I. J. Computer Network and Information Security, **2018, 1, 44-59**
Published Online January 2018 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2018.01.06

Using Homomorphic Cryptographic Solutions on E-voting Systems

Ahmed A. Abu Aziz
Department of Computer Engineering Faculty of Engineering Islamic University of Gaza, Gaza Strip, Palestine
E-mail: a.abuaziz@gmail.com

Hasan N.Qunoo
Department of Software Engineering Faculty of Applied Engineering University of Palestine, Gaza Strip, Palestine
E-mail: hassanq@gmail.com

Aiman A. Abu Samra
Department of Computer Engineering Faculty of Engineering Islamic University of Gaza, Gaza Strip, Palestine
E-mail: aasamra@iugaza.edu.ps

Received: 12 August 2017; Accepted: 22 September 2017; Published: 08 January 2018

***Abstract*—**Homomorphic Cryptography raised as a new solution used in electronic voting systems. In this research, Fully Homomorphic encryption used to design and implement an e-voting system. The purpose of the study is to examine the applicability of Fully Homomorphic encryption in real systems and to evaluate the 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 research, fully homomorphic encryption used to provide both operations additive and multiplication, which ease the demonstration of none interactive zero-knowledge proof NIZKP. The proposed e-voting system achieved most of the important security issues of the internet-voting systems such as eligibility, privacy, accuracy, verifiability, fairness, and others. 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. Addition and multiplication properties of fully homomorphic encryption were 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 voters.

***Index Terms*—**Fully Homomorphic Encryption, FHE, E-voting, Non-Interactive Zero Knowledge Proof, NIZKP.

I. INTRODUCTION

Voting is a decision making system in modern societies depends on the proper administration of popular elections. In elections, each voter should be confident that his intents were correctly captured and no modification was done to his vote. In addition, all eligible votes were correctly tallied. On the other side, the voting system should ensure that each vote was done in the right way and voter coercion is unlikely. These conflicting requirements present 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.

E-voting is an interdisciplinary subject and should be studied from different domains, such as software engineering, cryptography, network security, politics, law, economics and social science. Mostly e-voting is known as a challenging topic in cryptography because of the need to achieve privacy, anonymity and vote encryption. Many e-voting systems based on complicated encryption schemes and other based on mix net model, blind signature model, and homomorphic encryption model.

Cryptographic solutions provide methods of storing or transferring data in a secure way, the amount of data generated is growing in a huge manner. So, cloud services are a suitable solution for storing such huge amount of data. Since cloud technologies are one of the most cost-saving and scalable solutions for processing and saving large data. The need to process encrypted data stored in the cloud becomes 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 be retrieved after decrypting cipher using the secret key. In asymmetric encryption, private and public keys generated, the 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 be made on two ciphertexts to calculate the result, which will be the same result if this operation performed on plaintext.

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. Anyway, it is still applicable to some protocols concern in security. Section II.A and section II.B describes in detail the homomorphic encryption.

The research structured as follow, section II introduce the homographic encryption, then presents a literature review of previous fully homomorphic encryption schemes, properties, underlying principles, and limitations. This section also focuses on e-voting systems and give a brief explanation of the previous voting systems. Section III 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. Section IV presents the structure of the implemented voting system and describes the programming properties of each part of the system. Section V presents analysis and results of the implemented voting system, it shows traffic analysis, performance analysis, and stored data analysis. Section VIVI the final section a conclusion and the future developments described.

II. Related Works

This research studies an old problem in cryptography called a privacy homomorphism. It was introduced by Rivest, Adleman and Derto uzous [1] after the invention of RSA, which is a multiplicative homomorphic encryption scheme.

A. 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:

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

This property led Rives et al [1]. to think about what if we have a schema that is fully homomorphic: a schema $\boldsymbol{\varepsilon}$ have an efficient Evaluateε algorithm that can evaluate any circuit C contains any operation not just multiplication, for any public key pk, where:

$$\boldsymbol{c_i} = \textbf{Encrypt}\varepsilon(pk, m_i) \text{ Gives:} \quad (1)$$

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

Availed encryption of $C(m_1, \dots., m_t)$ under pk. This can arbitrarily compute on encrypted data, so 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.

