Unanimous Voting System using Homomorphic Encryption

CS4099D Project Final Report

Submitted by

Aaron Joseph	**B200047CS**
Amalkrishna A S	**B200729CS**
G Gautham Krishna	**B200739CS**

Under the Guidance of
Dr. Vinod Pathari

Department of Computer Science and Engineering
National Institute of Technology Calicut
Calicut, Kerala, India - 673 601

May 08, 2024

NATIONAL INSTITUTE OF TECHNOLOGY CALICUT, KERALA, INDIA - 673 601

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

2024

CERTIFICATE

Certified that this is a bonafide record of the project work titled

UNANIMOUS VOTING SYSTEM USING HOMOMORPHIC ENCRYPTION

done by

Aaron Joseph
Amalkrishna A S
G Gautham Krishna

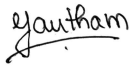
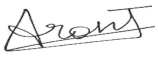
of eighth semester B. Tech in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering of the National Institute of Technology Calicut

Date
08-05-2024

Project Guide
Dr. Vinod Pathari
Associate Professor

DECLARATION

I hereby declare that the project titled, **Unanimous voting system using Homomorphic Encryption**, is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or any other institute of higher learning, except where due acknowledgement and reference has been made in the text.

	Name	Roll No.	
	G Gautham Krishna	B200739CS	
Place : NIT Calicut Date : 08-05-2024	Aaron Joseph	B200047CS	
	Amalkrishna A S	B200729CS	

Abstract

In the current era of technological advances, data has been crucial to provide the best results in Artificial Intelligence (AI), Machine Learning (ML), Internet of Things (IoT), Big Data Analytics, Cloud Computing and many more. Due to this substantial dependence on data, data security is of utmost importance where data should be protected during storage, transit and computation. Currently, methods employed ensures data is being encrypted in storage and during transit, but computation is still done on decrypted data.

The report discusses about our project on the application of Homomorphic Encryption. Homomorphic Encryption (HE) is a form of encryption that permits mathematical operations on encrypted data. In the project, we propose a design for a Voting System in which the votes are kept encrypted throughout the process with the help of Additive Homomorphic Encryption. In the later stage, we introduce a new parameter, group size, which increases the computational capacity of the system.

ACKNOWLEDGEMENT

We would like to thank our Project Guide, Dr. Vinod Pathari, for his guidance, mentorship, and patience and our Evaluator, Dr. Muralikrishnan K, for his valuable suggestions, comments and reviews during the course of our Project. Without their active guidance, help, cooperation and encouragement, we would not have made headway in the project. We would like to thank our parents and the faculty members for motivating us and being supportive throughout our work. We also take this opportunity to thank our friends who have cooperated with us throughout the course of the project.

Contents

1 Introduction **2**

2 Literature Survey **4**

3 Problem Definition **14**

4 Methodology **15**
- 4.1 Basic Scenario 15
- 4.2 Encoding 16
- 4.3 Encryption and Computation 18
- 4.4 Introducing Group Size 19
  - 4.4.1 Derivation of number of operations 22
  - 4.4.2 Minimum number of bits required to encode a vote . . 23
- 4.5 Implementation Details 23

5 Results **24**
- 5.1 Comparing different Partial Homomorphic Encryption 24
  - 5.1.1 Number of voters (n) v/s Time (t) 24
  - 5.1.2 Number of candidates (m) v/s Time (t) 25
  - 5.1.3 Group size (g) v/s Time (t) 25
- 5.2 Performance Variation based on Group Size (g) 26
  - 5.2.1 Bitmask Length 26
  - 5.2.2 Memory 27
  - 5.2.3 Number of Operations 27
  - 5.2.4 Time 28
- 5.3 Performance Variation based on Number of Voters (n) 28
  - 5.3.1 Bitmask Length 29
  - 5.3.2 Number of Operations 29
  - 5.3.3 Time 30
  - 5.3.4 Memory 31

5.4 Performance Variation based on Number of Candidates (m) . 31
5.4.1 Bitmask Length 32
5.4.2 Number of Operations 32
5.4.3 Memory 33
5.4.4 Time 34

6 Conclusion and Future work **35**

List of Figures

4.1 Encoding scheme 17
4.2 Variation of the number of votes with each round 21

5.1 bitmask length v/s g 26
5.2 memory (KB) v/s g 27
5.3 number of operations v/s g 27
5.4 time (ms) v/s g 28
5.5 bitmask length v/s n 29
5.6 number of operations v/s n 29
5.7 time (ms) v/s n 30
5.8 memory(KB) v/s n 31
5.9 bitmask length v/s m 32
5.10 number of operations v/s m 32
5.11 memory (KB) v/s m 33
5.12 time (ms) v/s m 34

List of Tables

5.1 Table comparing different PHE Algorithms with respect to Number of voters 24
5.2 Table comparing different PHE Algorithms with respect to Number of candidates 25
5.3 Table comparing different PHE Algorithms with respect to group size 25

Chapter 1

Introduction

Cryptography is generally used while data is either in transit or when the data is being stored in a location. An encrypted data would be decrypted while any computation is performed over it and this compromises data privacy. So Computation on Encrypted Data (COED) techniques [1] are practised to ensure data privacy. Homomorphic Encryption is one such type of COED, where addition or multiplication operations can be performed over encrypted data.

Homomorphic encryption (HE) can neatly be categorized under three types of schemes with respect to the number of operations allowed on the encrypted data as follows: Partial HE (PHE), Somewhat HE (SWHE) and Fully HE (FHE) [2]. Firstly, PHE allows either addition or multiplication operations for an unlimited number of times (i.e., no bound on the number of usages). Next, there is SWHE [3] that allows some type of operations with a limited number of usage before noise accumulates in the ciphertext (CT). In 2009, Dr. Craig Gentry published a thesis with the first FHE scheme that used bootstrapping [2] technique along with an existing SWHE scheme. FHE [4] can handle an unlimited number of operations on ciphertexts because bootstrapping is applied appropriately to manage the noise.

While Homomorphic Encryption has a wide range of potential applica-

tions in the commercial space, it is mainly bottled-necked by its requirement for demanding computational resources, particularly in the case of FHE. HE has the potential to revolutionize the way we process and store data, as it would make it possible to securely process and store sensitive data in a variety of settings.

In today's world, voting systems are always under threat from foreign interference, data breaches and insider attacks. Cyber-security techniques practised in this domain ensures the voting data to be secure only during transit and storage. But the result computations are performed over the decrypted form of the data, which is a vulnerability. In this scenario, Homomorphic Encryption Algorithms addresses the issue effectively by encrypting the data in a specific manner. This allows computation on the encrypted data while it remains anonymous and indistinguishable to any entity in the system.

Chapter 2

Literature Survey

- **Asymmetric and Symmetric Encryption:** [5] Symmetric encryption schemes are those which use a single key for encryption and decryption. These are generally more performance than asymmetric schemes but require the sender and receiver to agree on a key beforehand.

  The asymmetric encryption scheme uses a public key (one which can be shared with anyone) to encrypt the plain text and a private key to decrypt. The security of these schemes generally relies on the mathematical complexity of some well-identified mathematical problems.

- **Block Cipher & Stream Cipher:** [5] A stream cipher encrypts/decrypts the plain text bit by bit or byte by byte. It relies on confusion property for security and is much faster than block ciphers.

  A block cipher encrypts or decrypts in blocks of 64 bits or larger in size. It even relies on the diffusion properties for its security.

- **Confusion & Diffusion:**[5] Confusion makes the relationship between plain text (PT) and cipher text (CT) as complex as possible. Relationship between CT and PT are obscured. Given the CT, an adversary should not be able to derive any information about the PT.

In diffusion, change in each bit of a plain text bit should affect as many cipher text bits as possible.

