THE REALIZATION OF A BLOCKCHAIN-BASED E-VOTING SOLUTION WITH A NEW CONSENSUS ALGORITHM

A Thesis Submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Computer Engineering

by
Mustafa KARAÇAY

December 2022
İZMİR

ACKNOWLEDGMENTS

I would like to express my deepest and warmest thanks to my supervisor Asst. Prof. Serap Şahin for her guidance, patience, and invaluable effort she put into making this thesis possible.

I would also like to thank my family for the strength and great support they have given me throughout my life.

ABSTRACT

THE REALIZATION OF A BLOCKCHAIN-BASED E-VOTING SOLUTION WITH A NEW CONSENSUS ALGORITHM

Security and transparency issues in the paper-based voting system and technological advances popularized e-voting systems. Many academic research and industrial solutions have recently been proposed, designed, and implemented with a Homomorphic Cryptography Scheme or HTTPS. However, there is a new popular player in the game which is called blockchain technology. This study analyzes the requirements of a well-designed e-voting system and the technology behind the blockchain, and proposes an e-voting system with a novel consensus algorithm.

Different strategies are designed and implemented to satisfy all requirements.

First, RSA and Paillier Homomorphic Cryptosystem are applied to meet requirements such as individual verifiability, secrecy, etc. So that no one can modify the vote; however, any voter can verify his/her vote during the whole vote period.

Second, different blockchains are used to meet requirements such as eligibility, privacy, authentication, etc. So that the system detects whether the data is coming from an eligible or a non-eligible voter. The system ensures that votes and voters can not be correlated if it is an eligible voter. So, the privacy of eligible voters is always protected.

Third, our blockchains ensure Consensus throughout the voting process. Fully replicated, distributed, transparent, and secure blockchains ensure that everything is under control.

Fourth, internal control mechanisms are applied to meet requirements such as non-reusability, coercion-resistance, etc. So that eligible voters can cast just one vote within the specified period. The system keeps every sensitive data encrypted so that no one manipulates the results before the vote ends.

ÖZET

BLOKZİNCİR-TABANLI ELEKTRONİK SEÇİM ÇÖZÜMÜNÜN YENİ BİR UZLAŞMA ALGORİTMASI İLE GERÇEKLENMESİ

Kağıt bazlı oylama sistemindeki güvenlik ve şeffaflık sorunları, teknolojik gelişmelerle beraber elektronik oylama sistemlerini popüler hale getirdi. Son zamanlarda, Homomorfik Kriptografi Şeması veya HTTPS ile pek çok akademik araştırmalar ve endüstriyel çözümler önerilmiş, tasarlanmış, ve uygulanmıştır. Ancak, artık oyunda blokzincir teknolojisi olarak adlandırılan yeni bir oyuncu daha bulunuyor. Bu çalışmada, iyi tasarlanmış bir elektronik oylama sisteminin gereksinimleri ve blokzincirin arkasındaki teknoloji detaylı bir şekilde analiz edilmiştir. Çalışma içerisinde, yeni bir Uzlaşma Algoritması kullanılarak elektronik oylama sistemi önerilmiş ve gerçeklenmiştir.

Elektronik oylama sisteminin tüm gereksinimlerini karşılamak için çeşitli stratejiler tasarlanıp, uygulanmıştır.

Bireysel doğrulanabilirlik ve gizlilik gibi gereksinimleri karşılamak için RSA ve Paillier Homomorfik Kriptosistem uygulanmıştır. Bu sayede, hiç kimse oy verisini değiştiremez, aynı zamanda, herhangi bir seçmen tüm oylama süreci boyunca kendi oyunun blokzincirdeki varlığını doğrulayabilir.

Uygunluk, gizlilik, ve kimlik doğrulama gibi gereksinimleri karşılamak için farklı blokzincirler kullanılmıştır. Bu sayede verilerin uygun veya uygun olmayan bir seçmenden gelip gelmediğini tespit edilebilir. Oy kullanan kişi uygun bir seçmen ise, oy ve seçmen arasında herhangi bir ilişki kurulmasına izin vermez. Böylelikle uygun seçmenlerin mahremiyeti her zaman korunur.

Uzlaşma Algoritmasını, önerilen sistem için tasarlanan ve gerçeklenen blokzincirler sağlar. Tamamen kopyalanmış, dağıtılmış, transparan, ve güvenli blokzincirler her şeyin kontrol altında olmasını sağlar.

Yeniden kullanılamazlık ve baskıya karşı direnç gibi gereksinimleri karşılamak için iç kontrol mekanizmaları uygulanmıştır. Böylece, uygun seçmenler belirtilen süre içinde yalnızca bir oy kullanabilir. Sistem, tüm hassas verileri şifrelenmiş halde tutar, böylece oylama bitmeden sonuçları kimse manipüle edemez ve değiştiremez.

TABLE OF CONTENTS

LIST OF FIGURES vii

LIST OF TABLES x

CHAPTER 1. INTRODUCTION 1

1.1. Thesis Goal and Motivation 3

1.2. Contribution 3

1.3. Outline 4

CHAPTER 2. BACKGROUND 5

2.1. Blockchain: Definition 5

2.2. Blockchain: Under the Hood 7

2.3. Types of Blockchains 9

2.4. Consensus Algorithms 10

2.5. Types of Consensus Algorithms 10

2.6. Comparison of Blockchains Together with Consensus Algorithms. 11

2.7. Additional Applied Technologies 12

CHAPTER 3. DESIGN OF THE SYSTEM 14

3.1. System Assumptions 14

3.2. System Requirements 15

3.3. Proposed Solution and Design 15

3.3.1. Setup Phase 18

3.3.2. Registration Phase 21

3.3.3. Voting Phase 25

3.3.4. Blockchain Phase 26

3.3.5. Individual Verifiability During Voting Phase 33

3.3.6. Counting Phase 34

3.3.7. Announcement Phase 36

3.3.8. Individual and Universal Verifiability After Results Announced 37

CHAPTER 4. IMPLEMENTATION OF THE SYSTEM 39
4.1. Installation 39
4.2. Architecture 40
4.2.1. main.py 40
4.2.2. setup_phase.py 63
4.2.3. register_phase.py 65
4.2.4. voting_phase.py 67
4.2.5. counting_phase.py 68
4.2.6. announcement_phase.py 69
4.2.7. dashboard.py 70
4.3. Simulation of a Vote Event 71

CHAPTER 5. RELATED WORKS 73
5.1. Academic Researches 73
5.2. Current Blockchain-Based E-Voting Systems 75

CHAPTER 6. DISCUSSION 77

CHAPTER 7. FUTURE WORK 81

CHAPTER 8. CONCLUSION 84

LIST OF FIGURES

Figure **Page**

2.1. The Total Cryptocurrency Market Cap. 7
2.2. The Structure of Blockchain. 8
2.3. The State Transition Model for Raft Leader Election. 12
3.1. The System Overview. 16
3.2. The EPC Diagram of the Review Process. 17
3.3. The Usage of the System's RSA Key Pair, and the Voter's Paillier Key Pair. . 20
3.4. The Consensus with Four Blockchain. 32
3.5. The Comparison of Four Blockchain. 34
4.1. The Interaction Between the Voters and Our Blockchains. 43

LIST OF CODE SNIPPETS

3.1. A Sample Block of EligibleVoterBlockchain 18
3.2. A Sample Block of RegisteredVoterBlockchain 24
3.3. A Sample Block of PrivateKeyBlockchain 25
3.4. A Sample Block of VoteBlockchain 30
4.1. Instances of Blockchains and Phases 42
4.2. Calculate Hash of Given Values 42
4.3. Calculate Merkle Root Hash of Given Block 43
4.4. EligibleVoterBlockchain 44
4.5. RegisteredVoterBlockchain 46
4.6. PrivateKeyBlockchain 48
4.7. VoteBlockchain 50
4.8. Setting the Start/End Times of Each Phase 52
4.9. Initialize Candidates 53
4.10. Generate RSA Key Pair Endpoint 54
4.11. Generate RSA Key Pair Function 55
4.12. Paillier and RSA Key Generation Class 55
4.13. Initialize Eligible Voter's SSN 56
4.14. Voter Registration 57
4.15. Vote Attempt 58
4.16. Individual Verify During Voting Phase 59
4.17. Count Votes 60
4.18. Announce the Results 61
4.19. Individual Verify After Voting Completed 62
4.20. Setup Phase - Initialize Candidates 64
4.21. Setup Phase - Initialize Eligible Voter's SSN 64
4.22. Register Phase - Voter Registration 65
4.23. Voting Phase - Vote Attempt 67
4.24. Counting Phase - Count Votes 68
4.25. Announcement Phase - Announce the Results 69

4.26. Dashboard Data 70
4.27. Simulate of a Vote Event 71

LIST OF TABLES

Table **Page**

8.1. An Overview of the System's Phases in terms of Requirements. 85

CHAPTER 1

INTRODUCTION

Voting is the most critical part of democracy, and the current use of paper-based voting systems has serious transparency, trust, and security issues. This problem is sometimes referred as voter fraud or vote rigging. In some cases, there may be more votes used than voters in other words duplicate voting, votes may be stolen, voters may be forced to vote for the other candidate, or identity thieves may vote rather than the eligible voter. The examples can be extended, but they will not change the result. Classical paper-based voting systems also cause a huge waste of time, money, and sources.

There is a need, a demand, and a necessity for a secure, reliable, and transparent voting system. At this point, blockchain technology could be the answer to this problem domain.

Blockchain technology is changing how organizations do business across all functions and industries. Finance, commerce, and judiciary are some of the areas where blockchain technology is applied successfully.

In this study, a well-designed blockchain-based e-voting system is examined. Therefore, when making a comparison with the literature, the key required elements can be expanded to twelve elements by the study of [28] and they are listed as follows:

i. Inalterability: Once a vote is casted, it can not be modified.

ii. Non-reusability: A voter must have only one valid vote.

iii. Eligibility: Only eligible voters should cast a vote.

iv. Fairness: Unless the voting process ends, the counting process can not be started.

v. Individual verifiability: Every voter should keep track of whether her own casted vote is still in the system or not.

vi. Universal verifiability: Everyone can verify the correctness of the whole voting process and results according to announced system keys and data.

vii. Privacy: Votes can not be correlated with voters with the help of vote anonymization.

viii. Authentication: Checking the credentials of anyone to see whether the proffered identity is consistent or not.

ix. Integrity: While applying some operations on data, protecting the accuracy and consistency of it.

x. Coercion-resistance: Providing a fake credential for every voter to use in any possible coercion case.

xi. Receipt-freeness: Attackers can not find any receipt of a voter's casted vote.

xii. Secrecy: Ensuring that no one can read the message except the intended receiver.

All these requirements clearly cover the critical points of a well-designed e-voting system. In addition to these requirements, there may be an uncovered important requirement.

To identify this requirement, we need to look at it from the voter's point of view. Then, we need to ask following questions.

- How can we trust the entire election process?
- Is the truth of the result guaranteed?
- How secure is this electoral system?

The answer to the first and second questions is to re-process. Not only voters but also any user should be able to re-process the same phases of the election and check differences between each other, which are expected to be identical.

The answer to the last question is to perform cyber attacks. Any voting system's security level can be determined by performing security tests on it. Making cyber-attacks on the system by developers/authors or any user would work. Furthermore, the number of completed tests/cyber-attacks increases the probability of identifying the possible loopholes and weaknesses of the system. Many cyber-attacks exist, like Man in the middle, Brute Force, or DDoS (Distributed Denial of Service); these are the most familiar ones. These cyber-attacks force the system to make mistakes and naturally expose the problems and required solution domain [8].

The common point between these answers is that replaying of the election can be accessible to any user so that each user can observe the election process by themselves and try to find out insecure parts of the proposed system.

In conclusion, the following thirteenth requirement should be covered by the system to make the election process more secure, reliable, and transparent.

xiii. Election replay: Any user can re-process the entire election phase and perform tests/attacks against the system after the election is fully completed with given election data.

With the help of all requirements, including the last explained election replay capability, the system will be able to get the necessary robustness. In later chapters, the technical details will be provided.

1.1. Thesis Goal and Motivation

The proposed thesis briefly analyzes the requirements of a voting system and aims to satisfy each requirement efficiently. The thesis aims to design and implement an e-voting system that can be considered robust, trusted, secure, and easy to use in real life so that elections will be more transparent, equal, and fair. Even if the proposed solution is not used actively in real life, the thesis aims to contribute to Computer Science with the ideas.

1.2. Contribution

The requirements of a secure e-voting system are briefly analyzed, and even a new requirement: *xiii. Election replay* is discovered. The system uses different approaches to satisfy election replay and accomplish voter privacy.

A unique Authentication-Based Consensus Algorithm is designed and implemented instead of popular ones using our four blockchains, called *EligibleVoterBlockchain*, *RegisteredVoterBlockchain*, *PrivateKeyBlockchain*, and *VoteBlockchain*. These four blockchain will be used to make an agreement between nodes in different phases. So that, the system makes sure that non-eligible voters can not register to the system, non-registered voters can not vote, only valid votes will be counted, etc. The details of the consensus algorithm will be introduced in section *3.3.4. Blockchain Phase*.

Each of these blockchains can be replicated and distributed under different virtual machines - VMs. Thanks to our blockchains, the system does not require dis-

tributed/shared databases, relational/non-relational databases, browser databases, indexed databases, etc. The traces of data can lead to unacceptable consequences, and the details of this problem will be discussed in further chapters.

The agility of the system architecture is always considered during the design and implementation. Paillier encryption is used in the thesis; however, any asymmetric encryption mechanism can be chosen and implemented without breaking anything. The encryption algorithms provide an extra security level, and the system can be re-implemented according to desired encryption mechanism quickly thanks to the system's maintainability.

1.3. Outline

In Chapter 2, the required background information is introduced. *In Chapter 3*, the proposed solution and design of the system are briefly explained. System assumptions and requirements are listed, and then a detailed life-cycle of the system has introduced. *In Chapter 4*, implementation of the system is given with explanations and source code. The simulation tool is introduced at the end of the chapter. *In Chapter 5*, related works are examined and compared with the proposed solution in terms of advantages and disadvantages. Both academic research and industrial solutions are explained. *In Chapter 6*, we discussed the proposed system according to different parameters. *In Chapter 7*, the possible improvements are introduced to make the system more robust, secure, and flexible. *In Chapter 8*, the conclusion is presented.

CHAPTER 2

BACKGROUND

2.1. Blockchain: Definition

What is Blockchain? Blockchain is a technology that keeps any data, and the data is distributed across the network. Blockchains are shared and immutable so that any participant can trace the data thanks to the design of blockchains. Blockchains consist of blocks, and each block points to the next block. Furthermore, each block indirectly contains all the previous blocks' data. The sensitive data are stored in encrypted form using cryptography. The last block in the blockchain is called the last block, and the first block is called the genesis block [9].

History of Blockchain: David Chaum proposed a protocol in his thesis "Computer Systems Established, Maintained, and Trusted by Mutually Suspicious Groups" in 1982 [13]. David Chaum explains how a distributed computer system can be set up, maintained, and trusted by mutually suspicious groups.

Leslie Lamport developed the Paxos protocol in 1989 and submitted the paper "The Part-Time Parliament" in 1990. However, the paper was published in 1998 [27].

Stuart Haber and Scott Stornetta introduced a cryptographically secured chain of blocks in the study "How to time-stamp a digital document" in 1991 [22]. The study describes the hash, digital signature, linking, and distributed trust.

Stuart Haber, Scott Stornetta, and Dave Bayer introduced the hash functions, collisions, and Merkle trees in the study "Improving the Efficiency and Reliability of Digital Time-Stamping" in 1992 [7]. The purpose is to digitalize the document using a cryptographic hash function, and the hash function ensures that two different documents' hash values can not be the same. And then, the study explains the Merkle trees and the use cases of Merkle trees.

