we wish to compare them then, we have $\alpha_3.\beta_3'$ (where $'$ gives complement) = 1 if $\alpha_3 > \beta_3$ and 0 otherwise.
Which can be seen from the following truth table -

α_i	β_i	$\alpha_i.\beta_i'$
0	0	0
0	1	0
1	0	1
1	1	0

Also, let x_i denote if α_i and β_i are equal. ($x_i = 1$ if they α_i = β_i and 0 otherwise) and if we have to compute the numbers we will have $\rightarrow$

$Z = \alpha_3\beta_3' \vee x_3.(\alpha_2\beta_2') \vee x_3.x_2.(\alpha_1\beta_1') \vee x_3.x_2.x_1.(\alpha_0\beta_0')$
(that is $\mathrm{MSB}_\alpha > \mathrm{MSB}_\beta \vee (\mathrm{MSB}_\alpha)$=$(\mathrm{MSB}_\beta) \wedge (\mathrm{LSB}_\alpha > \mathrm{LSB}_\beta)$). Here "." is the AND operator or multiplication operator.

Similar logic can be extended for N digits.

Example - α = 14 $=$ (1110) and $\beta = 13 = (1101)$

Now, $Z = 1.1'$ OR $1.(1.1')$ OR $1.1.(1.0')$ OR $1.1.1.(0.1') = 1$ where OR is the logical operator. Which can be represented as $Z = (1111)$ which is an array or a vector by filling all the slots of plaintext or ciphertext by 1. Similarly if $\alpha \leq \beta$ then the result Z can be represented by
Now since we can perform addition and multiplication which convert to XOR and AND operations in $\mathbb{Z}_2$ we can also compare two Encrypted Integers by using FHE.

- COM(**Enc**(α), **Enc**(β)) = C. (C is an encrypted value)
- **Dec**(C) = 1 if $\alpha > \beta$
- **Dec**(C) = 0 otherwise.

Hence we can define a function -

$$COM(\alpha,\beta) = \begin{cases} 1 & \alpha > \beta \\ 0 & \alpha \leq \beta \\ Enc(1) & Enc(\alpha) > Enc(\beta) \\ Enc(0) & Enc(\alpha) \leq Enc(\beta) \end{cases}$$

Z = COM(α, β), if one bit is inequal then it is set 0 then due to rotation all the bits of the number are set to 0. If $Z = (0001)$ then Z.rotate(1) = (1000).

We get $Z = (1111)$ when the $\alpha > \beta$ are equal and $Z = (0000)$ when α and β are unequal. We do so, so that

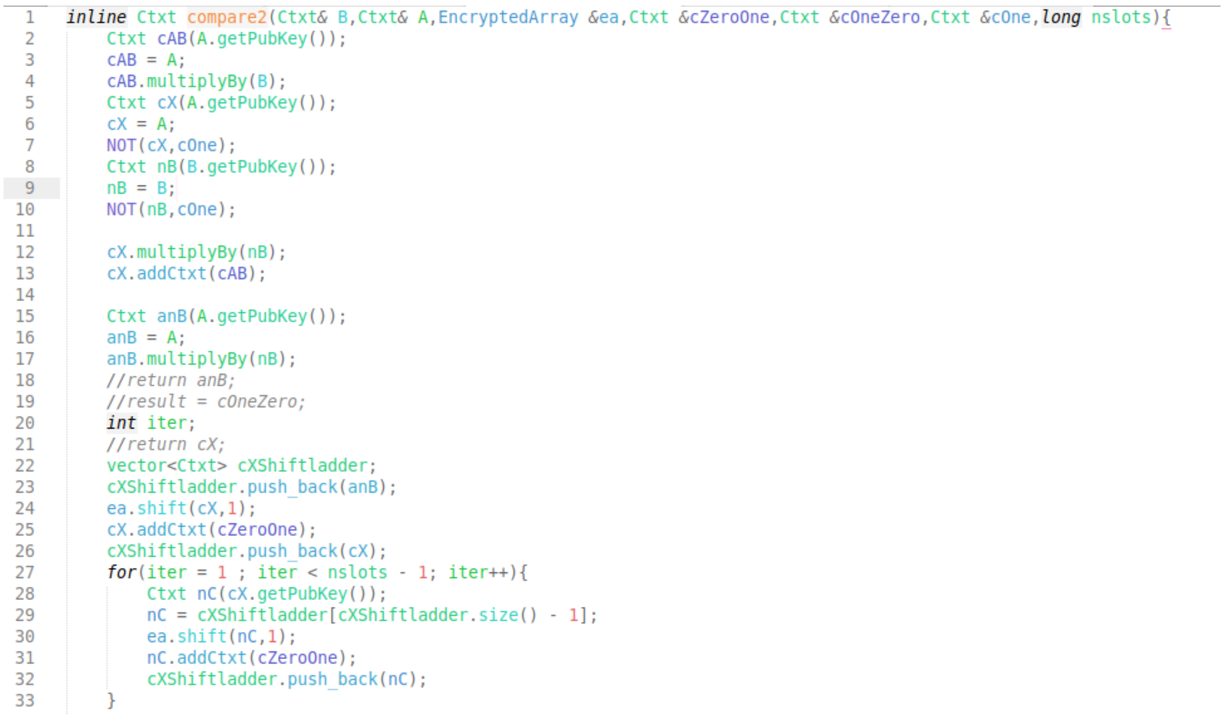
Algorithm 1 Rotating the array to spread 1 or 0

```
Z = COM(α, β)
1: for $i = 1$ to $N - 1$ do
2:     $Z = Z \oplus Z$.rotate(1)
3: end for
```

$$COM(\alpha,\beta).A = \begin{cases} A & \alpha > \beta \\ 0 & \alpha \leq \beta \end{cases}$$

Where "." represents the multiplication operator or AND in $\mathbb{Z}_2$.
Since the value is revealed only after decryption no information is leaked.