- **IND CPA, IND CCA1, IND CCA2:**[5] The security of an encryption scheme is generally evaluated by the adversary's computational capabilities and the scope of its access to cipher texts and plain texts. Based on this, there are three levels of security for encryption schemes.
  - **Indistinguishability under Chosen-Plaintext Attack (IND-CPA):** In an IND-CPA secure encryption scheme, an adversary is unable to distinguish between two ciphertexts generated from different plaintexts, even if the adversary has access to an encryption oracle (a function that encrypts messages of the adversary's choice).
  - **Indistinguishability under Chosen-Ciphertext Attack (IND-CCA):** An IND-CCA secure scheme provides security even when the adversary has access to a decryption oracle. There are two levels of IND-CCA - IND-CCA1 and IND-CCA2.
- **IND-CCA2:**[5] In an IND-CCA2 secure scheme, the adversary is allowed to make adaptively chosen-ciphertext queries to a decryption oracle. This means that the adversary can choose ciphertexts for decryption based on previous responses from the oracle.
- **IND-CCA1:**[5] In an IND-CCA1 secure scheme, the adversary is allowed to make non-adaptive chosen-ciphertext queries to a decryption oracle. This means that the adversary must submit all their queries upfront and cannot choose subsequent queries based on the responses received from the oracle.

Functions of Homomorphic Encryption:
H = {Key Generation, Encryption, Decryption, Evaluation} [6]

1. **Key generation:** Client will generate pair of keys public key (p_k) and secret key (s_k) for encryption of plaintext.

2. **Encryption:** Using secret key (s_k) client encrypt the plain text (PT) and generate E_sk(PT) and along with public key pk this cipher text (CT) will be sent to the server.

3. **Evaluation:** Server has a function f for doing evaluation of cipher text CT and performed this as per the required function using p_k.

4. **Decryption:** Generated *Eval(f(PT))* will be decrypted by client using its s_k and it gets the original result.

Properties of Homomorphic Encryption: [7]

1. **Additive Homomorphic Encryption -**

$$E_k(PT1 \bigoplus PT2) = E_k(PT1) \bigoplus E_k(PT2)$$

Identities:

- The product of two cipher texts will decrypt to the sum of their corresponding plaintexts, $D\left(E(m_1, r_1) \cdot E(m_2, r_2) \mod n^2\right) = m_1 + m_2 \mod n$.
- The product of a cipher text with a plaintext raised to g will decrypt to the sum of the corresponding plaintexts, $D\left(E(m_1, r_1) \cdot {m_2}^g \mod n\right) = m_1 + m_2 \mod n$.

Pallier Cryptosystem [8] [9]

- Key Encryption
  (a) Choose 2 large prime numbers p and q randomnly and independent of each other.
  (b) Compute $n = pq$.
  (c) Compute the Carmichael's function λ(n). For the primes p and q, λ(n) = lcm(p-1, q-1).
  (d) Choose a random integer g such that g is in the set $\mathbf{Z}^*_{n^2}$.
  (e) Ensure n divides the order of g by checking: $\gcd(L(g^\lambda \mod n^2), n) = 1$, where $L(x) = \frac{x-1}{n}$.
  (f) Compute the modular multiplicative inverse of $(L(g^\lambda \mod n^2)$ modulo n to get μ.

  Public Key: (n,g)
  Private Key: (λ, μ)
- Encryption:
  To encrypt a message m where 0 $\leq$ m $<$ n.
  (a) Choose a random r where 1 $\leq$ r $<$ n.
  (b) Compute ciphertext c as: $c = g^m \cdot r^n \mod n^2$.
- Decryption:
  To decrypt a ciphertext c:
  (a) Compute plaintext m as: $m = L(c^\lambda \mod n^2) \cdot \mu \mod n$.
- Homomorphic Properties:
  (a) Addition: $D(E(m_1) \cdot E(m_2) \mod n^2) = m_1 + m_2 \mod n$.
  (b) Scalar Multiplication: $D(E(m_1^{m_2} \mod n^2)) = m_1 \cdot m_2 \mod n$.
  - $CT1 = g^{m1} x_1^n \mod n^2$
  - $CT2 = g^{m2} x_2^n \mod n^2$
  - $CT1 \cdot CT2 = g^{m1} x_1^n \cdot g^{m2} x_2^n \mod n^2$

So, additive property: $g^{m1+m2}(x_1x_2)^n \mod n^2$

Okamoto-Uchiyama Cryptosystem

- Key Generation
  (a) Choose two large prime numbers p and q such that $p = 2p'+1$ and $q = 2q'+1$, where p' and q' are also prime.
  (b) Compute $n = pq$ and $g = h^{(p'-1)(q'-1)} \mod n$, where h is a randomly chosen integer modulo n.
  (c) Generate a random x such that $1 < x < n$ and $\gcd(x, n) = 1$.
  (d) Compute $y = g^x \mod n$.

  Public Key: (n, g)
  Private Key: x
- Encryption
  (a) Choose a random r such that $1 < r < n$ and $\gcd(r, n) = 1$.
  (b) Compute $c_1 = g^r \mod n$ and $c_2 = (y^r \cdot m) \mod n$, where m is the plaintext message.
- Decryption
  (a) Given the ciphertext (c_1, c_2), compute $u = c_1^x \mod n$.
  (b) Find the multiplicative inverse of u modulo n, denoted as u^{-1}.
  (c) Compute $m = (c_2 \cdot u^{-1}) \mod n$, which yields the plaintext message.
- Homomorphic Properties
  (a) Addition: $E(m_1) \cdot E(m_2) \equiv E(m_1 + m_2) \mod n$, where $E(m)$ denotes the encryption of plaintext m.
  (b) Scalar Multiplication: $E(m_1)^{m_2} \equiv E(m_1 \cdot m_2) \mod n$, where $E(m)$ denotes the encryption of plaintext m.

Damgård-Jurik Cryptosystem

- Key Generation
  (a) Choose two large prime numbers p and q such that $p \equiv q \equiv 3 \pmod 4$.
  (b) Compute the modulus $N = pq$.
  (c) Select a random integer g such that g is a quadratic residue modulo N.
  (d) Generate a random integer α such that α is coprime to N.
  (e) The public key is (N, g), and the private key is α.
- Encryption
  (a) Represent the plaintext message as an integer m such that $0 \leq m < N$.
  (b) Choose a random integer r such that $0 < r < N$.
  (c) Compute the ciphertext c as: $c = g^m \cdot \alpha^r \mod N$.
- Decryption
  (a) Given the ciphertext c, compute the plaintext message m using the private key α as: $m = c \cdot (\alpha^{-r} \mod N) \mod N$.
- Homomorphic Properties
  (a) Addition: $D(E(m_1) \cdot E(m_2) \mod N) = m_1 + m_2 \mod N$, where $E(m)$ denotes the encryption of plaintext m and $D(c)$ denotes the decryption of ciphertext c.
  (b) Scalar Multiplication: $D(E(m_1^{\alpha_2} \mod N)) = m_1 \cdot \alpha_2 \mod N$, where $E(m)$ denotes the encryption of plaintext m and $D(c)$ denotes the decryption of ciphertext c.

2. **Multiplicative Homomorphic Encryption -**

$$E_k(PT1 \bigotimes PT2) = E_k(PT1) \bigotimes E_k(PT2)$$

RSA [10] [11] - Rivest, Shamir and Adleman created a cryptosystem that is a Multiplicative HE.