In 2008, an anonymous person or a group of people known as Satoshi Nakamoto published a white paper [46]. The paper conceptualized blockchain technology and described a peer-to-peer digital payment system. The architecture of the system uses a

combination of data structures, cryptography, and decentralization. The revolutionary part can be considered as this combination and consensus algorithms. Since the paper describes a payment system, trust between participants is crucial. Satoshi Nakamoto tries to solve this problem using the Proof of Work Consensus Algorithm. In 2009, Nakamoto implemented the first blockchain as the public ledger for transactions made using bitcoin. The idea and system can be considered a game-changer since the system claims there is no need for a central bank to authorize transactions between participants. Or, there is no need to pay a fee for sending money from one account to another. Or, there is no need to expose identity to third-party organizations.

And nowadays, the research related to blockchain technology can be separated into three different classes [30]. The first class focuses on the fundamentals of blockchains and aims to increase the security level of blockchain using new techniques, and ideas [4] [19]. The second class examines the specifically consensus algorithms, and aims to improve existing consensus algorithms with modifications, or propose a unique consensus algorithm. The application areas of blockchain technology are vast, and these different areas try to solve different problems. Moreover, these problems create their consensus solutions. That is why there are too many consensus algorithms [56]. The third class examines the application areas of blockchain technology. These types of research contribute to the adaptation and usage of blockchain technology in the real world [41] [31] [33] [24]. The proposed thesis is the combination of these three classes.

The core architecture of blockchain covers the security and transparency of the proposed system in Nakamoto's paper. These features, verifiability of the system by the users, and resistance against cyber attacks gain the investors' trust. So, investors around the world paid attention to the cryptocurrency market.

Usage of Blockchain Technology: Blockchains are suitable for logistic services, supply chain tracking, the digitalization of Art, etc. However, cryptocurrencies are the most popular application of blockchains.

Nowadays, digital payment systems are the most known and one of the hot topics in the world. Bitcoin [46], and Ethereum [54] dominate the cryptocurrency market. The number of cryptocurrencies differs in many resources. However, the minimum given number is nearly ten thousand active cryptocurrencies. As explained in the history of blockchain, the first blockchain was designed and implemented for bitcoin. Cryp-

tocurrencies entered people's lives glamorously. The global crypto market cap reached $2.9T ($2,904,441,786,523) on 30 November 2021, which is 3.54 times the current volume. Even if the decrease is tremendous, the current total market cap is nearly $820B ($819,644,611,127) as of 21 November 2022 [12]. The following Figure 2.1 shows the total cryptocurrency market cap over the past years.

Figure 2.1. The Total Cryptocurrency Market Cap.

Blockchain technology aims to exclude the central authority (i.e., a bank, company, or government). And, the technology ensures that coin transactions are carried safely from one peer to another. Once a transaction is completed, the data is inserted into the blockchain. No one can change the true form of block, and immutable data is stored in distributed blockchains. Each peer has its own copy of the blockchain, and each new data consists of the previous data thanks to Merkle root hash. Even if the original aim of cryptocurrencies was to replace traditional currency, it becomes an investment option.

As we already explained in the introduction of the thesis, blockchain technology can also be adopted into traditional e-voting systems.

2.2. Blockchain: Under the Hood

The Structure of Blockchain: The blockchain consists of a set of the block, and it is a version of linked blocks, which means each block inside the blockchain points to the next block, and a block contains certain data, such as sensitive data related to the transaction, hash values, timestamp, and possibly a nonce [16].

The following Figure 2.2 shows the different parts of the block, and attributes inside the block [29].

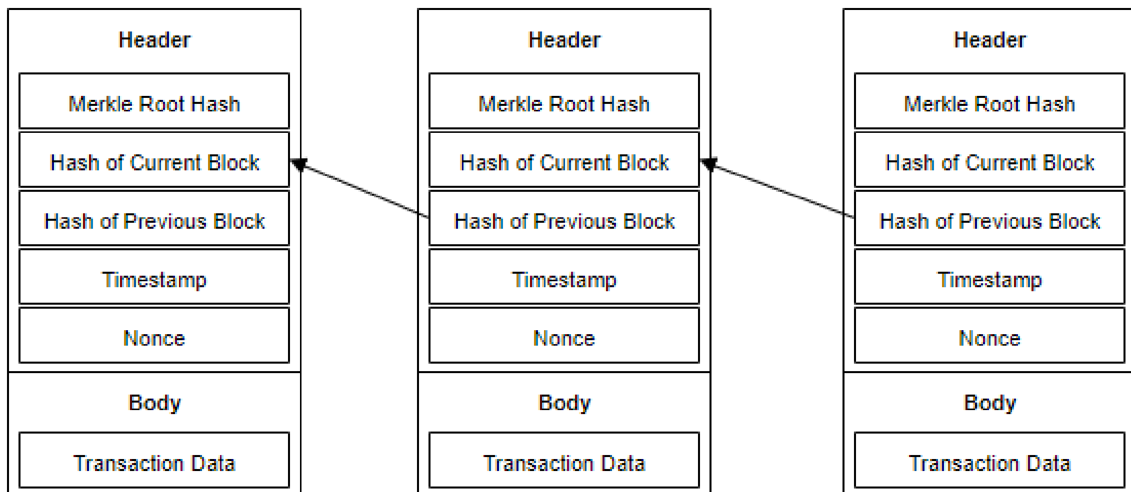

Figure 2.2. The Structure of Blockchain.

A block consists of a header and a body. The body contains sensitive transaction data, which is generally stored encrypted. The header contains the current block's hash, the previous block's hash, Merkle root hash, a timestamp, and possibly a nonce.

Encryption: Transaction data can be stored in encrypted or non-encrypted form. If the data is sensitive such as the vote choice of the voter, we should keep it encrypted. The good thing about blockchain, these technologies are independent, and the architecture can change drastically. That is why any cryptographic encryption algorithm can be applied as long as it satisfies the necessary security level.

Hash of Block and Merkle Root Hash: Hash values ensure that if we change even a single character in any block, then the hash value of that block will be entirely different from the original one.

The current block's hash value provides the integrity and immutability of that block. The previous block's hash value provides the integrity of the current and previous block pair's integrity and immutability. However, we should keep in mind that a blockchain must be immutable, and all blocks' integrity must be preserved as a whole, not just pairs. That is why we need another hash value called Merkle root hash.

Merkle root hash can be calculated using the Merkle tree. To calculate Merkle root hash, the system combines the hash of block pairs and constructs a tree with these hash values. In the end, the top of the tree represents the Merkle root hash. Moreover, the last block's Merkle root hash contains all block's hash values, and this feature ensures the whole blockchain's immutability and integrity.

In other words, if an attacker tries to change data in any block, the attacker must also change the remaining block's hash value. Otherwise, the system can easily detects fraud data and rejects it.

Like encryption, any secure cryptographic hash function can be applied to calculate the above hash values such as SHA-256. While choosing a hash function,
- We must ensure that our hash function is irreversible, which means that the original value can not be figured out using the hash value.
- We must ensure that a hashed value must be a unique value, which means that we can not calculate the same hash value using different input data.

SHA stands for Secure Hash Algorithm, and there are a couple of variations, such as SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512. These algorithms are iterative and one-way hash functions, which ensures the above constraints. As we explained, any change in the input data will directly affect the evaluated hash value, and this feature determines the integrity of the data [38].

A Timestamp: A timestamp value of the transaction time is added into the block so that the traceability of the block can be provided. The current timestamp value indirectly supports the hash functions to obtain unique block data.

A Nonce: A nonce value is generally used if the consensus algorithm is proof-based such as Proof of Work. Nonce value changes the difficulty of the mathematical problem.

Transactions: A transaction represents an interaction between nodes, which means that the data inside the transaction depend on the blockchain's design and architecture. If we consider cryptocurrencies, the transaction represents transferring money/cryptocurrency from one people/node to another. In voting systems, the transaction represents the voter's vote data and contains the voter's choice.

2.3. Types of Blockchains

Permissionless/Public Blockchains: Permissionless blockchains are also known as public blockchains because the system allows any user to participate in the blockchain network, and to append a new block into the blockchain.

Permissioned/Private Blockchains: Permissioned blockchains are also known as private blockchains because the system restricts specific users to participate in the blockchain network, and to append a new block into the blockchain. The blockchain is controlled by a group of node, people, or trusted authority.

Consortium/Hybrid Blockchains: Consortium blockchains are a combination of Permissionless and Permissioned blockchains. To append a new block to the blockchain, the user must prove that he/she is authorized to do such action. The system uses different Consensus Algorithms to verify each block's authority. The ultimate goal is to make an agreement between all participants/nodes that the block is real data/transaction.

2.4. Consensus Algorithms

What is Consensus Algorithm? Blockchain technology needs a consensus algorithm to find a common agreement between all the participants. The consensus algorithm ensures that each recently appended block to the blockchain is the one and only version of the truth, even if some peers fail. That means the system must be fault tolerant [1] [10].

2.5. Types of Consensus Algorithms

There are mainly three types of consensus algorithms, and most of the resources only mention the first and second one. However, too many consensus algorithms exist nowadays. The problem domain causes the variation of consensus algorithms. Each unique problem requires a unique solution, agreement, or in other words, consensus. The most popular and known consensus algorithms are as follows: Proof of Work (PoW), Practical Byzantine Fault Tolerance (PBFT), Proof of Stake (PoS), Delayed Proof of Work, Delegated Proof of Stake, Proof of Authority, Proof of Weight, Paxos, Raft, etc.

Proof-Based Consensus Algorithms: The first main type is Proof-Based Consensus Algorithms. In this type of consensus algorithm, the nodes in the blockchain are expected to solve a computationally hard mathematical problem to verify that they are eligible to add a new block [39]. PoW and PoS are one of the most popular types of proof-based consensus algorithms.

Voting-Based Consensus Algorithms: The second main type is Voting-Based Consensus Algorithms. In this type of consensus algorithm, it is essential that before adding a new block to the blockchain, a peer has to broadcast the outcome of mining a new block or transaction [39]. Raft or Paxos is one of the most popular types of voting-based consensus algorithms.

Authentication-Based Consensus Algorithms: The third main type is Authentication-Based Consensus Algorithms. In this type of consensus algorithm, the nodes have to pass pre-defined authentication mechanisms. Otherwise, the node can not generate a new block. Proof of Authentication (PoAh) is one of the most popular type of authentication-based consensus algorithm.

2.6. Comparison of Blockchains Together with Consensus Algorithms

Consensus Algorithms are designed for different types of Blockchains, and problem domain specifies the requirements of consensus algorithm [5].

If the system uses a permissionless blockchain, then the central idea behind the algorithm is to solve a complex (computationally hard) mathematical puzzle. Proof of Work is one of the most famous examples of a consensus algorithm, which is also used in Bitcoin. It is designed for permissionless blockchain in this manner. To solve this puzzle, computers consume a lot of power, time, and emit heat.

If the network is private, then it means permissioned blockchain is used in the system. Permissioned blockchain does not require solving a complex problem to make an agreement. If the system does not require solving any complex mathematical problem, then choosing a voting-based consensus algorithm is obviously a better idea for the design.

Raft is particularly efficient and straightforward compared to Paxos and PBFT. Moreover, it is largely adopted in distributed systems. Raft is a leader-based algorithm [23] and Figure 2.3 shows the Raft Consensus Algorithm's leader election. [1]

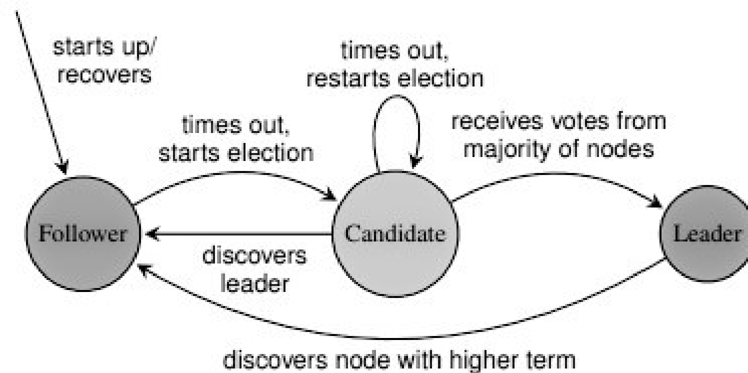

Figure 2.3. The State Transition Model for Raft Leader Election.

On the other hand, Paxos and its variations do not use leader election, which means that load balancing between nodes is well-distributed because any node can commit commands [34]. Nevertheless, supporting practical systems takes complicated changes due to the architecture of Paxos. The safety performance that is achieved with Raft is more or less equal to Paxos. Raft consensus algorithm can not tolerate malicious nodes and can tolerate up to 50% nodes of crash fault [25].

2.7. Additional Applied Technologies

Private Cloud: It is an environment that runs on the cloud, and the resources related to the hardware and software are dedicated exclusively. Only authorized users can access the private cloud. So the environment is isolated from the other private clouds [26].

Virtual Machine (VM): It is a virtual environment isolated from other resources. Users can install any operating system on VM. However, VMs does not require physical computers; instead, they need physical host machine. Therefore more than one VM can run one host machine with different operating systems [52].

[1] To see a visual simulation of Raft algorithm, check out https://raft.github.io/

Verifiable Secret Sharing (VSS): It is a cryptographic protocol; in this protocol, the dealer distributes part of the secret to n authority. Each authority receives different parts of the secret so that they can not reconstruct it with their piece. The secret can be reconstructed using many sufficient pieces [11].

SSL/TLS/HTTPS: They respectively stand for Secure Sockets Layer, Transport Layer Security, and Hyper Text Transfer Protocol Secure. SSL is the standard technology that ensures incoming and outgoing internet connections are transferred securely. The data is encrypted before sending between the server and the client, so an attacker can not read or modify the data. TLS is the updated version of SSL since TLS considered a more secure standard. HTTPS is a protocol for users to transfer data between web server and web browser securely. HTTPS runs over SSL, so an SSL certificate ensures that the data is sent after encrypted [14].

CHAPTER 3

DESIGN OF THE SYSTEM

In this study, an e-voting system is designed while RSA and Paillier Homomorphic Cryptosystem are applied with blockchain technology using a novel consensus algorithm based on the requirements in chapter *1. Introduction*.

In addition to specified requirements of a voting system, the proposed solution aims to cover the following topics also:

i. Accessibility: The system should be physically or virtually accessible during the whole voting process.

ii. Availability: The system should be available during the whole voting process.

iii. Adaptability: The system should be adaptable for different use cases.

The *accessibility* of the system is vital for people with disabilities and older people. That is why the system runs in the VMs, and any voter can vote using a mobile phone or personal computer anywhere, anytime.

The system is distributed among different VMs and takes backup of each valuable data in case of emergencies. If needed, the system can restart itself and continue to become *available*.

The system is designed to support the plural voting system, and the results will be announced according to it. In this type of voting system, the winner is the one that is mostly chosen regardless of whether it has the majority of votes or not. However, the system is easily maintainable, and it can be *adaptable* for other types of voting systems as well.

3.1. System Assumptions

(1) Admin is a trusted actor, and authorities are honest throughout the whole election process.

(2) The authority defines an eligible voter list, and the list is shared with the proposed system in the setup phase.

(3) The authority defines candidates, and the list is shared with the proposed system in the setup phase. In other words, it is defined before the voter registration phase.

(4) The application runs in a private cloud, and virtual machines (VMs) are dedicated to protecting the system.

(5) TLS/SSL technology secures the internet browser connections and transactions so any sensitive data transfers from the voter to the system or vice versa through encrypted data. HTTPS connection in the URL ensures that the system is certified and that the certificate will be automatically renewed every 60 days.

(6) The study uses the Hybrid Blockchain type to get maximum advantage from both Permissionless and Permissioned Blockchains.

(7) A unique Authentication-Based Consensus Algorithm is designed and implemented using four different blockchains.

