إقـــرار

أنا الموقع أدناه مقدم الرسالة التي تحمل العنوان :**استخدام حلول التشفير المثلي في أنظمة الاقتراع الإلكتروني.**

Using homomorphic cryptographic solutions on E-voting Systems.

أقر أن ما اشتملت عليه هذه الرسالة إنما هو نتاج جهدي الخاص، باستثناء ما تمت الاشارة إليه حيثما ورد، وإن هذه الرسالة ككل أو أي جزء منها لم يقدم من قبل لنيل درجة أو لقب علمي أو بحثي لدى أي مؤسسة تعليمية أو بحثية أخرى.

Declaration

The work provided in this thesis, unless otherwise referenced, is the researcher's own work, and not has been submitted elsewhere for any other degree or qualification.

Student's :**Ahmed Abd Allh Abu Aziz**

Signature:

Date:19/4/2015

اسم الطالب: أحمد عبد الله أبو عزيز

التوقيع:

التاريخ:19/4/2015

`

The Islamic University Gaza
Higher Education Deanship
Faculty of Engineering
Department of Computer Engineering
Master of Computer Engineering

الجامعة الإسلامية – غزة
عمادة الدراسات العليا
كلية الهندسة
قسم هندسة الحاسوب
ماجستير هندسة الحاسوب

استخدام حلول التشفير المثلي في أنظمة الاقتراع الالكتروني.

Using homomorphic cryptographic solutions on E-voting Systems.

Submitted by:

Eng. Ahmed Abu Aziz

Supervised by:

Dr. Hasan Qunoo

A Thesis Submitted in Partial Fulfillment of Requirements for the Degree of Master in Computer Engineering.

1436 هـ -2015 م

بسم الله الرحمن الرحيم

الجامعة الإسلامية – غزة
The Islamic University - Gaza

مكتب نائب الرئيس للبحث العلمي والدراسات العليا هاتف داخلي 1150

الرقم Ref ...../35/غ

التاريخ Date 2014/03/25

نتيجة الحكم على أطروحة ماجستير

بناءً على موافقة شئون البحث العلمي والدراسات العليا بالجامعة الإسلامية بغزة على تشكيل لجنة الحكم على أطروحة الباحث/ **أحمد عبدالله سالم أبوعزيز** لنيل درجة الماجستير في **كلية *الهندسة*** قسم **هندسة الحاسوب** وموضوعها:

استخدام حلول التشفير المثلي في أنظمة الاقتراع والانتخابات الالكترونية

Using homomorphic cryptographic solutions on E-Voting Systems

وبعد المناقشة التي تمت اليوم الأربعاء 05 جمادى الآخر 1436هـ، الموافق 2014/03/25م الساعة الثانية مساءً، اجتمعت لجنة الحكم على الأطروحة والمكونة من:

د. حسن نجيب قنوع	مشرفاً ورئيساً
د. أيمن أحمد أبو سمرة	مناقشاً داخلياً
د. عبد الحميد بشير زغبر	مناقشاً خارجيًا

وبعد المداولة أوصت اللجنة بمنح الباحث درجة الماجستير في **كلية *الهندسة*/ قسم** وموضوعها: **هندسة الحاسوب.**

واللجنة إذ تمنحه هذه الدرجة فإنها توصيه بتقوى الله ولزوم طاعته وأن يسخر علمه في خدمة دينه ووطنه.

والله ولي التوفيق،،،

مساعد نائب الرئيس للبحث العلمي وللدراسات العليا

أ.د. فؤاد علي العاجز

P.O. Box 108, Rimal, Gaza, Palestine fax: +970 (8) 286 0800 فاكس Tel: +970 (8) 286 0700 ص.ب. 108 الرمال، غزة، فلسطين هاتف
public@iugaza.edu.ps www.iugaza.edu.ps

`

DEDICATION

This thesis is dedicated to:

The sake of Allah, my Creator and my Master;

My great teacher and messenger, Mohammed (May Allah bless and grant him), who taught us the purpose of life;

My homeland Palestine;

The great martyrs and prisoners, the symbol of sacrifice;

The Islamic University; my second magnificent home;

My great parents, who never stop giving of themselves in countless ways;

My dearest wife, who leads me through the valley of darkness with light of hope and support,

My beloved brothers;

My beloved kids: Osama, and Waseem, whom I can't force myself to stop loving.

To all my family, the symbol of love and giving;

My friends who encourage and support me;

All the people in my life who touch my heart;

I dedicate this research.

`

ACKNOWLEDGEMENT

I would like to express my special appreciation and thanks to my advisor Dr. Hasan Qunoo, you have been a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow as a researcher. I would especially like to thank Dr. Aiman A. Samra for encouraging me and helping to complete this research, he always offered his advices to improve my work. I would also like to thank my committee members, Dr. Aiman A. Samra and Dr. Abd Al Hameed Zoghbr for serving as my committee members even in hardship. I also want to thank you for letting my defense be an enjoyable moment, and for your brilliant comments and suggestions, thanks to you.

A special thanks to my family. Words cannot express how grateful I am to my mother and my father for all of the sacrifices that you've made on my behalf. Your prayer for me was what sustained me thus far. I would also like to thank all of my brothers and friends who supported me, and supported me to achieve my goal. At the end, I would like to express appreciation to my dear wife, who spent sleepless nights with and was always my support in the moments when there was no one to answer my queries.

`

ملخص الدراسة

ظهر التشفير المثلي كحل جديد في أنظمة التصويت الالكتروني، في هذا البحث تم استخدام حلول التشفير المثلي التام لتصميم وتنفيذ نظام تصويت الكتروني، بهدف اختبار ودراسة أداء وقابلية التشفير المثلي التام للتطبيق في أنظمة تصويت الكتروني حقيقية. العديد من أنظمة التصويت الالكتروني التي تعتمد التشفير المثلي مبنية على التشفير المثلي الجمعي أو الضربي. في هذا البحث تم استخدام التشفير المثلي التام لتوفير كلا العمليتين الجمع والضرب، مما سهل انشاء الاثبات الصفر المعرفة غير التفاعلي. وقد حقق نظام التصويت المصمم بناء على التشفير المثلي التام أهم خصائص أنظمة التصويت عبر الانترنت الأمنية مثل: الأهلية والخصوصية والدقة والتحقق والانصاف والحرية وعدم الاجبار وحرية النزاع والمتانة والتوسع وقابلية التطبيق. بالإضافة الى أحد أهم خصائص نظام التصويت عبر الانترنت المطبق هو إمكانية تطبيقه عبر تقنية السحاب، مع الحفاظ على الخصائص الأمنية.

تم التنفيذ باستخدام مكتبة التشفير المثلي التام HELib، استنادا الى مخطط التشفير المثلي التام BGV، تمت كتابة الكود باستخدام لغة C++. ينقسم النظام المصمم الى ثلاثة أقسام: خادم التحقق وخادم التصويت والمصوتين. تم استخدام خصائص الجمع والضرب للتشفير المثلي التام للتحقق من صحة بنية الصوت كإثبات صفري المعرفة غير تفاعلي NIZKP، ولحساب ناتج عملية التصويت بطريقة مشفرة. حيث أظهرت النتائج أن نظام التصويت عبر المنفذ قابل للتطبيق على عدد كبير من الناخبين يصل الى 10 مليون ناخب.

`

ABSTRACT

Homomorphic Cryptography raised as a new solution used in electronic voting systems. In this thesis, Fully Homomorphic used to design and implement an e-voting system, for the purpose of examination and studying the applicability in real systems and performance of fully homomorphic encryption in e-voting systems. Most of homomorphic cryptography e-voting systems based on additive or multiplicative homomorphic encryption. In this thesis, fully homomorphic encryption used to provide both operations additive and multiplication, which ease the demonstration of none interactive zero knowledge proof NIZKP. The implemented e-voting systems achieved most of the important security issues of the internet-voting systems such as, eligibility, privacy, accuracy, verifiability, fairness, receipt-freeness, incoercibility, dispute-freeness, robustness, scalability and practicality. One of the most important properties of the implemented internet voting system its applicability to work on cloud infrastructure, while preserving its security characteristics.

The implementation is done using homomorphic encryption library HELib, which based on BGV fully homomorphic encryption scheme, written in C++ language. The implemented voting systems divided into three parts: Authentication Server, Voting Server and Voters. Addition and multiplication properties of fully homomorphic encryption used to verify the correctness of vote structure as a NIZKP, and for calculating the results of the voting process in an encrypted way. The results show that the implemented internet voting system is secure and applicable for a large number of voters up to 10 million voter.

`

List of Contents

`

`

LIST OF ABBREVIATIONS

FHE	Fully Homomorphic Encryption
HE	Homomorphic Encryption
SHE	Somewhat Homomorphic Encryption
LFHE	Leveled Fully Homomorphic Encryption
LWE	Learning With Error
RLWE	Ring Learning With Error
SIMD	Single Instruction Multiple Data
BGV	Brakerski Gentry Vaikuntanathan scheme
HELib	Homomorphic Encryption Library
MPC	Multiparty Computaion
WSN	Wireless Sensor Network
ZKP	Zero Knowledge Proof
NIZKP	None Interactive Zero Knowledge Proof
AS	Authentication Server
VS	Voting Server
BB	Bulletin Board
V	Voter

`

LIST OF FIGURES

`

LIST OF TABLES

`

CHAPTER 1: INTRODUCTION

Voting is a fundamental decision making instrument in any consensus-based society and democracy depends on the proper administration of popular elections. In any election, there exists a set of requirements among which any voters should be confident that their intents was correctly captured and no modification done to them votes. In addition, all eligible votes were correctly tallied. On the other side, the voting system should ensure that each vote done in the wright way and voter coercion is unlikely. These conflicting requirements presents a significant challenge. The changing from the traditional paper based voting methods used in many countries into electronic election systems, removes such challenge. The challenge transferred to build secure voting systems that able to run in real life situations and preserve privacy and anonymity for voters, and have verifiability properties to prove a correct tally of votes.

1.1 E-voting solutions

Many countries have begun to use modern technology into their voting procedures. Some solutions presented as Electronic voting systems (e-voting) as a secure method to perform secure elections. e-voting refers to the use of computers or computerized voting equipment to cast votes in election [1]. e-voting is an interdisciplinary subject and should studied from different domains, such as software engineering, cryptography, network security, politics, low, economics and social science. Mostly e-voting is known as a challenging topic in cryptography, because the need to achieved privacy, anonymity and vote encryption. Many e-voting systems proposed among the last decade, a lot of them based of complicated encryption schemes and other based on mix net model, blind signature model and homomorphic encryption model. Homomorphic Cryptography raised as a new solution used in electronic voting systems. E-voting systems tries to resolve many security issues such as eligibility, privacy, accuracy, verifiability, fairness, receipt-freeness, incoercibility, dispute-freeness, robustness, scalability and practicality.

1.2 Conventional Cryptography

Cryptographic solutions provide methods of storing or transferring data in a secure way, the amount of data generated is growing in a huge way, while cloud services is suitable

`

solution for storing such huge amount of data. Since cloud technologies is one of the most cost-saving and scalable solution for processing and saving large data, the need to process encrypted data stored in the cloud become more insistent.

Cryptographic techniques can separate into two general forms, Symmetric and Asymmetric encryption:

In symmetric encryption a common secret key defined between sender and receiver, the same key is used for encryption $E(m, k)$ and decryption $D(c, k)$ process, where m is the message and c is the generated ciphertext after encryption. The original message could retrieved after decrypting cipher using the secret key.

In asymmetric encryption, private and public keys generated, user can share his public key to the public, any sender can use the public key to encrypt a message $E(m, pk)$, then the receiver can decrypt using his private key $D(c, sk)$. All public key cryptography depends on numeric theory and modular operations, this provides a powerful property called homomorphism, and thus preserves group operations performed on ciphertexts, add, multiply or both can made on two ciphertext to calculate the result, which will be the same result if this operation performed on plaintext.

1.3 Homomorphic Cryptography

Homomorphism property preserves new secure method to perform a group of operations on ciphertexts in untrusted third party without knowledge of any secret information. The ability to perform simple computation on ciphertexts leads to a lot of applications and security protocols, but the complicated structure of homomorphic cryptosystems limits applicability in some protocols that need fast computation, and it still applicable to some protocols concern in security section 2.1 and section 2.2 describes in details the homomorphic encryption.

1.4 Problem Statement

Study the applicability of using fully homomorphic encryption in e-voting systems based on cloud infrastructure through designing an efficient and practical e-voting system depends on fully homomorphic encryption.

`

1.5 Research Aim

This research aims to study the efficiency and applicability of using fully homomorphic encryption in e-voting systems and their effectiveness to be built in untrusted platforms such as cloud infrastructure. The thesis also studies, performance, security and privacy issues of e-voting system.

1.6 Research Importance

The importance of this research that it discusses the practical applications of homomorphic encryption. It also studies the applications that could be implemented using fully homomorphic encryption and how much it is corresponding the infrastructure and the properties, security issues and performance of cloud environment.

In this topic, an open question raised "Can Homomorphic encryption be efficient enough to be practical?" Our thesis addresses this open problem. The proposed protocol clarifies a method of using fully homomorphic encryption to develop a practical voting system and implements a new non-interactive zero knowledge proof method.

1.7 Research Structure

Chapter 2 introduce the homographic encryption, then presents a literature review of previous fully homomorphic encryption schemas, properties, underlying principles and limitations. It also presents a survey of the most possible applications of homomorphic encryption. It gives a brief explanation of what is fully homomorphic encryption, how we can use it, and what practical implementations done using FHE. This chapter also focus on e-voting systems and give a brief explanation of the previous voting systems.

Chapter 3 presents the research methodology, the scope, implementation tools and testing methods.

Chapter 4 presents the implemented e-voting system using fully homomorphic encryption, and discuss a designed method of non-interactive zero knowledge proofs. It also describes the presentation method used and NIZKP.

`

Chapter 5 presents the structure of the implemented voting system, and describes the programming properties of each part of the system.

Chapter 6 presents analysis and results of the implemented voting system, it shows traffic analysis, performance analysis and stored data analysis.

Chapter 7 in the final chapter a conclusion and the future developments described.

`

CHAPTER 2: LITERATURE REVIEW

This research study an old problem in cryptography called a privacy homomorphism. It was introduced by Rivest, Adleman and Dertouzous [3] after the invention of RSA [2] which is a multiplicative homomorphic encryption schema.

2.1 Homomorphic cryptography.

If the RSA public key *pk = (N,e)*, then encryption of message *x* is given by $E(m_i) = m_i^e mod\ N$, then the homomorphism property is $\prod_i E(m_i) = (\prod_i m_i)^e mod\ N$ in other words:

$$E(m_1).E(m_2) = m_1^e m_2^e\ mod\ N = (m_1 m_2)^e mod\ N = E(m_1.m_2) \quad (1)$$

This property led Rives et al. [3] to think about what if we have a schema that is fully homomorphic: a schema $\boldsymbol{\varepsilon}$ have an efficient **Evaluate**ε algorithm that can evaluate any circuit C contains any operation not just multiplication, for any public key pk, where:

$\boldsymbol{c_i} = \textbf{Encrypt}\varepsilon(pk, m_i)$ Gives:

$$\boldsymbol{c} \leftarrow \boldsymbol{Evaluate}\varepsilon(pk, C, \boldsymbol{c_1}, \dots, \boldsymbol{c_t}), \quad (2)$$

A vailed encryption of $C(m_1, \dots., m_t)$ under pk. This can arbitrarily compute on encrypted data, so there many applications could be applied using this theory, such as query, calculate and write to data without decryption, any operation could be applied while it could be expressed efficiently as a circuit C.

Suppose we have two values $m_1 = 5$, $m_2 = 9$, we want to do an arithmetic operation multiplication in an untrusted party without revealing any value of m_1, m_2 result r. Encryption done on m_1, m_2 using homomorphic encryption schema ε, with generated public key *pk.*

$E_{pk}(5) = 83$

$E_{pk}(9) = 57$

`

Which indicate for c_1, c_2 respectively, the circuit C need to do in this example is multiplication operation. c_1, c_2 sent to the untrusted third party to be evaluated through **Evaluate**ε function which input C, c_1, c_2 *and* pk.

$$r = c_1 * c_2 = 83 * 57 = 4731$$

The result sent back after calculation and decryption done using generated private key *pk*.

$$D_{sk(r)} = D_{sk}(4731) = 45$$

Decryption must give the same result of the operation if done in clear, this powerful property can work for more complicated circuits, along with other operations based on addition and multiplication.

Figure 2.1 shows the general evaluation process, while the delegator is any user want to use the resources of third party evaluator without revealing any information about message *m* and result *r*. Evaluator could be cloud server, public processing infrastructure or even any untrusted PC

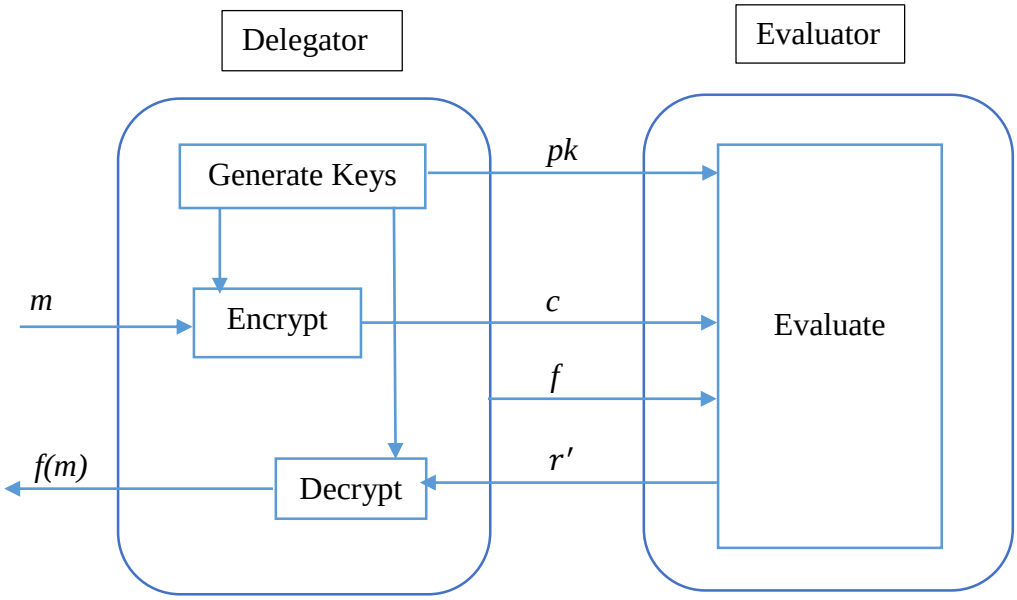


Figure 2.1Homomorphic Encryption Evaluation.