- Key Encryption
  (a) Each user generates a public/private key pair by selecting two large primes at random - p, q.
  (b) Compute their system modulus $N = p \times q$ and $\phi(N) = (p-1)(q-1)$ - Euler's totient function..
  (c) Selecting at random the encryption key 'e', where $1 < e < \phi(N)$, gcd(e,$\phi(N)$=1.
  (d) Publish their public encryption key {e,N} and keep the secret private decryption key {d,p,q}.
- Encryption:
  (a) Obtain public key of recipient {e,N}
  (b) Compute ciphertext c as: $c = M^e \mod N, 0 \leq M < N$.
- Decryption:
  (a) Use private key {d,p,q}.
  (b) Compute plaintext M = $C^d \mod N$.
- Homomorphic Properties:
  - $CT1 = (m_1^e) mod n$
  - $CT2 = (m_2^e) mod n$
  - $CT1 \cdot CT2 = m_1^e \cdot m_2^e \mod n$

  So, multiplicative property: $(m_1 \cdot m_2)^e \mod n$

ElGamal Encryption Algorithm [10] [12]

- Key Encryption

1. Choose a large prime p and a primitive root g modulo p
2. Choose a private key x randomly from [1,p-1].
3. Compute the public key $y = g^x \mod p$.

Public key: {p,g,y}
Private key: {x}

- Encryption:
  1. Choose a random value k from [1, p-1].
  2. Compute $c_1 = g^k \mod p$.
  3. Compute $c_2 = m \times y^k \mod p$.
- Decryption:
  To decrypt the ciphertext (c_1, c_2) with the private key x:
  1. Compute $s = c_1^x \mod p$.
  2. Compute the multiplicative inverse of s modulo p, denoted as s^{-1}.
  3. Compute the message m $= c_2 \times s^{-1} \mod p$.
- Homomorphic Properties:
  - $E(m_1) = (c_{11}, c_{21})$
  - $E(m_2) = (c_{12}, c_{22})$
  - $E(m_1) \times E(m_2) = (c_{11} \times c_{12} \mod p, c_{21} \times c_{22} \mod p)$

  So, multiplicative property: $D(c_{11} \times c_{12}, c_{21} \times c_{22}) = m_1 \times m_2 \mod p$

Pallier can be used for preserving the additive property of homomorphic encryption while ElGamal and RSA can be used for multiplicative property.

Homomorphic Encryption Schemes

1. Brakerski-Gentry-Vaikuntanathan(BGV) Encryption Scheme: [10]

   - Deals with integer vectors (whose security is dependent on the hardness of decisional LWE (Learning with Errors).
   - Deals with the integer polynomials (whose security is dependent on the hardness of the decisional R-LWE (Ring LWE).
   - BGV scheme's significance is its ability to handle both "leveled" and "somewhat" homomorphic encryption.
   - This means it can evaluate circuits of bounded depth without bootstrapping and can evaluate deeper circuits with periodic bootstrapping to reduce noise.

2. Gorti's Enhanced Homomorphic Cryptosystem (EHC)[10]:

   – Key Encryption

   (a) Choose a large prime p and another prime number q

   (b) Calculate m = p × q.

   (c) Generate a random number r

   Shared key: {p}

   Private key: {r,q,m}

   – Encryption:

   (a) Encrypt(X,m,p,q,r)

   (b) Assume X ∈ Z_p

   (c) Compute Y = $(X + r \times m \mod m)$

   (d) Output $Y \in Z_m$

   – Decryption:

   (a) Decrypt(Y,p)

   (b) Input Y ∈ Z_m

(c) Compute X = Y $\mod p$

(d) Output X $\in Z_p$

3. Non-interactive Exponential Homomorphic Encryption Scheme [NEHE][10] Encryption Scheme:

   - Alice (the originating host) has an algorithm to compute a function f.
   - Bob (the remote host) has input x and is willing to compute f(x).
   - Alice wants Bob to learn nothing substantial about f.
   - Bob should not need to interact with Alice during the computation of f(x).

4. Algebra Homomorphic Encryption Scheme Based On Updated ElGamal (AHEE) [10]:

   - It is a modified form of the digital signature standard DSS presented by the NIST in America.

     (a) Select any two prime numbers p and q.

     (b) Calculate the product of those two prime numbers. $N = p \times q$, where p and q are confidential and N is public.

     (c) Select random number x and a root g of GF(p) where g and x are smaller than p.

     (d) Calculate y = $g^x \mod p$, use this for encryption.

     (e) Encryption will be performed on following 2 steps:

        i. Select random integer number r and apply following homomorphic encryption - $E_1(M) = (M + r \times p) \mod N$

        ii. Then, select a random integer number k and the encryption algorithm will be - $E_g(M) = (a, b) = (g^k mod\, p, y^k\, E_1(M) \mod p)$

     (f) Decrypted algorithm $D_g()$ is $M = b \times (a^x)^{-1}(\mod p)$

Chapter 3

Problem Definition

Any voting system always has the risk of vote tampering from an intruder. Hence it is critical that the votes are encrypted throughout the process to ensure an unbiased result. We have to design a potential solution to secure a Voting System that can provide reliable outcomes.
The main objectives of the project are -

- The voter's votes should always be anonymous in the voting system.
- The voting results should not be distinguishable to a computing system at any stage of computation.
- The system should be able to operate on large number of voters.

Chapter 4

Methodology

In this section, we describe the methodology that was employed to tackle the aforementioned problem. Initially, we elaborate on the encoding technique employed to encode each vote in the specified format. Following that we will explore how Partial Homomorphic Encryption can be utilised to determine the winners in the voting system. Partial Homomorphic Encryption allows computation to be carried out on top of encrypted data thereby providing an additional layer of security. Subsequently, we go on to introduce a new parameter group size, and elaborate as to how it helps in optimising the proposed algorithm. We will discuss the rationale behind the new parameter and the potential impact it possesses.

4.1 Basic Scenario

In a Unanimous Voting System, the winner is determined through the unanimous support from every participating individual in the voting process towards a specific candidate. It is essential for the votes of a participant to be anonymous and protected from tampering, which helps create a secure environment for voting. Every candidate is encoded with a bit representation. A voter's vote is decided by performing a bitwise-OR operation over their

desired candidate's bit representation. The voter's votes are encrypted using an Additive Homomorphic Encryption Algorithm before sending it to a central system, which could be a remote server or a third-party cloud computing system for computation. The application of Partial Homomorphic Encryption allows us to operate on the encrypted votes and evaluate the results, without the central system ever distinguishing the votes or discovering the results at any stage of the computation.

4.2 Encoding

In this section we will describe about the encoding technique employed.

The below parameters are represented by

Number of Voters - n

Number of Candidates - m

Here we will try to represent the number of votes received by each of the m candidates by using a single bit-mask.

The maximum number of votes a candidate can receive = n

$\therefore$ The minimum number of bits required to represent the number of votes received by a candidate = $\lceil log_2\ (n+1) \rceil$

$\therefore$ Total number of bits required to represent the votes received by the m candidates separately in a single bitmask = $m\ \times\ \lceil log_2\ (n+1) \rceil$

Initially we will represent each candidate by using a bit-mask of size $m \times \lceil log_2\ (n+1) \rceil$. Later the required vote can be generated by taking the Bitwise OR of the bit-masks corresponding to all the desired candidates. Here the bit-mask corresponding to the i^{th} candidate is given by

$$bitmask_i\ =\ 1 \ll ((i-1)\ \times\ \lceil log_2\ (n+1) \rceil)$$

(where $\ll$ represents the left shift operator and $1 \leq i \leq m$)

Thus any *vote* can be represented as

$$vote_x \;=\; bitmask_{j_1} \vee\; bitmask_{j_2} \vee .... \vee\; bitmask_{j_l}$$

where $j_1, j_2, ..., j_l$ represent the candidates supported by the voter.
For example:

Consider a scenario with n=30 and m=10

$$\begin{aligned}\text{Length of bit-mask} &= m \;\times\; \lceil log_2\ (n+1) \rceil \\ &= 10 \times \lceil log_2\ 31 \rceil \\ &= 10 \times 5 \\ &= 50\end{aligned}$$

In this case, the bit-mask associated with each candidate will be as follows

$bitmask_1$ = 00000000000000000000000000000000000000000000000001

$bitmask_2$ = 00000000000000000000000000000000000000000000100000

.