(8) Both small-scale such as local/regional elections, and large-scale such as national elections, are supported thanks to the hybrid blockchain technology.

(9) The proposed solution has its own RSA public-private key pair.

(10) Each voter has one public-private key pair to encrypt and decrypt the vote data. To accomplish this, Paillier Homomorphic Cryptosystem is used in the proposed system. However, any asymmetric encryption system ensures the same security level because the system will perform the same.

(11) The login operation requires Two-Factor Authentication while voting. Username, password verification, and 6-digit verification code will be used together in the study.

3.2. System Requirements

As described in chapter *1. Introduction*; a well-designed, secure, reliable, and transparent blockchain-based e-voting system must satisfy the following requirements: *i. Inalterability, ii. Non-reusability, iii. Eligibility, iv. Fairness, v. Individual verifiability, vi. Universal verifiability, vii. Privacy, viii. Authentication, ix. Integrity, x. Coercion-resistance, xi. Receipt-freeness, xii. Secrecy,* and *xiii. Election replay.*

3.3. Proposed Solution and Design

The system's process is built by the following steps:

(1) Setup Phase
(2) Registration Phase
(3) Voting Phase
(4) Blockchain Phase
(5) Individual Verifiability During Voting Phase
(6) Counting Phase
(7) Announcement Phase
(8) Individual and Universal Verifiability After Results Announced

A simplified but descriptive system overview is represented in Figure 3.1, and EPC diagram of a review process in Figure 3.2.

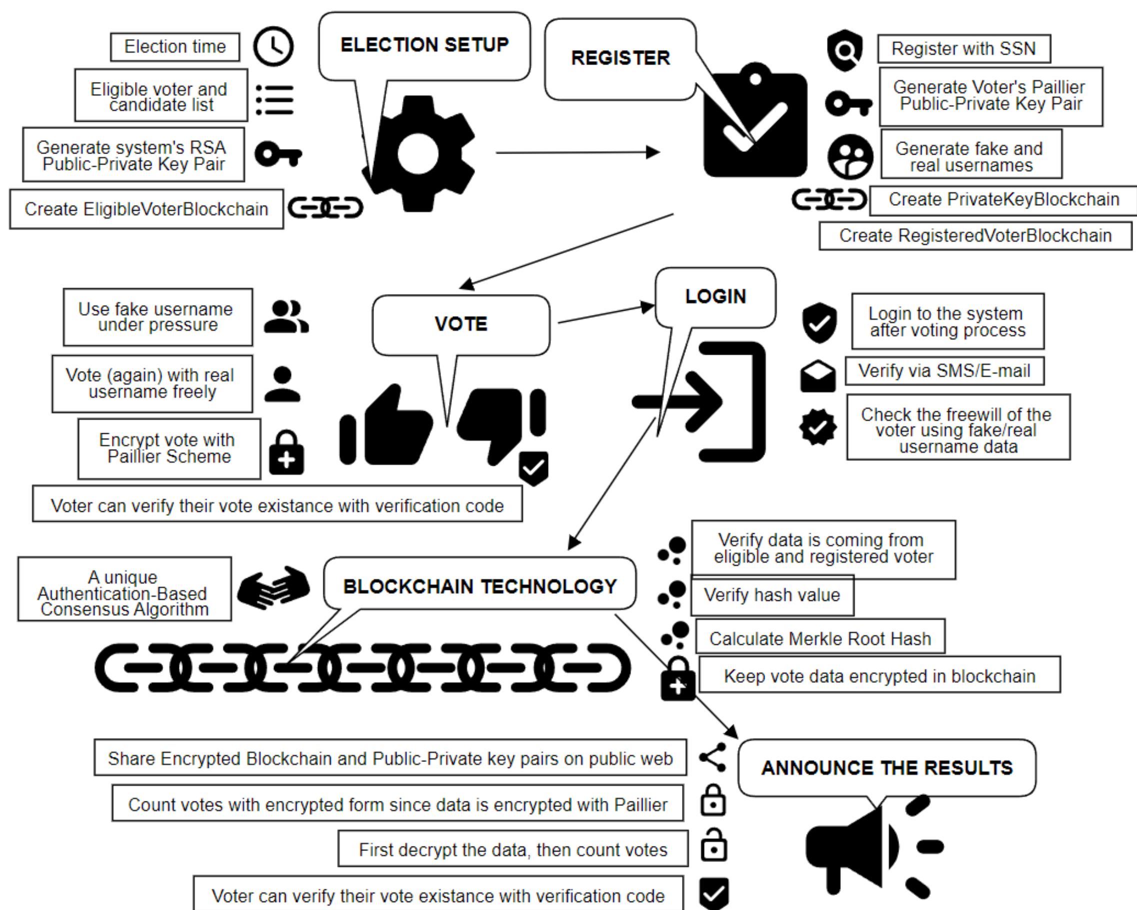

Figure 3.1. The System Overview.

Figure 3.2. The EPC Diagram of the Review Process.

3.3.1. Setup Phase

This phase can be considered as the beginning of the voting process. All requirements and preparations are handled in this phase.

The authority supplies the eligible voter list to the system. This list will be stored in a blockchain called 'EligibleVoterBlockchain', and the blockchain can be accessible via the system itself. The private and sensible data related to voter's identity will be kept private.

Code Snippet 3.1. A Sample Block of EligibleVoterBlockchain

```
class EligibleVoterBlock:
    def __init__(self, data=None, prev_block=None,
        verify_block_ssn_hash=None, current_block_hash=None,
        prev_block_hash=None, merkle_root_hash=None):
         self.enc_ssn = data['enc_ssn']
         self.verify_block_ssn_hash = verify_block_ssn_hash
         self.prev_block = prev_block
         self.current_block_hash = current_block_hash
         self.prev_block_hash = prev_block_hash
         self.merkle_root_hash = merkle_root_hash
         self.timestamp = data['timestamp']
         self.is_registered = False
```

The above code snippet 3.1 shows the attributes inside EligibleVoterBlock class, and each block in the EligibleVoterBlockchain contains the following attributes.

- a social security number (SSN), *enc_ssn*, the data is kept in encrypted form to satisfy *xii. Secrecy* requirement.

- a hashed value of SSN, *verify_block_ssn_hash*, will be used to verify whether the voter is eligible to register or not. So the requirement *iii. Eligibility* is ensured.

- a boolean variable, *is_registered*, to make sure that no one registered twice or more to the system using the same SSN. Even if the voter has the authority to cast a vote as an eligible voter, the voter must have only one valid vote in the system. Therefore, requirement *ii. Non-reusability* is satisfied.

Since the eligible voter's SSN is stored in the blockchain, we ensure that the requirement *ix. Integrity* is satisfied using the advantage of hash values.

Like the eligible voter list, candidates are also identified by the authority, and the list is shared with the proposed system. This data will be used as a ballot option.

And then, the start-end times of all phases are determined. When the registration phase closes, then the voting phase must start. After the voting phase is completed, then the counting phase must start. These restrictions increase the chance of satisfying the requirement *iv. Fairness*.

Finally, precisely one public-private key pair is generated by the system using RSA cryptosystem [44]. This key pair is generated for the proposed solution, and **the system's RSA public key** will be used to encrypt **voter's Paillier private key** during *3.3.2. Registration Phase*. So that no one can decrypt the vote data during the voting process. **The system's RSA private key** will be used to decrypt **voter's Paillier private key**. So that the votes can be decrypted during *3.3.6. Counting Phase* using the voter's Paillier private keys.

In other words; the system will be able to access voter's Paillier private key, after the RSA decryption is completed. Which means that an attacker must access both the system's RSA private key and the voter's Paillier private key to decrypt any vote data. So, if we securely store the system's RSA private key, the votes can not be counted even if the voter's Paillier private keys are leaked.

The requirement *xi. Receipt-freeness* is satisfied with the help of encryption of the voter's Paillier private key.

In addition to the above usage during *3.3.6. Registration Phase*, the system's RSA public key will be used during *3.3.1. Setup Phase* and *3.3.2. Registration Phase* again. First, to encrypt eligible voter's SSN during *3.3.1. Setup Phase*. Second, to encrypt voter's sensitive data such as username, and password during *3.3.2. Registration Phase*. The system's RSA private key can decrypt these encrypted data; however, the system does not require a such operation.

The following Figure 3.3 shows the relationship between keys and phases as time-line. As we can see from the Figure 3.3, the system's RSA private key should be kept safe from the setup phase to the counting phase. Otherwise, the eligible voter's sensitive data can expose, or the voter's Paillier key becomes vulnerable to leak.

Below, we will explain how we store the system's RSA private key. Moreover, we will discuss why we prefer to use RSA encryption specifically in chapter *6. Discussion*.

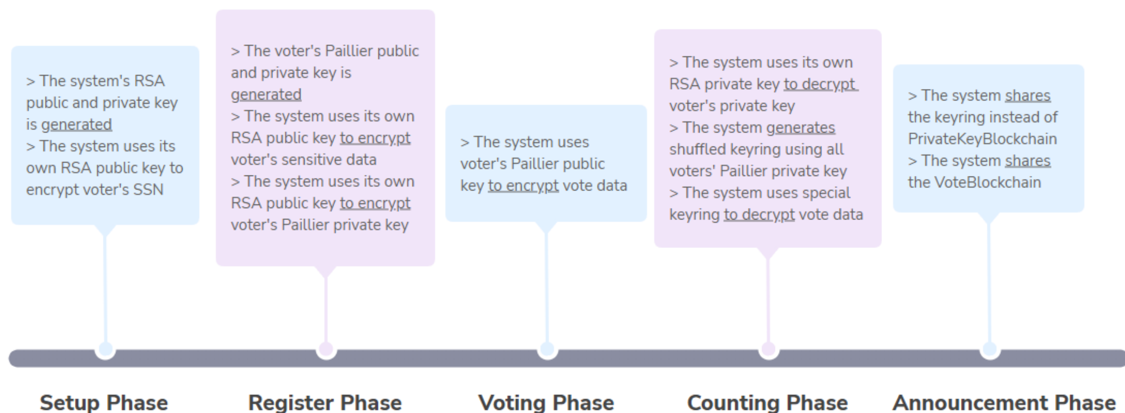

Figure 3.3. The Usage of the System's RSA Key Pair, and the Voter's Paillier Key Pair.

The critical question is how the system's RSA private key should be stored securely. As we discovered above, the system's RSA private key should not be reachable before *3.3.6. Counting Phase*. We have two options to store the system's RSA key isolated.

The first option is the time-lock puzzles. Solving the puzzle refers to having the system's RSA decryption/private key to read the vote data in our case. Time-lock puzzles are based on computational power similar to the Proof of Work-PoW Consensus Algorithm and require much time to solve the puzzle; by this way, we expect that we have required delay time to reach the *3.3.6. Counting Phase*. However, time-lock puzzles can only be solved sequentially. This puzzle tries to make "CPU time" and "real time" as close to each other as possible. However, CPU time can drastically change according to hardware quality. More computational resources can solve the puzzle more quickly, which means that vote data can be decrypted during the voting process, which can not be tolerated [45].

The second option is the trusted agents. The primitive version of this approach can be that a trusted agent stores data until the desired time t has elapsed, and then shares the data with the system. A better and improved version uses Verifiable Secret Sharing. VSS was first introduced by Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch in 1985 [11]. VSS uses more than one trusted agent to share the RSA private key parts (decryption key for votes), these authorities are responsible for keeping part of the decryption key, and when the time is up, they share their key parts with other trusted

agents. When one of the agents has the required number of key parts from other agents, the decryption key can be rebuilt.

Using VSS rather than time-locks is much more suitable for the proposed solution because the technology is developing and evolving quickly, and the accessibility to computationally powered hardware is easier than ever.

Therefore, the moment the system's RSA private key is generated, the system will use the VSS approach.

Finally, the following algorithms describe the procedures of RSA cryptosystem, such as key generation, encryption, and decryption [44] [3].

Algorithm 1 RSA - Key Generation

1: Choose two different large prime numbers p and q. Such that $p \neq q$

2: Compute the modulus n; $n = p \cdot q$

3: Calculate $\phi(n)$, where ϕ is the Euler's totient function:

$$\phi(n) = (p - 1) \cdot (q - 1)$$

4: Choose an integer e which is the public key such that e is coprime with $\phi(n)$:

$$\gcd(\phi(n), e) = 1 \; ; 1 < e < \phi(n)$$

5: Calculate d, which is the private key such that it is the modular multiplicative inverse of e:

$$d = e^{-1} mod \phi(n)$$

6: Eventually, the public key is $[e, n]$, and the private key is $[d, n]$

Algorithm 2 RSA - Encryption

1: For plaintext P, public key $[e, n]$ calculate the Chiphertext C as:

$$C = P^{e} \; modn$$

Algorithm 3 RSA - Decryption

1: Decrypt Chiphertext C, using the private key $[d, n]$:

$$P = C^{d} \; modn$$

3.3.2. Registration Phase

After the setup phase is completed and the designated start time of the registration phase is reached, voters can register to the system using any internet-connected device to vote at further stages. As explained in section *3.1. System Assumptions*, TLS/SSL certificate, and HTTPS connection provide the necessary security. The data transfers in encrypted form so that an attacker can not read or manipulate the voter's data.

Since the system already generated the EligibleVoterBlockchain in *3.3.1. Setup Phase* and each block in that blockchain contains the verification hash of the eligible voter's SSN, the system can verify whether the user is an eligible voter or not. With the help of verification hash, which is called verify_block_ssn_hash in the code snippet of EligibleVoterBlockchain, the requirement *iii. Eligibility* is satisfied.

The voter enters the SSN, and then the system receives the SSN via HTTPS. Then, the system calculates the hash value of the given SSN. After that, the system traverses the EligibleVoterBlockchain from the last node to the genesis node. The system compares the calculated hash value and verifies the verify_block_ssn_hash value inside the block. Suppose that there is n number of eligible voters. In that case, the system performs n iteration at worst case to verify whether the voter is eligible to register or not.

If the hash values are not matched, the system rejects the block; therefore, unauthorized users can not register to the system.

If hash values are matched, and SSN is not used to register before, then the system will ask the voter to fill in an email and a phone number. These data will be used to cover *viii. Authentication* requirement of the eligible voters during login. The voter must specify either email or phone number. In phase *3.3.3. Voting Phase*, the system will send a 6-digit verification code to the specified contact option.

Then, the system asks the voter to fill real username, fake username, and password. The usernames must be unique. The voter can either create these values or use the auto-generated data feature of the system. Automatically generated usernames and password ensure that ordinary computers can not identify your private data using simple brute-force attacks. That is why the system recommends that the voters should use generated data instead of manually created. Recent research indicates that users tend to choose familiar, easy-to-remember, and predictable usernames and passwords [55]. Moreover, this leads to

vulnerability to crack usernames and passwords with simple brute force attacks. Beyond the simple brute-force attacks; a simple and well-designed web scraping application can easily get your animal's name, your favorite song's name, your birthday, your family's name and surname, or even your home address within a second, of course if you share it anywhere online. That could be on any social media platform such as Facebook, Twitter, or Instagram. Even if General Data Protection Regulation (GDPR) restricts and protects the user's data, there are still no certain lines. And finally, we should remember that most users accept terms of use without reading them, and people give permission to process their data without notice.

In addition to the above explanations, using two different usernames provides simple yet excellent agility. The voter can use both real and fake usernames in the voting phase under different circumstances. Let us assume someone, an attacker, is trying to force eligible voters to vote for the opposite candidate, then the voter can use the fake username to get rid of this situation. Not only will this vote not be counted in the counting phase, but also, a voter can re-vote with the real username freely. This approach provides *x. Coercion-resistance* against an attacker.

After fake username, real username, and password are generated/created; the voter has two options to note that data. First, the voter can select to receive these data via email or SMS. Second, if the voter is more like a skeptical and does not want to keep this data online, he/she can disable the email and SMS features and note the data himself/herself.