`

The function f represents an arithmetic circuit or a Boolean circuit the scheme called circuit-based, if function f defined as a mathematical function, the scheme called non-circuit based.

Next section discusses in more details homomorphic encryption properties, definitions and lists many of famous fully homomorphic encryption schemas.

2.2 Homomorphic encryption schemes

2.2.1 Additive Homomorphic Encryption scheme:

2.2.1.1Paillier scheme

Paillier crypto system invented by Pascal Paillier in 1999 [4], that relies on the Decisional Composite Residuosity Assumption. It is a probabilistic asymmetric algorithm for public key cryptography, which computation considered difficult. This cryptosystem based on modular operations, with the property of homomorphism over additive.

Note that if we have two ciphertexts $E(m_0, r_0)$ and $E(m_1, r_1)$ which are encryptions of m_0, m_1 respectively, then:

$$D(E(m_0, r_0).E(m_1, r_1) \ mod \ n^2) = m_0 + m_1 \ mod \ n^2 \qquad (3)$$

So, the product of two ciphertexts will decrypt to the sum of their corresponding plaintexts, given encryption of m_0, m_1 we get the encryption of m_0+m_1 without having to know the secret key. Paillier cryptosystem used widely in voting systems applications due to additive property.

2.2.2 Multiplicative Homomorphic Encryption scheme

2.2.2.1RSA scheme

RSA is one of the most practical and popular cryptosystems for asymmetric encryption, it was introduced by Rivest, Shamir and Adleman in 1978 [3]. It is one of the first homomorphic cryptosystems, and the most widely used public key cryptosystem. This cryptosystem is based on the practical difficulty of factoring the product of two large prime

`

numbers, which used to provide both secrecy and digital signatures and key exchange protocols.

RSA has a multiplicative homomorphic property, which means a multiplication operation can done on encrypted messages without revealing their underlying information. If the RSA public key is modulus *n* and exponent *e*, then the encryption of a message *m* given by $E(m) = m^e \ mod \ n$. The homomorphic property then is:

$$E(m_0).E(m_1) = \ m_0^e \ m_1^e \ mod \ n = (m_0 m_1)^e \ mod \ n = E(m_0.m_1) \qquad (4)$$

So, decryption of result using secret key gives the result of multiplication of messages.

2.2.3 Fully homomorphic encryption scheme

2.2.3.1 Gentry's Scheme

Gentry described the first Fully Homomorphic Encryption scheme in 2009 [5] [6], which considered as a breakthrough. It solved an old problem of homomorphic cryptosystems, which provide addition and multiplication on ciphertexts.

Gentry derived a new method for solving this problem, by building a fully homomorphic scheme form "*somewhat homomorphic scheme"*, instead of directly creating a fully homomorphic scheme. Somewhat schema was only able to evaluate low degree polynomials on the encrypted data, it can perform a limited number of addition and multiplication operations on ciphertexts. Next he needs to "*squash*" the decryption procedure so that it can be expressed as a low-degree polynomial in the bits of the ciphertext and the secret key (a circuit of small depth). This squashing needed because every ciphertext has a noise component and any homomorphic operation applied to ciphertexts increases the noise in the resulting ciphertext. Once this noise reaches a certain threshold the resulting ciphertext does not decrypt correctly anymore; this limits the degree of the polynomial that can be applied to ciphertexts [7].

Gentry applied a breakthrough idea by evaluating the decryption of polynomial not on the bits of ciphertext and secret key directly as in regular, but he performs it homomorphically on the encryption of those ciphertexts and secret key. Instead of recovering the plaintext, it

`

gets an encryption of bits for ciphertext, but with less noise if the polynomial degree small enough in the ciphertext and this becomes the ciphertext for the original plaintext. This process called "*ciphertext refresh*" procedure.

The refresh process on a ciphertext make the refreshed ciphertext applicable for the homomorphic operation (addition or multiplication), while it's not possible for the original ciphertext due the noise threshold. Using this procedure the number of permissible homomorphic operations becomes unlimited and we get a fully homomorphic encryption scheme.

Finally, he applied a "bootstrapping" transformation to obtain fully homomorphic scheme. The crucial point in this process is to obtain a scheme that can evaluate polynomials of high-enough degree, and at the same time has decryption procedure that can be expressed as a polynomial of low-enough degree. Once the degree of polynomials that can be evaluated by the scheme exceeds the degree of the decryption polynomial (times two), the scheme is called "bootstrappable" and it can then be converted into a fully homomorphic scheme [8].

2.2.3.2 Implementation of Gentry's blueprint - Smart-Vercauteren

The first attempt to implement Gentry's scheme was made in 2010 by Smart and Vercauteren [9], they used a variant based on principal ideal lattices and requiring that the determinant of the lattice be a prime number. Such lattices can be represented implicitly by just two integers (regardless of their dimension), and moreover Smart and Vercauteren described a decryption method where the secret key is represented by a single integer.

Smart and Vercauteren were able to implement the underlying somewhat homomorphic scheme. But they were not able to support large enough parameters to make Gentry's squashing technique go through, because that required a lattice dimension of at least n = 227, whereas due to the prime determinant requirement they could not generate keys for dimensions n > 2048, which is essential for security purposes. As a result they could not obtain a bootstrappable scheme or a fully homomorphic scheme.

`

2.2.3.3 Gentry-Halevi Scheme

Gentry and Halevi described the first implementation of Gentry's scheme [8]. They follow the same direction as Smart and Vercauteren. They make some optimizations to implement the bootstrapping functionality, which not implemented by Smart and Vercauteren. The main optimization is a key-generation method, for the underlying somewhat homomorphic encryption, that does not require full polynomial inversion. They eliminate the requirement that the determinant is a prime.

Additionally, they present many clever optimizations that reduce the asymptotic complexity and practically reducing the time from many hours/days to a few seconds/minutes. The authors of [8] report that for an optimized implementation on a high-end workstation, key generation takes 2.2 hours, encryption takes 3 minutes, and ciphertext refresh takes 30 minutes.

2.2.3.4 Improvements on Gentry's scheme

2.2.3.4.1 Stehle-Steinfeld optimizations

Stehle and Steinfeld described two improvements [10] on Gentry's fully homomorphic scheme based on ideal lattices and its analysis. They provide a more aggressive analysis of one of the hardness assumptions (the one related to the Sparse Subset Sum Problem) and introduced a probabilistic decryption algorithm that can be implemented with an algebraic circuit of low multiplicative degree. Combined, these improvements lead to a faster fully homomorphic scheme. These improvements also apply to the fully homomorphic schemes of Smart and Vercauteren [9] and van Dijk et al [11].

2.2.3.4.2 SIMD Gentry optimization

In [9] Smart and Vercauteren presented a variant of Gentry's fully homomorphic scheme and mentioned that the scheme could support SIMD style operations. SIMD means simple instruction mutable data. While Gentry's original schema [5] was just able to perform encryption and decryption on a plaintext of one bit length.

`

Gentry and Halevi [8] addressed the slowness of key generation process of the Smart–Vercauteren system [9], but their key gen eration method excluded the SIMD style operation offered by Smart and Vercauteren.

In [12] Smart and Vercauteren show how to select parameters to enable such SIMD operations, to implement Gentry and Halevi scheme. Moreover, how to obtain a somewhat homomorphic scheme supporting SIMD operations. This somewhat homomorphic scheme can be made fully homomorphic in a naive way by recrypting all data elements separately. This result a substantial speed-up. This make performance 2.4 times faster than the standard FHE scheme and the ciphertext size reduced by a factor 1/72.

2.2.3.4.3 Gentry-Halevi without squashing

Gentry and Halevi describe in [13] a new approach to construct a fully homomorphic scheme encryption without the need to squash process. Previous schemes follows Gentry's blueprints in first constructing somewhat homomorphic encryption scheme, and next squash the decryption circuit until it is simple enough to be handled within the homomorphic capacity of the somewhat homomorphic encryption scheme. Finally perform bootstrapping to get fully homomorphic encryption scheme.

Gentry and Halevi show in their approach constructing of fully homomorphic encryption schema as a hybrid of somewhat homomorphic encryption scheme and multiplicatively homomorphic encryption scheme. This construction eliminate the need for squashing step. But it still using bootstrapping step to get fully homomorphic encryption scheme.

The main technique is to express the decryption function of somewhat schemes as a depth-3 ($\sum\prod\sum$) arithmetic circuit of a particular form. When evaluating this circuit homomorphically scheme temporarily switch to a multiplicatively homomorphic encryption scheme to handle the multiplication part. Due to the special form of the circuit, the switch to the multiplicative scheme can be done without having to evaluate anything homomorphically. Then the result translated back to the somewhat scheme by homomorphically evaluating the decryption function of the multiplicative scheme. The

`

somewhat homomorphic scheme only needs to be capable of evaluating the multiplicative scheme's decryption function, not its own decryption function. This avoids the circularity that necessitated squashing in the original blueprint.

2.2.3.4.4 Gentry-Halevi-Smart scheme

Gentry, Halevi and Smart [14] solved the bottleneck in the bootstrapping process, which need to evaluate homomorphically the reduction of one integer modulo another. This is typically done by emulating a binary modular reduction circuit, using bit operations on the binary representation of integers. Gentry, Halevi and Smart present a simpler approach that bypasses the homomorphic modular-reduction bottleneck to some extent. The method is easier to describe and implement and is likely to be faster in practice. The scheme reduced the size of the public key, and work with SIMD homomorphic computations.

2.2.3.5 DGHV fully homomorphic scheme over the integers

DGHV fully homomorphic scheme over the integers described in [11] a fully homomorphic scheme, that constructed from very simple somewhat homomorphic encryption scheme using only elementary modular arithmetic. The somewhat homomorphic scheme merely uses addition and multiplication over the integers rather than working with ideal lattices over a polynomial ring.

As in Gentry's scheme the authors first describe a somewhat homomorphic scheme supporting a limited number of additions and multiplications over encrypted bits. Then they apply Gentry's "squash decryption" technique to get a bootstrappable scheme and then Gentry's "ciphertext refresh" procedure to get a fully homomorphic scheme.

The main appeal of the scheme (compared to the original Gentry's scheme) is its conceptual simplicity. However, the public-key was $\tilde{o}(\lambda^{10})$ in which is too large for any practical system.

The major achievement of DGVH over the original Gentry scheme, was that the plaintext consisted of integers rather than single bits leading a better blueprint improve upon [15].

`

2.2.3.5.1 DGHV shorter public key

Coron et al, [16] reduced the public key size to $\tilde{o}(\lambda^7)$ by encrypting with a quadratic form in the public key elements, instead of a linear form. They proved that the scheme remains semantically secure, based on a stronger variant of the approximate-GCD problem, already considered by van Dijk et al.

Coron et al, described also the first implementation of the resulting fully homomorphic scheme. Borrowing some optimizations from the Gentry-Halevi [8] implementation of Gentry's scheme, obtained roughly the same level of efficiency. This shows that fully homomorphic encryption can be implemented using simple arithmetic operations.

2.2.3.6 Learning With Error LWR- FHE

Gentry's blueprint suffers from many problems, which first all schemes based on squashing decryption, squashing use "sparse subset sum assumption" in decryption circuit. Also the large size of keys and ciphertext, the evaluation time per gate, time of encryption and decryption. All these reasons make a bottleneck in practical deployment of FHE.

A new series works address these concerns. Brakerski and Vaikuntanathan [17] show that (leveled) FHE can be based on the hardness of the much more standard "learning with error" (LWE) problem. LEW show that it is hard to solve various short vector problems on arbitrary (not ideal) lattices in the worst case.

In effect, Brakerski and Vaikuntanathan show how to obtain a direct construction of a bootstrappable encryption scheme without having to squash the decryption circuit and thus, without relying on the non-standard sparse subset sum assumption. This construction improves the previous works in firstly showing how somewhat homomorphic can based on LWE using a new re- linearization technique. While all the previous work relied on complexity assumptions related to ideals in various rings. Second, the show to avoid the "squashing paradigm" used in all previous works, by introducing a new dimension-modulus reduction technique, which shortens the ciphertexts and reduces the decryption complexity of our scheme, without introducing additional assumptions [18].

`

The scheme has very short ciphertexts and therefore used to construct an asymptotically efficient LWE-based single-server private information retrieval (PIR) protocol.

2.2.3.7 Brakerski-Gentry-Vaikuntanathan BGV scheme

Brakerski, Gentry and Vaikuntanathan in [2] [19] presented a new FHE scheme based on previous work of Brakerski and Vaikuntanathan in [17]. This scheme based on LWE problem and Ring LWE. They constructed a new way of leveled fully homomorphic encryption schemes (capable of evaluating arbitrary polynomial-size circuits), without Gentry's bootstrapping procedure. Instead of recryption, this new scheme uses other light weighted methods to refresh the ciphertexts to limit the growth of the noise so that the scheme can evaluate much deeper circuits. The recryption process will serve as an optimization to deal with over complicated circuits instead of a necessary for most circuits.

The most significant development of BGV compared to BV [17] is the use of well-known security assumptions based on RLWE, where the introduces RLWE over standard LWE introduce a more efficient fully homomorphic scheme. Also, a fully homomorphic encryption without the need for bootstrapping achieved using modulus switching.

2.2.3.7.1 Encryption Scheme

The general encryption of BGV scheme that can be instantiated to both LWE and RLWE. We will describe RLWE which used by HELib. The RLWE-based public key encryption scheme as follows. Most of the description and equations taken from [19] [20].

In general, homomorphic encryption scheme is a tuple (HE.*KeyGen*, HE.*Enc*, HE.*Dec*, HE.*Eval*) of probabilistic polynomial time algorithms. In BGV, the message space of the scheme will always be some ring R_M and our computational model will be arithmetic circuits over this ring (i.e. addition and multiplication gates).

1. HE.*KeyGen* takes the security parameter (and possibly other parameters of the scheme) and produces a secret key *sk* and a public key *pk*.

2. HE.*Enc* takes the public key *pk* a message *m* and produces a ciphertext *c*, which is the encryption of *m*.

`

3. HE.*Dec* takes the secret key *sk* and a ciphertext *c* and produces a message *m*.

4. HE.*Eval* takes the public key *pk*, an arithmetic circuit *f* over R_M, and ciphertexts c_1, ... ,c_ℓ where ℓ is the number of inputs to *f*, and outputs a ciphertext c_f.

Given the security parameter λ and an additional parameter μ, first choose a μ-bit modulus q. Where q an odd positive modulus $q = q(\lambda)$. For RLWE scheme, chose the degree $d = d(\lambda, \mu)$, a "noise" distribution $\chi = \chi(\lambda, \mu)$, let the "dimension" $n = [3 \log q]$. Let $R_q = \mathbb{Z}_q[x]/(f(x))$ with $f(x)$ a polynomial of degree d. $f(x) = x^d + 1$ and $d = d(\lambda)$ is a power of 2. To get the secret key, first draw $\boldsymbol{s}'$ uniformly from χ. The secret key is then

$$\boldsymbol{s} = (1, \boldsymbol{s}') \in R_q^2. \quad (5)$$

To get the public key, first generate vectors $\boldsymbol{A}' \leftarrow R_q^n$, $e \leftarrow \chi^n$, then set $b = -\boldsymbol{A}'\boldsymbol{s}' + 2e$. Set public key $A = (b | \boldsymbol{A}') \in R_q^{n \times 2}$. Note that $A.s = 2e$. (6)

Suppose $m \in \{0,1\}$ Is the bit we wanting to encrypt. To encrypt, we do the following:

1- Select a random $r \in R_2^n$ and expand the message $m = (m, 0) \in R_q^n$.

2- Output $= m + A^T r \in R_q^n$. (7)

According to $\text{RLWE}_{d,q,\chi}$ where χ is a uniform distribution over R_q, we can use this scheme a polynomial number of times with negligible probability that an adversary can guess $\boldsymbol{s}$.

To decrypt, do the following:

1- Compute $b' = [\langle \boldsymbol{c}, \boldsymbol{s} \rangle]_q$. (8)

2- Output $m = [b']_2$. (9)

2.2.3.7.2 Key Switching (Dimension Reduction).

This technique used by [17] to reduce the dimension of ciphertext after homomorphic operation done. In BGV it can be used to not only reduce the dimension of the ciphertext, but more generally, can be used to transform a ciphertext c_1 that is decryptable under one secret key vector s1 to a different ciphertext c_2 that encrypts the same message, but is now decryptable under a second secret key vector s_1. The vectors c_2, s_2 may not necessarily be of

`

lower degree or dimension than c_1 , s_1 . Because of this generality, it's called key switching procedure. It consist of two basic operations as follows:

- $BitDecomp(x \in R_q^n, q)$ *decomposes* x *into its bit representation* $u_i \in R_q^n$, *then* $R_q^{n.[log\, q]}$. *We do this by first writing* $x = \sum_{i=0}^{[log\, q]} 2^i . u_i$ *with all* $u_i \in R_q^n$ *then output* $u = (u_0, u_1, \dots\ u_{[log\, q])} \in R_q^{n.[log\, q]}$. *(10)*
- $Powerof2(x \in R_q^n, q)$ *expands* x *into* $u \in R_q^{n.[log\, q]}$ *that has copies of* x *multiplied by power of 2 . The output is* $(x, 2, x, \dots, 2^{[log\, q]}x) \in R_q^{n.[log\, q]}$. *(11)*

Lemma 2.1 $\langle BitDecomp(\boldsymbol{c}, q)\rangle, Powerof2(s, q) = \langle \boldsymbol{c}, s\rangle mod\ q$. Detailed proof described in *[19]*.

The key switching technique can be defined by the following two operations.