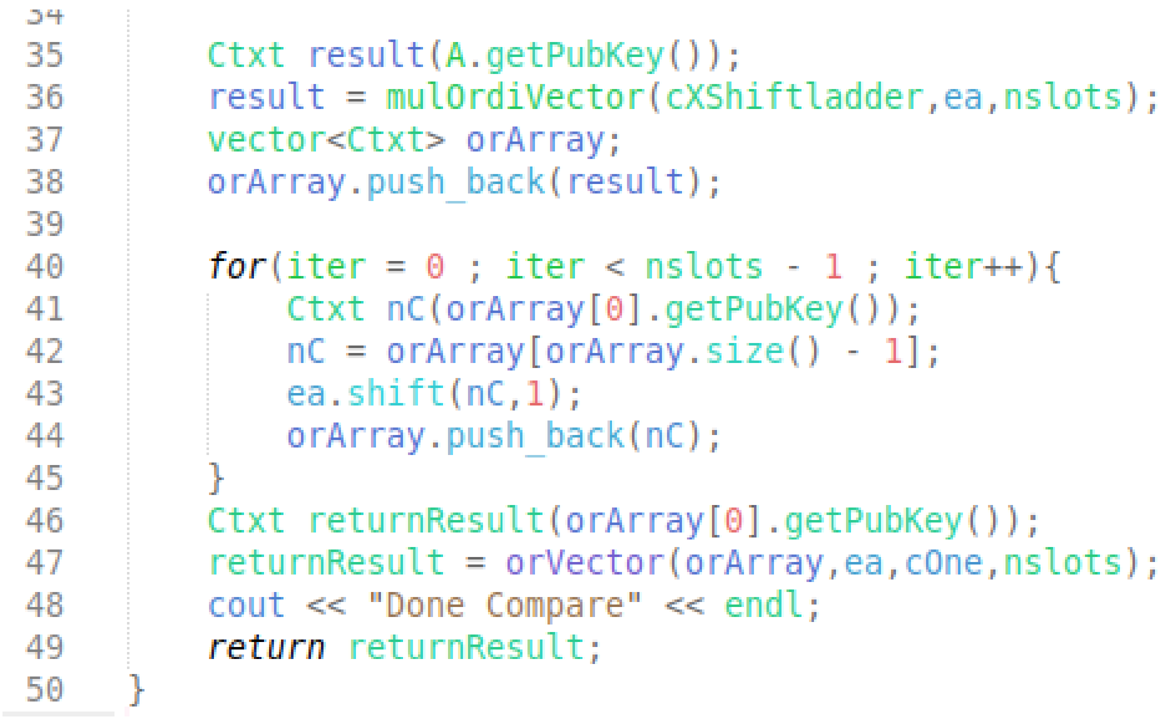
Here in Fig. 7.2 we present the code snippet for our compare function as written in HElib -



```
inline Ctxt compare2(Ctxt& B,Ctxt& A,EncryptedArray &ea,Ctxt &cZeroOne,Ctxt &cOneZero,Ctxt &cOne,long nslots){
    Ctxt cAB(A.getPubKey());
    cAB = A;
    cAB.multiplyBy(B);
    Ctxt cX(A.getPubKey());
    cX = A;
    NOT(cX,cOne);
    Ctxt nB(B.getPubKey());
    nB = B;
    NOT(nB,cOne);

    cX.multiplyBy(nB);
    cX.addCtxt(cAB);

    Ctxt anB(A.getPubKey());
    anB = A;
    anB.multiplyBy(nB);
    //return anB;
    //result = cOneZero;
    int iter;
    //return cX;
    vector<Ctxt> cXShiftladder;
    cXShiftladder.push_back(anB);
    ea.shift(cX,1);
    cX.addCtxt(cZeroOne);
    cXShiftladder.push_back(cX);
    for(iter = 1 ; iter < nslots - 1; iter++){
        Ctxt nC(cX.getPubKey());
        nC = cXShiftladder[cXShiftladder.size() - 1];
        ea.shift(nC,1);
        nC.addCtxt(cZeroOne);
        cXShiftladder.push_back(nC);
    }
```



```
    Ctxt result(A.getPubKey());
    result = mulOrdiVector(cXShiftladder,ea,nslots);
    vector<Ctxt> orArray;
    orArray.push_back(result);

    for(iter = 0 ; iter < nslots - 1 ; iter++){
        Ctxt nC(orArray[0].getPubKey());
        nC = orArray[orArray.size() - 1];
        ea.shift(nC,1);
        orArray.push_back(nC);
    }
    Ctxt returnResult(orArray[0].getPubKey());
    returnResult = orVector(orArray,ea,cOne,nslots);
    cout << "Done Compare" << endl;
    return returnResult;
}
```

Figure 7.2: Code snippet for compare function

7.2.2 Equality

Let us now consider the method for computing equality between two numbers which returns 1 if the numbers are the same and 0 otherwise. Let us see how to check for equality of two bits -

EQU(α, β) = (1 $\oplus$ α $\oplus$ β)

We can define a function -

$$EQU(\alpha,\beta) = \begin{cases} 1 & \alpha = \beta \\ 0 & \alpha \neq \beta \\ Enc(1) & Enc(\alpha) = Enc(\beta) \\ Enc(0) & Enc(\alpha) \neq Enc(\beta) \end{cases}$$

Suppose we have two 4 bit numbers - α = (α_3, α_2, α_1, α_0) and β = (β_3, β_2, β_1, β_0).

Now we know that the EQU() of two bits α_i and β_i gives us 1 if $\alpha_i = \beta_i$ and 0 if $\alpha_i \neq \beta_i$.