Decryption must give the same result of the operation as the operation done in clear, this powerful property can work for more complicated circuits, along with other operations based on addition and multiplication. Fig.*1*. Homomorphic Encryption Evaluation 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 an untrusted PC. 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. Homomorphic encryption proved to be the ultimate cryptographic solution to ensure the security of data on cloud [2], e.g. Location Privacy using Homomorphic Encryption over Cloud [3].

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

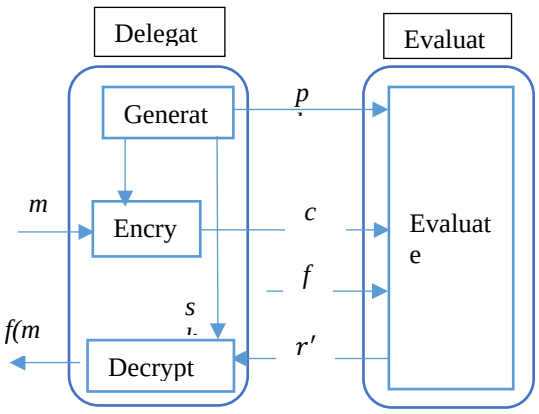

Fig.1. Homomorphic Encryption Evaluation

B. Fully Homomorphic Encryption Scheme

1. Gentry's Scheme

Gentry described the first Fully Homomorphic Encryption scheme in 2009 [4], which considered 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.

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 which makes the refreshed ciphertext applicable for the homomorphic operation (addition or multiplication), while it's not possible for the original ciphertext due to 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 [5].

2. Implementation of Gentry's blueprint - Smart-Vercauteren

The first attempt to implement Gentry's scheme was made in 2010 by Smart and Vercauteren [6], 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 could not obtain a bootstrappable scheme or a fully homomorphic scheme.

3. Gentry-Halevi Scheme

Gentry and Halevi described the first implementation of Gentry's scheme [5]. 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.

4. Improvements on Gentry's scheme

a) Stehle-Steinfeld optimizations

Stehle and Steinfeld described two improvements [7] 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 [6] and van Dijk et al [8].

b) SIMD Gentry optimization

In [6] 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 [4] was just able to perform encryption and decryption on a plaintext of one-bit length.

Gentry and Halevi [5] addressed the slowness of key generation process of the Smart–Vercauteren system [6], but their key generation method excluded the SIMD style operation offered by Smart and Vercauteren.

c) Gentry-Halevi without squashing

Gentry and Halevi describe in [9] a new approach to construct a fully homomorphic scheme encryption without the need to squash process. Previous schemes follow 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.

d) Gentry-Halevi-Smart scheme

Gentry, Halevi and Smart [10] 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.

5. DGHV fully homomorphic scheme over the integers

DGHV fully homomorphic scheme over the integers described in [8] 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.

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 [11].

6. DGHV shorter public key

Coron et al, [12] 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 [5] 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.

7. 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 [13] 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.

8. Brakerski-Gentry-Vaikuntanathan BGV scheme

Brakerski, Gentry and Vaikuntanathan in [14] [15] presented a new FHE scheme based on previous work of Brakerski and Vaikuntanathan in [13]. 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 [13] is the use of well-known security assumptions based on Ring Learning with Error RLWE, where the introduced RLWE over standard LWE provide a more efficient fully homomorphic scheme. Also, a fully homomorphic encryption without the need for bootstrapping achieved using modulus switching.

9. Gentry-Sahai-Waters scheme

Gentry, Sahai and Waters described in [16] 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.

10. NTRU based FHE

Lopez-Alt, Tromer and Vaikuntanathan in [17] 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 the main contribution of independent interest. Although the transformation to a fully homomorphic system deteriorates the efficiency of NTRU somewhat.

C. E-voting Systems

E-voting systems have a large space of research in cryptography literature, which many secure ballot election schemes have been offered, homomorphic encryption raised as one of those solutions for election schemes, which provide security, trust, and scalability. In such scheme, 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 [18].