Username-password verification and a 6-digit verification code ensure that Two-Factor Authentication is satisfied to increase security during the voting phase.

Finally, the system generates Paillier public-private key pair for each registered voter. Paillier or any asymmetrical cryptosystem can be used as a solution, for instance, ElGamal, ECC, RSA, or Lattice-based NTRU. If Lattice-based NTRU is preferred, the built solution becomes resistant to Post-Quantum cryptographic attacks [32] [15].

Voter's Paillier public key will be used to encrypt vote data in phase *3.3.3. Voting Phase* and it will be stored together with the registered voter data in a blockchain called 'RegisteredVoterBlockchain'. Voter's Paillier Private key will be used to decrypt vote data in phase *3.3.6. Counting Phase* and it will be stored in a different blockchain called 'PrivateKeyBlockchain'. Each private key will be encrypted with the system's RSA public key before adding into PrivateKeyBlockchain. One block will be created for each voter's

Paillier private key, and the system keeps secure these private keys until the voting process is ended thanks to the extra encryption level with RSA and VSS approach.

Since the eligible voter's sensitive data is stored in the blockchain, we ensure that the requirement *ix. Integrity* is satisfied using the advantage of hash values.

Code Snippet 3.2. A Sample Block of RegisteredVoterBlockchain

```
class RegisteredVoterBlock:
    def __init__(self, data=None, public_key=None, prev_block=None,
        verification_hash_fake=None, verification_hash_real=None,
        current_block_hash=None, prev_block_hash=None, merkle_root_hash
        =None):
         self.public_key = public_key
         self.fake_username = data['fake_username']
         self.real_username = data['real_username']
         self.password = data['password']
         self.verification_hash_fake = verification_hash_fake
         self.verification_hash_real = verification_hash_real
         self.email = data['email']
         self.phone = data['phone']
         self.location = data['location']
         self.prev_block = prev_block
         self.current_block_hash = current_block_hash
         self.prev_block_hash = prev_block_hash
         self.merkle_root_hash = merkle_root_hash
         self.timestamp = data['timestamp']
         self.is_voted = False
```

The above code snippet 3.2 shows the attributes inside RegisteredVoterBlock class, and each block in the RegisteredVoterBlockchain contains the following attributes:

- a public key of the eligible voter to encrypt vote data, *public_key*, the key is generated using Paillier Homomorphic Cryptosystem

- a fake username, *fake_username*, the voter can use this username under pressure while being forced to vote for another candidate

- a real username, *real_username*, the voter can use this username in normal circumstances

- a password, *password*, the password needs to match with one of the usernames

- a hashed value of combination of fake username and password, *verification_hash_fake*, the system will detect that the voter is casting his/her vote under pressure during voting phase, the system will reject this vote attempt and it is designed to let the voter use his/her

vote with the real username again freely

- a hashed value of combination of real username and password, *verification_hash_real*, the system will detect that the voter is casting his/her vote with free will, the system accepts the vote attempt and make sure that the same voter can not cast another vote

- an email, *email*, a personal email address of the voter for Two-FA

- a phone, *phone*, a personal phone number of the voter for Two-FA

- a boolean variable, *is_voted*, to represent that not only an eligible but also a registered voter casted his/her vote once. Therefore, requirement *ii. Non-reusability* is satisfied.

Code Snippet 3.3. A Sample Block of PrivateKeyBlockchain

```
class PrivateKeyBlock:
    def __init__(self, data=None, prev_block=None, current_block_hash=
        None, prev_block_hash=None, merkle_root_hash=None):
        self.encrypted_private_key = data['enc_private_key']
        self.prev_block = prev_block
        self.current_block_hash = current_block_hash
        self.prev_block_hash = prev_block_hash
        self.merkle_root_hash = merkle_root_hash
        self.timestamp = data['timestamp']
        self.is_counted = False
```

The above code snippet 3.3 shows the attributes inside PrivateKeyBlock class, and each block in the PrivateKeyBlockchain contains following attributes:

- a private key of the eligible voter to decrypt vote data, the key is generated using Paillier Homomorphic Cryptosystem and encrypted with the system's RSA public key to satisfy requirements *vii. Privacy* and *xii. Secrecy*, *enc_private_key*

- a boolean variable to make sure that no one cast more than one vote with the same key, *is_counted*

3.3.3. Voting Phase

After the setup and registration phase is completed and the scheduled start time of the voting phase is reached, voters will cast their vote via any web browser.

The voter selects one of the candidates and then enters the username and password. The system shows the eligible candidates as ballot options to the voters, and eligible

candidates are defined during the setup phase. The voter can use either real username or fake username under different circumstances so that the system will detect whether the voting attempt is made under pressure or with free will. HTTPS and SSL provide sufficient security to transfer data between client and server. After the server, in this case the proposed solution, receives the data; extra security levels will be used for sensitive data as explained in early phases.

The given username and password are used to verify that the voter is eligible and registered voter. At the same time username indicates the voter's free will. And then, Two-FA is required. The system asks the 6-digit verification code if the voter is eligible to vote. If the 6-digit verification code is matched with the system-generated one, then the vote is evaluated by the system and appended into the blockchain which is called 'VoteBlockchain'. The details of these processes will be explained in section *3.3.4. Blockchain Phase*.

Two-FA ensures that the requirement *viii. Authentication* is satisfied.

To satisfy the requirements *xi. Receipt-freeness*, and *xii. Secrecy* the vote data is kept in encrypted form.

Since the vote data is stored in the blockchain, we ensure that the requirement *ix. Integrity* is satisfied using the advantage of hash values.

And as a final step, the system generates a hash value and sends this hash value to the voter's email or phone number. The preferred authentication method is used by the system automatically. As explained in section *3.3.2. Registration Phase*, each voter must choose one of the contact options to authenticate his/her vote in the voting phase. Suppose a voter enters an email address during registration, then the system sends the vote verification hash value to that registered email address.

v. Individual verifiability requirement will be satisfied using the verification hash value.

This cycle will be evaluated for each registered voter during *3.3.3. Voting phase*. The requirement *vii. Privacy* of the voter is always protected, and votes can not be correlated with voters. The details of this process will be introduced in *3.3.4. Blockchain Phase* and *3.3.6. Counting Phase*.

3.3.4. Blockchain Phase

Voting Phase and Blockchain Phase start simultaneously since the vote data transfers from the client (a voter) to the server (the proposed system). In this phase, the same process in the voting phase will be re-evaluated; however, we will explain the architecture behind the system taking into account blockchain technology.

After the choice of candidate, username, and password are gathered from voters, the following sub-phases/steps will be executed.

Login Verification to Vote: As the first step, the system calculates the hash value of received username-password pair. And then, the system checks whether username-password pair is matched with one of the blocks in RegisteredVoterBlockchain or not. The system traverses from the last node to the genesis node and checks every block's *verification_hash_real* and *verification_hash_fake* value. In the worst case scenario, the system performs n iteration to reach the end of blockchain, suppose there exists *n* registered voter. There may be three different situations.

- Case-1, calculated hash value does not match any of the block's both *verification_hash_fake* and *verification_hash_real*.

  If the login credentials are failed, $Case-1$, then it means that voter did not register during the registration phase. Therefore, the system rejects the vote.

- Case-2, calculated hash value matches with one of the block's *verification_hash_fake*.

  If the voter logged in with fake username, $Case-2$, then it means that an attacker is possibly forcing voter to vote for opposite candidate. Therefore, the system rejects the vote.

- Case-3, calculated hash value matches with one of the block's *verification_hash_real*.

  If the voter logged in with real username, $Case-3$, then verification of the login is succeed.

This control mechanism provides an excellent feature to the voter. The voter can vote with his/her real username password pair again in case of emergencies. So the requirement *x. Coercion-resistance* is satisfied using fake username and real username approach.

Login Authentication to Vote: As the second step, according to the result of login attempt verification, the system requires extra authentication. If the case is either two or three, then the system gets the preferred contact information of that voter from the RegisteredVoterBlockchain. Then, the system randomly generates a 6-digit verification code and sends it via email or SMS. The voter receives the verification code and should enter it into the system. After that, the system checks that the received 6-digit verification code is matched with the randomly generated 6-digit code.

If the values do not match, then the voter can generate new 6-digit verification codes. If the values match, then it means that the vote data is verified, validated, and authenticated by both the voter and the system.

Vote Encryption: Homomorphic encryption is a type of encryption that allows you to perform specific types of computations on encrypted data without revealing any information about the data itself [17].

To satisfy the requirements, *xi. Receipt-freeness* and *xii. Secrecy* the vote data is kept in encrypted form until the counting phase.

There are two types of homomorphic encryption: Partially Homomorphic Encryption and Fully Homomorphic Encryption. The partially homomorphic encryption schemes support one homomorphic property, which is usually additive or multiplicative homomorphism, while the fully homomorphic encryption schemes support all the homomorphic properties [17].

The Paillier is a partially homomorphic encryption scheme that allows two types of computation: Addition of two ciphertexts and multiplication of a ciphertext by a plaintext number. This scheme was invented by Pascal Paillier in 1999 [40].

The Paillier's scheme can be considered under three main parts; key generation, encryption, and decryption algorithms. The following algorithms describe these procedures. A well-explained and numerical example of these algorithms can be found in the following paper: The Homomorphic Other Property of Paillier Cryptosystem [49].

Algorithm 4 Paillier - Key Generation

1: Choose randomly different p and q as two large prime numbers with:

$$gcd(pq, (p-1)(q-1) = 1$$

2: Compute RSA modulus $n = pq$ and calculate Carmichael's function that can be computed with:

$$\lambda = \frac{(p-1)(q-1)}{gcd(p-1.q-1)}$$

3: Select random g as generator where $g \in \mathbb{Z}^*_{n^2}$ with:

$$gcd(\frac{g^\lambda \mod n^2 - 1}{n}, n) = 1$$

4: Find modular multiplicative inverse with:

$$\mu = (L(g^\lambda \mod n^2))^{-1} \mod n$$

The function L is defined as $L(u) = \frac{u-1}{n}$

Algorithm 5 Paillier - Encryption

1: Let m be a message to be encrypted where $m \in \mathbb{Z}_n$

2: Select random r where $r \in \mathbb{Z}^*_n$

3: Compute ciphertext as: $c = g^m \cdot r^n \mod n^2$

Algorithm 6 Paillier - Decryption

1: Ciphertext $c \in \mathbb{Z}^*_{n^2}$

2: Compute message: $m = L(c^\lambda \mod n^2) \cdot \mu \mod n$

After we introduce the basics of Paillier Cryptosystem, as the third step, each vote data is encrypted using the voter's Paillier public key. Since the system already knows *verification_hash_real*, the Paillier public key of the voter can be accessible by traversing the RegisteredVoterBlockchain.

Hashing for Vote Verification: As the fourth step, the vote verification hash value are calculated by the system, and again this value is sent to the voter's preferred contact information. The hash value is generated using the encrypted version of the vote data. Since the vote data is encrypted using the voter's Paillier public key, hash values will be unique for each voter. Moreover, any voter can use this hash value to verify that his/her vote is in the system.

After the vote is encrypted with the voter's Paillier public key and *vote_verification_hash* is generated by the system, the Merkle root hash will be evaluated using SHA-256 algorithm. Each block in the blockchain includes the hash value of itself, the hash of the previous block, and Merkle root hash. With this approach, every block actually knows the data inside all the previous blocks. This makes the blockchain immutable.

Once a vote is inserted into the blockchain an attacker can not change a block because of the hash values inside that block. This feature satisfies *i. Inalterability* and *ix. Integrity* requirements.

The block is finally ready to append into VoteBlockchain.

Adding New Block: As the final step, according to the result of login attempt verification, the system either appends vote data into VoteBlockchain or does not append. Actually, the decision is made by the voter itself, and there are three possible use cases.

In first scenario, a voter may enter an incorrect username and password. In this case, the system warns the voter about it and expects to re-enter login credentials since it may be just typing issue. So, a block will not be appended to the VoteBlockchain.

In second scenario, a voter may enter fake username and correct password. In this case, the system returns a successful message to the voter on purpose. Since an attacker is still there and possibly watching the voter's device. In the background, the system rejects the vote data. So that even if an attacker forces any voter to vote for opposite candidate, the attacker will not be able to add a new block to the VoteBlockchain. The voter can vote with the real username later. And, the system appends this free will vote attempt instead the forced one.

In third scenario, a voter may enter real username and correct password. In this case, the system appends a new block into the VoteBlockchain.

Code Snippet 3.4. A Sample Block of VoteBlockchain

```
class VoteBlock:
    def __init__(self, data=None, prev_block=None, current_block_hash=
        None, prev_block_hash=None, merkle_root_hash=None):
         self.enc_choice = data['enc_choice']
         self.vote_verification_hash = data['vote_verification_hash']
         self.prev_block = prev_block
         self.current_block_hash = current_block_hash
         self.prev_block_hash = prev_block_hash
```

```
        self.merkle_root_hash = merkle_root_hash
        self.timestamp = data['timestamp']
```

The above code snippet 3.4 shows the attributes inside VoteBlock class, and each block in the VoteBlockchain contains following attributes:
- a choice data of the voter in encrypted form, the data is encrypted with voter's Paillier public key, *enc_choice*
- a hashed value for verification during voting phase, the voter can verify that his/her vote is added as expected during voting phase, *vote_verification_hash*

A Fully Replicated, Distributed, Transparent, and Secure Blockchain: So far, we have introduced how the system works generally. But more importantly, we must ensure that our VoteBlockchain is fully replicated, distributed, transparent, and secure. To do that, let us examine them one by one.

To make our blockchain *fully replicated*, we need to make multiple copies of our blockchain. So we must store the exact copy of our blocks locally. So that, we can import these exact copies in case of emergencies. To do that, the system stores the vote data as JSON in encrypted form.

To make our blockchain *distributed*, we need to copy these locally stored data into other VMs. So that, even if our main VM shuts down because of DDOS attacks, we may continue to serve with the backup-VMs. To do that, the system broadcasts the encrypted vote data to backup-VMs.

To make our blockchain *transparent*, the system shares the Merkle root hash of the EligibleVoterBlockchain, RegisteredVoterBlockchain, and PrivateKeyBlockchain to the World Wide Web. In addition to this transparency, the system broadcasts the size of the blockchain and Merkle root hash of the last block in VoteBlockchain. Everyone can access these data and make sure that everything is clear and nothing is under-covered, changed, corrupted, masked, or concealed.

We have already explained the ways and techniques of making our blockchain *secure* in many sections. And we will continue to demonstrate it. Let us re-examine these features again.

We store the block's hash value together with Merkle root hash, so that we can ensure the integrity of vote data. These hash values also prove the immutability of the

vote data. Actually, the chain between nodes and Merkle root hash ensures the security of whole VoteBlockchain, not just one specific block.

Each phase is designed according to one simple principle: always ensure voters' privacy. That is why, we encrypt any sensitive data before inserting it into our blockchains. The system generates two different key pairs. The first key pair is generated using RSA for the system itself. The second key pair is generated using Paillier for each registered voter. The usage of these key are demonstrated in Figure 3.3.

We should remember that replicated and distributed data are directly related to the security of the blockchain.

Finally, the security of blockchain leads us to talk about the *Consensus Algorithm*.

The Consensus: The consensus makes sure that *only eligible voters* can add new block into the *RegisteredVoterBlockchain* and *PrivateKeyBlockchain*. Furthermore, *only registered voters* can add a new block into the *VoteVBlockchain*. Our consensus also makes sure that the integrity of the data is kept.

As introduced in section *1.2. Contribution*, a unique Authentication-Based Consensus Algorithm is applied using our four blockchains. The Figure 3.4 shows the interactions between these four blockchains and the trusted authority. The consensus of the whole system relies on each other, and each blockchain is created for different purposes.