$SwitchKeyGen(s_1 \in R_q^{n_2}, s_2 R_q^{n_2})$:

1- Generate a public key A as previously described, but with secret key s_2 and parameter $n = n_1 . [\log q]$. (12)

2- Set $B = [Powersof2(s_1)|O]$, that is the matrix with the first column containing $Powersof2(s_1)$ and augmenting some columns with all elements zero until it matches the size of $\boldsymbol{A}$.

3- Set $\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B}$, and output $\tau_{s_1 \to s_2} = \boldsymbol{C}$. (13)

4- $SwitchKey(\tau_{s_1 \to s_2}, c_1)$: Output $c_2 = BitDecomp(c_1)^T . \boldsymbol{C}$. (14)

The following lemma proves that key switching works.

*Lemma 2.*2 Let s_1, s_2, q, A, B,C be as in $SwitchKeyGen(s_1, s_2)$, and let $A . s_2 = 2e_2 \in R_q^N$. Let $c_1 \in R_q^{n_1}$ and $c_2 \leftarrow SwitchKey(\tau_{s_1 \to s_2}, c_1)$. Then we have.

$$\langle c_2, s_1\rangle = 2\langle BitDecomp(c_1), e_2\rangle + \langle c_1, s_1\rangle\ mod\ q. \quad (15)$$

This lemma implies that key switching only produces an error $\| 2\langle BitDecomp(c_1, e_2)\rangle \|_2$ which is small because $BitDecomp(c_1)$ only has coefficients 0 or 1 in the inner product.

The performance of the BGV [19] scheme is as follows (RLWE case):

`

• Secret key: The secret key is 2 ring elements, which require $2d\ log\ q$ bits.

• Single ciphertext: The ciphertext also consists of 2 ring elements, which require $2d\ log\ q$ bits.

• Public key: The public key $A \in R_q^{n\times 2}$ consists of $2n$ ring elements, which require 2dn log q bits.

2.2.3.8 Gentry-*Sahai*-*Waters* scheme

Gentry, Sahai and Waters described in [21] a comparatively simple fully homomorphic encryption (FHE) scheme based on the learning with errors (LWE) problem. In previous LWE-based FHE schemes, multiplication is a complicated and expensive step involving "relinearization". This scheme proposed a new technique for building FHE schemes that called the "approximate eigenvector" method. Homomorphic addition and multiplication considered as just matrix addition and multiplication. This makes the scheme both asymptotically faster and easier to understand.

In previous schemes, the homomorphic evaluator needs to obtain the user's "evaluation key", which consists of a chain of encrypted secret keys. This scheme has no evaluation key. The evaluator can do homomorphic operations without knowing the user's public key at all, except for some basic parameters. They construct the first identity-based FHE scheme. Using similar techniques.

2.2.3.9 NTRU based FHE

Lopez-Alt, Tromer and Vaikuntanathan in [22] construct a multikey FHE scheme based on NTRU, a very efficient public-key encryption scheme proposed in the 1990s. It was previously not known how to make NTRU fully homomorphic even for a single party. They viewed the construction of (multikey) FHE from NTRU encryption as a main contribution of independent interest. Although the transformation to a fully homomorphic system deteriorates the efficiency of NTRU somewhat.

`

2.3 Application of homomorphic Encryption

H.E has many security applications, this large scope of theoretical and practical applications of H.E gives the wide interest of information security researchers and soon it will be one of the big interest of information technology manufacturers specially cloud services vendors and providers. In this section, we will list some of the main applications of homomorphic schemas and summarize the idea behind them

2.3.1 E-voting systems

E-voting systems have a large space of research in cryptography literature, which many secure ballot election schemas has been offered, homomorphic encryption raised as one of those solutions for election schemas, which provide security, trust and scalability.

In such schema, a user simply sends a valid encrypted vote to the server, while the server can compute this vote while it encrypted, this property made election systems more simple and secure [32] [33].

Electronic voting solutions or e-voting systems used by many countries around the world. Internet voting systems used for general elections by countries like Switzerland, Estonia Norway, France, Germany, Spain, Paraguay, Netherlands and the United Kingdom. These countries used special cryptosystems to preserve security for the election process [47] [48]. Electronic voting (e-voting) can be mainly classified into two different systems: machine based systems and Internet voting (i-voting) systems. Machine based e-voting means that both casting a vote and tallying the votes are performed using dedicated electronic devices. I-voting is a voting method that transmits casted votes via public Internet. Development of i-voting systems have been attractive for many researchers and developers, because it uses the wide spread of mobiles, smart phones and personal computers. Providers can construct secure systems with new technologies like cloud via the public internet. I-voting systems still have many security and privacy concerns and there a lot of research in this field.

`

Counting process in i-voting systems can classified into two main methods, mix-nets model, blind signature model and homomorphic model.

2.3.1.1 I-voting Systems Models

Mix-nets model: In the mix-nets a several linked servers called mixes, each mix randomizes input messages and outputs the permutation of them, such that the input and output messages are not linkable to each other, it provides anonymity for a group of voters. Several schemes based on mix-nets are proposed in [49] [50] [51] [52].

The blind signatures model: In blind signature schemes the voter first obtains a token, which is a message blindly signed by the administrator or the authority and known only to the voter himself. Later the voter sends his vote anonymously, with this token as proof of eligibility. Even if later the (un-blinded) signature is made public, it is impossible to connect the signature to the signing process, i.e. to the voter. Schemes based on blind signatures usually use anonymous channels in order to send the un-blinded signature and the encryption of the ballot to a voting authority, assuring the anonymity of the sender [49] [53] [54] [55].

Homomorphic Model: In homomorphic model the tally process depends on encryption of a vote using homomorphic encryption scheme, where add or multiplication process performs homomorphically on encrypted votes to get the results. The voter needs to make proof of his valid vote; this proof must be zero knowledge proof. Schemes based on homomorphic encryptions possess property of verifiability, while preserving privacy. As shown earlier in chapter 2 the property of homomorphism is performed on addition and multiplication ($\oplus$,$\otimes$) which also described in 2.2.1and 2.2.2. Many homomorphic voting systems derived from the theory of ElGamal cryptosystem [56] which additive homomorpthe homomorphic voting system of homomoa multiplicative homomorphic Paillier cryptosystemorphic Paillier cryptosystem [4] are proposed in [57] [58] [33].

3.1.1 Zero Knowledge proofs:

Zero knowledge proofs can be used to demonstrate the truth of a statement without revealing anything else. Which one party (the *prover P*) can prove to another party (the *verifier V*) that

`

a given statement is true, without conveying any information apart from the fact that the statement is indeed true. In ZKP, the prover proves that he/she knows a secret without revealing it [59].This statement assumed as a secret, the interactions are designed that they cannot lead to revealing or guessing the secret. After exchanging messages, the verifier only knows that the prover does or does not have the secret, nothing more. The result is a yes/no situation, just a single bit of information.

Zero-knowledge proofs needs interactive communication between Prover and Verifier, where input from Verifier needed. The prover must respond with usually in the form of a challenge or challenges such that the responses from the prover will convince the verifier if and only if the statement is true. This type is called Interactive Zero- knowledge proofs.

A zero-knowledge proof must satisfy three properties:

Completeness: The prover can convince the verifier if the prover knows a witness testifying to the truth of the statement.

Soundness: A malicious prover cannot convince anybody if the statement is false, except with some small probability.

Zero-knowledge: A malicious verifier learns nothing except that the statement is true. This is formalized by showing that every cheating verifier has some simulator that, given only the statement to be proved (and no access to the prover), can produce a transcript that "looks like" an interaction between the honest prover and the cheating verifier [60].

3.1.2 Non Interactive Zero-knowledge Proofs:

Non-interactive zero-knowledge (NIZK) proof systems [61] yield proofs that can convince others about the truth of a statement without revealing anything but this truth. It has been shown under standard cryptographic assumptions that NIZK proofs of membership exist for all languages in NP. NIZKP does not need the interactive communications between the prover and verifier. We will consider statements of the form $x \in L$, where L can be an arbitrary language in NP. We require that the NIZK proof be *complete*, *sound*, and *zero-knowledge*.

`

In NIZKP model the prover and the verifier are in possession of a reference string sampled from a distribution D by a trusted setup $\sigma \leftarrow Setup(1^k)$. To prove statement $y \in L$ with witness w, the prover computes $\pi \leftarrow Prove(\sigma, y, w)$and sends the proof π to the verifier. The verifier accepts if $Verify\ (\sigma, y, \pi) = Accept$, and rejects others.

Gentry [5] proposed a fully homomorphic encryption scheme and demonstrated that fully homomorphic encryption can be used to construct NIZK proofs whose size depends only on the size of the witness and on the security parameter, but not on the size of the circuit used to verify the witness. Gentry proposed to encrypt every bit of the witness using a fully homomorphic encryption scheme. Using the operations of the fully homomorphic encryption scheme, it is possible to evaluate the circuit on the plaintexts to get a ciphertext that contains the output. Using an NIZK proof the prover then constructs a proof for the public key being valid, the encrypted inputs being valid ciphertexts and the output ciphertext being an encryption of 1. Since the proof contains $|w|$ ciphertexts and $|w|$ proofs of their correctness, the total complexity is $|w|$. *poly(k)* [62].

2.3.2 Cloud services

The most trending application of fully homomorphic encryption is cloud services. One of the major problems in Cloud Computing is the fact that the customer cannot technically validate the security and confidentiality of a remote resource. Current cloud solutions did not preserve the optimal privacy for users. Some solutions depend on data encryption, but still unsupported with privacy on processing level.

FHE provide the solutions for processing data while still encrypted, and execute an encrypted program as shown [42]. Which the program execution performed in binary, an encryption of a circuit can be done by encoding a bit in a cipher text's property of having an even or odd remainder modulo a secret prime key. This can be easily reduced to a boolean algebra by mapping 0-bits to even integers and 1-bits to odd integers. An XOR-operations will be represented by the integer addition, while the integer multiplication represents a boolean AND-operation. This allows to simulate chains of boolean operations by means of simple

`

integer arithmetics. A formulation of processor components and memory operations done, to determine each circuit for each part or operation. This process, like virtualization of process in an encrypted way, which provide execution of program in secret.

This model can be used in many cases like, delegation of computation to a remote resource, remote Search with encrypted search functions, Mobile Code and Multi-Party Computation.

Another type of secret program execution described in [43], which describes a method of formulating programming function such if statement, loops and function calls in encrypted forms using FHE and use it in program structure.

2.3.3 Protection of mobile agents

It is one of the most famous applications that is applicable by using homomorphic cryptography. Since all conventional computer architectures are based on binary strings, and only requires the addition and multiplication operations, such fully homomorphic encryption schema would offer the possibility to encrypt a whole program so that it is still executable [25]. The protection of mobile agents could be based on two ways: (a) computing with encrypted functions such as making encrypted queries to search engine. (b) computing with encrypted data, while the data is being encrypted in advanced then sent to server to be computed without decrypting data with the use of homomorphic properties, finally the computed results sent back to user, then he can decrypt it and review the results.

2.3.4 Making encrypted queries to search engines

F.H.E could be used to query a search engine, without revealing what is being searched for. [5] Alice want to use this property of F.H.E, she generates ciphers $(\boldsymbol{c_1}, \dots, \boldsymbol{c_t})$ of here query $(\boldsymbol{m_1}, \dots, \boldsymbol{m_t})$ under pk, using fully homomorphic encryption schema with $\mathbb{C}$ circuit express the server's search function. The server sets $\boldsymbol{c} \leftarrow$ Evaluate (pk, $\mathbb{C}$, $\boldsymbol{c_1}, \dots, \boldsymbol{c_t}$). The server sends these cipher texts to Alice. We know, by the correctness requirement, that Decrypt (sk, c) = $\mathbb{C}(\boldsymbol{m_1}, \dots, \boldsymbol{m_t})$.These latter values constitute precisely the answer to Alice's query, which she recovers through decryption.

`

2.3.5 Searching over encrypted data

Alice as many of users want to use the cloud to store her data, because of the benefits of its processing power and large storage size, but she does not trust the cloud. She needs her data to be stored securely in an encrypted way, because otherwise the server could read or leak her private data. In this case Alice needs to encrypt here data using F.H.E schema, store it on the cloud, when she wants to perform a search query, which will represent as a circuit $\mathbb{C}$.

As discussed before the result of evaluation process c, represents the data manipulated with function$\mathbb{C}$, decryption gives the right answer of here query without the server learn anything of query or data being processed.

2.3.6 Multiparty Computation (MPC)

Which is a central problem in theoretical cryptography, in these problem n parties, holding a private input $\boldsymbol{m_1, \ldots, m_n}$, which to compute a given function $\boldsymbol{f(m_1, \ldots, m_n)}$. This problem belongs to the area of computing with encrypted data [26]. One of famous approaches in this field is F.H.E introduced by Gentry [27], which all parties encrypt their input first under the F.H.E schema, then they evaluate the desired function on the cipher texts using the homomorphic properties, and finally they perform a distributed decryption on the final cipher texts to get the results.

MCP concept was used to perform a secure dynamic programming protocol that utilizes homomorphic encryption. Which can compute optimization problem on servers receives encrypted inputs from mutable agents. Servers should not know anything about the data being processed, the result will sent to agents to be decrypted [28].

2.3.7 Secret sharing scheme

In secret sharing schemas, there is m parties have "sub-secret", and there exist a "super-secret" which is the composition of the sub-secret under specified function such as the sum of or the multiplication of sub-secrets, each party want to determine the super-secret without revealing his sub-secret [29].

`

If the secrets of each party are (s_1, s_2, s_3), so each party encrypts his secret $c_{1,2,3} = E(s_{1,2,3})$, then users can share their secrets encrypted with homomorphic schema, so if suppose the function is the sum, so the super-secret is $S = D(c_1 + c_2 + c_3)$.

In other words the sum of shares of the secrets are the shares of the sum of the secrets.

$$E(s_1) + E(s_2) + E(s_3) = E(s_1 + s_2 + s_3). \quad (16)$$

2.3.8 Threshold schemes

Threshold schemes allow any *t* out of *l* individuals to recompute a secret (key). In other words A key pair is generated jointly between multiple parties, and whereas the public key is used for encryption as in ordinary asymmetric cryptosystems, the private key will only exist as a shared secret throughout its lifetime. Multiparty Computation and secret sharing are types of threshold schemas.

2.3.9 Zero-knowledge proofs

Zero-knowledge proofs could be used to demonstrate the truth of a statement without revealing anything else, by which one party (the prover) can prove to another party (the verifier) that a given statement is true, without conveying any additional information apart from the fact that the statement is indeed true. In such case a user wants to log in a host, he has to prove his identity by logging his username and password, which the user need to be private and not leaked during the protocol operation. An example of using homomorphic cryptographic properties in zero-knowledge proofs in [30] and [31].

2.3.10 Watermarking and fingerprinting schemes

Watermarking schemas provide a solution for saving copyrights, illegal redistribution of copies and violation of ownership, it enables the owner to embed some information on the contents and to extract it, this information indicates to the owner copyrights. A fingerprinting scheme embeds the information related to a buyer and enables a merchant to trace the buyer from the redistributed copy [34].

Homomorphic properties enable the merchant to add or multiply encrypted watermark to the encrypted message, then he can prove his ownership of digital data. In fingerprinting, a buyer

`

encrypts his identity, then sends to the merchant using zero-knowledge proofs. The merchant embeds encrypted identity to encrypted digital data and returns it to the buyer.

The buyer decrypts it and gets fingerprinted digital data without disclosing his identity to the merchant [35].

2.3.11 Commitment schemes

Commitment schemes are of great importance in cryptography fundamentals, it allows to party to choose a value and commit to his choice while keeping it secret, and then he can reveal this value later. It's designed so that the party cannot change his mind and cannot change the committed value.

Commitment schemas have various applications such as secure coin flipping, zero-knowledge proofs and secure computation. Homomorphic property was used efficiently to implement some commitment schemas [36].

2.3.12 Lottery protocols

Usually in cryptographic lottery, which have to compute a random number in order to indicate the winning, ticket from all participants. In such homomorphic lottery schemas, a random number chosen by the players, then each number encrypted with the homomorphic cryptosystem of the lottery. Nobody except the lottery can learn the chosen numbers. Using homomorphic property, the encryption of the sum of the random numbers. Then the combination of this and a threshold decryption scheme leads to the winning ticket of the lottery without the ability to compute the winner during the purchase time [37].

2.3.13 Mix-nets

Mix-nets are protocols that provide anonymity for senders by collecting encrypted messages from several users. It uses a set of servers to establish private communication channels that hard to trace. One type of mix network accepts as input a collection of ciphertexts, and outputs the corresponding plaintexts in a randomly permuted order. In such a scenario, privacy is achieved by requiring that the permutation that matches inputs to outputs is kept secret to anyone except the mix-net. A desirable property to build such mix-nets are re-

`

encrypted, which is achieved by using homomorphic encryption. In a re- encryption mixnet, the inputs are submitted encrypted under the public-key of the mixnet. Each mix server processes the batch of input ciphertexts sequentially. The First server takes the set of input ciphertexts, re-encrypts them, and outputs the re-encrypted ciphertexts in a random order. Each server in turn takes the set of ciphertexts output by the previous server, and re-encrypts and mixes them. The set of ciphertexts produced by the last server may be decrypted by a quorum of mix servers to yield plaintext outputs [38].

2.3.14 Data aggregation in wireless sensor networks

Wireless Sensor Networks (WSN) consist of less expensive and low power sensor nodes that are capable of computation, storage and communication. These sensor nodes have low computation power and storage space. The purpose of deploying a sensor node is to monitor an area of interest with respect to some physical quantity. Information gathered by the sensor nodes is reported to the base station.

Data aggregation in wireless sensor networks (WSN) helps eliminate information redundancy and increase the lifetime of the network. It is a technique combines partial results at the intermediate nodes in route to the base station, thereby reducing the communication overhead and optimizing the bandwidth utilization in the wireless links [39] [40].

When homomorphic encryption is used for data aggregation, end-to-end encryption is achieved and aggregation function like average or minimum/maximum can be computed on the encrypted data. It's applied to protect the privacy of input data while computing an arbitrary aggregation function in a wireless sensor network [41].

2.4 Implementations of Homomorphic Encryption schemas

In the last section, we reviewed the most important theoretical applications of Homomorphic Encryption, all these applications still discussed in researches with experimental implementations.