.

$bitmask_{10}$ = 00001000000000000000000000000000000000000000000000

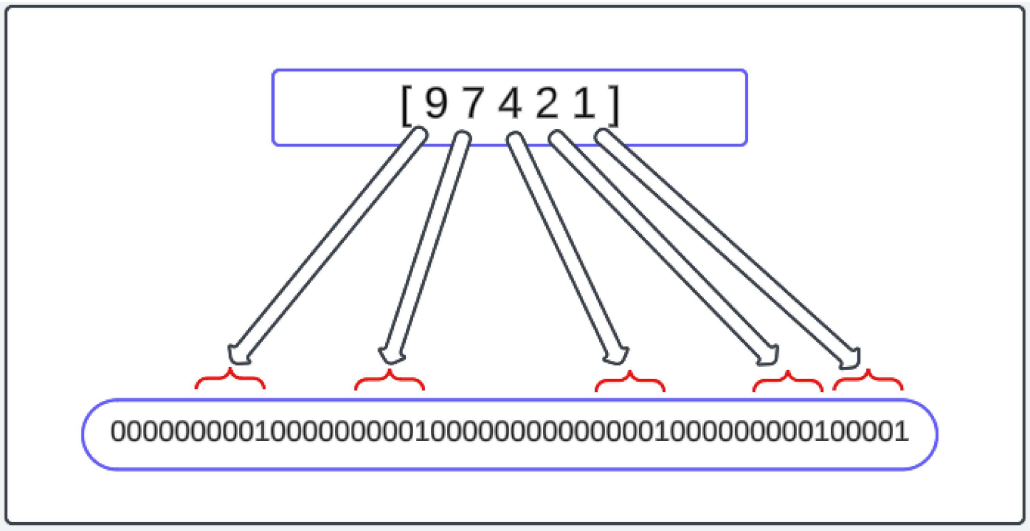


Figure 4.1: Encoding scheme

Let 1,2,4,7 and 9 be the candidates $voter_x$ approves of, the encoding of the vote is given by the bitwise OR of the bit-masks corresponding to all the specified candidates.

$$vote_x = bitmask_1 \vee bitmask_2 \vee bitmask_4 \vee bitmask_7 \vee bitmask_9$$

$$= 00000000010000000001000000000000001000000000100001$$

All the votes can be encoded similarly.

4.3 Encryption and Computation

In this section, we describe how the votes are encrypted and how the computations are carried out.

Now each of the encoded votes is encrypted by using any additive partially homomorphic encryption technique (H) [13].

$$encrypted\_vote_x = Enc_H(vote_x)$$

(where Enc_H is the encryption corresponding to H)

As the encryption technique employed is additive homomorphic we can perform addition operations on the encrypted data. The number of votes received by each candidate can be determined by adding all the $encrypted\_vote_i$ and decrypting afterwards.

$$\therefore Resultant\ Encrypted\ Vote\ = \Sigma\ encrypted\_vote_i,\ \ where\ 1 \leq i \leq n$$

Now we will decrypt $Resultant\ Encrypted\ Vote$ using Dec_H to obtain the $Resultant\ Decrypted\ Vote$.

$$Resultant\ Decrypted\ Vote = Dec_H(Resultant\ Encrypted\ Vote)$$

(where Dec_H is the decryption corresponding to H)

The length of the bitwise representation of *Resultant Decrypted Vote* is $m \times \lceil log_2\ (n+1) \rceil$. This in turn can be divided into m blocks of size $\lceil log_2\ (n+1) \rceil$. Where $block_i$ gives the number of votes received by the i^{th} candidate.

As we are able to obtain the number of votes received by each candidate, we can determine the winner by checking if the number of votes any candidate received is equal to n, then he can be declared the winner.

The algorithm that was described till now can be used for any electoral systems (Not just confined to unanimous voting system)where the winner is determined based on the number of votes.

4.4 Introducing Group Size

In the algorithm proposed above, the number of bits required in the bit-mask depends on both m and n. But generally, n can have really huge values and thereby confine the usability of the system.

So in this section, a new parameter, group size, is introduced to remove the dependency of the number of bits on the value of n.

Till now the votes were being decrypted only at the end of the complete computation cycle. But to remove that constraint, a trusted system is introduced that can decrypt and re-encrypt the results in the intermediary stages.

Now, instead of computing the result considering all the n votes, we go on to compute intermediate results. This only depends on a smaller subset of votes, which is used for further computations.

The new parameter group size (g) denotes the number of votes that are considered to compute the intermediate results.

So in each operation, a set of g votes are taken, and their sum is computed just as mentioned in the above section. This intermediate result is decrypted to figure out the number of votes each candidate received from the selected

set of g votes. Based on this a new vote named *group_vote*, which represents the collective opinion of all the g votes, is generated.

During computation, a candidate gets maximum g votes, so only $\lceil log_2(g+1) \rceil$ bits are required to represent the number of votes received by each candidate in the encoding scheme.

$\therefore$ The number of bits required to represent each vote $= m \times \lceil log_2(g+1) \rceil$

Hence the length of the encoded vote only depends on m and g. Therefore the system should be able to handle computations for larger values of n.

The updated bit-masks *u_bitmask* is given by

$$u\_bitmask_i = \ 1 \ll ((i-1) \times \lceil log_2 \ (g+1) \rceil) \qquad where 1 \leq i \leq m$$

Similarly, $vote_x$ can be computed by using

$$vote_x \ = \ u\_bitmask_{j_1} \vee \ u\_bitmask_{j_2} \vee .... \vee \ u\_bitmask_{j_l}$$

where $j_1, j_2, ..., j_l$ represent the candidates supported by the voter.

These votes are encrypted using the additive homomorphic encryption algorithms as mentioned in the previous subsection.

$$encrypted\_vote_x = Enc_H(vote_x)$$

Now in each operation we take the sum of g votes to obtain intermediate_encrypted_vote

$$intermediate\_encrypted\_vote = \Sigma \ encrypted\_vote_i, \quad where \ 1 \leq i \leq g$$

This *intermediate_encrypted_vote* is decrypted to get *intermediate_result*.

$$intermediate\_result = Dec_H(intermediate\_encrypted\_vote)$$

The length of the bitwise representation of *intermediate_result* is given

by $m \times \lceil log_2\ (g+1) \rceil$. This can be divided into m blocks of size $\lceil log_2\ (g+1) \rceil$.

Here $block_i$ provides the number of votes received by the i^{th} candidate from the pool of g voters.

Now we create a new vote say *group_vote* which includes the collective opinion of the g voters. A group is said to vote for a candidate only if all the g voters vote for the candidate.

$$\therefore group\_vote\ =\ u\_bitmask_{j_1} \vee\ u\_bitmask_{j_2} \vee .... \vee\ u\_bitmask_{j_x}$$

where $j_1, j_2, ..., j_x$ represent the candidates having $block_i = g$.

Similarly in each round, we divide all the remaining votes into blocks of size g, and their corresponding *group_votes* can be computed. Therefore at each round, the number of votes reduces by a factor of g. We go on performing the operation until only a single vote remains. If at any point the number of votes remaining is greater than 1 but is less than g, then we can append *unity element* until the number of votes becomes g.

$unity\ element\ =\ u\_bitmask_1 \vee\ u\_bitmask_2 \vee .... \vee\ u\_bitmask_m$

Let $final\_vote$ represent the only vote remaining at the end of all the rounds. This can be decrypted to obtain *decrypted final vote*. This also can be broken down into m blocks which provide the number of votes received by each candidate.

All the candidates having $block_i = g$ will be declared winners.