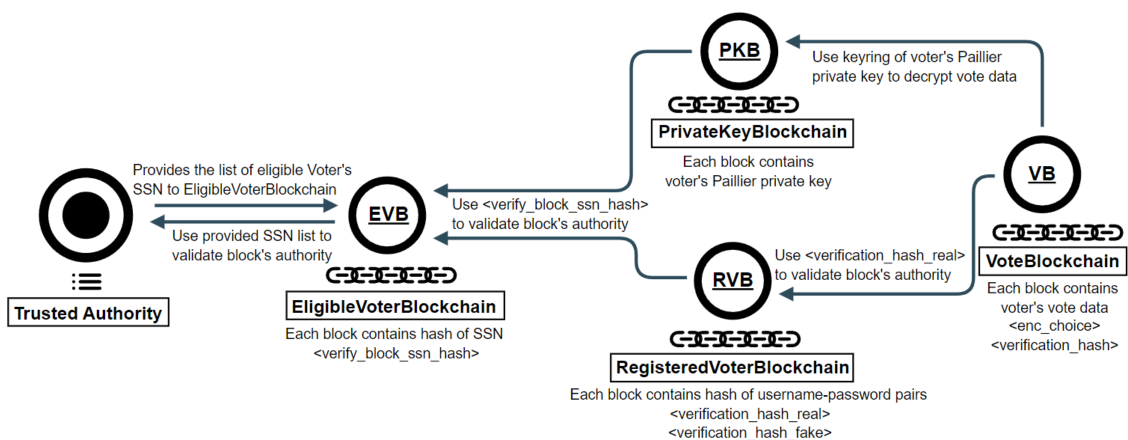

Figure 3.4. The Consensus with Four Blockchain.

At first, the trusted authority provides the eligible voter's SSN list to the system. Our first blockchain, *EligibleVoterBlockchain*, directly trusts the authority and generates

a blockchain that consists of eligible voters' data. Each block inside the blockchain represents one eligible voter. The primary attribute will be the hash of the voter's SSN.

These SSN hash values will be used to validate the block's authority by our second and third blockchains, *RegisteredVoterBlockchain* and *PrivateKeyBlockchain*. In this case, only eligible voters can access and append a new block to our Registered-VoterBlockchain and PrivateKeyBlockchain. The PrivateKeyBlockchain will be used to decrypt vote data, so there is no active part in the consensus. On the other hand, RegisteredVoterBlockchain contains the verification hash for the real username and the verification hash for the fake username.

These real and fake username hash values will be used to validate the block's authority by our fourth blockchain, *VoteVBlockchain*. VoteBlockchain will use the second and third blockchains for different purposes. Before analyzing their relationship, let us first understand one small detail.

Right now, we created a *chain* between our blockchains, and VoteBlockchain indirectly requires the voter's eligibility thanks to the other blockchains. VoteBlockchain directly interacts with the PrivateKeyBlockchain and uses our third blockchain to generate a keyring. However, this relationship does not affect the consensus part; it only affects the decryption success of votes. The consensus part continues with the relation between RegisteredVoterBlockchain and VoteBlockchain. VoteBlockchain uses the RegisteredVoterBlockchain to validate vote data's authority using the verification hash for the real username.

In conclusion, the introduced chain relation between the four blockchains ensures consensus. So, we know the relation between our blockchains which means we can compare them with following Figure 3.5. In addition to the comparison of our blockchains, Figure 4.1 shows the interaction between the voters and our blockchains.

We will discuss the details of why we prefer to design such a unique consensus algorithm in chapter *6. Discussion*.

3.3.5. Individual Verifiability During Voting Phase

As an e-voting requirement, *v. Individual verifiability* during the voting process, voters can verify that their votes exist inside the blockchain. So that the system has

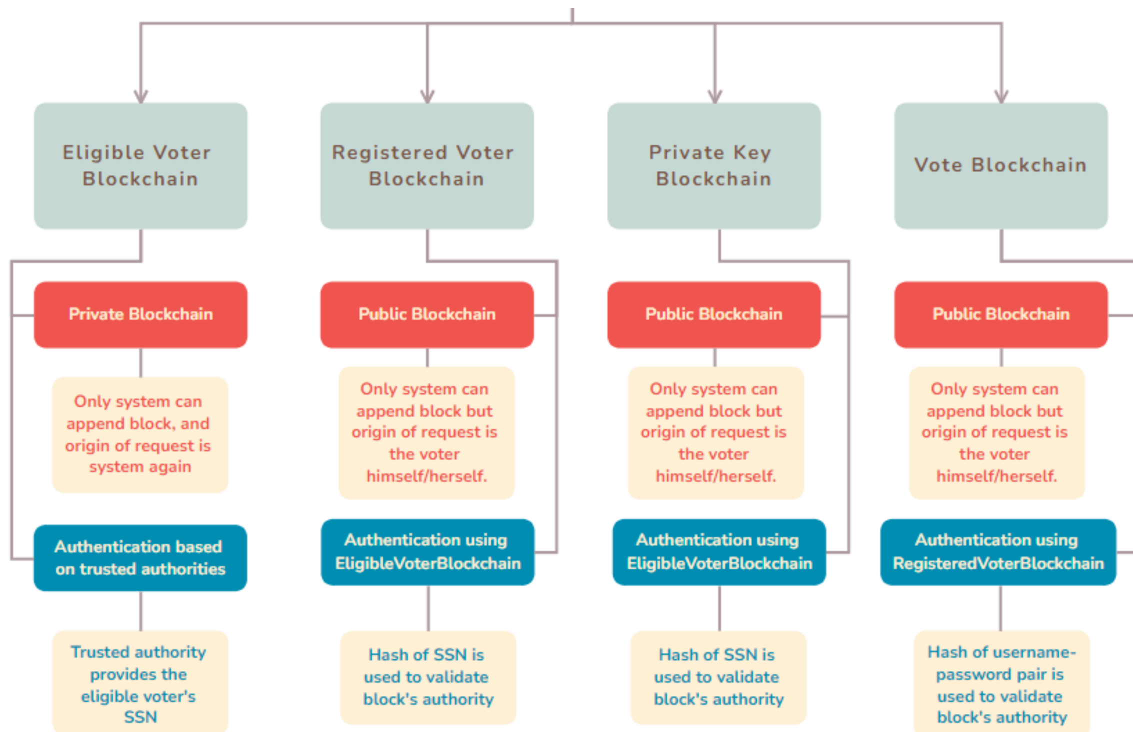

Figure 3.5. The Comparison of Four Blockchain.

successfully processed the vote data. To do that, voters can use the calculated hash value of the vote block. In sub-phase Hashing for Vote Verification, the system calculated the block's hash value and the value sent to the voter. This hash value will be used to verify the vote's existence in the blockchain.

Simple REST API using HTTPS can easily handle this communication between the voter and the system. The voter will use hashed value just like an API Key, to attach this hashed value there exist possible ways;

- Via the apiKey querystring parameter.
- Via the X-Api-Key HTTP header.
- Via the Authorization HTTP header.

The last two option is recommended, so that voter's hashed value is not visible to others in logs or via request sniffing [37].

If the hashed value exists in one of the blocks in VoteBlockchain, then the system will return the necessary information to the voter in JSON format with 200 status code. If it does not exist in the blockchain, then 404 status code will return with no information.

In conclusion, the requirement *v. Individual verifiability* is satisfied.

3.3.6. Counting Phase

As mentioned in section *3.3.1. Setup Phase*, the proposed system ensures that before the voting process ends, the counting phase can not start and any vote data can not be decrypted by attacker or the system itself since the system's RSA private key is distributed among different trusted authorities using VSS.

All data will be kept in encrypted form, and any decryption operation of vote data can start only in the counting phase due to *iv. Fairness* requirement. If the system or an attacker reads the data during the voting process, it may affect the results of the election with a high probability.

After the voting process is completed, first, the system will announce the Merkle root hash of the last block and the total used vote to the World Wide Web. In other words, the system will broadcast the specified data so that anyone can access these data. Merkle root hash contains all block's hash values, and this feature ensures the whole blockchain's immutability and integrity. Therefore, requirements *i. Inalterability* and *ix. Integrity* are satisfied.

After that, the system will traverse the blockchain from the last block to the genesis block and announce additional information, such as; total/current vote count, total accepted/rejected vote count, current block's hash value, Merkle root hash of the current block, etc. All these various information provides the security, transparency, and reliability of the whole voting process directly or indirectly.

Merkle root hash contains all block's hash values, and this feature ensures the whole blockchain's immutability and integrity.

In registration phase, each voter's Paillier private key is appended into PrivateKeyBlockchain after encrypted with the system's RSA public key and this blockchain ensures that, even if an attacker some-how manages to append a new block into VoteBlockchain, the vote data can not be decrypted using any of the PrivateKeyBlockchain block. The decryption keys are defined during registration phase and the blockchain's Merkle root hash is already shared to the World Wide Web. No one can append fraud private key, so no one can cast a fraud vote, as explained.

To decrypt each block in the VoteBlockchain, the system first accesses the PrivateKeyBlockchain. However, the voter's Pailler private keys inside each block is kept in

encrypted form. To decrypt these private keys, the system collects all RSA decryption key pieces from trusted authorities and obtains the system's RSA private key. After that, each voter's encrypted Paillier private key is decrypted using the system's RSA private key. This decryption is performed for each block, and all Paillier private keys are appended to the keyring.

In this keyring, all Paillier private keys of the voters are mixed, and there is no way to match them with voters, and this satisfies the requirement *vii. Privacy*. Finally, the system obtains all registered voter's private keys as one special keyring, and then the system will be able to decrypt each block in the VoteBlockchain.

We should remember that there are high probably different number of eligible voter, registered voter, and, valid vote respectively $n_{eligible_voter}$, $n_{registered_voter}$, n_{valid_vote} $n_{eligible_voter} \geq n_{registered_voter} \geq n_{valid_vote}$

To count the votes, the system traverses VoteBlockchain from the last node to the genesis node, and each block will be decrypted using the special keyring. With this method, the vote data and the voter can not be correlated. If one of the Paillier private keys in the keyring manages to decrypt the vote data, then the system keeps track of each candidate's statistics. After all blocks are decrypted and counted, the results will be announced to the World Wide Web.

- VoteBlockchain
- Shuffled keyring

We should also specify that the following Merkle root hash values are also shared with the World Wide Web to make this system more secure and transparent. The integrity of the blockchain is preserved using Merkle root hash so that the requirement *ix. Integrity* is satisfied during the *3.3.1. Setup Phase* and *3.3.2. Registration Phase*

- Merkle root hash of EligibleVoterBlockchain was already shared to the World Wide Web after Setup phase.
- Merkle root hash of RegisteredVoterBlockchain was already shared to the World Wide Web after Registration phase.
- Merkle root hash of PrivateKeyBlockchain was already shared to the World Wide Web after Registration phase.

3.3.7. Announcement Phase

After votes are successfully counted, this phase can start. In this phase, the system announces the statistical data to the World Wide Web. The following list shows the available statistical data.

- Setup phase - number of candidate, number of eligible voter.
- Registration phase - number of voter's Paillier private key, total number of register attempt, number of succeed register attempt, number of failed register attempt because of the eligibility problem, number of failed register attempt because of the second time register attempt.
- Voting phase - total number of used vote, number of valid used vote, number of invalid used vote because of the login with fake username, number of invalid used vote because of the login with incorrect credentials.
- Counting phase - total decrypted vote count, number of decryption succeed and valid vote count, number of decryption succeed and invalid vote count, number of decryption failed vote count.
- Announcement phase - vote results.

3.3.8. Individual and Universal Verifiability After Results Announced

Before introducing these two new phases, let us re-examine the differences between *v. Individual verifiability*, *vi. Universal verifiability*, and *xiii. Election replay*.

In individual verifiability, the voter can verify the existence of his/her vote using the hash value of the vote data. The voter can perform this request either during the voting phase or after the results are announced. As mentioned in section *3.3.5. Individual Verifiability During Voting Phase*, the voter can verify vote data with the help of HTTP requests. The detailed process is described in that phase. The only difference between these two phases is the level of information. Suppose the voter tries to verify the vote during the voting phase. In that case, the system will inform the voter whether the vote is appended to the VoteBlockchain successfully or not. If the voter tries to verify the vote after the results are announced, then the system will inform the voter with all the

information inside the block, such as vote time, vote choice, hash Of block, etc. In other words, the voter can track his/her vote so that the requirement *v. Individual verifiability* is satisfied.

In universal verifiability, anyone can verify the correctness of the whole voting process using system keys and data. In our case, the shuffled keyring and VoteBlockchain will be shared to the World Wide Web. With these two data, anyone can decrypt the whole election data without violating the voter's privacy.

In election replay, in addition to the election verification capability, anyone can replay the whole election process from beginning to end. After the votes are counted and the system announces the results, the system will share other necessary data such as anonymous voters' Paillier public key, shuffled keyring, VoteBlockchain, different Merkle root hash values, and so on. Instead of traditional solutions, the vote data in our VoteBlockchain is cumulative. Each block contains the hash of previous blocks, and the integrity of the vote data is always preserved. Moreover, we shared the three parts of the result: the anonymous voters' Paillier public key, the shuffled keyring, and the VoteBlockchain. In election replay, the procedure will be like:

1. Traverse the VoteBlockchain and try to decrypt each vote using shuffled keyring.
2. Obtain all non-encrypted anonymous choices.
3. Count the votes.
4. Compare the announced result with locally performed.
5. Iterate through obtained data. (the processing order of choice data is not important)
6. Encrypt choice with any of the voter's Paillier public key (the processing order of the voter's Paillier public key is not important)
7. Append encrypted vote data into VoteBlockchain.
8. Do the same thing for the rest of (choice, Paillier public key) pair.
9. At the end, the VoteBlockchain will be generated again with a high possibly different order, but the result will be the same.
10. Do the same procedure from step 1 to step 4.
11. Asynchronously, perform cyber-attacks.

Actually, this simple algorithm exposes a remarkable note. In our system, *everyone is equal, no matter their choice*.

In conclusion, the requirements *vi. Universal verifiability* and *xiii. Election replay* are satisfied.

CHAPTER 4

IMPLEMENTATION OF THE SYSTEM

In this chapter, the implementation of the proposed system will be examined in detail. The source code can be found on GitHub [35].

4.1. Installation

Ubuntu 20.04 operating system and Python programming language are chosen to implement in the system with version 3.8.10, and the following guide is prepared according to this specification. The hardware specification of the development environment is listed below: CPU: Intel(R) Core(TM) i5-10200H CPU @ 2.40GHz, GPU: Nvidia GeForce GTX 1050 Ti, Memory: 16GB, SODIMM DDR4 Synchroous 2933 MHz.

To clone source-code from GitHub, type the following command in your terminal.

```
$ cd /home/<username>/Desktop
$ git clone https://github.com/mustafakaracay/blockchain-based-e-voting-system
```

For a better and clean installation, it is recommended to use virtual environments while installing the dependencies. Since the dependencies are external Python packages and the same package's different version might already be installed in your local system.

To create a virtual environment, use the following command.

```
$ cd /home/<username>/Desktop/blockchain-based-e-voting-system/
$ python3.8 -m venv venv
```

Once you create a virtual environment, you need to activate it with the following command.

```
$ source venv/bin/activate
```

The required dependencies are exported into $requirements.txt$ with the exact version number. The dependencies can be installed with the following command.

```
$ pip3 install requirements.txt
```

Alternatively, the above commands are combined in the installation.sh script file; for simplicity, you can directly run this script and complete the installation. To do that, you need to give the necessary permission to the installation script and run it with the following command.

```
$ sudo chmod 777 installation.sh
$ sudo ./installation.sh
```

4.2. Architecture

In chapter *3.3. Proposed Solution and Design*, we described the details of the voting process and divided it into different phases such as setup phase, register phase, etc. In this section, we will introduce the source code of the thesis.