In this section, we will discuss the implemented schemas practically and try to explain how it work by examples.

`

2.4.1 Homomorphic Encryption Library HELib

HElib is a software library that implements homomorphic encryption (HE). Available as an implementation of the Brakerski-Gentry-Vaikuntanathan (BGV) scheme [2], along with many optimizations to run homomorphic evaluation runs faster, focusing mostly on effective use of the Smart-Vercauteren [12] ciphertext packing techniques and the Gentry-Halevi-Smart [44] optimizations.

At its present state, it is fairly low-level provides low-level routines (set, add, multiply, shift, etc.). This library is written in C++ and uses the NTL mathematical library (version 6.1.0 or higher). It is distributed under the terms of the GNU General Public License (GPL) [1]. Shai Halevi and Victor Shoup developed this library. More details in annex 1.

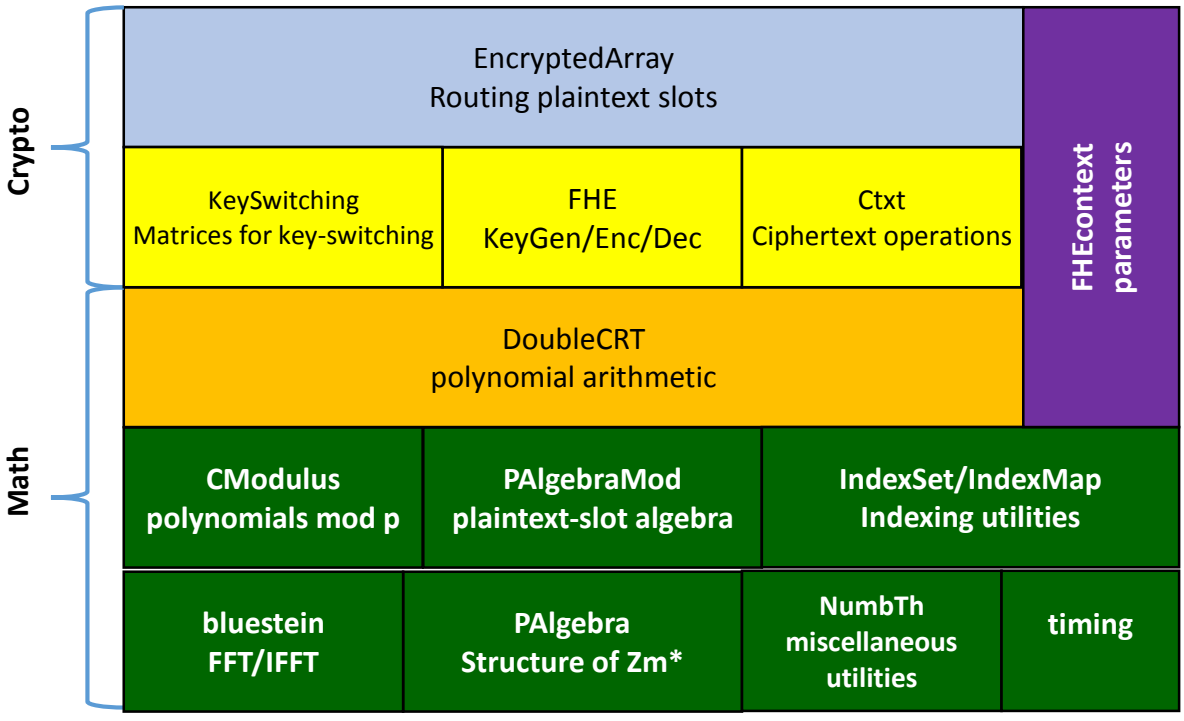


Figure 3.1 A block diagram of the Homomorphic-Encryption library

Figure 3.1 shows the structure of HELib library for method and functions.

HELib [45] consists of four layers: in the bottom layer the modules for implementing mathematical structures and various other utilities. The second layer implements the Double-CRT representation of polynomials. The third layer implements the cryptosystem itself (with

`

the "native" plaintext space of binary polynomials). The top layer provides interfaces for using the cryptosystem to operate on arrays of plaintext values. The bottom two layers identified as the "math layers". The top two layers identified as the "crypto layers". Figure 3.1 A block diagram of the Homomorphic-Encryption library. Roughly, the modules `NumbTh`, `timing`, `bluestein`, `PAlgebra`, `PAlgebraMod`, `Cmodulus`, `IndexSet` and `IndexMap` belong to the bottom layer, `FHEcontext`, `SingleCRT` and `DoubleCRT` belong to the second layer, `FHE`, `Ctxt` and `KeySwitching` are in the third layer, and `EncryptedArray` is in the top layer.

Implantation of HELib variant original BGV [19] in some issues, where its defined over polynomial rings of the form $\mathbb{A} = \mathbb{Z}[X]/\Phi_m(X)$, where m is a parameter and $\Phi_m(X)$ is the m'th cyclotomic polynomial. The "native" plaintext space in this scheme is usually the ring $\mathbb{A}_2 = \mathbb{A}/2\mathbb{A}$ namely binary polynomials modulo $\Phi_m(X)$. HELib uses the Smart-Vercauteren CRT-based encoding technique to "pack" a vector of bits in a binary polynomial, so that polynomial arithmetic in $\mathbb{A}_2$ translates to entry-wise arithmetic on the packed bits. The ciphertext space in this scheme consists of vectors over $\mathbb{A}_q = \mathbb{A}/q\mathbb{A}$, where q is an odd modulus that evolves with the homomorphic evaluation.

Secret keys are polynomials $\mathfrak{s} \epsilon \mathbb{A}$ with "small" coefficients, and we view $\mathfrak{s}$ as the second element of the 2-vector $\vec{s} = (1, \mathfrak{s})$. A level-$i$ ciphertext $\vec{c} = (c_0, c_1)$ encrypts a plaintext polynomial $m \in \mathbb{A}_2$with respect to $\vec{s} = (1, \mathfrak{s})$ if we have the equality over $\mathbb{A}$, $[\langle \vec{c}, \vec{s} \rangle]_{q_i} \equiv m(mod\ 2)$, moreover the polynomial $[c_0 + \mathfrak{s}.c_1]_{q_i}$ is small, all its coefficients are considerably smaller than q_i. Roughly, that polynomial is considered the "noise" in the ciphertext, and its coefficients grow as homomorphic operations are performed.

The basic operations done in this scheme are key-generation, encryption, and decryption, the homomorphic evaluation routines for addition, multiplication and automorphism (and also addition-of-constant and multiplication-by-constant), and the "ciphertext maintenance" operations of key-switching and modulus-switching. More details of how HELib work described in the report [45] founded in the documentation HELib source code.

`

Halevi and Shoup the Creators of HELib described in [46] some of the algorithms and optimization techniques that are used in HElib for data movement and simple linear algebra over HELib "platform".

2.4.2 Scarab library

It's an open source implementation of a fully homomorphic encryption scheme using large integers, based on Gentry [1], and N. Smart and F. Vercauteren [10] for the integer-based approach used in this implementation.

The software requires other libraries:

1. GMP: GNU Multiple Precision Arithmetic Library.
2. FLINT: Fast Library for Number Theory **version 1.6.**
3. MPIR: Multiple Precision Integers and Rationals.
4. MPFR: Multiple-precision floating-point computations with correct rounding.

The libraries are implemented using C language, and supporting Linux only, the installation requires installing the provided libraries before installation of "libScarab".

2.5 Summery

The mentioned schemes separated into three major types of homomorphic encryption. First, additive homomorphic encryption schemes such as Paillier scheme. Second, multiplicative homomorphic schemes such as RSA and ElGamal scheme, which is a widely used public key scheme. Some applications developed on additive and multiplicative homomorphic schemes such as multiplicative homomorphic e-voting system [23]. The limitation to a single operation resulted the reduction of the efficiency and effectiveness of these schemas, especially for bigger and more complex applications. This led to looking forward to schemas that supports both operations, which provides in the third type of homomorphic encryption called Fully Homomorphic Encryption which first show was by Gentry in 2009 [5].

Gentry's construction consists of several steps: He first constructed a "somewhat homomorphic" scheme that supports evaluating low-degree polynomials on the encrypted data, next he needed to "squash" the decryption procedure so that it can be expressed as a

`

low-degree polynomial which is supported by the scheme, and finally he applied a "bootstrapping" transformation to obtain a fully homomorphic scheme [8].

A lot of developments and schemas based on Gentry's blueprints, most of these schemas suffer from the squashing process. Although these earlier schemes have achieved full homomorphism, the performance of these schemes becomes the bottleneck. To address this problem, some To address this problem, some newer FHE schemes were proposed in recent years. In [19] Brakerski, Gentry, and Vaikuntanathan proposed a new FHE scheme (BGV) based on LWE problems. Instead of recryption, this new scheme uses other light weighted methods to refresh the ciphertexts. These methods can limit the growth of the noise so that the scheme can evaluate much deeper circuits. The recryption process will serve as an optimization to deal with over complicated circuits instead of a necessary for most circuits; this scheme based on leveled terminology where the depth of circuit defined early [24].

An implementation of BGV scheme was founded by IBM research team in 2013, called "HELib" which described as the first well-structured and fully documented implementation library of FHE. HELib has a lot of optimization variant of the original BGV, these optimizations advance the performance and usability issues.

In this thesis, HELib chosen as a basic platform to express the properties of FHE, and make the implementation of the proposed voting system based on FHE. HELib provide many classes and methods to examine performance and timing, which ease results examination.

The research focuses on election and e-voting systems as a case study, in this work NIZKP also discussed as an application on Fully Homomorphic Encryption. Also cloud services used as infrastructure of the system.

`

Chapter 3: Research Methodology.

In this research, a proof of concept methodology followed to present that using fully homomorphic encryption is applicable in such e-voting systems. Many e-voting protocols were implemented; some of them uses homomorphic encryption through additive or multiplicative homomorphic property. Our contribution is to design and implement e-voting system based on fully homomorphic encryption and can work in cloud infrastructure.

To prove that using fully homomorphic encryption on e-voting system can give applicable performance and security, we designed and implemented e-voting system based on fully homomorphic encryption.

This research tried to prove that the implemented e-voting system met the security properties required in such e-voting systems such as eligibility, privacy, accuracy, verifiability, fairness, receipt-freeness, incoercibility, dispute-freeness, robustness, scalability and practicality.

3.1 The basic tool "HELib"

The implementation deployed using the HELib library [1], which implements homomorphic encryption (HE), currently available is an implementation of the Brakerski-Gentry-Vaikuntanathan (BGV) scheme [2].

HELib considered as the first applicable implementation library of homomorphic encryption, that well-structured and documented. It attracted the attention of many researchers. HELib is an open source C++ library focusing on effective use of ciphertext packing and the GHS optimizations. It includes an implementation of the BGV scheme itself with all its basic homomorphic operation, and some higher-level procedures implementing the GHS data-movement procedures and simple linear algebra. The operation done on a vector of plaintext values, the plaintext arrays in HELib often hold a few hundred plaintext slots (sometimes even a few thousand). It also support Single instruction multiple data processing, which increase the performance of the library.

HELib focuses on two factors time and noise, These correspond roughly to size and depth of the corresponding SIMD circuits, but the correspondence is not quite one-to-one since different operations have different time and noise behavior. Since multiplication operation

`

have much higher noise than addition operation. Many optimization done in HElib to reduce the time and noise of the original BGV scheme [63]. HELib described in details in section 2.4.1 and annex 1.1.

HELib used to generate public-private keys, used to demonstrate a circuit have multiplication and addition operation to perform proof operation, and finally used to count results through addition operation.

3.2 Implementation

An implementation of the enhanced system done using C++ language in Ubuntu Linux 12, because help build on C++ and run only on Linux. The implementation divided into three parts, the first called Authentication Server program, second called Voting Server program and the third called Voting program. Each part of the system has a different role, the authentication server responsible for public-private key generation and this provided by HELib. It's also holding authentication credentials such as user passwords and secret keys for voting server, also generated random secret, its basic functionality decrypting ciphers. It was considered as a trusted party.

The voting server program is responsible for performing homomorphic operations on encrypted cipher which in this case the encrypted votes. It does not do any decryption process and considered as an interested party, because it's hosted in cloud infrastructure and the cloud provider can disclose some data from the hosted machines.

The voter program responsible for making authentication with authentication server and voting server, formatting the vote according the voter input and encrypt the vote. The voter program considered as untrusted and its turn to trust after authentication.

All communication channels considered as untrusted, and all traffic transferred between system parts were encrypted and integrity checked.

3.3 Testing

The testing and result analysis based on testing performance and measuring resulted ciphers size and processing time, this done by the provided class from HELib “timing”. The class

`

can measure the time of generating public-private key, encryption time, decryption time and homomorphic operation time. From these results, we measured the performance of the system according to the number of processed votes and the produced size of each vote. We designed number of experiment to do that, where the number of vote calculated was variable for each experiment. Also *p* value which indicate the number of users of the systems measured. The *p* value affect the public-private keys and the size of the vote and mask.

3.4 Scope

The scope of this research was related to the maximum number of users could participate in election process, this number is tied to the maximum value of *p* produce acceptable public key size and processing time for mask and vote tallying. In our experiment, we measured the performance for 10 million voter, which able to be applied in about of 70% of countries all over the world. In addition, it is applicable to applied to in our country Palestine, and many other countries.

The experiments tested up to 40 thousands votes generated randomly and stored locally in one machine, encrypted, masked, homomorphicaly checked and tallied.

`

CHAPTER 4: Proposed E-Voting System

This chapter describe in details the proposed system

4.1 The proposed E-Voting system

Our protocol based on using cloud services as an infrastructure for components of the system, cloud provides high performance-processing capabilities and can deal with huge numbers of communications done by voter that they want to make voting in a short period.

Cloud considered untrusted platform for such sensitive process, but homomorphic encryption solves some of the security issues related to tallying and proving votes, which need the biggest part of processing, we needed a part of our system to secure for containing private keys, and voter identification information's. Our system consists of:

a) Authentication Server (AS): responsible for authentication, verifying the correctness of the vote, and valid encrypted with the public key.
b) Voting Server (VS): responsible for masking the vote and tallying.
c) Bulletin Board (BB): responsible for display the checksum of vote for public and other public information.

Protocol Steps

1) **Registration**: Voter need to have Identification information to be able access and authenticated by the system, he need make registration process personally to have his secret key, which is required with other information like his national ID number, and this information provided by authority office and delivered using the secure method as shown in figure 4-1.

$$\text{Voter } (V) \xrightarrow{ID\ info} \text{Authority Office } (AO)$$

$$V \overset{SK}{\leftarrow} AO$$

`

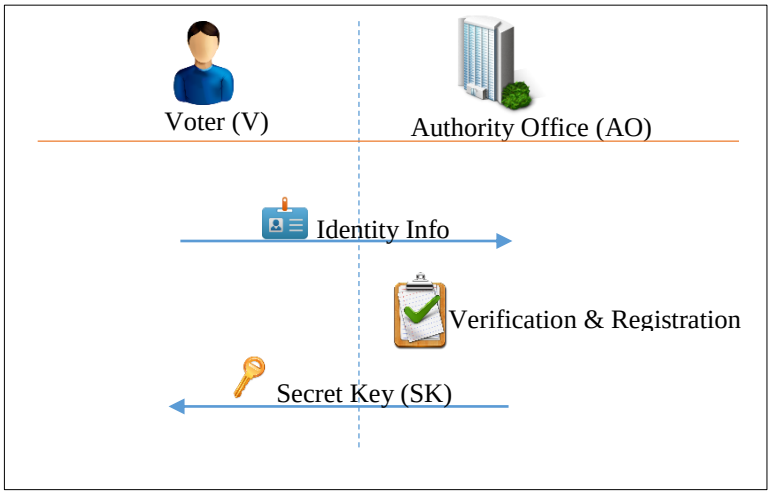


Figure 4.1 Voters Registration Process

2) **Authentication**: When the voting process starts, the voter needs to connect to the Authentication Server to authenticate his identity using his international ID number and secret key. This connection to server done via SSL protocol to preserve privacy, authenticity and verifiability. Once the voter authenticated, a new Random Secret Key (RSK) generated in AS, this new RSK encrypted with AS secret key $E_{sk}(RSK)$. The resulted cipher sent to both Voter and Voting Server. V and VS can reveal RSK by decrypting the received cipher using Public Key of Authentication Server $RSK = D_{pk}(E_{sk}(RSK))$.

   Another method to do that, once a voter authenticated, a new Random Secret Key (RSK) generated in AS, this new RSK encrypted with voter password, and sent him. Also RSK encrypted with predefined key between VS and AS, and sent to VS.

User allowed to communicate with Voting server using the random secret key generated by AS to be authentication secret of session between voter and VS.

The User can send Hello message to VS with encrypting M using RSK, $E_{RSK}(M)$ and using Hash-based message authentication code HMAC [63], which used to verify both the data integrity and the authentication of a message. VS also can send response message using the same way. The HMAC will be used for the rest of communications with RSK as the secret key.