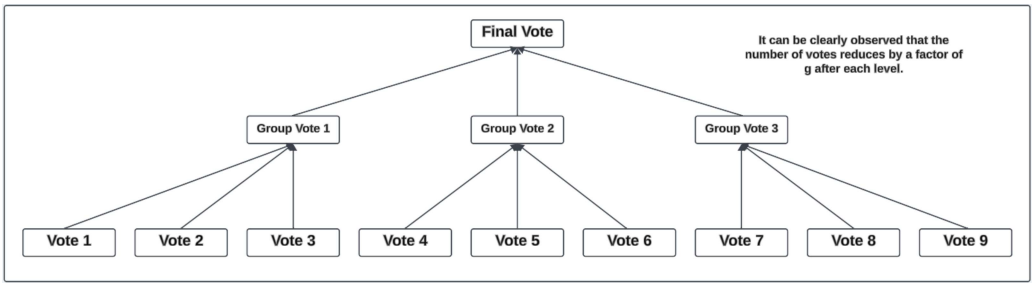


Figure 4.2: Variation of the number of votes with each round

4.4.1 Derivation of number of operations

In this subsection we will derive the equation for the number of operations performed.

Let l represent the number of rounds after which only 1 vote remains and n_k denote the number of votes after the k^{th} round

$$Total\ no\ of\ operations = n_1 + n_2 + n_3 ... + n_l$$

The value of g is greater than 1, so $n_j = 0$ for all $j > l$

$$Total\ no\ of\ operations\ = \sum n_i, where\ i > 0$$

In each operation, we replace g votes with a single vote. Therefore after each round the number of votes reduces by a factor of g.

$$n_{k+1} = (n_k)/g$$

$$\implies n_k = n/g^k$$

$$Total\ no\ of\ operations\ = \sum n/g^i, where\ i > 0$$

This forms an infinite geometric progression with $a = n/g$ and $r = 1/g$.

Also, the sum of all the terms of an infinite geometric progression is given by $a/(1-r)$

$$Total\ no\ of\ operations\ = (n/g)/(1-(1/g))$$

$$\boxed{\text{Total no of operations} = \text{n/(g-1)}}$$

4.4.2 Minimum number of bits required to encode a vote

Case 1: Without group size

Minimum number of bits $= m \times \lceil log_2(n+1) \rceil$

Case 2: With group size

Minimum number of bits $= m \times \lceil log_2(g+1) \rceil$

Here we can select any value of g, greater than 1. As we want to minimise the number of bits we can use $g = 2$.

Minimum number of bits $= m \times \lceil log_2(3) \rceil$

$= 2 \times m$

Let B, denote the bit length of the maximum number that can be encrypted using the encryption methodology utilised.

The Algorithm proposed can be used without the concept of *group size* if $1 \leq m \leq B/(log_2\ n+1)$. But with the introduction of *group size* we can extend the algorithm to $1 \leq m \leq B/2$.

4.5 Implementation Details

A proof of concept implementation of the algorithm is made available on Git Hub. The web app allows you to tweak the different parameters such as the number of voters, number of candidates and group size to analyse the performance of the algorithm. It displays the time and memory consumption of the algorithm and the number of operations for the supplied set of parameters. Additionally, it visualizes the relationship between the parameters by plotting corresponding graphs. The web app is entirely written in Python programming language and primarily utilises the open-source library LightPHE for implementing homomorphic operations. The frontend interface is built using another python framework - Streamlit.

Chapter 5

Results

5.1 Comparing different Partial Homomorphic Encryption

5.1.1 Number of voters (n) v/s Time (t)

Constant parameters followed are:
Group size = 15
Number of Candidates = 8

	Paillier	Damgard-Jurik	Okamoto-Uchiyama
n = 10	1234.1ms	1410.1ms	1292.7ms
n = 100	2417.6ms	3942.1ms	2240.2ms
n = 1000	14151.7ms	29410.5ms	12392.9ms
n = 10000	134256.8ms	281852.5ms	116580.3ms

Table 5.1: Table comparing different PHE Algorithms with respect to Number of voters

5.1.2 Number of candidates (m) v/s Time (t)

Constant parameters followed are:

Group size = 15

Number of Voters = 1000

	Paillier	Damgard-Jurik	Okamoto-Uchiyama
m = 10	13824.6ms	28902.5ms	12596.0ms
m = 20	14225.6ms	29920.7ms	12975.0ms
m = 30	14254.4ms	30527.8ms	13146.2ms
m = 40	14359.6ms	31026.4ms	13514.6ms
m = 50	14631.9ms	31360.1ms	13858.5ms

Table 5.2: Table comparing different PHE Algorithms with respect to Number of candidates

5.1.3 Group size (g) v/s Time (t)

Constant parameters followed are:

Number of Candidates = 10

Number of Voters = 1000

	Paillier	Damgard-Jurik	Okamoto-Uchiyama
g = 3	28527.4ms	59347.9ms	21169.9ms
g = 7	17113.0ms	35828.1ms	14037.7ms
g = 15	14096.8ms	29293.8ms	12197.2ms
g = 31	12619.1ms	26776.2ms	11573.6ms
g = 63	11993.3ms	25433.3ms	10834.6ms

Table 5.3: Table comparing different PHE Algorithms with respect to group size

5.2 Performance Variation based on Group Size (g)

Constant parameters followed in the graphs below
Number of Voters = 1000
Number of Candidates = 10

5.2.1 Bitmask Length

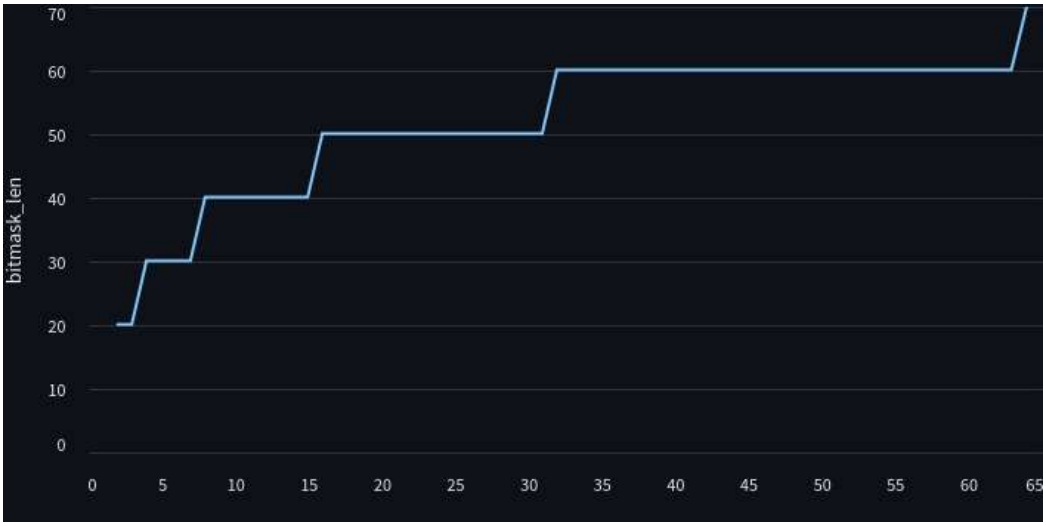


Figure 5.1: bitmask length v/s g

Justification: The length of the bitmask is given by $m \times \lceil log_2\ (g+1) \rceil$. Therefore as the value of g increases, the length of bitmask also increases.