That is 1 $\oplus$ α_i $\oplus$ β_i = 1 if $\alpha_i = \beta_i$ and 1 $\oplus$ α_i $\oplus$ β_i = 0 if $\alpha_i \neq \beta_i$.

α_i	β_i	Z_i
0	0	1
0	1	0
1	0	0
1	1	1

Here $Z_i = 1 \oplus \alpha_i \oplus \beta_i$

Now since we can perform addition and multiplication which convert to XOR and AND operations in $\mathbb{Z}_2$ we can also compare two Encrypted Integers by using FHE.

- EQU(**Enc**(α), **Enc**(β) = C. (C is an encrypted value)
- **Dec**(C) = 1 if $\alpha = \beta$
- **Dec**(C) = 0 otherwise.

Now again let us take,

$\alpha = 14 = (1110)$ and $\beta = 14 = (1110)$

When we perform $Z = 1 \oplus \alpha \oplus \beta$ we have $Z = (1111)$ as all the α_i's and β_i's are equal and $Z = (0000)$ when even one of the α_i's and β_i's are unequal. We do so, so that

$$EQU(\alpha, \beta).A = \begin{cases} A & \alpha = \beta \\ 0 & \alpha \neq \beta \end{cases}$$

Where "." represents the multiplication operator.

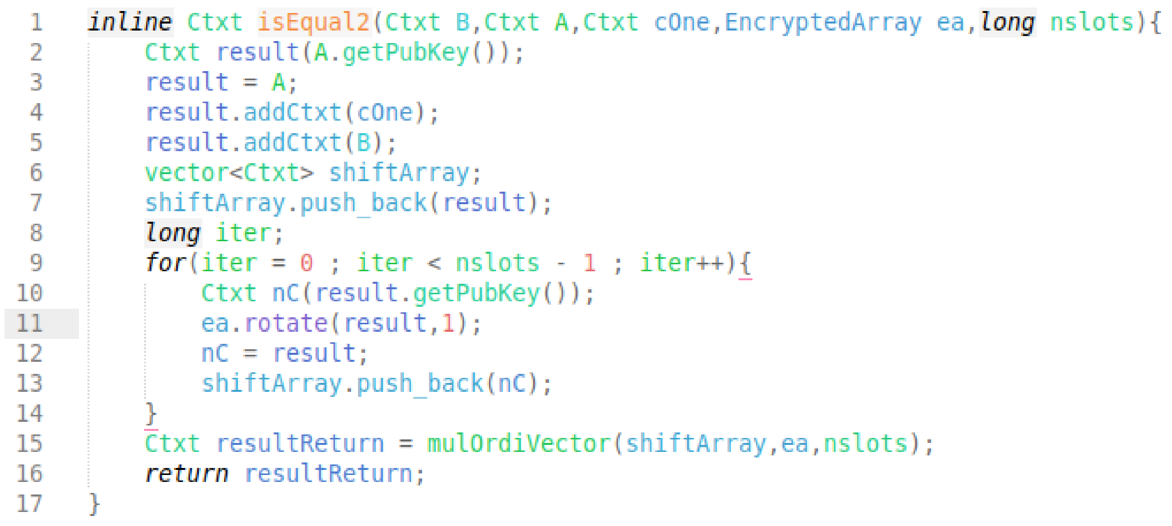
Now we want that when $\alpha_i \neq \beta_i$ we should a get a binary string R made up completely of 0's.

Algorithm 2 Rotating the array to spread 1 or 0

```
Z = 1 ⊕ α ⊕ β
1: for $i = 1$ to $N - 1$ do
2:     $Z = Z \times Z$.rotate(1)
3: end for
```

$Z = 1 \oplus \alpha \oplus \beta$, if one bit is inequal then it is set 0 then due to rotation all the bits of the number are set to 0. If $Z = (0001)$ then Z.rotate(1) = (1000).

Here in Fig. 7.3 we present the code snippet for our equality function as written in HElib -



```
inline Ctxt isEqual2(Ctxt B,Ctxt A,Ctxt cOne,EncryptedArray ea,long nslots){
    Ctxt result(A.getPubKey());
    result = A;
    result.addCtxt(cOne);
    result.addCtxt(B);
    vector<Ctxt> shiftArray;
    shiftArray.push_back(result);
    long iter;
    for(iter = 0 ; iter < nslots - 1 ; iter++){
        Ctxt nC(result.getPubKey());
        ea.rotate(result,1);
        nC = result;
        shiftArray.push_back(nC);
    }
    Ctxt resultReturn = mulOrdiVector(shiftArray,ea,nslots);
    return resultReturn;
}
```

Figure 7.3: Code snippet for equality function

7.2.3 Maximum

We see that in line 18 of algorithm given in Fig. 6.8 requires us to compute the maximum value of two numbers. We will describe a function MAX(α, β) using EQU(α, β) and COM(α, β) already defined in the subsections above.

$$MAX((\alpha,\beta)) = \begin{cases} \alpha & \alpha > \beta \\ \beta & \beta > \alpha \\ Enc(\alpha) & Enc(\alpha) > Enc(\beta) \\ Enc(\beta) & Enc(\beta) > Enc(\alpha) \end{cases}$$

We describe MAX(α, β) as -

MAX(α, β) = COM(α, β).α + COM(α, β).β + EQU(α, β).α

Let us take the example of α = 5 and β = 7,

MAX(5,7) = COM(5,7).5 + COM(7,5).7 + EQU(5,7).5

$= 0.5 + 1.7 + 0.5 = 7$

We will get the same resullt when we compare them in the form of binary strings.