`

To prevent attacker form identifying any unencrypted messages sent between VS, AS and V, a symmetric encryption used with salting communication messages to prevent cipher duplication. The key for the symmetric encryption is the RSK generated from AS, then HMAC used along with encrypted messages.

$$V \xrightarrow{ID,SK} AS$$

$$V \xleftarrow{RSK} AS \ , VS \xleftarrow{RSK} AS$$

$$V \xrightarrow{RSK} VS$$

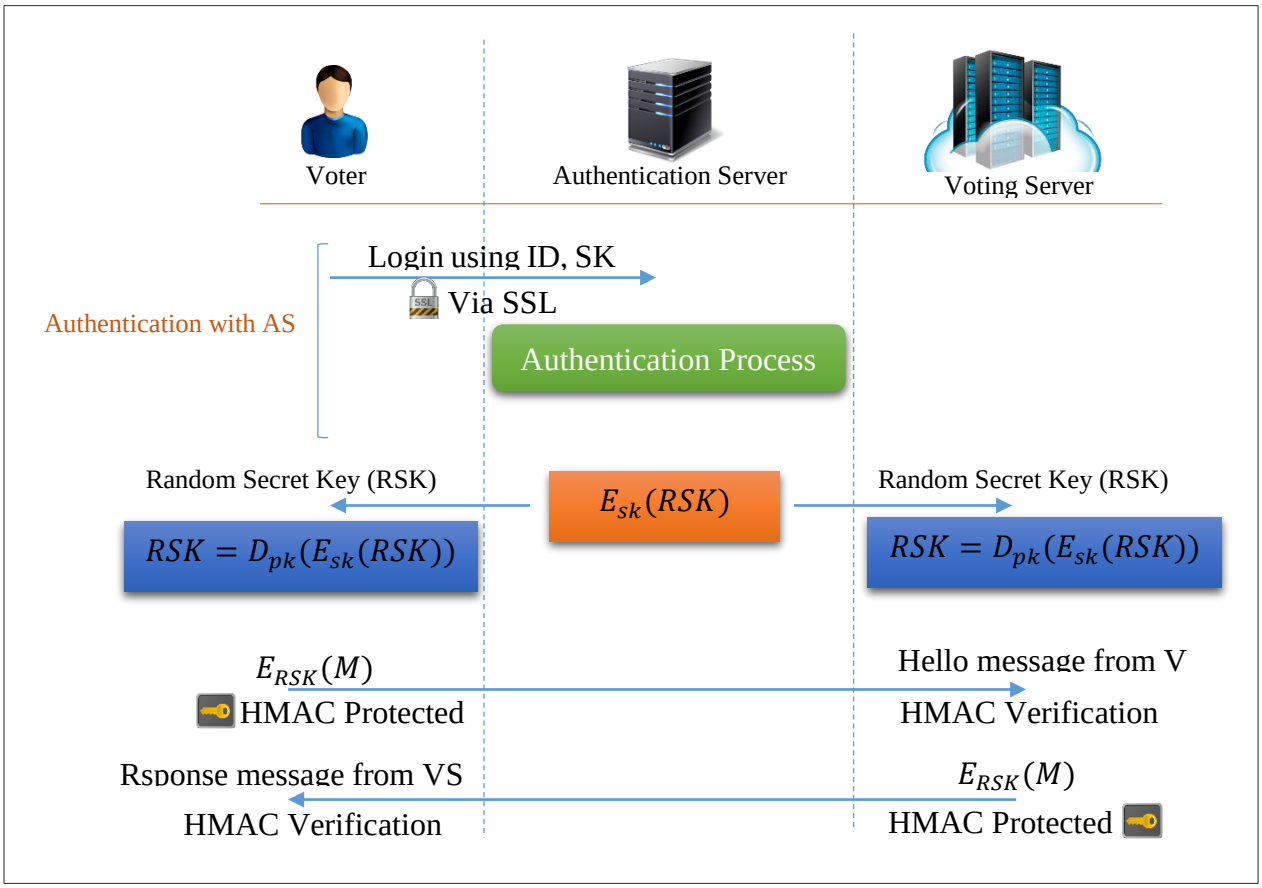


Figure 4.2 Voter Authentication with Authentication Server & Voting Server

3) **Voting Process**: suppose that the voter wants to vote for some candidates N_i , where i is the number of candidates. Vote v represented by {0, 1} for each candidate, where if V is voting for $C_{i=1}$ for Yes the v_1= 1, if No v_1= 0. Additional digit d is considered as verification of the correct tallying of votes with value of 1, where $v = \{v_1, \ldots\ldots, v_i, d\}$ as shown in Figure 4.3Vote Structure Example.

`

$$v \begin{matrix} C_1 & C_2 & C_3 & C_4 & C_5 & d \end{matrix} \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Figure 4.3Vote Structure Example

a. Vote Encryption: v encrypted by public key pk of VS $E_{pk}(v) = c$. V need to calculate checksum of $c = H_c$, which is used to verify that c is tallied without any modification, and it arrived correctly to VS, in this stage HMAC used to preserve integrity.

$$V \xrightarrow{c} VS$$

$$V \xrightarrow{HMAC} VS$$

Encrypted cipher sent to VS, which calculate$H(c)$, and store both H, c and then send H to the Bulletin Board, V can check for H in BB. If the values are identical, c arrived correctly.

$$VS \xrightarrow{H} BB.$$

b. Vote Verification: at this stage we present a None Interactive Zero Knowledge proof method which the voter wants to prove that he used a valid pk and valid voting where no additional number added to some v_i and restricted to i number of candidates, so that the vote is well formatted. The verifier is our system with his both separated parts AS and VS.

VS process c to make it masked, so that AS can't identify the original vote and still able to verify the correctness of valid encryption and formatting.

Mask function calculated for c,

$$\boldsymbol{M = c\ XOR\ m^0 + c\ XOR\ m^1} \quad (17)$$

Where $m^0 = \{0_1, \dots, 0_{i+1}\}$, $m^1 = \{1_1, \dots, 1_{i+1}\}$. M is sent to AS, a decryption of masked vote is being done $D_{sk}(M) = U$, so the result must be

`

1 for each i,$U_{i+1} = 1$. If it's not, a reject flag sent to VS, V told that he tried to enter invalid c, and c, H deleted for that V.

For each valid c, AS count 1 valid voting, the number of valid c in VS must be identical with number in AS.

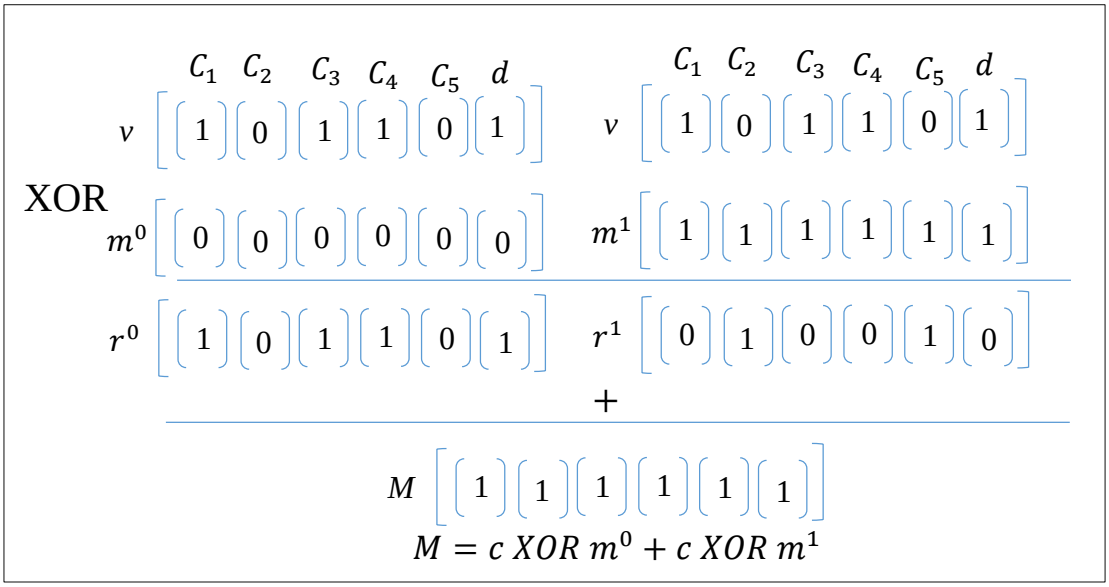


Figure 4.4 Masking process example

As shown in Figure 4.4 Masking process example, the addition process done in decimal form not in binary form, the intruder may try to add some core to a specified candidate, in such case the vote slot will increase by the value entered by intruder, it will calculated in the final results. This easy to cover after tallying process because the summation of result of each result must be equal to the number of voters. No one can identify the vote that have the additional score before the tallying process. Here come the NIAZKP role, to identify any invalid vote, without decrypt the vote and before the tally process. This process can handled using FHE easily as described early. It just need to two parties to make this operation away from the voter to preserve the correctness of masking process.

`

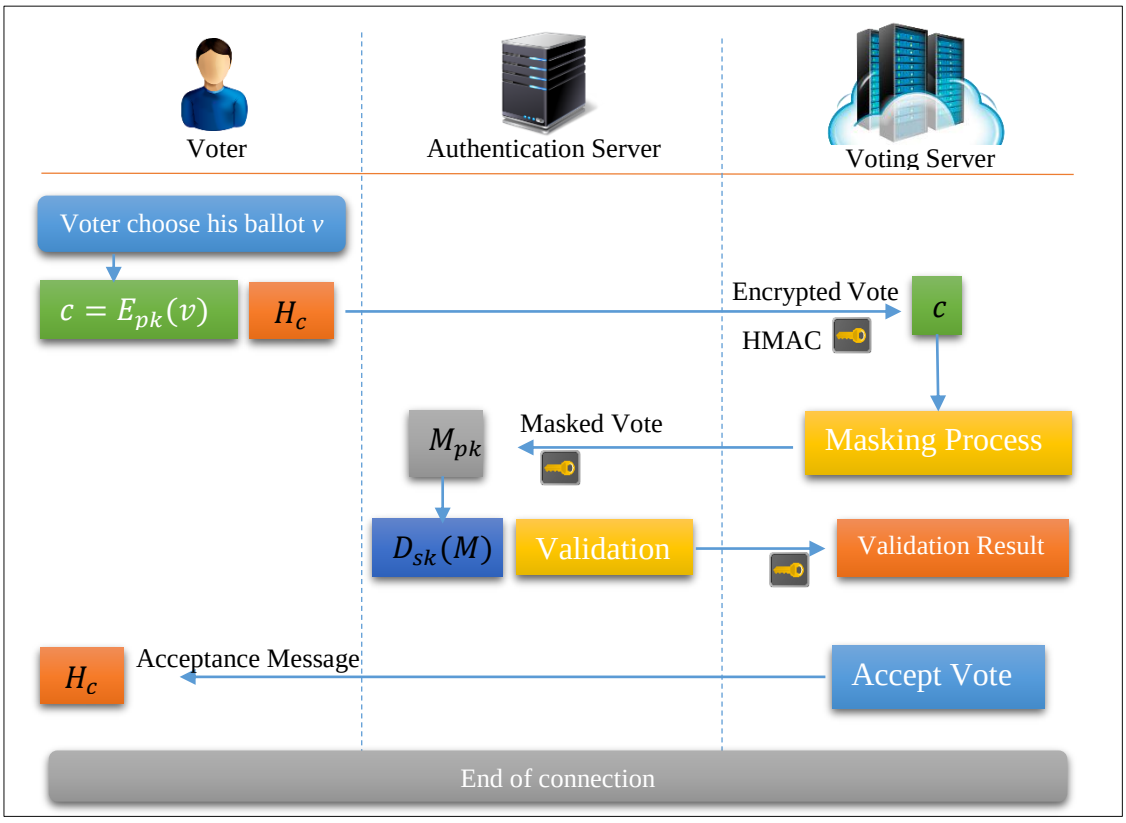


Figure 4.5Vote Encryption & Validation with NIZKP

4) **Tally process**: after the specified period form authorities finished, the tallying process starts, let the number of valid votes is j, so $\sum_j c = C$, which is the final result of the voting process

$VS \xrightarrow{C} AS$

Decryption of results processed, $D_{sk}(C) = R$, where $R = \{r_1, \ldots., r_i, j\}$. (18)

a. R, C *and* sk put in the BB, so regulatory institutions can verify the tally process.

`

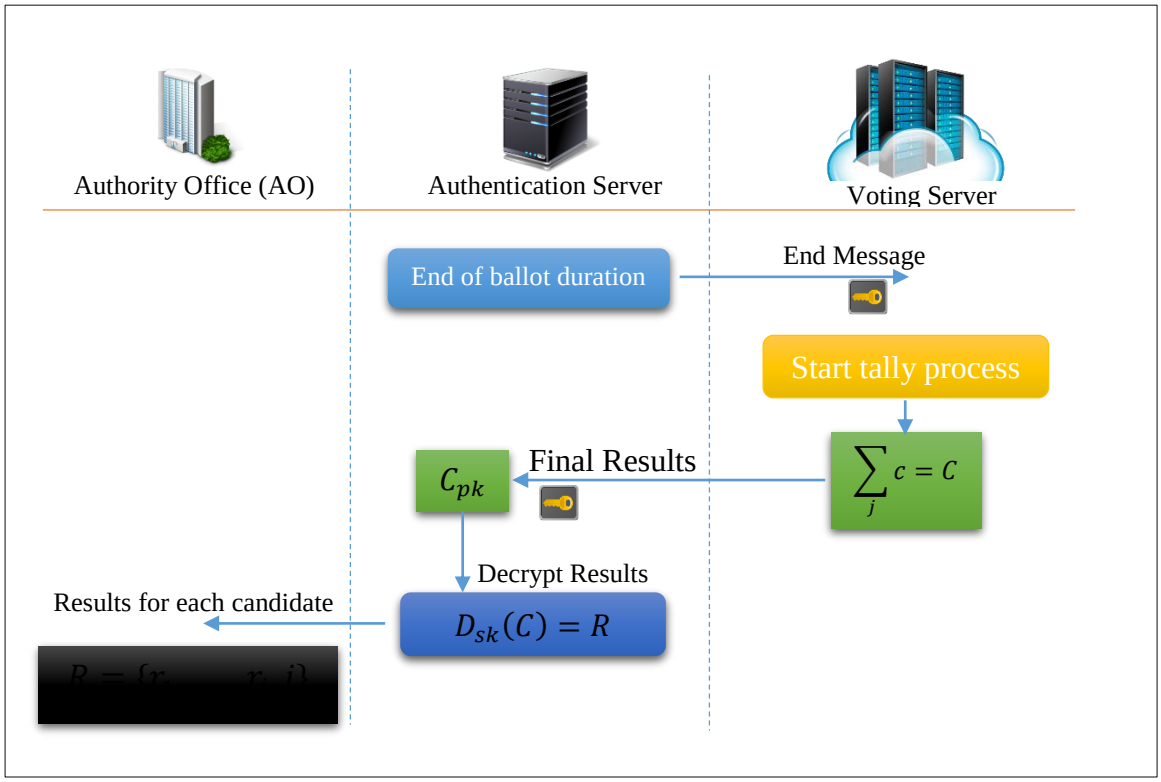


Figure 4.6 Votes tally & results decryption

4.2 Security analysis

Any voting system must be able to deal with some security issues related to preserve privacy of voting and accuracy of results.

4.2.1 **Eligibility**: Only persons who meet certain pre-determined criteria are allowed to cast permitted number of votes. To achieve this, authority needs to verify the eligibility of voters and record their casted votes, in registration process voter need to introduce all information's to be considered eligible.

4.2.2 **Privacy**: No one except voters can know their votes. To achieve this, any traceability between voters and their votes must be removed during the whole election. In our protocol no one can connect the user to his vote.

4.2.3 **Accuracy**: In the elections, voters expect that their votes are correctly captured and that all eligible voters are correctly tallied. As we introduced, the tally process is verified by the digit d added to each vote, the number of valid votes in AS and V. Another verification done by NIZKP which satisfy accuracy and verifiability.

`

4.2.4 **Verifiability**: Verifiability is the ability to determine whether only and all valid votes are counted in final tally or not i.e. to determine the accuracy of the election. Accuracy of the election can be verified in two ways, one is the individual verifiability where only voters can verify their own votes in the tally which done by our NIZKP method. Therefore the accuracy of the election consists of *N* voters is ensured when there are less than or equal to *N* votes and all *N* voters verify their votes. The other is universal verifiability, which enables any third party to verify the accuracy of the election which accomplished by putting all *R,C,sk* on BB for any third party to check tally process.

4.2.5 **Fairness**: In order to conduct the impartial election, anyone should not be able to compute the partial tally before the end of the election which may influence the remaining voters and may affect the voting result, and this accomplished by separating AS nd VS. so *sk* is stored in AS which cannot calculate any results until it receive *C* form AV.

4.2.6 **Receipt-freeness:** Receipt-freeness disables anyone including voters themselves to link voters to their votes, in order to protect voters from being coerced to follow intentions of other entities. To achieve receipt-freeness, the voting system didn't leave any information about the votes of voters. Also, votes should not include any information peculiar to the voters. Receipt-freeness shares the same notion with privacy. Our protocol is Receipt-free.

4.2.7 **Incoercibility**: Incoercibility protects voters against coercers who can communicate with the voters actively. In our protocol, we allow V to Revote which a method to overcome incoercibility. If V exposed from some incoercible person, he can revote again by authenticate to AS, then send revote to VS with his previous H, new vote should be replaced with old vote and new H added to BB.

4.2.8 **Dispute-freeness**: Even if dishonest voters are involved in elections, disputes among entities should be solved without involving irrelevant entities. The notion of universal verifiability is similar to dispute-freeness but it is limited to the voting and tallying stages. Dispute-freeness accomplished by a mutable verification method before

`

considering the vote is valid, and validation using digit and counting the number of valid votes in AS and VS.

4.2.9 **Robustness**: Any entity should not be able to disrupt the voting, i.e. the voting system must be able to detect dishonest entities and to complete the voting process without the help of detecting dishonest entities, which is satisfied in our protocol, while any illegible voter does not allowed to communicate with VS, and no invalid vote stored.

4.2.10 **Scalability**: A scheme has to be extended easily to suffice computation, communication and storage requirements of large-scale elections. Our system is scalable due to cloud based infrastructure where huge processing and communication can be done.

4.2.11 **Practicality**: A scheme should not have assumptions and requirements that are difficult to implement. Our scheme is very practical because it doesn't need any special equipment, its just need to rent some cloud servers and put your system on it for a specific period of time, it's also cost effective.

`

CHAPTER 5: Design and Implementation

5.1 Protocol Implementation:

The proposed protocol in 4 implemented using HELib library described in 2.4.1, the implementation done by C++ on Ubuntu 12. Installation method and example of HELib described in Annex 1.1.