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 [19] [20]. 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 the public Internet. Development of i-voting systems has been attractive for many researchers and developers because it uses the widespread of mobiles, smartphones 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 is a lot of research in this field. Counting process in i-voting systems can classified into three main methods, mix-nets model, blind signature

model, and homomorphic model.

D. 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 [21] [22].

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 [23] [24] [25].

Homomorphic Model: In the 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 the property of verifiability while preserving privacy. As shown earlier in section A the property of homomorphism is performed on addition and multiplication ($\oplus$,$\otimes$) which also described in section 2.0. Many homomorphic voting systems derived from the theory of ElGamal cryptosystem [26], which is additive homomorphic. Another voting systems based on multiplicative homomorphic Paillier cryptosystem [27] are proposed in [28] [29]. All these systems support only additive or multiplicative homomorphism only.

[30] uses the mobile application based systems with Smart Card based E-Governance System that allows the use of a mobile application to input user identification number using the Multipurpose Electronic Card (MEC) based E-Governance system. In case of successful authentication, the voter allowed to caste the original vote.

E. Zero-Knowledge Proofs

Zero-knowledge proofs could 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 [31]. 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 need 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 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 [32].

F. Non-Interactive Zero-knowledge Proofs

Non-interactive zero-knowledge (NIZK) proof systems [33] 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

Gentry [4] 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 plaintext 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 [34].

G. 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 [14], along with many optimizations to run homomorphic evaluation runs faster, focusing mostly on the effective use of the Smart-Vercauteren [35] ciphertext packing techniques and the Gentry-Halevi-Smart [36] 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) [37]. Shai

Halevi and Victor Shoup developed this [38].

III. The Proposed E-voting System

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

Cloud considered as 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 informations. Our system consists of:

Authentication Server (AS): responsible for authentication, verifying the correctness of the vote, and valid encrypted with the public key.

Voting Server (VS): responsible for masking the vote and tallying.

Bulletin Board (BB): responsible for displaying the checksum of the vote for public and other public.

A. Stages of the voting system

Registration: Voter need to have Identification information to be able to access and authenticated by the system, he needs to 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.

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 a predefined key between VS and AS, then sent to VS.

User allowed to communicate with the Voting server using the random secret key generated by AS to be authentication secret of a 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 [39], 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 an attacker from 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. Fig.2. Voter Authentication with Authentication Server & Voting Server.

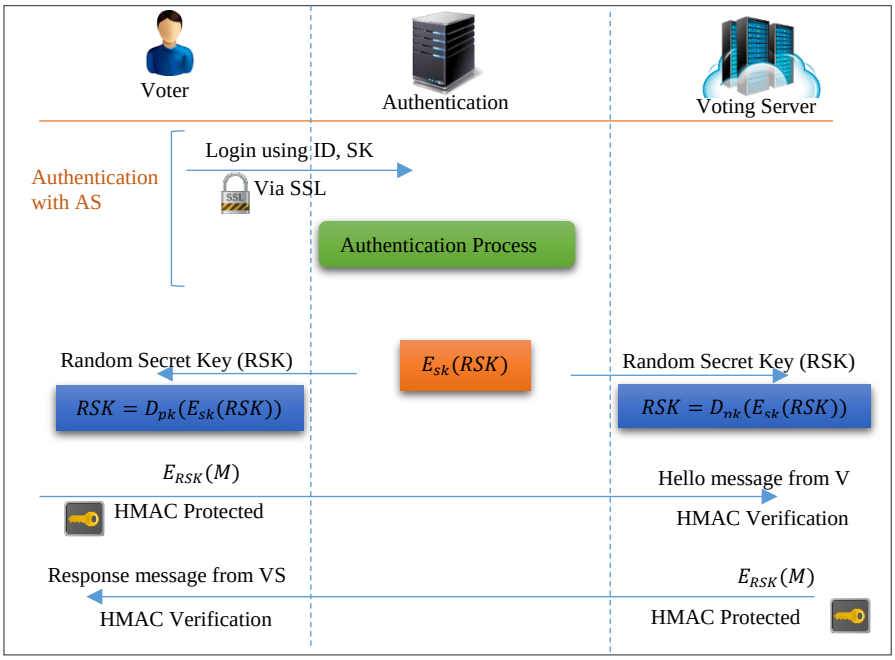

Fig.2. Voter Authentication with Authentication Server & Voting Server

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 $v = \{v_1, \dots, v_i, d\}$.