7.2.4 Minimum

We see that in line 18 of algorithm given in Fig. 6.8 requires us to compute the Minimum value of two numbers. We will describe a function MAX(α, β) using EQU(α, β) and COM(α, β) already defined in the subsections above.

$$MIN((\alpha,\beta)) = \begin{cases} \alpha & \alpha < \beta \\ \beta & \beta < \alpha \\ Enc(\alpha) & Enc(\alpha) < Enc(\beta) \\ Enc(\beta) & Enc(\beta) < Enc(\alpha) \end{cases}$$

We describe MIN(α, β) as -

MIN(α, β) = COM(α, β).β + COM(α,β).α + EQU(α, β).α

Let us take the example of α = 5 and β = 7,

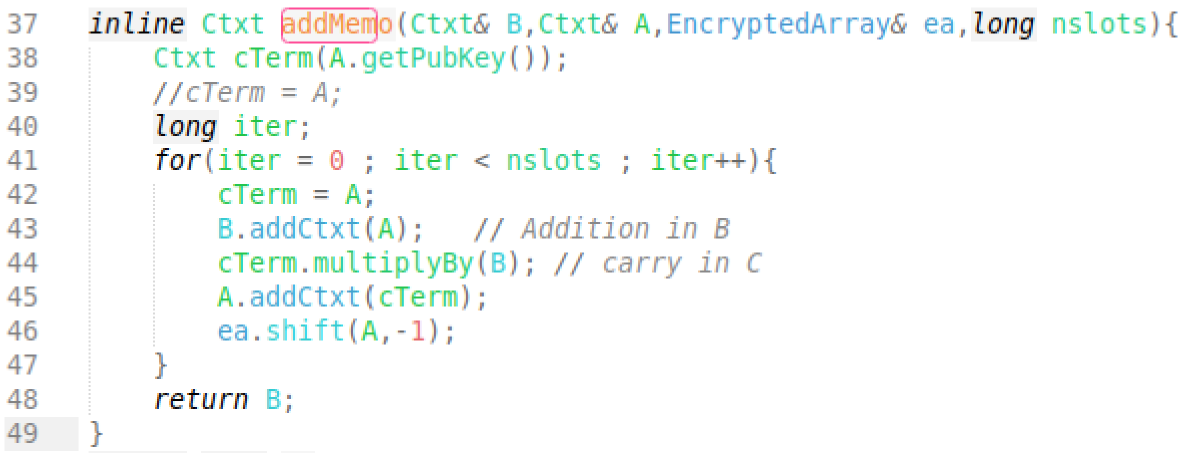
MIN(5,7) = COM(5,7).7 + COM(7,5).5 + EQU(5,7).5

$= 0.7 + 1.5 + 0.5 = 5$

We will get the same result when we compare them in the form of binary strings.

7.2.5 Addition

We will be using a binary circuit adder to add two encrypted numbers as they have been encrypted in the binary string format. It has been implemented in HElib as shown in Fig. 7.4 -

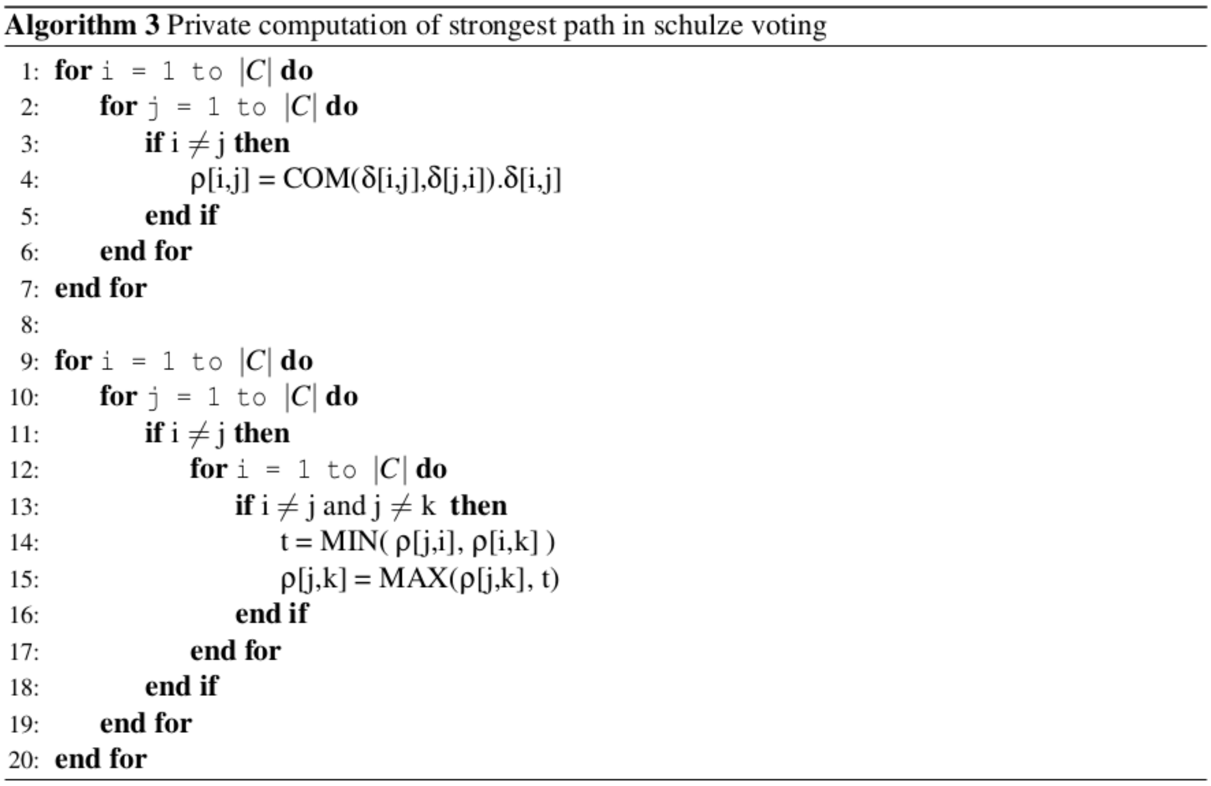


```
inline Ctxt addMemo(Ctxt& B,Ctxt& A,EncryptedArray& ea,long nslots){
    Ctxt cTerm(A.getPubKey());
    //cTerm = A;
    long iter;
    for(iter = 0 ; iter < nslots ; iter++){
        cTerm = A;
        B.addCtxt(A);   // Addition in B
        cTerm.multiplyBy(B); // carry in C
        A.addCtxt(cTerm);
        ea.shift(A,-1);
    }
    return B;
}
```

Figure 7.4: HElib implementation for binary carry adder

7.3 Private Computation of Schulze Voting Method

In this section we propose the algorithm for private computation of strongest paths in the schulze voting method using the help of COM(α, β), EQU(α, β) and MAX(α, β), MIN(α, β) functions as described in the above sections.