Once you cloned the source code from GitHub, the directory tree should look like:

```
|-- blockchain-based-e-voting-system
|    `-- simulation-data
|        `-- dummy-candidates.json
|        `-- dummy-eligible-voter-info.json
|        `-- dummy-eligible-voter-ssn.json
|        `-- dummy-usernames-password.json
|        `-- dummy-vote.json
|        `-- postman_collection.json
|    |-- announcement_phase.py
|    |-- block_eligible_voter.py
|    |-- block_private_key.py
|    |-- block_registered_voter.py
|    |-- block_vote.py
|    |-- config.ini
|    |-- counting_phase.py
|    |-- dashboard.py
|    |-- encryption.py
|    |-- helper.py
|    |-- installation.sh
|    |-- main.py
|    |-- register_phase.py
|    |-- requirements.txt
|    |-- setup_phase.py
|    |-- voting_phase.py
```

4.2.1. main.py

The implementation of the proposed solution can be started with the *main.py* file using the following command.

```
$ python3.8 main.py
```

Flask Framework: *main.py* file is basically a Flask application. Flask is a lightweight WSGI web application framework, and the details can be found on the official website [20].

First, the system imports other classes and python packages including Flask.

```
from flask import Flask, request, jsonify, Response
...
```

Next, the system creates an instance of the Flask class. This instance is a WSGI application. The first argument is the name of the application's module or package. *__name__* is a convenient shortcut so that Flask knows where to look for resources such as templates and static files.

```
app = Flask(__name__)
```

Then, the system binds a function to a URL using *route*() decorator to tell Flask what URL should trigger which function.

```
@app.route('/setupPhase/setStartEndTime', methods=['GET', 'POST'])
def setup_start_end_time_endpoint():
    do_something()
```

Finally, the system runs the application.

```
# app.run(host='127.0.0.1', port=5000, ssl_context='adhoc')   # HTTPS
app.run(host='127.0.0.1', port=5000)  # HTTP
```

Creating a New Instance: First, we created an instance of each blockchain such as EligibleVoterBlockchain, RegisteredVoterBlockchain, etc. And then, we created instances for each phase such as setup phase, register phase, etc.

Code Snippet 4.1. Instances of Blockchains and Phases

```
eligible_voter_blockchain = EligibleVoterBlockchain()
registered_voter_blockchain = RegisteredVoterBlockchain()
private_key_blockchain = PrivateKeyBlockchain()
vote_blockchain = VoteBlockchain()

setup_phase = SetupPhase()
register_phase = RegisterPhase()
voting_phase = VotingPhase()
dashboard_data = Dashboard()
counting_phase = CountingPhase({})
announcement_phase = AnnouncementPhase()
```

Functions to Calculate Hash: Let us first introduce our hash calculator functions before our classes and the endpoints. These two functions will be used by each of our class.

Code Snippet 4.2. Calculate Hash of Given Values

```
def calculate_hash(data, key_attributes=None):
    merged_content = ""

    if key_attributes is not None and len(key_attributes) > 0:
        for key in key_attributes:
            value = str(data[key])
            merged_content += value
    else:
        for key in data.keys():
            value = str(data[key])
            merged_content += value

    encoded_content = merged_content.encode('utf-8')
    hashed_value = hashlib.sha256(encoded_content).hexdigest()
    return hashed_value
```

$def\ \ calculate_hash(data,\ \ key_attributes = None)$ function takes the Python dict as first parameter, and Python list as second parameter. $key_attributes$ set to $None$ by default. If the $key_attributes$ is None or empty list, then the system traverses the whole key-value pair of the given dictionary. This means that, if a given $key_attributes$ value contains some key, then the system evaluates only these key-value pairs to calculate the hash. The system merges the desired value into a new string, called $merged_content$. And then, the system encodes the merged string with $utf - 8$. UTF-8 stands for Unicode Transformation Format - 8 bits. After that, $sha256$ hash function is used to calculate

the hash value of encoded content. SHA stands for Secure Hash Algorithm. Finally, the function returns the calculated hash function.

Code Snippet 4.3. Calculate Merkle Root Hash of Given Block

```
def calculate_merkle_root_hash(new_block_hash, current_merkle_root_hash):
    merged_content = current_merkle_root_hash + new_block_hash
    encoded_content = merged_content.encode('utf-8')
    merkle_root_hash = hashlib.sha256(encoded_content).hexdigest()
    return merkle_root_hash
```

$def\ calculate_merkle_root_hash(new_block_hash, current_merkle_root_hash)$ function takes the hash of the new block as first parameter, and Merkle root hash of current block as second parameter. The system combines the new block's hash value and the current block's Merkle root hash. And then, encodes the combined string using UTF-8. After that, the system calculates the hash value of the encoded value using SHA256. Finally, the function returns the calculated hash function.

Blockchains: And then, we can introduce our classes before the endpoints. All of these classes represent a unique blockchain. We identified the relationships between these blockchains in Figure 3.4. And, we compared our blockchains in Figure 3.5. In below Figure 4.1, we will introduce the interactions between the voter and our blockchains.

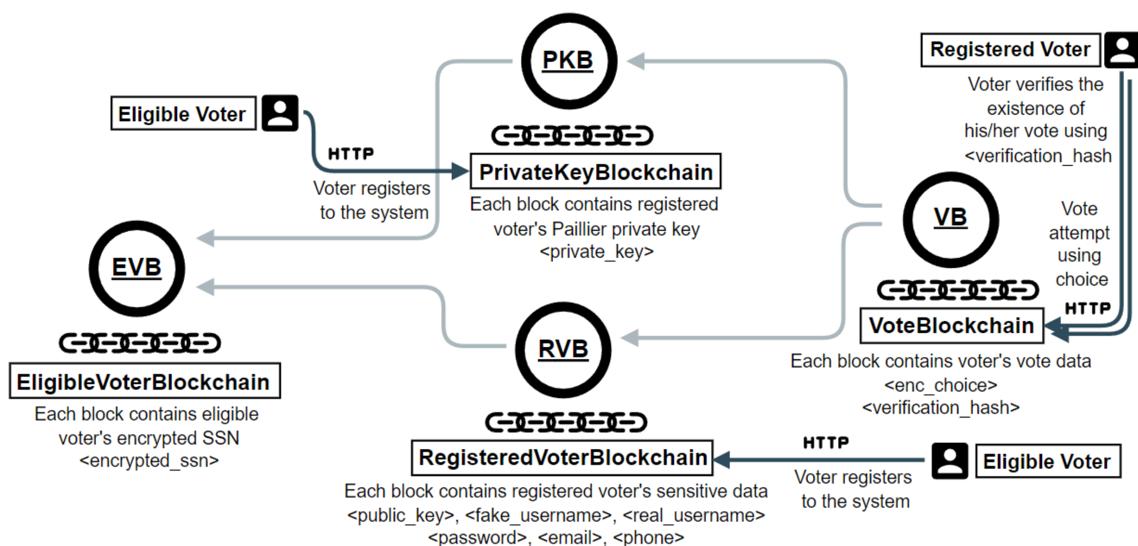

Figure 4.1. The Interaction Between the Voters and Our Blockchains.

An eligible voter can append a new block into the RegisteredVoterBlockchain using his/her sensitive data such as fake username, real username, password, etc.

An eligible voter can append a new block into the PrivateKeyBlockchain using his/her Paillier private key.

A registered voter can append a new block into the VoteBlockchain using his/her vote choice.

A registered voter can send HTTP POST request to VoteBlockchain to verify his/her vote's existence in the VoteBlockchain using his/her verification hash of vote data.

They have a common $def\ __init__(self)$ function to initialize $last_block$ to a $None$ value.

$def\ _get_merkle_root_hash(self)$ and $def\ _get_last_block_hash(self)$ functions are also common and performs exactly the same. $_single_leading_underscore$, $_$, represents that these functions are private for that class, and they can not be accessible from the outside of that class.

Since these classes are used to create unique blockchains, they all have their own $def\ insert(self, < params >)$ function. The function parameters differ from one class (or blockchain) to another.

Code Snippet 4.4. EligibleVoterBlockchain

```
class EligibleVoterBlockchain:
    def __init__(self):
        self.last_block = None

    def _get_merkle_root_hash(self):
        merkle_root_hash = self.last_block.merkle_root_hash
        return merkle_root_hash

    def _get_last_block_hash(self):
        last_block_hash = self.last_block.current_block_hash
        return last_block_hash

    def verify_register_attempt(self, ssn):
        register_data = {'ssn': ssn}
        hashed_ssn = calculate_hash(register_data, ['ssn'])

        # Traverse from last node to genesis node and check that calculated
            hash value is matched or not
        current_block = self.last_block
        while current_block:
```

```
            if hashed_ssn == current_block.verify_block_ssn_hash:
                return current_block  # Hashed value of SSN matched

            current_block = current_block.prev_block

        return None  # Hashed value of SSN did not match

    def insert(self, ssn, encrypted_ssn):
        non_encrypted_data = {'ssn': ssn}
        register_data = {'encryptedSsn': encrypted_ssn, 'timestamp': int(time.
            time() * 1000)}
        verify_block_ssn_hash = calculate_hash(non_encrypted_data, ['ssn'])
        current_block_hash = calculate_hash(register_data, None)

        if self.last_block:
            prev_block_hash = self._get_last_block_hash()
            prev_block_merkle_root_hash = self._get_merkle_root_hash()
            merkle_root_hash = calculate_merkle_root_hash(current_block_hash,
                prev_block_merkle_root_hash)

            new_node = EligibleVoterBlock(register_data, None,
                verify_block_ssn_hash, current_block_hash, prev_block_hash,
                merkle_root_hash)

            new_node.prev_block = self.last_block
            self.last_block = new_node

        else:  # self.last_block == None
            prev_block_hash = None  # since this is the first and only block
            merkle_root_hash = current_block_hash

            new_node = EligibleVoterBlock(register_data, None,
                verify_block_ssn_hash, current_block_hash, prev_block_hash,
                merkle_root_hash)

            self.last_block = new_node

    def deactivate_register_attempt(self, block):
        block.is_registered = True
```

class EligibleVoterBlockchain will be used during the setup phase. As explained in the system assumptions section, an eligible voter list is defined by the authority, and the list is shared with the proposed system in the setup phase. In this class, the system creates a new blockchain for the SSN of eligible voters.

$def\ \ insert(self,\ \ ssn,\ \ encrypted_ssn)$ function will be used to append a new block into the EligibleVoterBlockchain. The details of each block in the EligibleVoterBlockchain are shown in the code snippet 3.1. This function takes the SSN, and encrypted form of the SSN as function parameters.

$def\ \ verify_register_attempt(self,\ \ ssn)$ function will be used during Register Phase to check whether the given SSN is inside the EligibleVoterBlockchain or not. To do that, the system calculates the hash value of the given SSN and assigns it to the $hashed_ssn$ variable. And then, the system traverses the EligibleVoterBlockchain from the last node to the genesis node, and compare's the current block's $verify_block_ssn_hash$ value with the calculated $hashed_ssn$ value. If the values are matched then the system understands that a voter is eligible to register to the system. To traverse a blockchain the system just follows the link of the $prev_block$. The function either returns the $current_block$ (hashes matched) or returns the $None$ value (hashed did not match).

$def\ \ deactivate_register_attempt(self,\ \ block)$ function will be used during Register Phase. Any voter can register to the system only once. This boolean flag is stored separately for each block in the EligibleVoterBlockchain, and the system automatically set this flag to True, when a voter successfully registered to the system.

Code Snippet 4.5. RegisteredVoterBlockchain

```
class RegisteredVoterBlockchain:
    def __init__(self):
        self.last_block = None

    def _get_merkle_root_hash(self):
        merkle_root_hash = self.last_block.merkle_root_hash
        return merkle_root_hash

    def _get_last_block_hash(self):
        last_block_hash = self.last_block.current_block_hash
        return last_block_hash

    def verify_login_attempt(self, login_data):
        hashed_data = calculate_hash(login_data, ['username', 'password'])

        # Traverse from last node to genesis node and check that calculated
            hash value is matched or not
        current_block = self.last_block
        while current_block:
            if hashed_data == current_block.verification_hash_fake:
```

```
                return current_block, 2  # Login attempt with fake username
            elif hashed_data == current_block.verification_hash_real:
                return current_block, 1  # Login attempt with real username

            current_block = current_block.prev_block

        return current_block, 0

    def insert(self, data):
        verification_hash_fake = calculate_hash(data, ['fakeUsername', '
            password'])
        verification_hash_real = calculate_hash(data, ['realUsername', '
            password'])
        data['timestamp'] = int(time.time() * 1000)
        current_block_hash = calculate_hash(data, None)

        if self.last_block:
            prev_block_hash = self._get_last_block_hash()
            prev_block_merkle_root_hash = self._get_merkle_root_hash()
            merkle_root_hash = calculate_merkle_root_hash(current_block_hash,
                prev_block_merkle_root_hash)

            new_node = RegisteredVoterBlock(data, data['publicKey'], None,
                verification_hash_fake, verification_hash_real,
                current_block_hash, prev_block_hash, merkle_root_hash)

            new_node.prev_block = self.last_block
            self.last_block = new_node

        else:  # self.last_block == None
            prev_block_hash = None  # since this is the first and only block
            merkle_root_hash = current_block_hash

            new_node = RegisteredVoterBlock(data, data['publicKey'], None,
                verification_hash_fake, verification_hash_real,
                current_block_hash, prev_block_hash, merkle_root_hash)

            self.last_block = new_node
```

$class\ RegisteredVoterBlockchain$ will be used during the voting phase. In this class, the system creates a new blockchain for eligible voters, and the only way of adding a new block to the RegisteredVoterBlockchain is that the voter has to be eligible to register and complete the Register Phase successfully. In this case, we can call this blockchain as RegisteredVoterBlockchain.

$def\ \ insert(self,\ \ data)$ function will be used to append a new block into the RegisteredVoterBlockchain. The details of each block in the RegisteredVoterBlockchain are shown in the code snippet 3.2. This function takes the voter's data as function parameter, $dict\ \ data$ contains the voter's Paillier public key of the voter together with the voter's sensitive data such as fakeUsername, realUsername, password, etc. The sensitive data is always transferred and stored in encrypted form.

$def\ \ verify_login_attempt(self,\ \ login_data)$ function will be used during Voting Phase to check whether the given username-password pair is inside the Registered-VoterBlockchain or not. There are three different use cases in the return process. To detect each case, the system calculates the hash value of the given username and password and assigns it to the $hashed_login_data$ variable. And then, the system traverses the RegisteredVoterBlockchain from the last node to the genesis node, and compare's the current block's $verification_hash_fake$ and $verification_hash_real$ values with the calculated $hashed_login_data$ value. To traverse a blockchain the system just follows the link of the $prev_block$.

Code Snippet 4.6. PrivateKeyBlockchain

```
class PrivateKeyBlockchain:
    def __init__(self):
        self.last_block = None

    def _get_merkle_root_hash(self):
        merkle_root_hash = self.last_block.merkle_root_hash
        return merkle_root_hash

    def _get_last_block_hash(self):
        last_block_hash = self.last_block.current_block_hash
        return last_block_hash

    def insert(self, data):
        data['timestamp'] = int(time.time() * 1000)
        current_block_hash = calculate_hash(data, None)

        if self.last_block:
            prev_block_hash = self._get_last_block_hash()
            prev_block_merkle_root_hash = self._get_merkle_root_hash()
            merkle_root_hash = calculate_merkle_root_hash(current_block_hash,
                prev_block_merkle_root_hash)

            new_node = PrivateKeyBlock(data, None, current_block_hash,
```

```
                prev_block_hash, merkle_root_hash)

            new_node.prev_block = self.last_block
            self.last_block = new_node

        else:  # self.last_block == None
            prev_block_hash = None  # since this is the first and only block
            merkle_root_hash = current_block_hash

            new_node = PrivateKeyBlock(data, None, current_block_hash,
                prev_block_hash, merkle_root_hash)

            self.last_block = new_node

    def generate_keyring(self):
        voters_priv_keyring = phe.paillier.PaillierPrivateKeyring()

        current_block = self.last_block
        while (current_block):
            # update dashboard data for number of voter's Paillier private key
            Dashboard.increase_dashboard_data("numberOfVoterPrivateKey")

            voters_priv_keyring.add(current_block.private_key)
            current_block = current_block.prev_block

        # shuffle the keyring to protect voter's privacy
        random.shuffle(voters_priv_keyring)

        return voters_priv_keyring
```

$class\ \ PrivateKeyBlockchain$ will be used during the counting phase. In this class, the system creates a new blockchain for Paillier private keys of the registered voters.