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.

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.

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 its 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,

$$\boldsymbol{M = c\ XOR\ m^0 + c\ XOR\ m^1} \quad (3)$$

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.

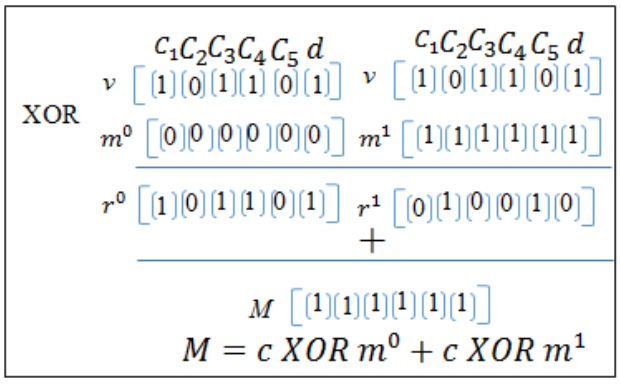

Fig.3. Masking Process Example

As shown in Fig.3. Masking Process Example, the addition process is done in a 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 an intruder, it will be calculated in the final results. This easy to cover after tallying process because the summation of the result of each result must be equal to the number of voters. No one can identify the vote that has the additional score before the tallying process. Here come the NIZKP role, to identify any invalid vote, without decrypting the vote and before the tally process. This process can be 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 this is shown in Fig.5. Vote Encryption & Validation with NIZKP

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 Decryption of results processed,

$$D_{sk}(C) = R, \text{ where } R = \{r_1, \dots, r_i, j\}.$$

R, C *and* sk put in the BB, so regulatory institutions can verify the tally process, this is shown in Fig.4. Votes Tally & Results Decryption.

B. Security analysis

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

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.

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.

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.

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. The accuracy of 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.

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 and VS. so *sk* is

stored in AS which cannot calculate any results until it receives *C* form AV.

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 of privacy. Our protocol is Receipt-free.

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, a new vote should be replaced with the old vote and new H added to BB.

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.

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.

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.

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.