5.2 System Structure

Our implemented software consists three main programs:

1) Authentication Server program
2) Voting Server program
3) Voter Program

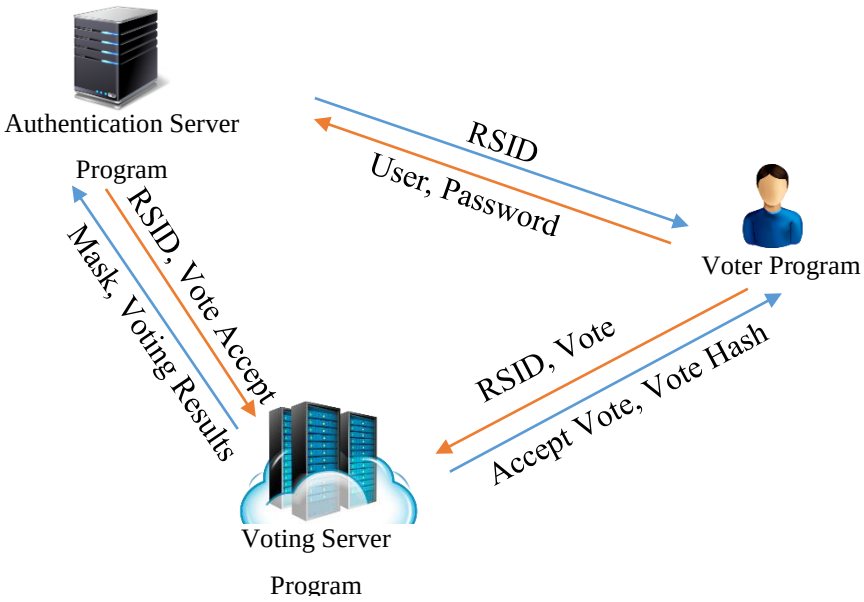


Figure 5.1 System Structure

All three programs can communicate with each other; all informations sent between programs encrypted in different ways, depending on the type of message.

`

5.2.1 Assumptions

The assumptions upon which we have built and the design this system are as follows:

1. Authentication server (AS) considered as a trusted party which fully controlled by authorities. AS contains sensitive data like users and passwords database, private keys and functions that generate RSK. AS should monitored and logged and controlled by the highest authorities of the Central Election Commission. Although this monitoring process does not reveal any information about votes or leaks partial results of the election process. Which this server does not contain any votes.
2. Voting Server (VS) considered as untrusted party of the system, it's hosted in some cloud service, and these cloud services considered as untrusted platform, where in some cased the vendor can access to the hosted services and serves and may reveal some sensitive data. Due to this issue, the FHE provided to solve security issues of untrusted platforms. VS could not reveal any data about users, votes and partial results. Where authentication with users done based on RSK provided by AS, and all votes and results are encrypted using FHE schema, and VS does not contain secret key for that scheme. All data are processed in encrypted form, which prevent any untrusted party form revealing any sensitive data.
3. Voter (V) consider as untrusted party, until it authenticate AS. The voter must provide secret credentials, which authenticate his identity. Then he transfer to the second level of trust, where he can authenticate VS using RSK provided by AS. A voter can encrypt his vote locally using the provided program, vote validated to check of correct encryption using a correct public key, and well defined vote according the condition provided by the Central Election Commission. The voter cannot prove his vote to anybody, and prevent coercion.
4. The communication between AS and VS considered as secured connection based on VPN services, or any other secure connection services. Although all messages transferred between AS and VS are encrypted and integrity checked.
5. The communication between Votes and system considered as untrusted anonymous connections, and all messages between Voters and system are encrypted and integrity checked.

`

5.2.2 Authentication Server Program

Authentication program responsible for:

1) Key generation:

   In the key generation process, public key and private key generated. Public key sent to both VS and BB.

   Before key generation, some credentials must be prepared depending on number of voters involved in the election process.

Table 5.1 Key Generation Parameters

Variable	Value	Description
m	0	*m, p and r* define the native plaintext space $\mathbb{Z}[X]/(\Phi_m(X), p^r)$
p	997	
r	1	*P* is defined as 997 for testing, while all results derived *mod p*. It must be a primary number higher final result.
l	3	The number of "levels", we chose it to 3 while the deepest circuit in our solution is Masking process see chapter 3)b)
c	3	Number of columns in key switching matrix
w	64	The Hamming weight of a secret key
Security	128	Used to derive *m*, with function `FindM(security,L,c,p, d, 0, 0);`
d	0	

The plaintext space defined over *G* (of type `ZZX`) a monic polynomial irreducible over $\mathbb{Z}_p$.

`

- **Public Key generation**: to initialize Public Key, an object `context` that holds (*m, p and r*) a part of class `FHEcontext,` that's responsible for maintaining the parameters. An object `FHEPubKey` is defined relative to a fixed `FHEcontext`, which must be supplied to the constructor and cannot be changed later. An `FHEPubKey` includes the public encryption key, which is a ciphertext of type `Ctxt`, a vector of key-switching matrices of type `KeySwitch`, and another data structure called `keySwitchMap` that is meant to help finding the right key-switching matrices use in different settings. The public key is just a ciphertext, encrypting the constant 0.

  Hamming weight of that key stored in `FHEPubKey` object, for every secret key in this instance. The `FHEPubKey` class provides an encryption method, and various methods to find and access key-switching matrices.

  Public key can be imported by providing context and creating an empty `FHEPubKey.` Context can be imported using:

  `readContextBase(inContxt, m, p, r).`

- **Secret Key generation:**

  Secret key generated via `FHESecKey` class provides for either generating a new secret-key, or importing a new secret key that was generated by the calling application. That is, we have the methods `ImportSecKey and GenSecKey.`

Table 5.2 PublicKey, SecretKey and Context sizes; p=997.

Key	Size
Public Key	20.3 MB (20,321,810 bytes)
Secret Key	20.3 MB (20,349,993 bytes)
Context	91 bytes

2) Voter authentication:

`

In vote generation stage VA program listens always for new voter requests. In this stage, the program establishes SSL connection to the voter as a response to the SSL request from the voter. The voter need to provide his international ID number and his secret password – provided by authority office in registration stage- to be verified and authenticated. This SSL connection used only for authentication stage to hide voter identity form any intruder.

The next stage of authentication is between voter and voting server. AS generates a random secret key, encrypt it with the voter secret password, sent it back to the voter with HMAC function used for message integrity. The same random secret key encrypted with pre-defined symmetric key between VA and VS, also it's sent to VS.

3) Vote verification:

After voter submit his vote to VS, VS calculates a vote mask described in section 3)b (vote verification procedure), AS program just always listen for mask verification requests came from VS. AS program decrypt mask and perform a check, which every field in the mask must be 1, else it's invalid vote.

Validation message (valid or invalid) saluted with a random number, encrypted with symmetric encryption with RSK as a key and sent to VS.

Size of masked vote about 204.3 kB (204,293 bytes). And the execution time of decryption is 0.019093/s. Also, decode function used internally in this step with execution time 0.038623/s. The total execution time of decryption is 0.057716/s, this considered a very small time for decryption which make the system applicable to work for many decryption processes.

4) Results decryption:

After the specified period of voting ends, end of vote message sent to VS. VS start tallying votes. AS program decrypt the results of voting using the private key, and finally validate the count of voters to valid vote count and send results to BB.

The execution time of the decryption function of final results came form VS is 0.011884/s and decode function is 0.0696/s, so the total decryption time is 0.081484/s.

5.2.3 Voting Server Program

`

The voting program responsible for

1) Voter authentication:

   The VS program receives RSK for AS, decrypt it and wait for voter to send hello message encrypted with same RSK. Once the voter provides correct RSK, he verified and become able to send vote to VS, if the provided RSK was wrong, VS sends reject message to the user, and store logs for that wrong RSK.

2) Vote mask calculation:

   The VS program calculates vote mask for every vote, each vote mask sent to AS for validation. If it's valid, vote acceptance sent to voter encrypted with RSK with HMAC.

3) Vote tallying:

   After the voting ends, a message received indicate that voting period ended, vote tallying starts. All results computed in one cipher and sent to AS to decrypt and publish results.

5.2.4 Voter Program

Voter program responsible for:

1) Voter Authentication:

   The first step in voter program is authentication with AS and then authenticate with VS. Voter first establishes SSL connection to AS then authenticate with his ID and password. Once authenticated he receives an RSK encrypted with his password, he decrypt it and use it for authentication with VS. Voter sends “hello message” encrypted with RSK using a symmetric encryption algorithm.

2) Ballot preparation:

   The Voter chooses his selection from candidates, and form his ballot in a specified way as described before in section 4. Voter program just presents just the candidate choices and user selects his choice. Voter program performs the preparation process.

3) Vote Encryption:

`

After ballot preparation voter program encrypt it using the public key provided on BB. The encrypted ballot size is about 136.1 kB (136,100 bytes) it's not constant value and vary for each user, but size almost the same with same parameters.

The encryption time is 0.027659/s. timing function in HELib provides easy to measure execution time of some basic functions. Function `printAllTimers()` used to print all timers in the program. Also function `setTimersOn()` needed to start the timer, and `setTimersOff()` to stop it. Every function needed to be monitored by individual times.

5.3 Security Properties

The implemented system achieved many security properties, some properties related to the voting process itself, which described in section 4.2, and some other propertied related to communication channels and hosting environments. This section discusses related issues.

5.3.1 Communication channels security

Communication done between servers AS and VS secured with two factors:

1. All messages sent between those servers are encrypted even messages like ACCPTED or REJECTED messaged are salted to be indistinguishable in the case of symmetric key encryption. All messages are equipped with an HMAC integrity check. This prevents any eavesdropper of intercept or change the messages transmitted over the channel.
2. The communication channel secured using VPN service, in this case, we suggest using OpenVPN service to secure connection, which an open source platform that provide high security and privacy of communication. OpenVPN can encrypt communications using many different symmetric key algorithms such as AES, and its use TLS protocol to provide secure commutations.

The communication between Voter side and AS and VS in other side secured using encryption and HMAC integrity check. All messaged between servers and Voter are encrypted and salted. All messages are equipped with an HMAC integrity check. This prevents any eavesdropper of intercept or change the messages transmitted over the channel.

`

5.3.2 Hosting environment security

AS hosted on dedicated servers that secured using Intrusion Detection Systems IDS and Intrusion Prevention Systems IPS, in addition to firewalls, to prevent any intruder from accessing AS. If such thing happened, the intruder will be able to access the most sensitive data in the system ID's and passwords and secret keys. To prevent this, we suggest securing AS by monitoring each communication trying to connect AS server, if any suspicious activity detected the connection must terminate.

VS hosted in cloud service, it also secured using IDS and IPS, which work to prevent any intruder from accessing VS. If such thing happened, the intruder is able to delete or corrupt some votes, this will lead to damage voting results. To prevent this, all connections must be monitored and if any suspicious activity detected, the connection must be terminated and event log of this activity registered, the voter can do authentication again to be verified.

Voter requested to secure his machine, any hacking to his local machine could lose him his vote. To prevent intruders form changing the structure of vote for example by infecting the victim of viruses that can change the vote structure or change the public key or corrupting votes. VS and AS are responsible for checking the validity of each vote. If the vote is corrupted or unverified, the user told with this issue and given some instruction to secure his machine again. The implemented Voter program should not be able to change or code recover.

`

CHAPTER 6: RESULTS AND ANALYSIS

6.1 Traffic analysis:

The presented protocol generates communication traffic between each part of the system, voter, VS and AS. The generated traffic achieved privacy, confidentiality and integrity. As shown in table 5.3, all traffic between system parts encrypted, checked with integrity function. This prevents intruders form changing the content of transferred messages and even change the message itself, while all messages encrypted with securely shared secret key.

Table 6.1 Traffic tracing and protection function

Sender	Message	Receiver	Confidentiality Function	Integrity Function
AS	Public Key	BB	Public	HMAC
AS	Public Key	VS	Public	HMAC
V	ID, Password	AS	SSL	SSL
AS	RSK	V	AES Encryption, key: password	HMAC
AS	RSK	VS	AES Encryption, key: predefined key	HMAC
V	Hello message	VS	Salted, AES Encryption, key: RSK	HMAC
V	Vote	VS	FHE, key: Public Key	HMAC
VS	Vote mask	AS	FHE, key: Public Key	HMAC
AS	Validation message	VS	Salted, AES Encryption, key: RSK	HMAC
VS	Acceptance message + Hash of Enc. Vote	V	AES Encryption, key: RSK	HMAC
AS	End of voting period message	VS	AES Encryption, Key: Predefined key	HMAC
VS	Final Result of tallied Vote Cipher	AS	FHE, Key: Public Key	HMAC
AS	Decrypted Final Results	BB	AES Encryption, Key: Predefined key	HMAC

`

6.2 Performance analysis

All previous results are done with p = 997 and a small number of candidates; p limits the number of voters. To achieve true decryption of results p must be larger than the number of voters, because all results are calculated modulo p. To examine the system scalability and capability to deal with large number of users and much candidates choices, we design a test to examine different p's and its reflections on key sizes , vote and mask size, also its reflection of encryption and decryption time.

The value of p in the test defines the maximum number of users should vote, which restricted to the number of calculated votes. All results of final tallying and mask calculation done modulo p. If the number of resulted value greater than p, the decryption result will be incorrect. We have to choose p greater than the maximum value of the result. Table 6.2, Results of testing reflection of p on key size , vote and mask size represent the results of described experiment.

Table 6.2, Results of testing reflection of p on key size , vote and mask size

	V ≈ 1,000	V ≈ 10,000	V ≈ 100,000	V ≈ 10,00,000	V≈ 10,000,000
	p=1,009	p= 10,007	p=100,003	p =1,000,003	p=10,000,019
Public key	6.1MB	7.1 MB	7.9 MB	12.4 MB	38.0 MB
Secret key	6.1MB	7.1 MB	7.8 MB	12.4 MB	38.1 MB
Vote	138.6 kB	133.2 kB	143.7 kB	235.4 kB	240.5 kB
Mask	208.0 kB	199.6 kB	215.4 kB	353.1 kB	360.7 kB

As shown in Figure 6.1 PublicKey size and Secretkey size for different. The result derived from Table 6.2, Results of testing reflection of p on key size , vote and mask size show that secret key size and public key size is identical for same p value. However, it differs when choosing a larger value of p. The largest key size hit in this experiment when p = 10,000,019, it has reached almost 38 MB for both secret key and public key. For public key, this

`

considered a large size, but it is necessary when voting made from a large number of persons such 10 million as described in Table 6.2, Results of testing reflection of p on key size , vote and mask size. For a less number of users such as 1 million keys decreased to 12.4 MB, which more affordable. Nevertheless, 38MB not too much size for growing speed of internet. We considered our system practical for such cases, because the public key will published on BB, and the user can take their time for receiving it.

An important issue is the stored votes total size, which if we considered each vote take an average of 250 kb of disk space, it need 2.328 terabit for 10 million users. This small size of storage compared to a large number of users, is suitable and affordable because this storage size available from most cloud providers and even for personal computer.

Figure 6.2 Vote and Mask sizes for different p values, the size of both vote and mask generally increase with greater values of *p*. Mask size greater than vote size, which mask is the vote itself process with a defined equation in section 4 which contains addition and multiplication operations which increase the size of resulted mask. The noise generated from the homomorphic addition with noise at most B is 2B and the noise generated form multiplication process is B^2. BGV provides a noise-management technique that keeps the noise in check by reducing it after homomorphic operations, its bases on “modulus switching” technique.

`

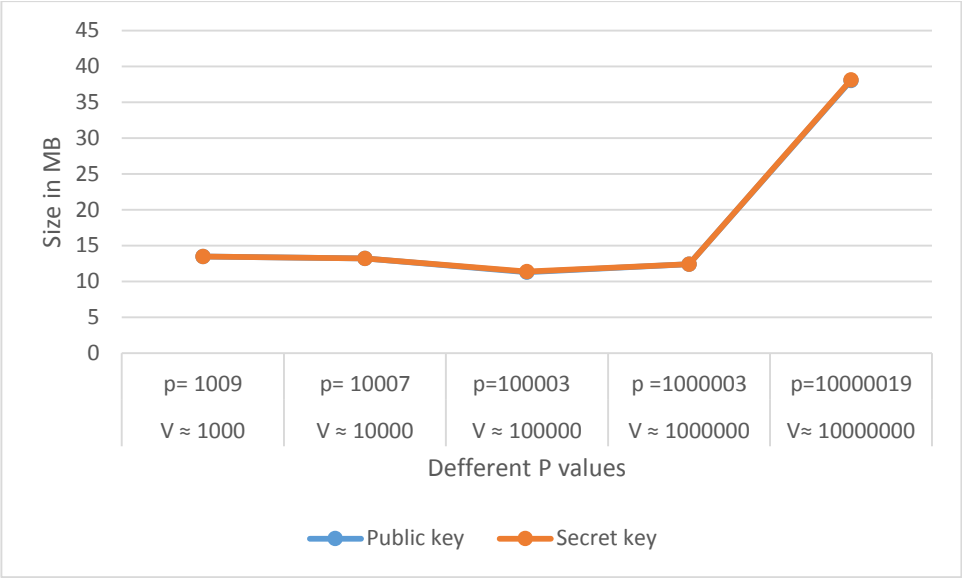


Figure 6.1 PublicKey size and Secretkey size for different p values

Figure 6.1 PublicKey size and Secretkey size for different p values shows that the mininumu recorded value public key and secret key on p = 100,003, and the maxixmun value of keys on p= 10,000,019.