$def\ \ insert(self,\ \ data)$ function will be used to append a new block into the PrivateKeyBlockchain. The details of each block in the PrivateKeyBlockchain are shown in the code snippet 3.3. This function takes the Paillier private key of the voter as function parameter.

$def\ \ generate_keyring(self)$ function will be used to generate a keyring with all registered voters' Paillier private keys. This means that, only the valid votes can be decrypted with this keyring. To generate such a keyring, the system traverses the PrivateKeyBlockchain from the last node to the genesis node, gets the voter's Paillier private key of the voter, and then adds that private key to the keyring. After the system reaches

the genesis block, all Paillier private keys are added to the keyring. However, we should remember that sequential traverse may create security problems. To solve this issue, the system shuffles the keyring, so that voters' Paillier private keys inside the keyring are now totally mixed. The system ensures that votes and voters can not be correlated so that no one can violate voters' privacy. Therefore, the requirement *vii. Privacy* is satisfied.

Code Snippet 4.7. VoteBlockchain

```
class VoteBlockchain:
    def __init__(self):
        self.last_block = None

    def _get_merkle_root_hash(self):
        merkle_root_hash = self.last_block.merkle_root_hash
        return merkle_root_hash

    def _get_last_block_hash(self):
        last_block_hash = self.last_block.current_block_hash
        return last_block_hash

    def insert(self, data):
        data['timestamp'] = int(time.time() * 1000)
        current_block_hash = calculate_hash(data, None)  # hash of (ssn+
            timestamp)

        if self.last_block:
            prev_block_hash = self._get_last_block_hash()
            prev_block_merkle_root_hash = self._get_merkle_root_hash()
            merkle_root_hash = calculate_merkle_root_hash(current_block_hash,
                prev_block_merkle_root_hash)

            new_node = VoteBlock(data, None, current_block_hash,
                prev_block_hash, merkle_root_hash)

            new_node.prev_block = self.last_block
            self.last_block = new_node

        else:  # self.last_block == None
            prev_block_hash = None  # since this is the first and only block
            merkle_root_hash = current_block_hash

            new_node = VoteBlock(data, None, current_block_hash,
                prev_block_hash, merkle_root_hash)

            self.last_block = new_node
```

```
def individual_verify_voting_phase(self, received_verify_hash):
    current_block = self.last_block
    while (current_block):
        if current_block.verification_hash == received_verify_hash:
            return True     # received verification hash matched with one
                of the block's hash value

        current_block = current_block.prev_block

    return False    # received verification hash did not match with any of
        the block's hash value

def individual_verify_announcement_phase(self, received_verify_hash):
    current_block = self.last_block
    while (current_block):
        if current_block.verification_hash == received_verify_hash:
            return current_block     # received verification hash matched
                with one of the block's hash value

        current_block = current_block.prev_block

    return None    # received verification hash did not match with any of
        the block's hash value

def get_last_block(self):
    return self.last_block
```

$class\ VoteBlockchain$ will be used during the voting phase, and counting phase. In this class, the system creates a new block for each valid vote attempt. The conditions for adding a new block to the VoteBlockchain will be explained in later sections.

$def\ \ insert(self,\ \ data)$ function will be used to append a new block into the VoteBlockchain. The details of each block in the VoteBlockchain are shown in the code snippet 3.4. This function takes the vote data as function parameter, $dict\ \ data$ contains the voter's choice in encrypted form.

$def\ \ individual_verify_voting_phase(self,\ \ received_verify_hash)$ function will be used during the voting phase to verify whether the voters' vote data is added into VoteBlockchain successfully or not. To do that the system uses a special hash value which is generated after the voter successfully votes. The system traverses the VoteBlockchain from the last node to the genesis node and compares the received hash value, $received_verify_hash$, with the current block's $verification_hash$ value. If any of the

block's hash value matches with the received hash value, then the system returns $True$. Otherwise, it returns $False$.

$def\ \ individual_verify_announcement_phase(self,\ \ received_verify_hash)$ function will be used during the announcement phase to verify whether the voters' vote data is counted successfully or not. To do that the system uses a special hash value which is generated after the voter successfully votes. The system traverses the Vote-Blockchain from the last node to the genesis node and compares the received hash value, $received_verify_hash$, with the current block's $verification_hash$ value. If any of the block's hash value matches with the received hash value, then the system returns matched $current_block$. Otherwise, it returns $None$.

$def\ \ get_last_block(self)$ function will be used during the counting phase to get the last block of the VoleBlockchain.

Endpoints: Now, we can introduce the Flask endpoints. Each endpoint is created using the $route$ decorator, and they can accept either POST or GET requests regarding the use case. If the endpoint accepts both HTTP methods, then it means that GET request is implemented for the simulation of the system with given dummy data. The simulation tool is different than the requirement *xiii. Election replay*, and we will introduce the simulation tool at the end of this part.

Code Snippet 4.8. Setting the Start/End Times of Each Phase

```
@app.route('/setupPhase/setStartEndTime', methods=['GET', 'POST'])
def setup_start_end_time_endpoint():
    try:
        if request.method == 'POST':
            data = request.get_json()
            setup_phase.set_setup_time(data['setup_start_time'], data['
                setup_end_time'])
            setup_phase.set_register_time(data['register_start_time'], data['
                register_end_time'])
            setup_phase.set_vote_time(data['vote_start_time'], data['
                vote_end_time'])

        else:  # request.method == 'GET'
            setup_phase.set_setup_time(config_data['SETUP_START_TIME'],
                config_data['SETUP_END_TIME'])
            setup_phase.set_register_time(config_data['REGISTER_START_TIME'],
                config_data['REGISTER_END_TIME'])
```

```
            setup_phase.set_vote_time(config_data['VOTE_START_TIME'],
                config_data['VOTE_END_TIME'])

    except Exception as error:
        # update dashboard data for number of internal error
        Dashboard.increase_dashboard_data("numberOfInternalError")
        error_message, error_code = Helper.text_parser(str(error))
        response = {"error_description": error_message}
        return jsonify(response), error_code

    response = {"description": "Start/End time of Setup, Register, Voting Phase
         successfully initialized."}
    return jsonify(response), 200
```

$'/setupPhase/setStartEndTime'$ endpoint sets the start time and end time of setup phase, register phase, and voting phase. Expected JSON for this endpoint is shown below.


```
{
    "setup_start_time": 0,
    "setup_end_time": 9999999999999,
    "register_start_time": 0,
    "register_end_time": 9999999999999,
    "vote_start_time": 0,
    "vote_end_time": 9999999999999
}
```


Code Snippet 4.9. Initialize Candidates

```
@app.route('/setupPhase/initCandidates', methods=['GET', 'POST'])
def setup_candidates_endpoint():
    try:
        setup_start_time, setup_end_time = setup_phase.get_setup_time()
        if Helper.is_time_interval_valid(setup_start_time, setup_end_time):
            if request.method == 'POST':
                data = request.get_json()
            else:  # request.method == 'GET'
                path = config_data['CANDIDATE_DATA_PATH']
                with open(path, "r") as fp:
                    data = json.load(fp)

            setup_phase.init_candidates(data)

        else:
            # update dashboard data for number of unauthorized requests
```

```
            Dashboard.increase_dashboard_data("numberOfUnauthorizedRequests")
            response = {"description": "The current time is not a valid time to
                 initialize candidates."}
            return jsonify(response), 403

    except Exception as error:
        # update dashboard data for number of internal error
        Dashboard.increase_dashboard_data("numberOfInternalError")
        error_message, error_code = Helper.text_parser(str(error))
        response = {"error_description": error_message}
        return jsonify(response), error_code

    response = {"description": "Candidates successfully initialized."}
    return jsonify(response), 200
```

$'/setupPhase/initCandidates'$ endpoint initializes the eligible candidates. This endpoint expects a list of JSON, and one sample expected JSON is shown below.

```
{
    "option": 1,
    "name": "Mustafa",
    "surname": "Karacay"
}
```

Code Snippet 4.10. Generate RSA Key Pair Endpoint

```
@app.route('/setupPhase/generateKey', methods=['GET'])
def setup_generate_key_endpoint():
    try:
        setup_start_time, setup_end_time = setup_phase.get_setup_time()
        if Helper.is_time_interval_valid(setup_start_time, setup_end_time):
            setup_phase.generate_rsa_key(config_data['RSA_KEY_SIZE'])

        else:
            # update dashboard data for number of unauthorized requests
            Dashboard.increase_dashboard_data("numberOfUnauthorizedRequests")
            response = {"description": "The current time is not a valid time to
                 generate RSA key-pair."}
            return jsonify(response), 403

    except Exception as error:
        # update dashboard data for number of internal error
        Dashboard.increase_dashboard_data("numberOfInternalError")
        error_message, error_code = Helper.text_parser(str(error))
        response = {"error_description": error_message}
```

```
        return jsonify(response), error_code

    response = {"description": "The system's RSA public and private key
        successfully generated."}
    return jsonify(response), 200
```

$'/setupPhase/generateKey'$ endpoint generates RSA public-private key pair, and these keys will be used by the system itself. This endpoint only accepts GET requests, and there is no query parameter for that. The endpoint calls the $def\ \ generate_rsa_key$ $(self,\ \ key_size)$ function, and details of this function is given below

Code Snippet 4.11. Generate RSA Key Pair Function

```
def generate_rsa_key(self):
    rsa_pub_key, rsa_priv_key = Encryption().generate_rsa_public_private_key(
        key_size)
    self.rsa_public_key = rsa_pub_key
    self.rsa_private_key = rsa_priv_key
    return rsa_pub_key, rsa_priv_key
```

$def\ \ generate_rsa_key(self,\ \ key_size)$ function generates RSA public and private key pair and initialize the $self.rsa_public_key$ and $self.rsa_private_key$ values using generated key pair. To do that, the function uses $classEncryption$ as a helper. The following code snippet shows the class Encryption.

Code Snippet 4.12. Paillier and RSA Key Generation Class

```
import phe
import rsa

class Encryption:
    def __init__(self):
        pass

    def generate_paillier_public_private_key(self, key_size):
        pub_key, priv_key = phe.paillier.generate_paillier_keypair(None,
            key_size)
        return pub_key, priv_key

    def generate_rsa_public_private_key(self, key_size):
        pub_key, priv_key = rsa.newkeys(key_size)
        return pub_key, priv_key
```

The related function uses rsa, and phe python packages to generate the RSA key pair, and Paillier key pair respectively. The key sizes can be configured using $config.ini$ file. The default values are 4096 bits for RSA, and 3072 bits for Paillier encryption in the configuration file.

Code Snippet 4.13. Initialize Eligible Voter's SSN

```
@app.route('/setupPhase/initEligibleVotersSsn', methods=['GET', 'POST'])
def setup_eligible_voters_ssn_endpoint():
    try:
        setup_start_time, setup_end_time = setup_phase.get_setup_time()
        if Helper.is_time_interval_valid(setup_start_time, setup_end_time):
            if request.method == 'POST':
                data = request.get_json()
            else:  # request.method == 'GET'
                path = config_data['ELIGIBLE_VOTER_SSN_DATA_PATH']
                with open(path, "r") as fp:
                    data = json.load(fp)

            setup_phase.init_eligible_voters_ssn(eligible_voter_blockchain,
                data)

        else:
            # update dashboard data for number of unauthorized requests
            Dashboard.increase_dashboard_data("numberOfUnauthorizedRequests")
            response = {"description": "The current time is not a valid time to
                initialize eligible voter's SSN."}
            return jsonify(response), 403

    except Exception as error:
        # update dashboard data for number of internal error
        Dashboard.increase_dashboard_data("numberOfInternalError")
        error_message, error_code = Helper.text_parser(str(error))
        response = {"error_description": error_message}
        return jsonify(response), error_code

    response = {"description": "Eligible voters' SSN successfully initialized."
        }
    return jsonify(response), 200
```

$'/setupPhase/initEligibleVotersSsn'$ endpoint initializes the SSN of eligible voters. At the end of this request, a new block will be added to the EligibleVoterBlockchain. Expected JSON for this endpoint is shown below.

```
{
    "ssn_list": [
        "67984846863",
        "52200070095",
        "92201740257",
        ...
    ]
}
```

Code Snippet 4.14. Voter Registration

```
@app.route('/registerPhase/initEligibleVotersInfo', methods=['GET', 'POST'])
def register_eligible_voters_info_endpoint():
    try:
        reg_start_time, reg_end_time = setup_phase.get_register_time()
        if Helper.is_time_interval_valid(reg_start_time, reg_end_time):
            if request.method == 'POST':
                data = request.get_json()
            else:
                path = config_data['ELIGIBLE_VOTER_INFO_DATA_PATH']
                with open(path, "r") as fp:
                    data = json.load(fp)

            register_phase.init_eligible_voter_info(eligible_voter_blockchain,
                registered_voter_blockchain, private_key_blockchain, data,
                config_data['PAILLIER_KEY_SIZE'])

        else:
            # update dashboard data for number of unauthorized requests
            Dashboard.increase_dashboard_data("numberOfUnauthorizedRequests")
            response = {"description": "The current time is not a valid time to
                 register."}
            return jsonify(response), 403

    except Exception as error:
        # update dashboard data for number of internal error
        Dashboard.increase_dashboard_data("numberOfInternalError")
        error_message, error_code = Helper.text_parser(str(error))
        response = {"error_description": error_message}
        return jsonify(response), error_code

    response = {"description": "Eligible voter(s) successfully registered to
        the system."}
    return jsonify(response), 200
```

$'/registerPhase/initEligibleVotersInfo'$ endpoint initialized the sensitive register data of the eligible voter. At the end of this request, a new block will be added to the RegisteredVoterBlockchain if the conditions are satisfied. This endpoint expects a list of JSON, and one sample expected JSON is shown below.


```
{
    "ssn": 67984846863,
    "fakeUsername": "TommieHampton",
    "realUsername": "TomHampton",
    "password": "601a73!46+8a5c.5",
    "email": "tommiehampton@mantro.com",
    "phone": "+90(890)455-3255"
}
```


Code Snippet 4.15. Vote Attempt

```
@app.route('/votingPhase/voteAttempt', methods=['GET', 'POST'])
def vote_attempt_endpoint():
    try:
        vote_start_time, vote_end_time = setup_phase.get_vote_time()
        if Helper.is_time_interval_valid(vote_start_time, vote_end_time):
            if request.method == 'POST':
                a_data = request.get_json()
                data = [a_data]
            else:  # request.method == 'GET'
                path = config_data['VOTE_DATA_PATH']
                with open(path, "r") as fp:
                    data = json.load(fp)

            voting_phase.init_vote_attempt(registered_voter_blockchain,
                vote_blockchain, data)

        else:
            # update dashboard data for number of unauthorized requests
            Dashboard.increase_dashboard_data("numberOfUnauthorizedRequests")
            response = {"description": "The current time is not a valid time to
                vote."}
            return jsonify(response), 403

    except Exception as error:
        # update dashboard data for number of internal error
        Dashboard.increase_dashboard_data("numberOfInternalError")
        error_message, error_code = Helper.text_parser(str(error))
        response = {"error_description": error_message}
        return jsonify(response), error_code
```

```
    response = {"description": "Vote successfully added to VoteBlockchain."}
    return jsonify(response), 200
```