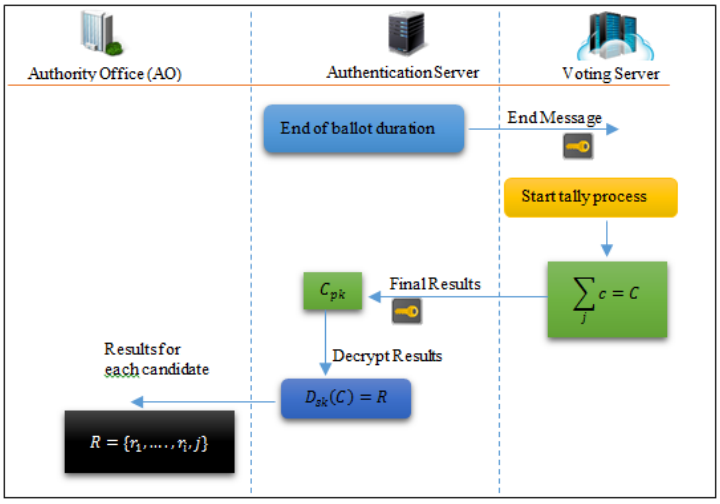

Fig.4. Votes Tally & Results Decryption

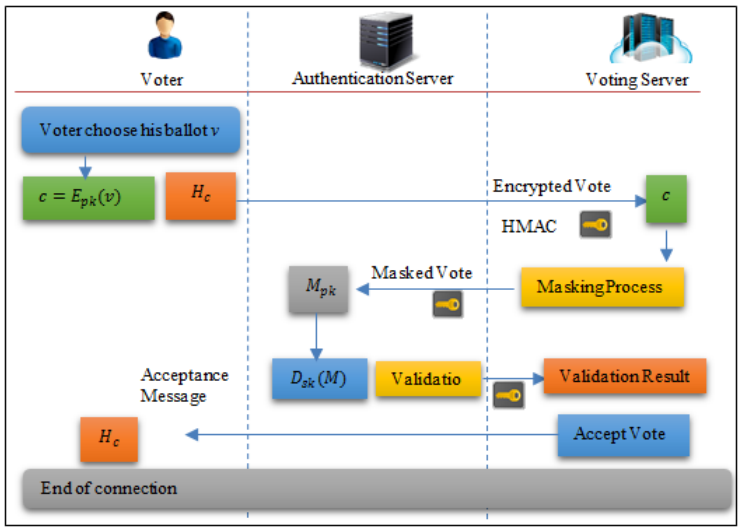

Fig.5. Vote Encryption & Validation with NIZKP

IV. Design and Implementation

The proposed E-voting system implemented using HELib library [37], the implementation done by C++ on Ubuntu 12.

A. System Structure

The proposed E-voting system software consists three main programs:

- Authentication Server program
- Voting Server program
- Voter Program

All three programs can communicate with each other; all information sent between programs encrypted in different ways, depending on the type of message. The structure is shown in Figure 6. System Structure.

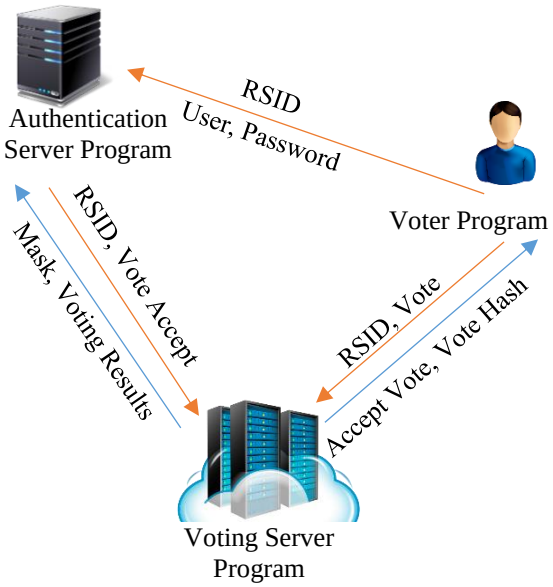

Fig.6. System Structure

1. Assumptions

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

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 be monitored and logged and controlled by the highest

1 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, wherein some cases 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 the secret key for that scheme. All data are processed in encrypted form, which prevents any untrusted party from revealing any sensitive data.
3 Voter (V) consider as untrusted party until it authenticates AS. The voter must provide secret credentials, which authenticate his identity. Then he transfers 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 to 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 a secure 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.

2. Authentication Server Program

Authentication program responsible for:

a) 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 the number of voters involved in the election process.

The generated Public key size was 20.3 MB, the same as Secret key, at the value of p=997, for experimental test.

b) 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 needs 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 the pre-defined

symmetric key between VA and VS, also it's sent to VS.

c) Vote verification:

After voter submits his vote to VS, VS calculates a vote mask described in section 0 (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 its 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 the masked vote about 204.3 kB (204,293 bytes). And the execution time of decryption is 0.019093/s. Also, decoding 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 makes the system applicable to work for many decryption processes.

d) 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 from VS is 0.011884/s and decode function is 0.0696/s, so the total decryption time is 0.081484/s.

3. Voting Server Program

The voting program responsible for

a) Voter authentication:

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

b) 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.

c) 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.

4. Voter Program

Voter program responsible for:

a) 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 decrypts it and uses it for authentication with VS. Voter sends "hello message" encrypted with RSK using a symmetric encryption algorithm.

b) Ballot preparation:

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

c) Vote Encryption:

After ballot preparation, voter program encrypts it using the public key provided on BB. The encrypted ballot size is about 136.1 kB (136,100 bytes) it's not a constant value and varies for each user, but size almost the same with same parameters. The encryption time is 0.027659/s.