Algorithm 3 Private computation of strongest path in schulze voting

```
for i = 1 to |C| do
    for j = 1 to |C| do
        if i ≠ j then
            ρ[i,j] = COM(δ[i,j],δ[j,i]).δ[i,j]
        end if
    end for
end for

for i = 1 to |C| do
    for j = 1 to |C| do
        if i ≠ j then
            for i = 1 to |C| do
                if i ≠ j and j ≠ k then
                    t = MIN( ρ[j,i], ρ[i,k] )
                    ρ[j,k] = MAX(ρ[j,k], t)
                end if
            end for
        end if
    end for
end for
```

Figure 7.5: Algorithm for private computation of strongest path in schulze voting

7.4 Results and Evaluation

In this section we will estimating the time taken by the algorithm to run against the different parameters. We will be estimating the time in an approximate manner using the time taken for COM(α, β), EQU(α, β) and the ADD(α, β) and MAX(α, β) function. We plot the time taken by each of these functions for one instance and then estimate the time taken for the whole algorithm for different number of candidates. The parameters used for implementation in HElib library are explained in the Table 7.1

Parameters	Meaning
p	Characteristic of plaintext space
r	Lifting parameter
d	Embedding degree
c	Number of columns in key-switching matrix
k	Security parameter
w	Hamming weight of secret key
s	Minimum number of slots
m	Chooses the mth cylotomic polynomial.
L	Number of levels in the modulus chain

Table 7.1: Meaning of parameters used for implementation in HElib

The following parameter values as shown in Table 7.2 have been used in HElib to implement the basic functions like ADD(α, β), EQU(α, β) and MAX/MIN(α, β) function.

Parameters	**Values**
Library	HElib
p	2
r	1
d	1
c	2
k	128
w	64
s	16
m	255

Table 7.2: Value of parameters used for implementation in HElib

We implemented the COM(α, β), EQU(α, β), ADD(α, β) and MAX(α, β) in C++ (gcc version 7.3.0) and deployed it on Intel(R) Core(TM) $i5-3230M$ CPU clocked at 2.60GHz with 4 GB RAM and we make use of these to estimate the time for the algorithm given in Fig. 7.5 **??** for different parameters. Throughout the implementation we make use of HElib for using BGV scheme.

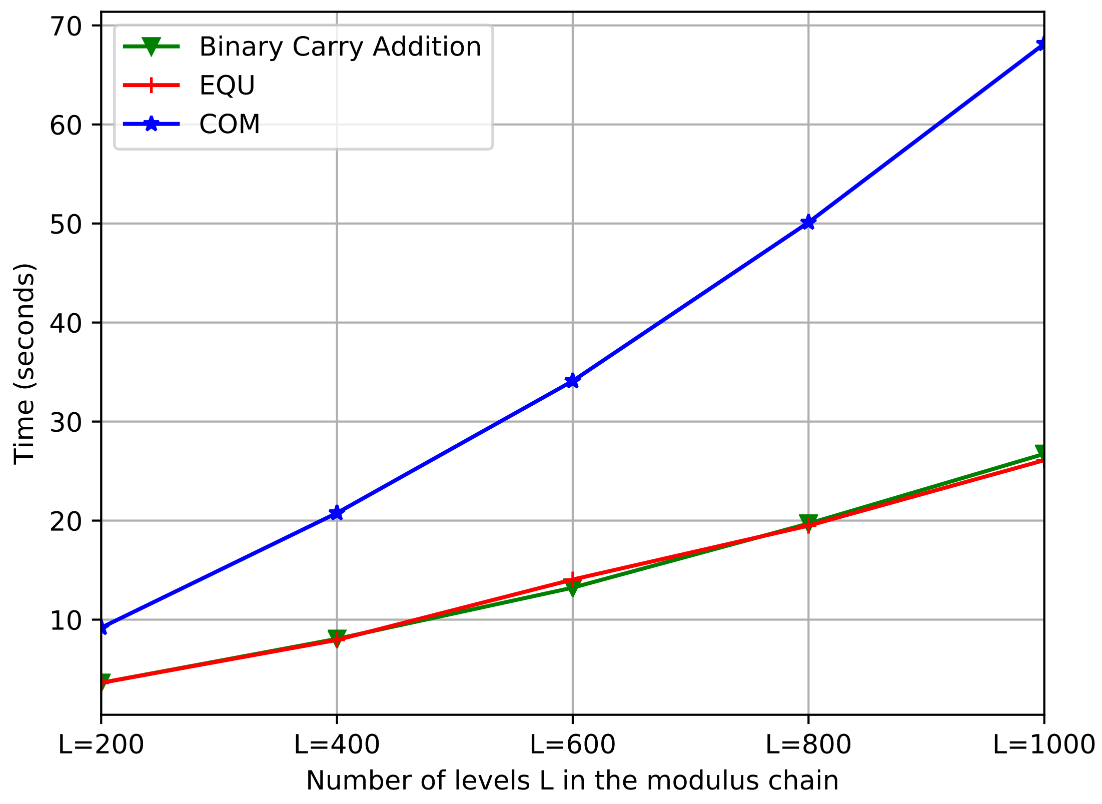


Figure 7.6: Number of Levels in the modulus chain vs Time plot for Binary Carry Addition: ADD(α, β) function and EQU(α, β), COM(α, β) functions

In the above plot i.e in Fig. 7.6 we have plotted the time taken by the functions ADD(α, β), EQU(α, β), COM(α, β) for the various number of levels i.e the length of modulus chain. We can clearly see that if we want to increase the depth of our circuit then the time taken to compute functions also increases.