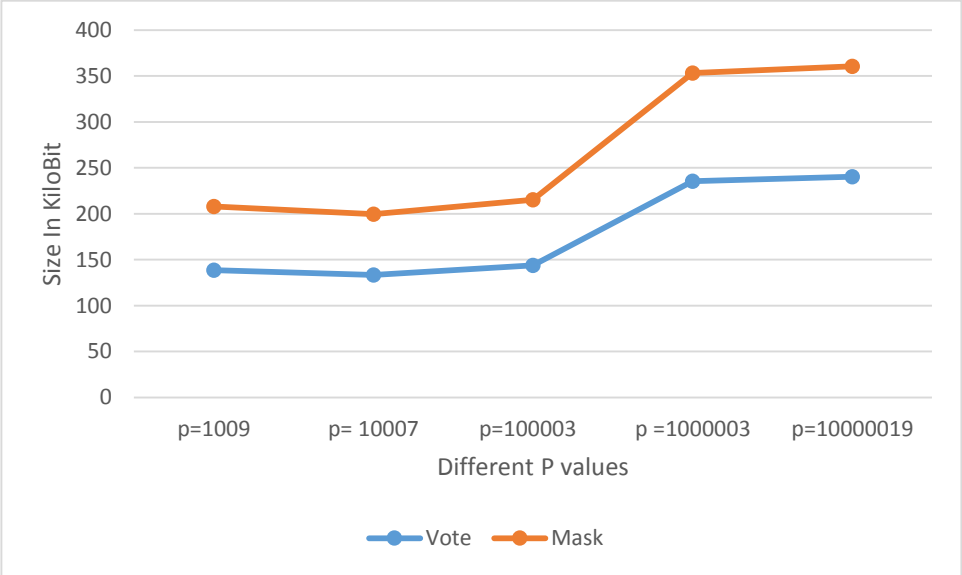


Figure 6.2 Vote and Mask sizes for different p values

values shows that the at p=1,000,003 and p=10,000,019 gives the largest value of vote and mask sizes, while the other p values gives almost the same size. This due to the change value of L=4 on p=10, 00,003 and p=10, 000,019, which gives an incorrect decryption of mask when L=3. Because NIZKP circuit contains addition and multiplication, we need to increase

`

the depth of the circuit to be compatible with resulted cipher while all result decryption done modulo p. NIZKP circuit increases the cipher text size and noise, which give incorrect decryption. For smaller p values it succeeded to decrypt mask correctly with smaller L=3, which the generated noise is smaller than *p*.

The second part of the test is distinguished the difference of encryption and decryption time for vote, mask and the result. In addition, mask calculation included which important component of system performance. Encryption and decryption time for deferent p values somewhat similar. The produced results are acceptable for our system because it's small and does not affected by changing *p* value. Mask calculation produced different results for different *p* value and in general produced higher results of encryption and decryption. This because of circuit size, which contains mutable addition and multiplication process.

Table 6.3 Vote encryption, mask calculation - decryption and result decryption for different p values

	V ≈ 1,000	V ≈ 10,000	V ≈ 100,000	V ≈ 1,000,000	V≈ 10,000,000
	p=997	*p*= 10,007	*p*=100,003	*p* =10,00,003	*p*=10,000,019
Vote Encryption	24.52/ms	27.51/ms	27.25/ms	31.65/ms	36.52/ms
Mask Calculation	269.8/ms	400.97/ms	446.57/ms	558.35/ms	608.39/ms
Mask Decryption	63.82/ms	37.85/ms	32.64/ms	54.73/ms	70.78/ms
Result Decryption	87.18/ms	79.76/ms	80.68/ms	78.27/ms	82.352/ms
Number of plaintext slots	53	54	84	60	66

The number of plaintext slots differs for each value of p; in our test, we used 31 slots for vote formulation. The vote itself takes 30 slot present voting for each candidate, the 31 slot is a

`

check digit described in section 3,b) . the rest plaintext value is zero, the number of plaintext slots is related to CRT technique used by HELib, described in [45] The resulted plaintext slot can fit up to 65 candidate when p=10,000,019. It's acceptable for most countries which number of candidates usually not big.

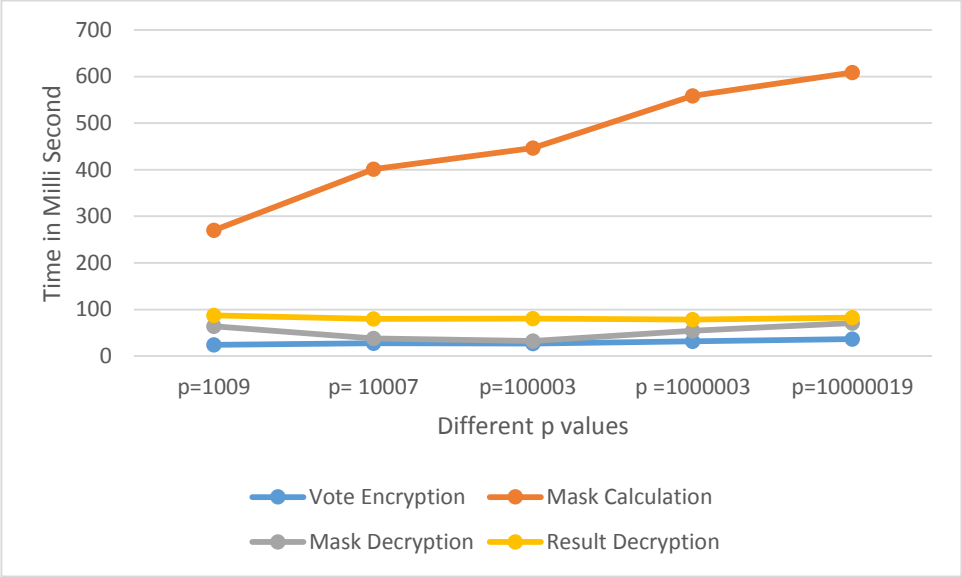


Figure 6.3 Vote encryption, mask calculation - decryption and result decryption for different p values

Figure 6.3 values shows the time consuming in vote encryption, mask decryption, mask calculation and result decryption different p values, which indicate that vote encryption is less time than other and it does not change with changing p value. Second mask decryption and third result decryption. The result of this test was by tallying 100 votes, thus size of the vote increase with sum calculation. The time is almost similar in vote encryption, mask decryption and mask calculation. The major difference is in mask calculation time, which increase by increasing p value.

Another part of our experiment, is examining the performance of different number of votes, in this part we designed a method to auto generate encrypted votes. Each vote filled randomly as a ballot between 30 candidates, and slot 31 filled with 1 as a check digit and encrypt it. For this experiment we use p=10, 000,019, which indicates the largest number of voters 10 million and the largest resulted public key. Table 6.4 Voting performance analysis for vary number of voters, where it takes about a half hour to generate 30,000 encrypted votes. This

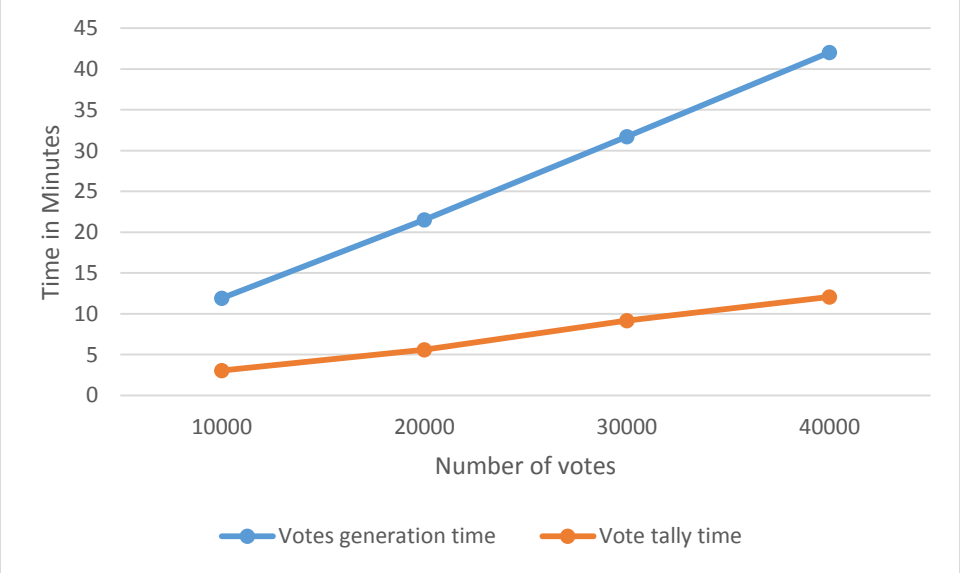
`

time seems to be linear with larger number of votes. In fact, this not as in real situations, where voters encrypt their votes at the same period separately, not in a sequential way as in this test.

Table 6.4 Voting performance analysis for vary number of voter

	Number of votes			
	10,000	20,000	30,000	40,000
Votes generation time	11m 56s	21m 12s	31m 43s	42m 2s
Vote tally time	3m 3s	5m 37 s	9m 9s	12m 4s
Result decryption time	70.303/ms	78.288/ms	84.312/ms	90.772/ms
Tallied result size	240.5/ kB	240.4/ kB	240.4/ kB	240.6/ kB
Total size of votes	2.4/GB	4.8/GB	7.3/GB	9.7/GB

The most important issue of this test is examinig the time of tallying a large number of votes, the largest number we examined in this test is 40,000 votes. It takes 42 minutes to tally this number of votes. The test was made on VMware virtual machine configured with 3G of RAM, 2 processors, 2 cores and 30G of hard disk. The host machine was core i5 processor.



`

Figure 6.4 Votes generation time, and Vote tally time for different number of votes

As shown in Figure 6.4, tallying time is somewhat linear in the number of votes. With these results, an expected time of tallying 10 million votes will be 50 hours, using one single virtual machine with the previous specifications. In the real situation this tally process will done in cloud, which may consist of several powerful nodes. The system is scalable, and it may contain hundreds of nodes, where tailing process is can be done in several nodes and the result of each node can be tallied to gather to get final results. This scalability will reduce the time of tallying much time.

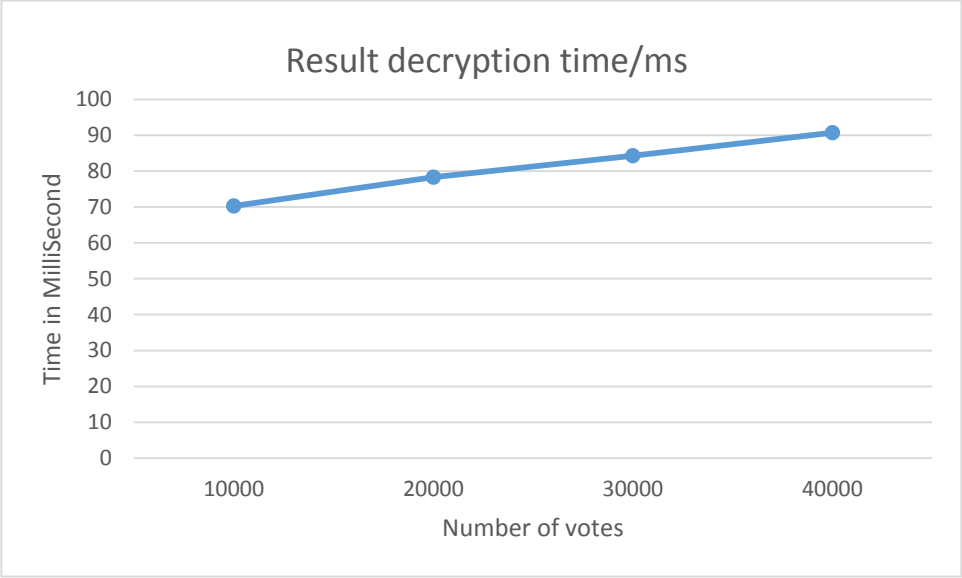


Figure 6.5 Final result cipher decryption time

The decryption of results after tallying finish shown in Figure 6.5 Final result cipher decryption time, it also increases linearly with the number of tallied votes, this because the noise generated by each homomorphic addition operation. The noise is not too much because addition has a small noise effect. Where addition of two ciphers generates 2B of noise, this is small compared with multiplication noise B^2.

Size of tallied results cipher is somewhat identical for different number of votes as shown in Table 6.4 Voting performance analysis for vary number of voters, this due to the reduction technique used by HELib.

`

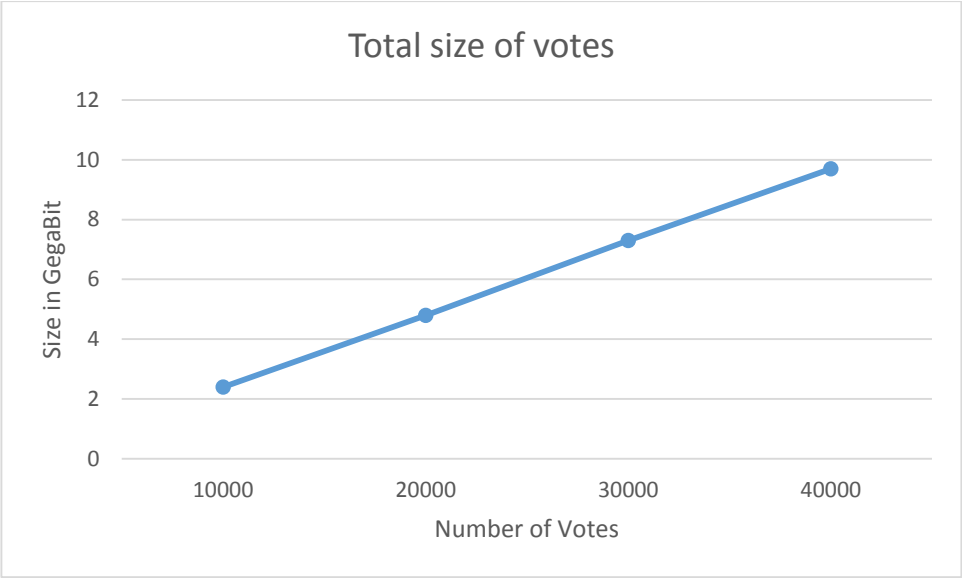


Figure 6.6 Total size of votes for different number of votes

Size of all votes is a big issue, while the size of each vote is small, the total size of large number of votes considered big. In this experiment, we examined the total size of votes at different number of votes. As shown in Figure 6.6 the largest size hit on 40,000 was 9.7/Gb. The size grows with the number of votes linearly. In an expectation for size of 10 million votes, it will take 2.4Tera bit of size, which very affordable in cloud systems. This size available now on some personal computers. For such systems, this considered acceptable size.

6.3 Stored data analysis

At some point, each part of the system has some data, this data maybe secret, public or useless data. In this section we analyze the data stored at each part and its security concerns.

6.3.1 Authentication Server stored data

The authentication server is the most critical part of the system, whereas it contains the most sensitive data in the system which private key, database of users – passwords and secret keys. This part of the system should be secured very well with the most recent ways of server security like an intrusion detection system IDS, intrusion prevention system IPS and firewall. It must be monitored in all the period of voting. AS also store temporary data such as RSK,

`

mask, mask decryption, and mask validation result with its salt and HMAC's. All these result deleted after voter commit his vote for each voter.

6.3.2 Voting Server stored data

VS stores vote cipher and hash function of that vote, until end of voting period ends. Other temporary data stored in VS such mask, RSK and HMAC's. Mask deleted immediately after chinking by AS. In addition, RSK deleted after the session ends with voter.

This provides the minimum information seen by VS, which could be any cloud service that considered untrusted and could reveal some information about the election process. Cloud provider or intruder could not have useful data can affect voting process or clarify vote or voter personality. It also could not leak partial results while all votes encrypted.

6.3.3 Voter stored data

The voter machine contains temporarily authentication credentials, ID and password. RSK, HMAC, vote and its encryption, which temporary data. The hash function of encrypted vote stored at the voter side for validation. Vote computed and revote process starts, if coercion happens.

6.4 General analysis

In general, the system divided to separated parts to prevent any intruder can access one part of the system from affecting the result of the election or leaking partial results or connect any voter to his vote.

No one other than registered users can vote or access system. Each voter can vote without revealing his identity and no one can connect a vote to a voter. Every vote checked whether it has encrypted with valid public key and formed in the correct format of voting and no addition values added to a specified candidate to increase his result, also no fake votes made.

No one can compute partial results, or interrupt voting process, all communication processes encrypted and integrity checked. Any manipulation tries should be discovered by system, reported and prevented. The user can revote when he felt coerced.

`

The system is scalable while it can deal with large number of users at the same time, and system structure can easily expand without affecting of system functionality. It's also very practical to be used in real election processes.

The system satisfies the major properties of an optimal voting system such as eligibility, privacy, accuracy, verifiability, fairness, receipt-freeness, incoercibility, dispute-freeness, robustness, scalability and practicality.

`

CHAPTER 7: CONCLUSIONS AND FUTURE WORK

This thesis examines the applicability of FHE in e-voting systems through designing and implementing internet based voting system. The implemented system able to work through cloud infrastructure. The conclusions of this work described below.

7.1 Conclusions

This thesis presented an electronic voting system based on fully homomorphic encryption as a case study, to understand how much fully homomorphic encryption is applicable in real life systems. The proposed e-voting system consists of main components, authentication server, voting server, bulletin board and on the other side voters. The separation of authentication server and voting server, let the voting server could be hosted in any cloud service provider or any datacenter service. This provides more privacy, which all votes stored in authentication server encrypted with fully homomorphic encryption and can processed or calculated in encrypted form. This led to another feature, scalability and cost effectiveness. The system could easily expand to more cloud server without compromising system structure or functionality. Using cloud services for a specified period of election obviates buying new hardware each election cycle. This is sufficient for us to afford the burden of maintaining and updating hardware for the next election cycle.

We implemented the proposed system using HELib [1] homomorphic library based on BGV [19] fully homomorphic encryption scheme. The implementation divided into three parts, authentication server program, voting server program and voter program. We tested results where the system should deal up to 10 million voter, which meets the need of about 70% of countries over the world according to the number of eligible users. The results were applicable for public key size, vote size, mask calculation time, mask decryption time, total size of votes before tallying, tallying time and decryption result.

Security concerns of voting systems considered in our work. The developed system was able to prevent intruder form make any fake votes or affect the voting process. The system disable anybody from linking between voters and their votes, even the voters themselves. Every vote checked for validation test. All communications encrypted and integrity checked. No one could calculate partial results even cloud provider. The system satisfies many security

`

concerns eligibility, privacy, accuracy, verifiability, fairness, receipt-freeness, incoercibility, dispute-freeness, robustness, scalability and practicality.

The implemented e-voting was acceptable to work in real elections, with providing more cloud processing power.

7.2 Future work

Fully homomorphic encryption has many applications, in this thesis we discussed in details one of these applications, which is voting system and its applicability to deploy to cloud services.

The implemented e-voting systems need to add usability features to be more user friendly. And it need to compared with other systems.

In future work we intended to discuss other types of applications that applicable to work with cloud infrastructure to study applicability performance and security issues of FHE.

The depth of the circuit in FHE considered as a limitation of the practicality of FHE, we intended to examine much larger in depth circuits to study its effects on performance and resulted ciphers.

In addition to, optimizing our implemented voting system to decrease public key size, vote size and mask size. In addition, to use some other functions of HELib, which deals with plaintext slots and noise optimization.