B. Security Properties

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

1. Communication channels security

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

1 All messages sent between those servers are encrypted even messages like ACCEPTED 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 provides 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 messages 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.

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 an 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 an 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 from 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.

V. Results and Analysis

A. Traffic Analysis:

The proposed E-voting system 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 1. Traffic Tracing and Protection Function, all traffic between system parts encrypted, checked by integrity function. This prevents intruders from changing the content of transferred messages and even change the message itself, while all messages encrypted with the securely shared secret key.

Table 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

B. 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 a 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.

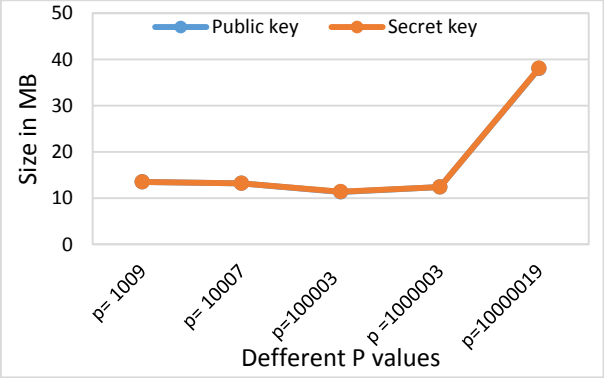

Fig.7. PublicKey size and Secretkey size for Different p Values

As shown in Fig.7. PublicKey size and Secretkey size for Different p Values, the 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. For a less, the 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 the internet. We considered our system practical for such cases, because the public key will be 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 needs 2.328 terabits for 10 million users. This small size of storage compared to a large number of users, is suitable and affordable because of this storage size available from most cloud providers and even for personal computers.

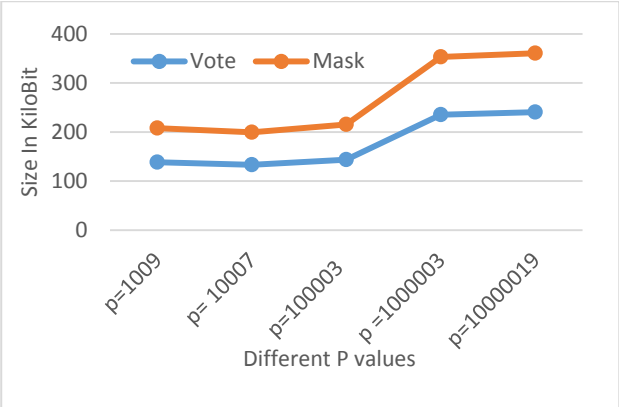

Fig.8. Vote and Mask sizes for Different p Values

Fig.8. 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 III.A which contains addition and multiplication operations which increase the size of the 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 under check, by reducing it after homomorphic operations, its bases on "modulus switching" technique.

Fig.7. PublicKey size and Secretkey size for Different p Values shows that the minimum recorded value public key and secret key on p = 100,003, and the maximum value of keys on p= 10,000,019.

Values show that at p=1,000,003 and p=10,000,019 gives the largest value of vote and mask sizes, while the other p values give 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 is done modulo p. NIZKP circuit increases the ciphertext size and noise, which gives 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 has 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.

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. the rest plaintext value is zero, the number of plaintext slots is related to CRT technique used by HELib, described in [38] 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.

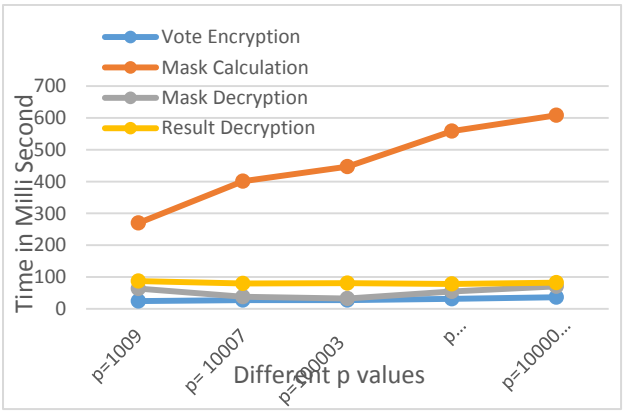

Fig.9. Vote Encryption, Mask Calculation - Decryption and Result Decryption for Different P Values

Fig.9. Vote Encryption, Mask Calculation - Decryption and Result Decryption for Different P 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 the size of the vote increase with sum calculation. The time is almost similar to vote encryption, mask decryption, and mask calculation. The major difference is in mask calculation time, which increases by increasing p value.