5.2.2 Memory

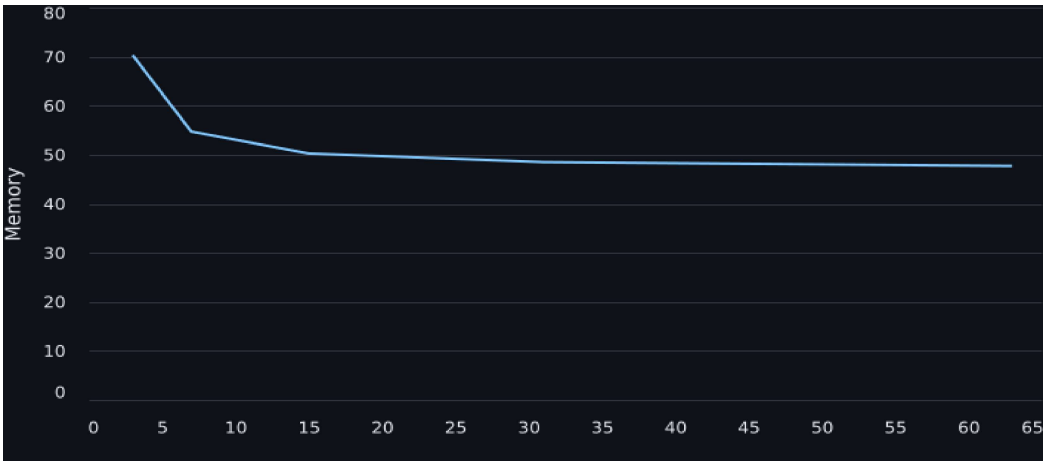


Figure 5.2: memory (KB) v/s g

Justification: As the group size increases the number of intermediate group votes, that are to be encrypted decreases, decreasing the total memory usage.

5.2.3 Number of Operations

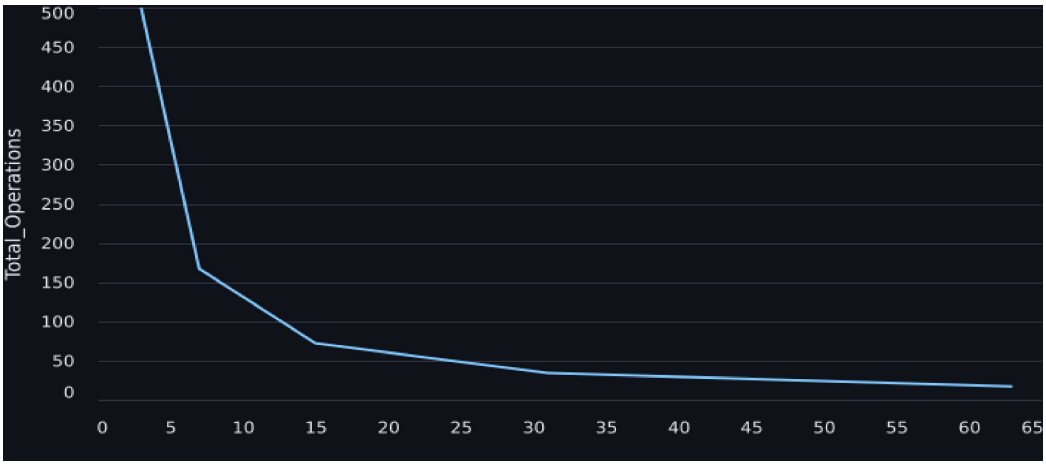


Figure 5.3: number of operations v/s g

Justification: The total number of operations is given by $n/(g-1)$.It is evident in the graph that the number of operations is inversely proportional to g.

5.2.4 Time

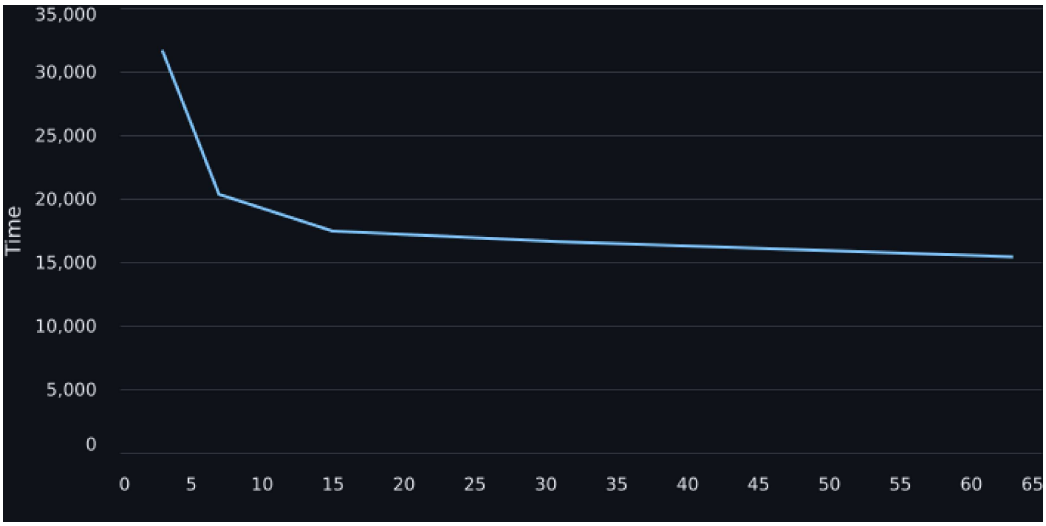


Figure 5.4: time (ms) v/s g

Justification: As the group size increases the number of operations decreases. This brings down the number of encryptions and decryptions to be performed thereby reducing the total time required.

5.3 Performance Variation based on Number of Voters (n)

Constant parameters followed in the graphs below
Group Size = 15
Number of Candidates = 8

5.3.1 Bitmask Length

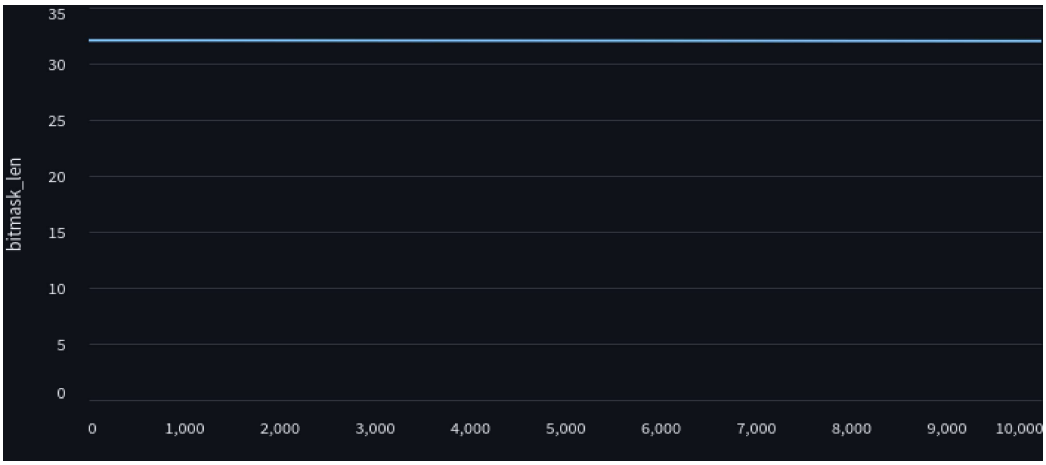


Figure 5.5: bitmask length v/s n

Justification: The length of bitmask is given by $m \times \lceil log_2 g \rceil$. Therefore the length of bitmask is independent of n.