`

1 ANNEX 1: Implementation of F.H.E

1.1 HELib library:

1.1.1 Install (Compilation)

Installing and compilation of HELib is easy, its depend on NTL (Number Theory Library) version 6.+, I tyried installing HELib on Ubuntu 12.

First we need to install NTL:

```
osama@ubntu:~$ sudo apt-get install g++
osama@ubntu:~$ sudo apt-get install libgmp3c2
osama@ubntu:~$ sudo apt-get install libgmp3-dev
osama@ubntu:~$ wget http://www.shoup.net/ntl/ntl-6.0.0.tar.gz
osama@ubntu:~$ tar -xvf ntl-6.0.0.tar.gz
osama@ubntu:~$ cd ntl-6.0.0/
osama@ubntu:~$ cd src
osama@ubntu:~$ ./configure
osama@ubntu:~$  make
osama@ubntu:~$  make check
osama@ubntu:~$ sudo make install
```

Then install HELib, first we need to install git application, to be able to download HELib from githup.

```
osama@ubntu:~$ sudo apt-get install git
osama@ubntu:~$ git clone https://github.com/shaih/HElib.git
osama@ubntu:~$ cd Downloads/
osama@ubntu:~$ cd HElib-master/
osama@ubntu:~$  cd src/
osama@ubntu:~$  make
   ; open Makefile and chnge the path inside it into the path
of your NTL include direcroty /home/osama/ntl-6.0.0/include
osama@ubntu:~$  nano Makefile
osama@ubntu:~$  make
osama@ubntu:~$ make test
```

1.1.2 Testing:

An example of HELib where created to show the basic idea of HE [64], this example where based on Test_* files included in HELib, the example showd below:

`

App.cpp

```
#include "FHE.h"
#include "EncryptedArray.h"
#include <NTL/lzz_pXFactoring.h>
#include <fstream>
#include <sstream>
#include <sys/time.h>

int main(int argc, char **argv)
{
    /* On our trusted system we generate a new key
     * (or read one in) and encrypt the secret data set.
     */

    long m=0, p=2, r=1; // Native plaintext space
                        // Computations will be 'modulo p'
    long L=16;          // Levels
    long c=3;           // Columns in key switching matrix
    long w=64;          // Hamming weight of secret key
    long d=0;
    long security = 128;
    ZZX G;
    m = FindM(security,L,c,p, d, 0, 0);
FHEcontext context(m, p, r);
    // initialize context
    buildModChain(context, L, c);
    // modify the context, adding primes to the modulus chain
    FHESecKey secretKey(context);
    // construct a secret key structure
    const FHEPubKey& publicKey = secretKey;
    // an "upcast": FHESecKey is a subclass of FHEPubKey

    //if(0 == d)
    G = context.alMod.getFactorsOverZZ()[0];

   secretKey.GenSecKey(w);
   // actually generate a secret key with Hamming weight w

   addSome1DMatrices(secretKey);
   cout << "Generated key" << endl;

   EncryptedArray ea(context, G);
   // constuct an Encrypted array object ea that is
   // associated with the given context and the polynomial G
```

`

```
    long nslots = ea.size();

vector<long> v1;
    for(int i = 0 ; i < nslots; i++) {
        v1.push_back(i*2);
    }
    Ctxt ct1(publicKey);
    ea.encrypt(ct1, publicKey, v1);

    vector<long> v2;
    Ctxt ct2(publicKey);
    for(int i = 0 ; i < nslots; i++) {
        v2.push_back(i*3);
    }
    ea.encrypt(ct2, publicKey, v2);

// On the public (untrusted) system we
    // can now perform our computation

    Ctxt ctSum = ct1;
    Ctxt ctProd = ct1;

    ctSum += ct2;
    ctProd *= ct2;
vector<long> res;
     ea.decrypt(ctSum, secretKey, res);

     cout << "All computations are modulo " << p << "." << endl;
     for(int i = 0; i < res.size(); i ++) {
         cout << v1[i] << " + " << v2[i] << " = " << res[i] <<
endl;
     }

     ea.decrypt(ctProd, secretKey, res);
     for(int i = 0; i < res.size(); i ++) {
         cout << v1[i] << " * " << v2[i] << " = " << res[i] <<
endl;
     }

     return 0;
}
```

`

1.1.3 Parameter Description:

R is the number of rounds

p is the plaintext base [default=2]

r is the lifting [default=1]

d is the degree of the field extension [default==1]

(d == 0 => factors[0] defined the extension)

c is number of columns in the key-switching matrices [default=2]

k is the security parameter [default=80]

L is the # of primes in the modulus chai [default=4*R]

s is the minimum number of slots [default=4]

m is a specific modulus

repeat is the number of times to repeat the test

You can compile App.cpp code with: (App.cpp in /scr directory)

```
sudo g++ Mytest.cpp fhe.a -o App \-L/usr/local/lib -lntl
```

1.2 Scarab library

1.2.1 **Installation**:

This installation was tested on Ubuntu 12.04 32bit, kernel Linux 3.2.0-60-genric-pae, processor Intel Core i5 CPU M60 2.67 GHz *4, memory 4GiB, virtualized machine using VMware.

Download:

- libScarab-1.0.0 [65].
- gmp-6.0.0
- flint-1.6
- mpfr-3.1.2
- mpir-2.6.0

`

```
# Install m4 and lzip

sudo apt-get install m4

sudo apt-get install lzip

# Install gmp

lzip -d gmp-6.0.0.tar.lz

tar xf gmp-6.0.0.tar

cd gmp-6.0.0

./configure

make

make check #(don't skip the checks!)

sudo make install

# Install mpfr

tar xjf mpfr-3.1.2.tar.bz2

cd mpfr-3.1.2

make

make check

sudo make install

# Install mpir

tar xjf mpir-2.6.0.tar.bz2

cd mpir-2.6.0

./configure

make

make check

sudo make install

# Install flint

tar xf flint-1.6.tgz

cd flint-1.6

source flint_env
```

`

```
make library

sudo cp libflint.so /usr/local/lib

sudo cp *.h /usr/local/include

sudo mkdir -p /usr/local/include/zn_poly/src

sudo cp zn_poly/include/*.h /usr/local/include/zn_poly/src/

# Run libscarab test

mkdir libscarab

cd libscarab

unzip ../libScarab-1.0.0.zip

make

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib

./integer-fhe
```

`

Bibliography

[1] S. Halevi, "GitHub -HELib," 31 3 2013. [Online]. Available: https://github.com/shaih/HElib. [Accessed 28 4 2014].

[2] Z. Brakerski, C. Gentry and V. Vaikuntanathan, "Fully Homomorphic Encryption without Bootstrapping," *Electronic Colloquium on Computational Complexity ECCC*, pp. 1-26, 2011.

[3] R. L. Rivest, L. Adleman and M. L. Dertouzos, "On data banks and privacy homomorphisms," *Foundations of Secure Computation*, p. 169{180, 1987.

[4] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," *Springer*, 1999.

[5] C. Gentry, A Fully Homomorphic Encryption Scheme, Stanford University, 2009.

[6] C. Gentry, "Fully homomorphic encryption using ideal lattices," *Michael Mitzenmacher , editor, STOC*, no. ACM, p. 169–178, 2009.

[7] J. Coron and A. Mandal, "Fully homomorphic encryption over the integers with shorter public," *Advances in Cryptology*, pp. 1-24, 2011.

[8] C. Gentry and S. Halevi, "Implementing Gentry ' s Fully-Homomorphic Encryption Scheme," *Advances in Cryptology–EUROCRYPT* , pp. 1-29, 2011.

[9] N. Smart and F. Vercauteren, "Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes," *Public Key Cryptography – PKC 2010 Berlin, Heidelberg, New York*, 2010.

[10] D. Stehlé and R. Steinfeld, "Faster Fully Homomorphic Encryption Damien," *Advances in Cryptology-ASIACRYPT 2010*, 2010.

`

[11] M. van Dijk, C. Gentry, S. Halevi and V. Vaikuntanathan, "Fully Homomorphic Encryption over the Integers," *Advances in Cryptology–EUROCRYPT 2010,* pp. 1-28, 2010.

[12] N. Smart and F. Vercauteren, " Fully Homomorphic SIMD Operations," *IACR Cryptology ePrint Archive,* 2011.

[13] C. Gentry and S. Halevi, "ully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits," *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on. IEEE,* pp. 107-109, 2011.

[14] C. Gentry, S. Halevi and N. . P. Smart, "Better Bootstrapping in Fully Homomorphic Encryption," *Public Key Cryptography–PKC 2011,* 2011.

[15] I. Sharma, "Fully Homomorphic Encryption Scheme with Symmetric Keys," *Master Thesis for Master of Technology Department of Computer Science & Engineering, Rajasthan Technical University, Kota,* 2013.

[16] J.-S. Coron, A. Mandal, D. Naccache and M. Tibouchi, "Fully Homomorphic Encryption over the Integers with Shorter Public Keys," *Advances in Cryptology–EUROCRYPT 2011y,* pp. 1-24, 2011.

[17] Z. Brakerski and V. Vaikuntanathan, "Efficient Fully Homomorphic Encryption from (Standard) LWE," *appears in this proceedings. Also available at Cryptology ePrint Archive,,* 2011.

[18] V. Vaikuntanathan, "Computing blindfolded: New developments in fully homomorphic encryption.," *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on. IEEE,* 2011.

[19] Z. Brakerski, C. Gentry and V. Vaikuntanathan, "(Leveled) Fully Homomorphic Encryption without Bootstrapping," *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference on - ITCS '12,* pp. 309-325, 2012.

[20] P. Fauzi, "On Fully Homomorphic Encryption," *Master's Thesis, University of Tartu ,Faculty of Mathematics and Computer Science Institute of Computer Science*, 2012.

[21] C. Gentry, A. Sahai and B. Waters, "Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based," *Cryptology ePrint Archive, Report 2013/340,* 2013.

[22] A. Lopez-Alt, E. Tromer and V. Vaikuntanathan, "On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption," *Cryptology ePrint Archive, Report 2013/094,* 2013.

[23] K. Peng, C. Boyd, E. Dawson, B. Lee and R. Aditya, "Multiplicative Homomorphic E-Voting," *Progress in Cryptology-INDOCRYPT 2004. Springer Berlin Heidelberg.*, pp. 61-72, 2005.

[24] Y. Hu, "Improving the Efficiency of Homomorphic Encryption Schemes," *Diss. Virginia Tech,* 2013.

[25] J. Sen , "Homomorphic Encryption: Theory & Application," in *Theory and Practice of Cryptography and Network Security Protocols and Technologies*, InTech, 2013, pp. 1-21.

[26] I. Damgard, V. Pastro, N. Smart and S. Zakarias, "Multiparty Computation from Somewhat Homomorphic," pp. 1-46, 2013.

[27] G. Gentry, "Fully homomorphic encryption using ideal lattices," *ACM,* pp. 169-178, 2009.

[28] M. Yokoo and K. Suzuki, "Secure Multi-agent Dynamic Programming based on Homomorphic Encryption and its Application to Combinatorial Auctions," *ACM,* pp. 102-119, 2002.

`

[29] J. C. Benaloh, "Secret Sharing Homomorphisms: Keeping Shres of a Secret Secret," *Springer*, pp. 251-260, 1998.

[30] J. Groth, "Minimizing Non-interactive Zero-Knowledge Proofs Using Fully," *International Association for Cryptologic Research IACR,* pp. 1-14, 2011.

[31] "Zero-Knowledge Proofs for Finite Field Arithmetic, Or: Can Zero-Knowledge be for Free?," *Advances in Cryptology - Proceedings of CRYPTO'98,* pp. 424-441, 1998.

[32] A. Kiayias and Y. Moti , "Tree-homomorphic encryption and scalable Hierarical Secret-Ballot Election.," *Springer*, 2010.

[33] R. Cramer, R. Gennaro and B. Schoenmakers, "Asecure and optimally Efficient Muti-Authority Election schema," *Springer* , 1998.

[34] B. Pfitzmann and M. Waidner, "Anonymous Fingerprinting," *Springer,* vol. 1233, p. 88–102, 1997.

[35] M. Kuribayashi and H. Tanaka, "Fingerprinting Protocol for Images Based on Additive Homomorphic Property," *IEEE TRANSACTIONS ON IMAGE PROCESSING*, vol. 14, no. 12, pp. 2129-2139, 2005.

[36] H. Delfs and H. Knebl, Introduction to Cryptography Principles and Applications, Berlin Heidelberg: Springer, 2007.

[37] P.-A. Fouque, G. Poupard and J. Stern, "Sharing Dycryption in the Context of Voting or Lotteries," *Proceedings of the 4 th International Conference on Financial Cryptography (FC'00),* vol. 1962, pp. 90-104, 2000.

[38] P. Golle, M. Jakobsson, A. Juels and P. Syverson, "Universal Re-encryption for Mixnets," *Topics in Cryptology–CT-RSA 2004. Springer Berlin Heidelberg,*, 2004.

[39] Y. Xiao., "Security in Sensor Networks," *Auerbach Publications,* pp. 275-290, 2007.

[40] J. Sen, "Secure and Privacy-Preserving Data Aggregation Protocols for Wireless Sensor Networks," *Cryptography and Security in Computing,* 2012.

[41] L. Ertaul and k. Vaidehi, "Computing Aggregation Function Minimum/Maximum using Homomorphic Encryption Schemes in Wireless Sensor Networks (WSNs)," *California State University, East Bay Hayward, CA, USA,* 2007.

[42] M. Brenner, J. Wiebelitz, G. von Voigt and M. Smith, "Secret Program Execution in the Cloud Applying Homomorphic Encryption," *Digital Ecosystems and Technologies Conference (DEST), 2011 Proceedings of the 5th IEEE International Conference on. IEEE,* pp. 114-119, 2011.

[43] D. Park, J. Kang, K. Heo, S. Cho, Y. Yoon and K. Yi, "Encrypted Execution," *Research on software analysis for error-free computing ROSAEC MEMO,* 2014.

[44] C. Gentry, S. Halevi and N. Smart, "Homomorphic Evaluation of the AES Circuit," *CRYPTO,* 2012.

[45] S. Halevi and V. Shoup, "Design and Implementation of a Homomorphic-Encryption Library," 2013.

[46] S. Halevi and V. Shoup, "Algorithms in HElib," *IACR Cryptology ePrint Archive.,* 2014.

[47] S. Drew, T. Finkenauer, Z. Durumeric, J. Kitcat, H. Hursti, M. MacAlpine and J. A. Halderman, "Security Analysis of the Estonian Internet Voting System.," *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM,* pp. 703-715, 2014.

[48] M. A. Bingol, F. Birinci, S. Kardas and M. S. Kiraz, "Norwegian Internet Voting Protocol Revisited: Security and Privacy Enhancements," *International Conference BulCrypt, Sofia, Bulgaria,* 2012.

[49] A. Fujioka, T. Okamoto and K. Ohta, "A practical secret voting scheme for large Scale Elections.," *Advances in Cryptology-AUSCRYPT. Springer-Verlag*, p. 244–251, 1992.

[50] C. Park , K. Itoh and K. Kurosawa, "Efficient anonymous channel and all/nothing election scheme," *In Advances in Cryptology - EUROCRYPT '93, LNCS, Springer-Verlag*, p. 248–259, 1993.

[51] J. K. K. Sako, "Reciept-free Mix-Type Voting Scheme," *Advances in Cryptology—EUROCRYPT'95. Springer Berlin Heidelberg,,* p. 393–403, 1995.

[52] D. C. a. M. J. A. Juels, "Coercion-Resistant Electronic Elections," *Proceedings of the 2005 ACM workshop on Privacy in the electronic society. ACM*, pp. 61-70, 2005.

[53] W.-S. Juang and C.-L. Lei, "A secure and practical electronic voting scheme for real world environments," *IEICE Transaction on Fundamentals of Electronics, Communications and Computer Science*, p. 64–71, 1997.

[54] A. Huszti, "A secure electronic voting scheme.," *Electrical Engineering 51*, pp. 141-146, 2008.

[55] I. R. a. N. N. I. Ray, "An anonymous electronic voting protocol for voting over the Internet," *Third InternationalWorkshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS '01),* 2001.

[56] T. ElGamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE TRANSACTIONS ON INFORMATION THEORY*, Vols. IT-31,, no. 4, 1985.

[57] O. Baudron, P.-A. Fouque, D. Pointcheval, J. Stern and G. Poupard, "Practical multi-candidate election system," *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing. ACM,*, 2001.

`

[58] I. Damgard and M. Jurik, "A generalisation, a simplification and some applications of Paillier's probabilistic public-key system.," *Public Key Cryptography. Springer Berlin Heidelberg,*, 2001.

[59] A. Mohr, "A Survey of Zero-Knowledge Proofs with Applications to Cryptography," *Southern Illinois University, Carbondale,* pp. 1-12, 2007.

[60] J. Groth, "Efficient Zero-Knowledge Arguments from Two-Tiered Homomorphic Commitments," *Advances in Cryptology–ASIACRYPT 2011,* 2001.

[61] M. Blum, P. Feldman and S. Micali, "Non-interactive zero-knowledge and its applications," *In STOC,* p. 103–112, 1988.

[62] C. Gentry, J. Groth, C. Peikert and A. Smith, "Using Fully Homomorphic Hybrid Encryption to Minimize Non-interative Zero-Knowledge Proofs," *Journal of Cryptology (2014),* pp. 1-22, 2014.

[63] M. Bellare, R. Canetti and H. Krawczyk, "Keying hash functions for message authentication," *Advances in Cryptology—CRYPTO'96. Springer Berlin Heidelberg,,* 1996.

[64] T. DuBuisson, "GitHub -Secure Computation with HELib," 13 5 2013. [Online]. Available: http://tommd.github.io/. [Accessed 28 4 2014].

[65] M. Brenner and H. Perl, "hcrypt project," 2011. [Online]. Available: https://hcrypt.com/scarab-library/. [Accessed 15 5 2014].

[66] S. Jaydip , Theory and Practice of Cryptography and Network Security Protocols and Technologies, InTech, 2013.

[67] N. P. Smart and F. Vercauteren , "Fully Homomorphic SIMD Operations," *Designs, Codes and Cryptography,* vol. 71, no. 1, pp. 57-81, 2012.