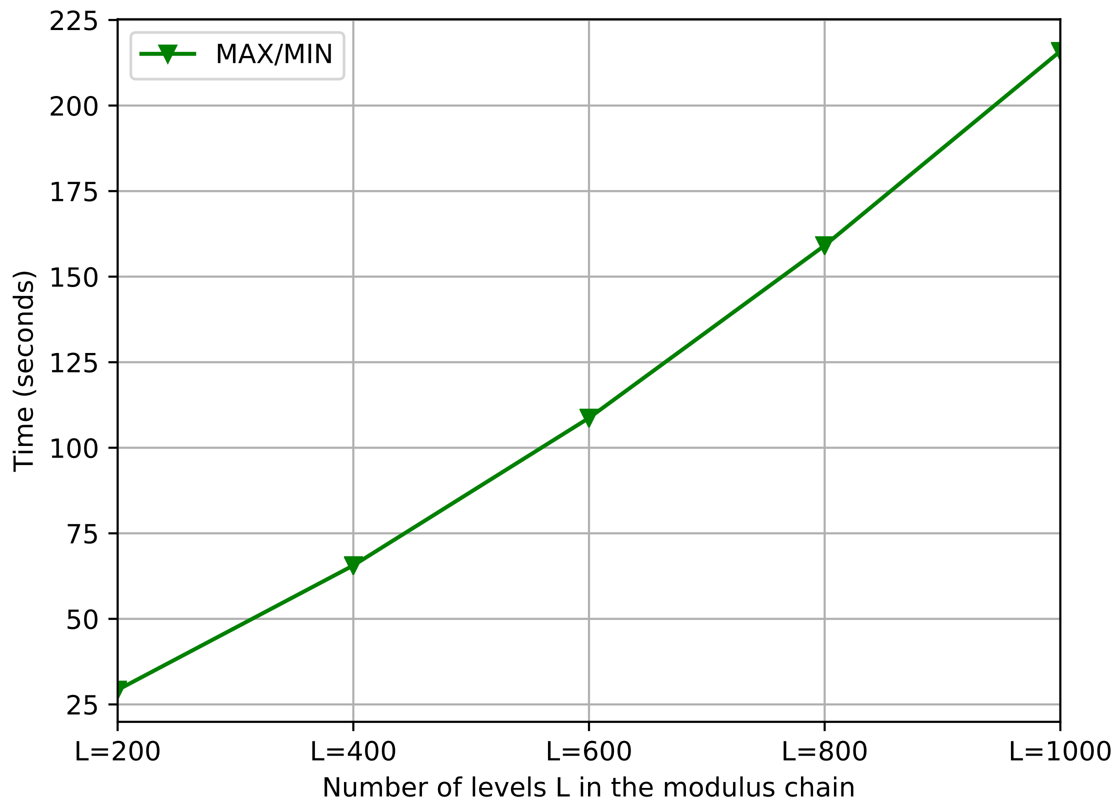


Figure 7.7: Number of Levels in the modulus chain vs Time plot for the MAX/MIN(α, β) functions

In the above plot i.e in Fig. 7.7 we have plotted the time taken by the function MAX(α, β)/MIN(α, β) for the various number of levels i.e the length of modulus chain. We can see again that if we want to increase the depth of our circuit then the time taken to compute functions also increases.