5.3.2 Number of Operations

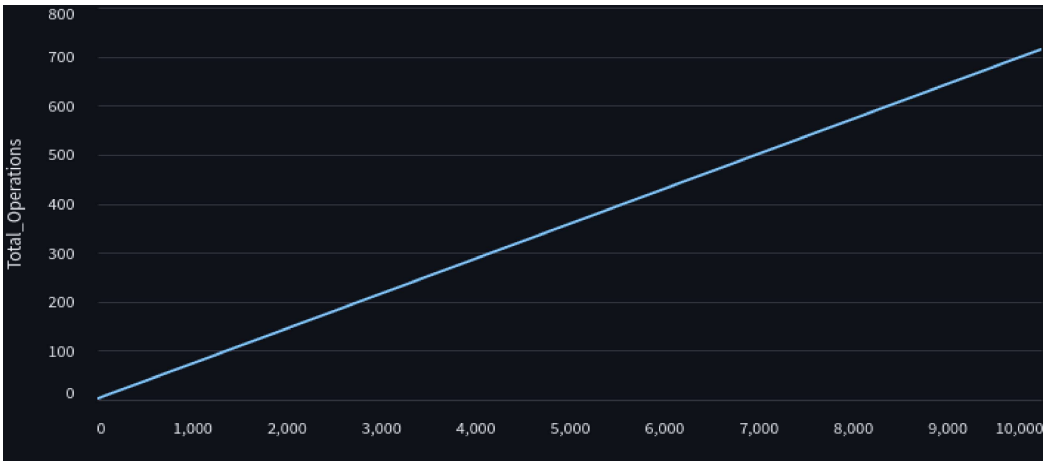


Figure 5.6: number of operations v/s n

Justification: The number of operations is given by $n/(g-1)$, Here from the graph also it is clearly evident that the number of operations is directly proportional to n.

5.3.3 Time

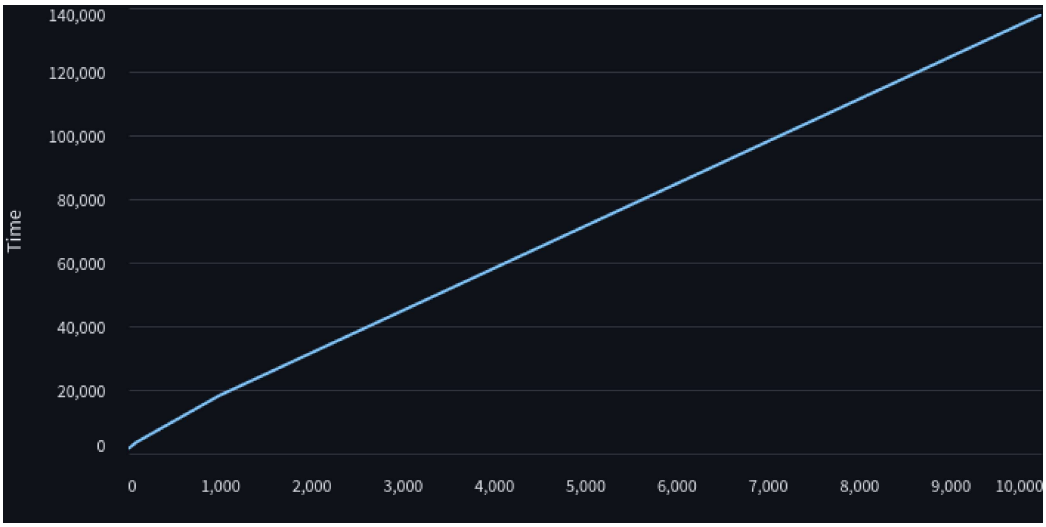


Figure 5.7: time (ms) v/s n

Justification: As the number of voters increases, the number of votes to be encrypted and the number of operations required increases, thereby increasing the amount of time required.

5.3.4 Memory

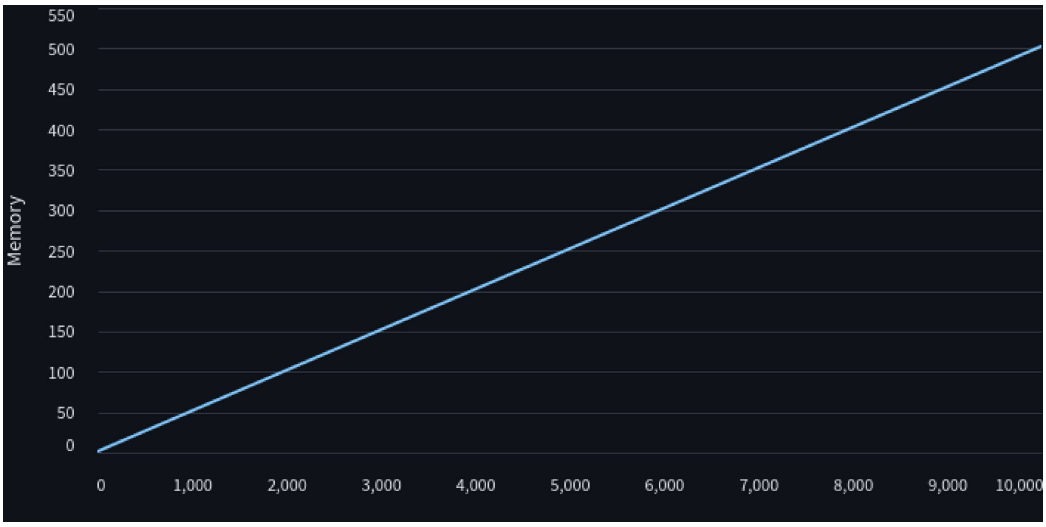


Figure 5.8: memory(KB) v/s n

Justification: As the number of voters increases, the number of votes to be encrypted, and the number of group votes generated also increases, thereby increasing the total amount of memory required.

5.4 Performance Variation based on Number of Candidates (m)

Constant parameters followed in the graphs below
Number of Voters = 1000
Group Size = 15

5.4.1 Bitmask Length

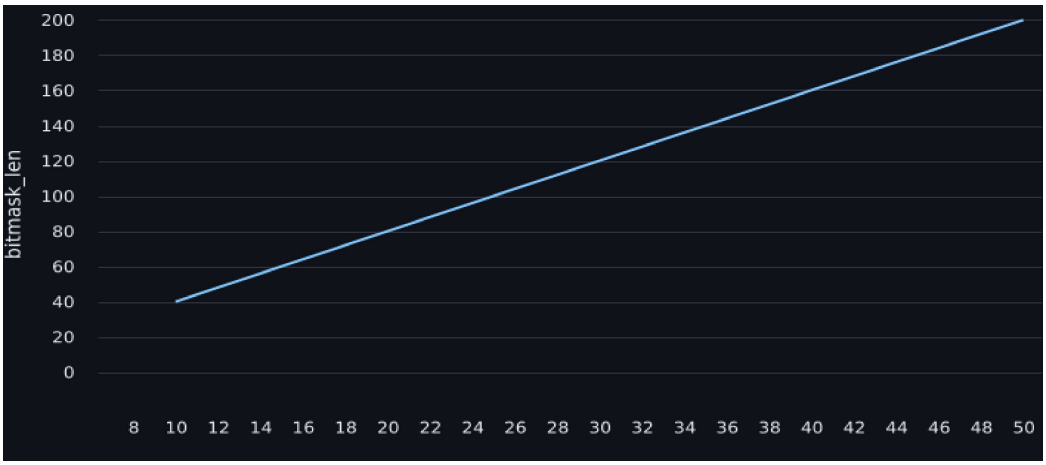


Figure 5.9: bitmask length v/s m

Justification: The length of bitmask is given by $m \times \lceil log_2(g+1) \rceil$. Here the length of the bitmask is directly proportional to the value of m.