Another part of our experiment is examining the performance of a 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. It takes about a half

hour to generate 30,000 encrypted votes. This time seems to be linear with a 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.

The most important issue of this test is qualifying 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.

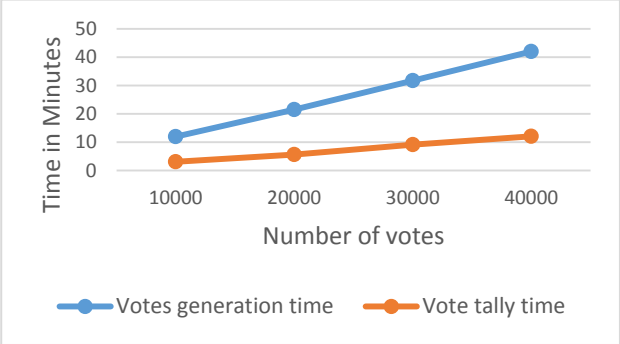

Fig.10. Final Result Cipher Decryption Time

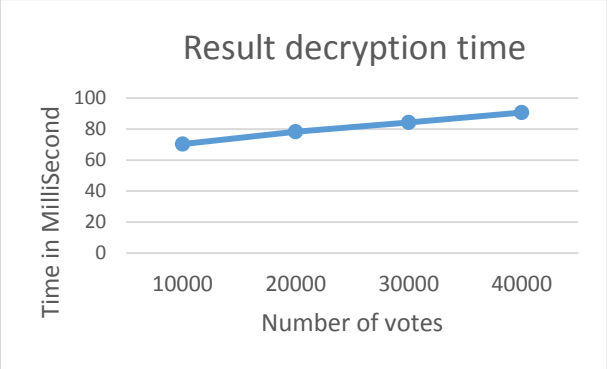

Fig.11. Votes Generation Time, And Vote Tally Time for Different Number of Votes

As shown in Fig.11. Votes Generation Time, And Vote Tally Time for Different Number of Votes, 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 be done in the 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.

The decryption of results after tallying finish shown in Fig.10. 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 the 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 to a different number of votes as shown in Fig.12. Total Size of Votes for Different Number of Votes, this due to the reduction technique used by HELib.

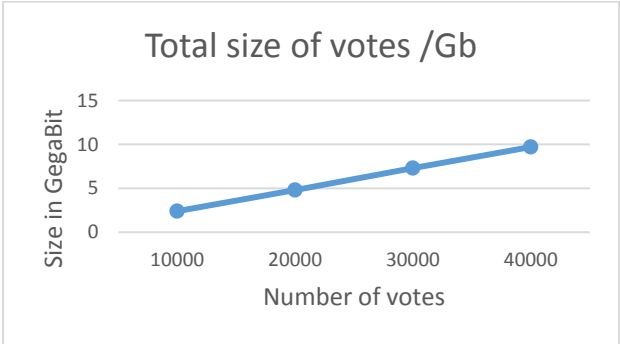

Fig.12. Total Size of Votes for Different Number of Votes

Size of all votes is a big issue, while the size of each vote is small, the total size of a large number of votes considered big. In this experiment, we examined the total size of votes at a different number of votes. As shown in Fig.12. Total Size of Votes for Different Number of Votes the largest size hit on 40,000 was 9.7/Gb. The size grows with the number of votes linearly. In an expectation for the size of 10 million votes, it will take 2.4 Terabit of size, which very affordable in cloud systems. This size available now on some personal computers. For such systems, this considered acceptable size.

C. Stored Data Analysis

At some point, each part of the system has some data, this data may be secret, public or useless data. In this section, we analyzed the data stored in each part and its security concerns.