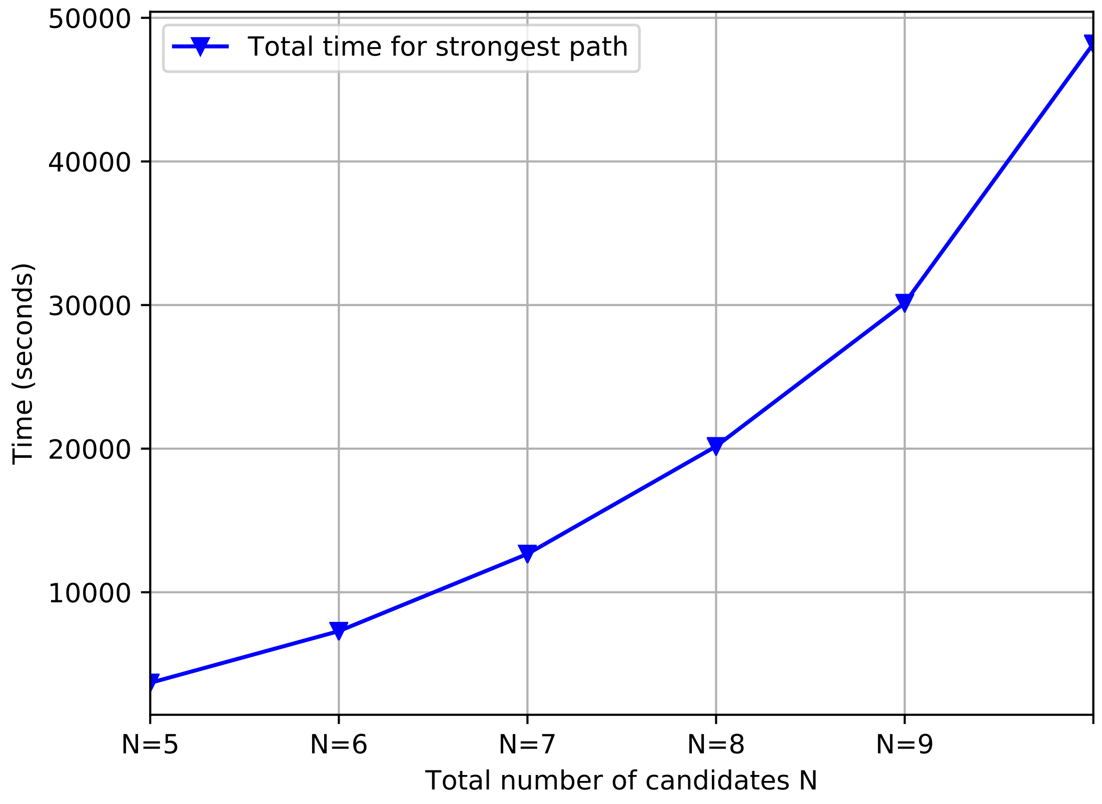


Figure 7.8: Number of candidates vs Time plot for the private computation of strongest path in Schulze voting method for $L = 200$

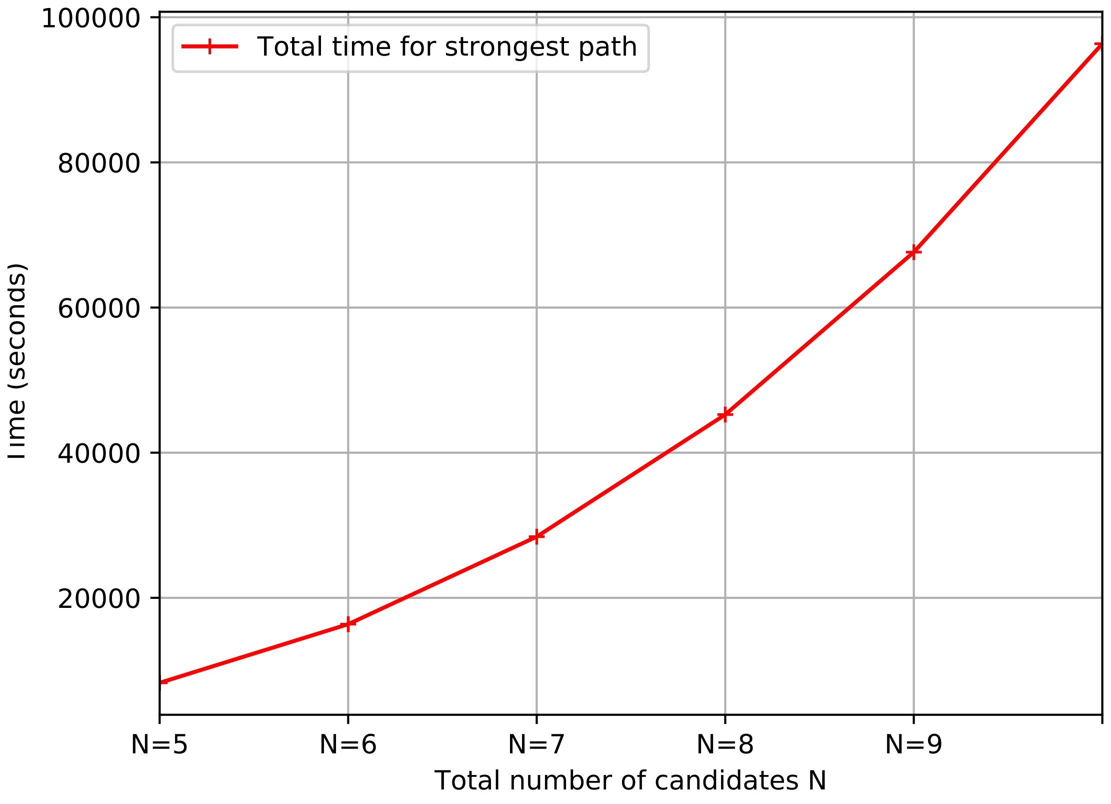


Figure 7.9

In the above plots i.e in Fig. 7.8 and Fig. 7.9 we have plotted the estimated time taken to compute the strongest path in the schulze voting method as shown in algorithm given in Fig. 7.5 for the various number of candidates. We can see again that if we increase the candidates in the election the time taken to compute the strongest path increases substantially. Since the time consumed increases too much and the depth of the circuit is not required to be too big we present here the results for only two values of levels i.e $L = 200$ and $L = 400$.

L	Public key size (Mb)	Secret key size (Mb)
200	6.3	6.6
400	12.6	13.1
600	18.7	19.5
800	25.1	26.1
1000	31.2	32.6

Table 7.3: Size of the public key and secret key for different values of L

We also present the values for the size of the secret key and public key used for encryption and decryption when using the parameters specified in the Table 7.3.

Chapter 8

Software and Hardware Requirements

1. C++ (gcc version 7.3.0 or higher)
2. NTL mathematical library (version 10.0.0 or higher)
3. Linux or Windows
4. x86 – 64 CPU
5. HElib
6. RAM 4 GB or above

Chapter 9

Conclusion

In this paper, we have provided a novel way to conduct condorcet voting in a private way. We have designed and provided an algorithm for computing the strongest paths in a directed weighted have provided an open source for the same for the research community to use. Through examples and actual implementations, the paper presents the challenges and a better understanding about computing on encrypting data using FHE. The paper also discusses the effect of different parameters on the time taken and the memory usage while computing on encrypted data. To the best of our knowledge, this is the first effort to homomorphically perform the Schulze voting method.