5.4.2 Number of Operations

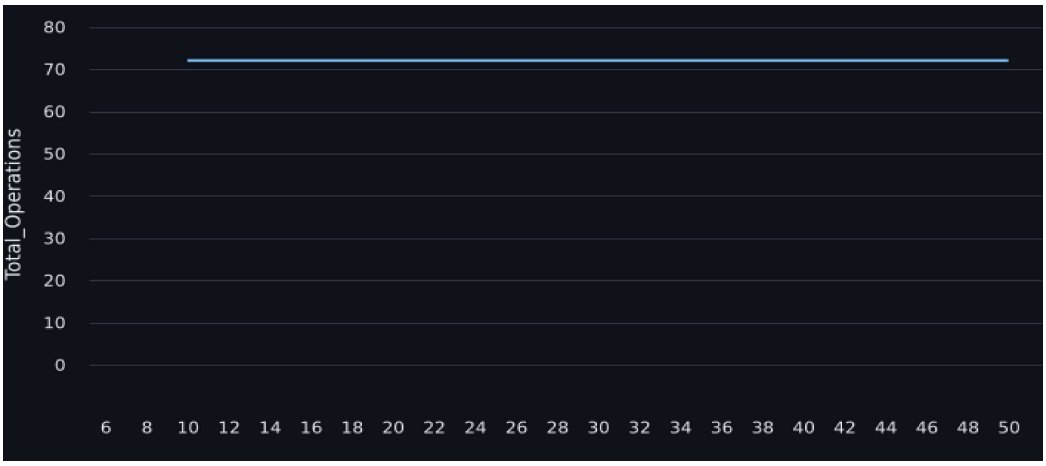


Figure 5.10: number of operations v/s m

Justification: The number of operations is given by $n/(g-1)$. Therefore it is independent of the number of candidates.

5.4.3 Memory

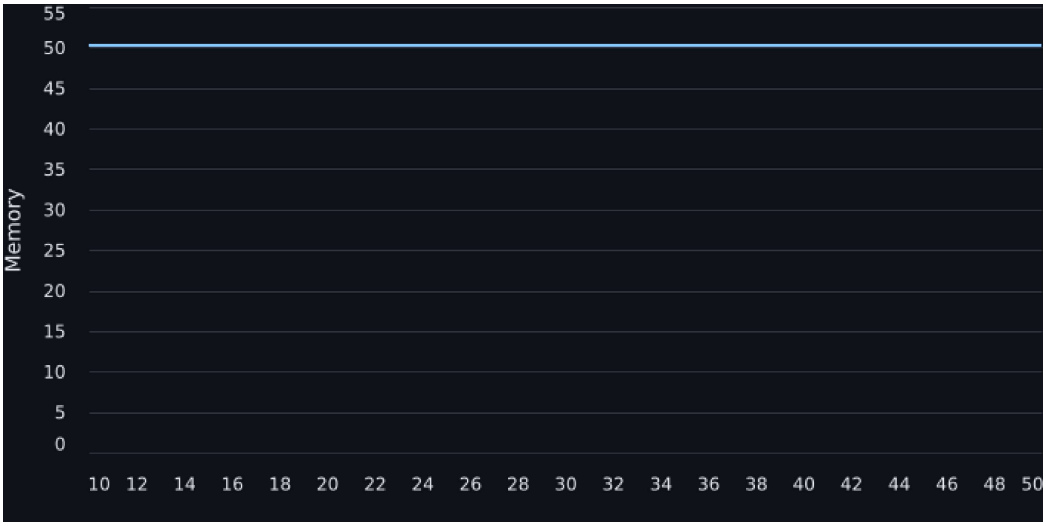


Figure 5.11: memory (KB) v/s m

Justification: As the number of voters and group size remains constant, the number of votes, and group votes to be encrypted remain the same. Therefore the total memory utilised remains constant.

5.4.4 Time

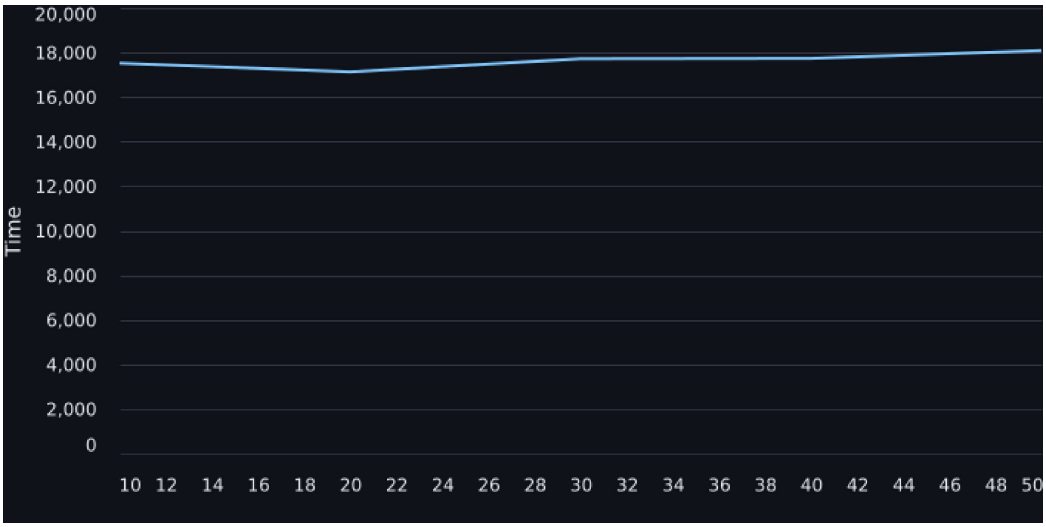


Figure 5.12: time (ms) v/s m

Justification: As the number of voters and group size remains constant, the number of votes, and group votes to be encrypted remain the same. The only difference is in the number of bitwise operations being performed. As the value of m increases, the number of bitwise operations to be performed increases thereby slightly increasing the time required.

Chapter 6

Conclusion and Future work

The project proposes an algorithm for a Unanimous Voting System that ensures anonymity of the voter's votes and conceals all computational results during the process from any third party. The application of Additive Homomorphic Encryption and bitwise encoding facilitated in the construction of the algorithm. The introduction of the *group-size* parameter further improves the algorithm to handle a larger number of voters in the voting system.

The proposed algorithm has the potential application for leader election systems in multiple domains such as distributed systems, computer networks, cluster computing and blockchain technology. Leader election is a fundamental problem in distributed systems where multiple nodes need to coordinate and collaborate to perform tasks. The algorithm can be extended to low-power devices such as in IoT where the devices in the network participate in multiple rounds of message exchanges and voting to decide upon a leader.

We would like to put forth this novel algorithm to the computer science community in the form of a research publication in the near future.

The proof of concept implementation of the algorithm can be found at https://github.com/AmalkrishnaAS/voting-algo-analysis

References

[1] N. Smart, "Computing on encrypted data," *IEEE Security Privacy*, vol. 21, 2023.

[2] C. Gentry, "A fully homomorphic encryption scheme," 2009.

[3] B. Hayes, "Alice and bob in cipherspace," *Americal Scientist*, vol. 100, 2012.

[4] S. H. . V. V. Marten van Dijk, Craig Gentry, "Fully homomorphic encryption over the integers," *Advances in Cryptology – EUROCRYPT 2010*, vol. 2010, 2010.

[5] F. G. Caroline Fontaine, "A survey of homomorphic encryption for nonspecialists," *EURASIP Journal on Information Security*, vol. 2007, 2007.

[6] A. S. U. M. C. Abbas Acar, Hidayet Aksu, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys (CSUR)*, vol. 51, 2018.

[7] P. D. Monique Ogburn, Claude Turner, "Homomorphic encryption," *Procedia Computer Science*, vol. 20, 2013.

[8] D. B. T. Saja J. Mohammed, "Performance evaluation of rsa, elgamal, and paillier partial homomorphic encryption algorithms," *2022 International Conference on Computer Science and Software Engineering (CSASE)*, vol. 2022, 2022.

[9] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," *Advances in Cryptology — EUROCRYPT '99*, vol. 1999, 1999.

[10] S. N. P. N. I. B. R. H. J. Payal Parmar, Shraddha B. Padhar, "Survey of various homomorphic encryption algorithms and schemes," *International Journal of Computer Applications*, vol. 91, 2014.

[11] A. S. R. Rivest and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, 1978.

[12] T. ElGamal, "A public-key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, 1985.

[13] R. R. R. R. S. B. Sai Sri Sathya, Praneeth Vepakomma, "A review of homomorphic encryption libraries for secure computation," *arXiv.org*, vol. 91, 2018.