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 results deleted after voter commit his vote for each voter.

2. Voting Server stored data

VS stores vote cipher and a 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 the 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 the voting process or clarify vote or voter personality. It also could not leak partial results while all votes encrypted.

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.

D. 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 a valid public key and formed in the correct format of voting and no additional 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 the system, reported and prevented. The user can revote when he felt coerced.

The system is scalable while it can deal with a 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.

VI. Conclusions and Future Work

This research 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.

A. Conclusions

This research 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 the 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 [37] homomorphic library based on BGV [15] 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 voters, 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 disables 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.

B. Future Work

Fully homomorphic encryption has many applications, in this research we discussed in detail 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, an also 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 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 the 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.

References

[1] R. L. Rivest, L. Adleman and M. L. Dertouzos, "On data banks and privacy homomorphisms," *Foundations of*

Secure Computation, p. 169{180, 1987.

[2] A. Rohilla, M. Khurana and M. Kumari, "Homomorphic Cryptosystem," *International Journal of Computer Network and Information Security(IJCNIS)*, pp. Vol.9, No.5, pp.44-51, 2017, 2017.

[3] A. Rohilla, M. Khurana and L. Singh, "Location Privacy using Homomorphic Encryption over Cloud," *International Journal of Computer Network and Information Security (IJCNIS).*, 2017.

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

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

[6] 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.

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

[8] 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.

[9] 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.

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

[11] 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.

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

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

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

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

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

[22] 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.

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

[24] 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.

[25] V. N. Kumar and B. Srinivasan , "A practical privacy preserving e-voting scheme with smart card using blind signature.," *International Journal of Computer Network and Information Security* , pp. 5(2), p.42., 2013.

[26] 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.

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

[28] 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.

[29] 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.

[30] R. Khatun, T. Bandopadhyay and A. Roy , "Data Modeling for E-Voting System Using Smart Card based E-Governance System," *International Journal of Information Engineering and Electronic Business*, pp. 9(2), p.45., 2017.

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

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

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

[34] 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.

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

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

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

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

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

Authors' Profiles

Ahmed Abu Aziz, is an enthusiastic information security engineer, experienced in Networks, Systems administration, Programming, Information Security training and consulting Ahmed has his MSc degree in Computer Engineering form the Islamic University of Gaza in 2015 in the field of Information Security and Cryptography. He has his BSC degree in Computer Systems Engineering from Palestine Technical Collage. He is very interested in systems and cloud security. He is working as a systems administrator and information security engineer in a local company.

Hasan Qunoo, is a young and active lecturer and researcher in computing. Hasan has a PhD and MSc degrees in Computer Security from the University of Birmingham and an extended experience and training in diverse and interactive teaching in the UK and Gaza.

He has taught a number of courses and has been an active member of the curriculum review committees at the department and faculty levels. He is a researcher and lecturer in Computing and Computer Security. He is a member of Computer Security Group at the University of Birmingham. He is now an Assistant Professor at the University of Palestine and was the head of department between 2014 - 2016. He has authored the Palestinian Information Technology Association of Companies strategy (2015-2018). In addition, he has worked in various other technical projects.

Dr. Aiman A. Abu Samra, is an Associate Professor at the Computer Engineering Department at the Islamic University of Gaza. He received his PhD degree from the National Technical University of Ukraine in 1996.

Dr. Aiman was the Assistant Vice President of IT Affairs at IUG between 2011- 2014. He was a supervisor of many Master thesis in mobile computer networks, computer security and other topics. His research interests include computer networks and mobile computing. Dr. Aiman is a member of the Technical Committee of the International Arab Journal of Information Technology (IAJIT). He was recognized as one of the most active reviewers during the year 2016.

How to cite this paper: Ahmed A. Abu Aziz, Hasan N.Qunoo, Aiman A. Abu Samra,"Using Homomorphic Cryptographic Solutions on E-voting Systems", International Journal of Computer Network and Information Security(IJCNIS), Vol.10, No.1, pp.44-59, 2018.DOI: 10.5815/ijcnis.2018.01.06