Chapter 10

Future Scope

In the work done in this thesis we have limited ourselves to the homomorphic encryption aspects of the Schulze voting. Also we have limited ourselves to the computation of strongest path only. A possible direction of future work that stems from our work can be to propose a full fledged private e-voting scheme with verifiability and additional security functionalities throguh secret sharing schemes. Another direction of work can be to implement the building blocks that is COM(), EQU() and MAX()/MIN() functions using some other cryptoscheme in a more efficient manner or to experiment with the different HElib parameters.

References

[1] C. Gentry, “A fully homomorphic encryption scheme,” Ph.D. dissertation, Stanford University, 2009, http://crypto.stanford.edu/craig.

[2] “How counting of votes is done in india,” https://www.mapsofindia.com/my-india/india/election-results-2017-how-counting-of-votes-is-done-in-india, accessed: 2019-04-15.

[3] M. Hirt and K. Sako, “Efficient receipt-free voting based on homomorphic encryption,” in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2000, pp. 539–556.

[4] D. Sandler, K. Derr, and D. S. Wallach, “Votebox: A tamper-evident, verifiable electronic voting system.” in *USENIX Security Symposium*, vol. 4, no. 0, 2008, p. 87.

[5] M. Schulze, “A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method,” *Social Choice and Welfare*, vol. 36, no. 2, pp. 267–303, 2011.

[6] N. P. Smart and F. Vercauteren, “Fully homomorphic encryption with relatively small key and ciphertext sizes,” in *International Workshop on Public Key Cryptography*. Springer, 2010, pp. 420–443.

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

[8] Z. Brakerski and V. Vaikuntanathan, “Efficient Fully Homomorphic Encryption from (Standard) Learning With Errors,” Cryptology ePrint Archive, Report 2011/344, 2011, http://eprint.iacr.org/.

[9] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “Fully homomorphic encryption without bootstrapping,” Cryptology ePrint Archive, Report 2011/277, 2011, http://eprint.iacr.org/.

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

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

[12] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, "Tfhe: Fast fully homomorphic encryption over the torus." *IACR Cryptology ePrint Archive*, vol. 2018, p. 421, 2018.

[13] S. Halevi and V. Shoup, "HElib," 2014, https://github.com/shaih/HElib.

[14] ——, "Algorithms in HElib," Cryptology ePrint Archive, Report 2014/106, 2014, http://eprint.iacr.org/.

[15] L. Ducas and D. Micciancio, "Fhew: bootstrapping homomorphic encryption in less than a second," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 617–640.

[16] J. H. Cheon, M. Kim, and K. Lauter, "Homomorphic computation of edit distance," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 194–212.

[17] G. S. Cetin, Y. Doröz, B. Sunar, and E. Savas, "Low depth circuits for efficient homomorphic sorting." *IACR Cryptology ePrint Archive*, vol. 2015, p. 274, 2015.

[18] O. Kocabas and T. Soyata, "Utilizing homomorphic encryption to implement secure and private medical cloud computing," in *2015 IEEE 8th International Conference on Cloud Computing*. IEEE, 2015, pp. 540–547.

[19] A. Chatterjee and I. Sengupta, "Translating algorithms to handle fully homomorphic encrypted data on the cloud," *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 287–300, 2018.

[20] J. Hoffstein, J. Pipher, J. H. Silverman, and J. H. Silverman, *An introduction to mathematical cryptography*. Springer, 2008, vol. 1.

[21] J. Beachy and W. Blair, *Abstract Algebra: Third Edition*. Waveland Press, 2006.

[22] B. Viray, “Ring homomorphism and the isomorphism theorems,” https://sites.math.washington.edu/~bviray/teaching/RingHomomorphismsAndIsomorphisms.pdf.

[23] J. O’Connor, “Ring homomorphism and isomorphism,” Lecture MT4517, http://www-history.mcs.st-and.ac.uk/~john/MT4517/Lectures/L7.html.

[24] W. Stein, “Algebraic number theory, a computational approach,” 2012.

[25] O. Regev, “On lattices, learning with errors, random linear codes, and cryptography,” in *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, ser. STOC ’05. ACM, 2005, pp. 84–93.

[26] ——, “The learning with errors problem,” 2010, http://www.cims.nyu.edu/~regev/papers/lwesurvey.pdf.

[27] A. Lange, “An overview of homomorphic encryption,” Class notes, 2011, http://www.cs.rit.edu/~arl9577/crypto/alange-presentation.pdf.

[28] G. Pötzelsberger, “Kv web security: Applications of homomorphic encryption,” 2013, http://www.fim.uni-linz.ac.at/Lva/Web_Security/Abgaben/Poetzelsberger-Homomorphic.pdf.

[29] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, ser. EUROCRYPT’99. Springer-Verlag, 1999, pp. 223–238.

[30] S. Goldwasser and S. Micali, “Probabilistic encryption & how to play mental poker keeping secret all partial information,” in *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, ser. STOC ’82. ACM, 1982, pp. 365–377.

[31] R. L. Rivest, L. Adleman, and M. L. Dertouzos, “On data banks and privacy homomorphisms,” *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[32] B. Hayes, “A new form of encryption allows you to compute with data you cannot read,” 2012, http://www.americanscientist.org/issues/pub/2012/5/alice-and-bob-in-cipherspace/99999.

[33] C. Stuntz, “What is Homomorphic Encryption, and Why Should I Care?” 2010, http://blogs.teamb.com/craigstuntz/2010/03/18/38566/.

[34] N. R. D. staff, “Securing the cloud with homomorphic encryption,” 2014, https://www.nsa.gov/research/tnw/tnw203/articles/pdfs/tnw203_article5.pdf.

[35] Wikipedia, "Schulze method," https://en.wikipedia.org/wiki/Schulze_method.

[36] M. Togan and C. Pleşca, "Comparison-based computations over fully homomorphic encrypted data," in *2014 10th International Conference on Communications (COMM)*. IEEE, 2014, pp. 1–6.