$'/votingPhase/voteAttempt'$ endpoint initializes the sensitive vote data of the registered voter. At the end of this request, a new block will be added to the VoteBlockchain if the conditions are satisfied. Expected JSON for this endpoint is shown below.


```
{
    "choice": 1,
    "username": "TomHampton",
    "password": "601a73!46+8a5c.5"
}
```


Code Snippet 4.16. Individual Verify During Voting Phase

```
@app.route('/votingPhase/individualVerify', methods=['GET', 'POST'])
def individual_verify_during_voting_endpoint():
    try:
        vote_start_time, vote_end_time = setup_phase.get_vote_time()
        if Helper.is_time_interval_valid(vote_start_time, vote_end_time):
            if request.method == 'POST':
                data = request.get_json()
            else:  # request.method == 'GET'
                data = {"verificationHash": "dummyHash"}

            verification_hash = data['verificationHash']
            isVoteExist = vote_blockchain.individual_verify_voting_phase(
                verification_hash)

            if isVoteExist is True:
                response = {"description": "Your vote is added to blockchain
                    successfully, and no one can change this truth."}
                return jsonify(response), 200
            else:
                response = {"description": "Verification hash could not find in
                     blockchain, make sure that value entered correctly."}
                return jsonify(response), 200

        else:
            # update dashboard data for number of unauthorized requests
            Dashboard.increase_dashboard_data("numberOfUnauthorizedRequests")
            response = {"description": "The current time is not a valid time to
                 verify vote data."}
```

```
            return jsonify(response), 403

    except Exception as error:
        # update dashboard data for number of internal error
        Dashboard.increase_dashboard_data("numberOfInternalError")
        error_message, error_code = Helper.text_parser(str(error))
        response = {"error_description": error_message}
        return jsonify(response), error_code
```

$'/votingPhase/individualVerify'$ endpoint checks whether the voter's vote data is added into the VoteBlockchain successfully or not. Expected JSON for this endpoint is shown below.


```
{
    "verificationHash":
        "ca957bba2ce8d0fec42c227bebeae064316c36c33e41dfbf921704dfe0a967f3"
}
```


Code Snippet 4.17. Count Votes

```
@app.route('/countingPhase/countVotes', methods=['GET'])
def count_votes_endpoint():
    try:
        # voting end time can be considered as counting phase start time
        # counting phase must start after voting phase is completed
        counting_padding_time = 1000 * 60 * 3    # 3 min just for in case
        counting_start_time = int(setup_phase.get_vote_end_time())
        counting_end_time = 9999999999999

        if Helper.is_time_interval_valid(counting_start_time +
            counting_padding_time, counting_end_time):
            # we are going to use below set() for checking vote choice is for
                an eligible candidate or not
            candidate_set = set()

            # get eligible candidate from setup phase
            candidates_dict = setup_phase.get_candidates_dict()
            initialized_candidate_votes_count = {}

            # iterate through keys in the eligible candidate dict
            for k in candidates_dict.keys():
                candidate_set.add(k)
                # set the total vote of 'candidate_x' to zero.
                initialized_candidate_votes_count[str(k)] = 0
            # set the total fraud vote to zero
```

```
            initialized_candidate_votes_count['fraudData'] = 0
            # set the total invalid candidate vote to zero
            initialized_candidate_votes_count['invalidCandidate'] = 0

            # we already created an object of 'class CountingPhase' in main
                body, and on the above code block we initialized the vote
                counts properly. Now, we just need to update it using
                init_vote_results function.
            counting_phase.init_vote_results(initialized_candidate_votes_count)

            # generate keyring with all registered voters' Paillier private key
            counting_phase.generate_voters_keyring(private_key_blockchain)

            counting_phase.count_votes(vote_blockchain, candidate_set)

        else:
            # update dashboard data for number of unauthorized requests
            Dashboard.increase_dashboard_data("numberOfUnauthorizedRequests")
            response = {"description": "The current time is not a valid time to
                 count votes."}
            return jsonify(response), 403

    except Exception as error:
        # update dashboard data for number of internal error
        Dashboard.increase_dashboard_data("numberOfInternalError")
        error_message, error_code = Helper.text_parser(str(error))
        response = {"error_description": error_message}
        return jsonify(response), error_code

    response = {"description": "All votes in the blockchain counted
        successfully."}
    return jsonify(response), 200
```

$'/countingPhase/countVotes'$ endpoint counts the votes using VoteBlockchain and PrivateKeyBlockchain. This endpoint only accepts GET requests, and there is no query parameter for that.

Code Snippet 4.18. Announce the Results

```
@app.route('/announcementPhase/announceResults', methods=['GET'])
def announce_results_endpoint():
    try:
        # vote counting end time can be considered as announcement phase start
            time
        # announcement phase must start after counting phase is completed
        isCountingCompleted = counting_phase.get_is_vote_counting_completed()
```

```
        if isCountingCompleted is True:
            dashboard_data = Dashboard.get_dashboard_data()
            vote_results = counting_phase.get_vote_results()
            announcement_phase.init_dashboard_data(dashboard_data)
            announcement_phase.init_vote_results(vote_results)

            announcement_phase.plurality_voting_counting()

        else:
            # update dashboard data for number of unauthorized requests
            Dashboard.increase_dashboard_data("numberOfUnauthorizedRequests")
            response = {"description": "The current time is not a valid time to
                 announce the results."}
            return jsonify(response), 403

    except Exception as error:
        # update dashboard data for number of internal error
        Dashboard.increase_dashboard_data("numberOfInternalError")
        error_message, error_code = Helper.text_parser(str(error))
        response = {"error_description": error_message}
        return jsonify(response), error_code

    response = {"description": "All votes in the blockchain counted
        successfully."}
    return jsonify(response), 200
```

$'/announcementPhase/announceResults'$ endpoint announces the results to the public in other words World Wide Web. This endpoint only accepts GET requests, and there is no query parameter for that.

Code Snippet 4.19. Individual Verify After Voting Completed

```
@app.route('/announcementPhase/individualVerify', methods=['GET', 'POST'])
def individual_verify_after_voting_completed_endpoint():
    try:
        # result announcement end time can be considered as start time of
            individual verify phase
        is_results_announced = announcement_phase.get_is_results_announced()

        if is_results_announced is True:
            if request.method == 'POST':
                data = request.get_json()
            else:  # request.method == 'GET'
                data = {"verificationHash": "dummyHash"}
```

```
            verification_hash = data['verificationHash']
            vote_block = vote_blockchain.individual_verify_announcement_phase(
                verification_hash)

            if vote_block is not None:
                # decrypt vote data
                keyring = counting_phase.get_voters_keyring()
                decrypted_choice = keyring.decrypt(vote_block.enc_choice)
                response = {'description': "Your vote is counted successfully,
                    and no one can change this truth.", 'choice':
                    decrypted_choice}
                return jsonify(response), 200
            else:
                response = {"description": "Verification hash could not find in
                     the blockchain, please make sure that value entered
                    correctly."}
                return jsonify(response), 200

        else:
            # update dashboard data for number of unauthorized requests
            Dashboard.increase_dashboard_data("numberOfUnauthorizedRequests")
            response = {"description": "The current time is not a valid time to
                 verify vote data."}
            return jsonify(response), 403

    except Exception as error:
        # update dashboard data for number of internal error
        Dashboard.increase_dashboard_data("numberOfInternalError")
        error_message, error_code = Helper.text_parser(str(error))
        response = {"error_description": error_message}
        return jsonify(response), error_code
```

$'/announcementPhase/individualVerify'$ endpoint checks whether the voter's vote data is counted successfully or not. Expected JSON for this endpoint is shown below.


```
{
    "verificationHash":
        "ca957bba2ce8d0fec42c227bebeae064316c36c33e41dfbf921704dfe0a967f3"
}
```

4.2.2. setup_phase.py

Let us examine the two important functions of *setup_phase.py* since the rest of the functions are simple getter-setter functions.

Code Snippet 4.20. Setup Phase - Initialize Candidates

```
def init_candidates(self, data):
    # check that candidates are finalized or not
    if self.is_candidates_finalized is False:
        try:
            for a_data in data:
                # initialize candidate
                self.candidates_dict[a_data['option']] = a_data['name'] + " " +
                     a_data['surname']

                # update dashboard data for number of candidates
                Dashboard.increase_dashboard_data("numberOfCandidate")

            self.is_candidates_finalized = True

        except Exception as error:
            # update dashboard data for number of internal error
            Dashboard.increase_dashboard_data("numberOfInternalError")
            raise Exception(str(error))

    else:  # candidates are already set, so do nothing
        # update dashboard data for number of unauthorized requests
        Dashboard.increase_dashboard_data("numberOfUnauthorizedRequests")
        # 403 status code used for 'Forbidden' The system understood the client
            's request; however, it refuses to response on purpose.
        raise Exception("Candidates are already set and finalized, no-one can
            modify./403")
```

This function receives the eligible candidate data as a list of JSON and iterates through this list. At each iteration, an eligible candidate is added into the $self.candidates_dict$ as key-value pair.

Code Snippet 4.21. Setup Phase - Initialize Eligible Voter's SSN

```
    def init_eligible_voters_ssn(self, eligible_voter_blockchain, data):
        # check that eligible voters are finalized or not
        if self.is_eligible_voters_ssn_finalized is False:
            try:
                for ssn in data['ssn_list']:
```

```
                # add encrypted SSN into EligibleVoterBlockchain
                encoded_ssn = ssn.encode('utf-8')
                encrypted_ssn = rsa.encrypt(encoded_ssn, self.
                    rsa_public_key)
                eligible_voter_blockchain.insert(ssn, encrypted_ssn)

                # update dashboard data for number of eligible voters
                Dashboard.increase_dashboard_data("numberOfEligibleVoter")

            self.is_eligible_voters_ssn_finalized = True

        except Exception as error:
            # update dashboard data for number of internal error
            Dashboard.increase_dashboard_data("numberOfInternalError")
            raise Exception(str(error))

    else:  # eligible voters ssn are already set, so do nothing
        # update dashboard data for number of unauthorized requests
        Dashboard.increase_dashboard_data("numberOfUnauthorizedRequests")
        # 403 status code used for 'Forbidden' The system understood the
            client's request; however, it refuses to response on purpose.
        raise Exception("SSN of eligible voters are already set and
            finalized, no-one can modify./403")
```

This function receives the eligible voters' data as JSON, and iterates through the JSON's inner list. At each iteration, encodes and encrypt the SSN, and then the data will be inserted into the EligibleVoterBlockchain.

4.2.3. register_phase.py

The one and only function of *register_phase.py* file is shown below.

Code Snippet 4.22. Register Phase - Voter Registration

```
def init_eligible_voter_info(self, eligible_voter_blockchain,
    registered_voter_blockchain, private_key_blockchain, data, key_size):
    try:
        for a_data in data:
            # update dashboard data for number of register attempt
            Dashboard.increase_dashboard_data("totalNumberOfRegisterAttempt")

            ssn = str(a_data['ssn'])
            ssn_matched_block = eligible_voter_blockchain.
                verify_register_attempt(ssn)
```

```
            if ssn_matched_block is not None:
                isVoterRegistered = ssn_matched_block.is_registered
                if isVoterRegistered is False:
                    # create Paillier public, private key pair for that voter
                    public_key, private_key = Encryption().
                        generate_paillier_public_private_key(key_size)

                    # create new block for the voter's Paillier private key to
                        PrivateKeyBlockchain
                    priv_data = {'privateKey': private_key}
                    private_key_blockchain.insert(priv_data)

                    # create new block for registered voter to
                        RegisteredVoterBlockchain
                    a_data['publicKey'] = public_key
                    registered_voter_blockchain.insert(a_data)

                    # update dashboard data for number of succeed register
                        attempt
                    Dashboard.increase_dashboard_data("
                        numberOfSucceedRegisterAttempt")

                    # update is_registered boolean value in the
                        EligibleVoterBlockchain
                    eligible_voter_blockchain.deactivate_register_attempt(
                        ssn_matched_block)
                else:
                    # update dashboard data for number of failed register
                        attempt bacuse of the SSN matched but voter is already
                        registered.
                    Dashboard.increase_dashboard_data("
                        numberOfFailedRegisterAttemptCase2")
                    print("Voter is already registered to the system. SSN: " +
                        str(a_data['ssn']))
            else:
                # update dashboard data for number of failed register attempt
                    bacuse of the SSN did not match.
                Dashboard.increase_dashboard_data("
                    numberOfFailedRegisterAttemptCase1")
                print("Eligible voter's SSN did not match. SSN: " + str(a_data[
                    'ssn']))

        except Exception as error:
            # update dashboard data for number of internal error
            Dashboard.increase_dashboard_data("numberOfInternalError")
            raise Exception(str(error))
```

4.2.4. voting_phase.py

The one and only function of *voting_phase.py* file is shown below.

Code Snippet 4.23. Voting Phase - Vote Attempt

```
def init_vote_attempt(self, registered_voter_blockchain, vote_blockchain, data):
    try:
        # each iteration represent one vote attempt.
        for a_data in data:
            # update dashboard data for total number of vote
            Dashboard.increase_dashboard_data("totalNumberOfVoteUsed")

            # 1-check that login attempt is valid or not
            registered_voter_block, login_attempt_enum =
                registered_voter_blockchain.verify_login_attempt(a_data)

            # login attempt with realUsername - password detected
            if login_attempt_enum == 1:
                # 2-encrypt the vote data (choice)
                voter_pub_key = registered_voter_block.public_key
                enc_choice = voter_pub_key.encrypt(int(a_data['choice']))
                a_data['encryptedChoice'] = enc_choice

                # 3-append verification_hash_real to a_data
                a_data['verificationHashReal'] = registered_voter_block.
                    verification_hash_real

                # 4-remove sensitive data from a_data
                a_data.pop("choice")
                a_data.pop("username")
                a_data.pop("password")

                # 5-add timestamp
                a_data['timestamp'] = int(time.time() * 1000)

                # 6-add vote data to vote_blockchain
                vote_blockchain.insert(a_data)

                # update dashboard data for number of valid vote used
                Dashboard.increase_dashboard_data("numberOfValidVoteUsed")

            # login attempt with fakeUsername - password detected
            elif login_attempt_enum == 2:
                # update dashboard data for number of invalid vote because of
                    the fakeUsername
```

```
                Dashboard.increase_dashboard_data("numberOfInvalidVoteCase2")
                print("Login attempt with fakeUsername detected. Block did not
                    appended into VoteBlockchain")

            # login attempt failed because login credentials did not match
            else:
                # update dashboard data for number of invalid vote because of
                    the username password credential
                Dashboard.increase_dashboard_data("numberOfInvalidVoteCase3")
                print("Username password credential failed. Block did not
                    appended into VoteBlockchain")

    except Exception as error:
        # update dashboard data for number of internal error
        Dashboard.increase_dashboard_data("numberOfInternalError")
        raise Exception(str(error))
```

4.2.5. counting_phase.py

The one important function of *counting_phase.py* file is shown below since the rest of the functions are simple getter-setter functions.

Code Snippet 4.24. Counting Phase - Count Votes

```
def count_votes(self, vote_blockchain, candidate_set):
    try:
        keyring = self.get_voters_keyring()
    except Exception as error:
        # update dashboard data for number of internal error
        Dashboard.increase_dashboard_data("numberOfInternalError")
        raise Exception("The system could not load the voters' Paillier private
             key/403")

    # get the last block of VoteBlockchain
    last_block = vote_blockchain.get_last_block()
    # traverse from last node to genesis node until block points to None
    while (last_block):
        # update dashboard data for number of total decrypted vote
        Dashboard.increase_dashboard_data("totalDecryptedVoteCount")

        try:
            decrypted_vote_choice = keyring.decrypt(last_block.enc_choice)
        except Exception as error:
            # an exception has been catched, none of the registered voter's
```

```
                Paillier private key manages to decrypt vote data. so, update
                dashboard data for number of decryption failed vote data
            Dashboard.increase_dashboard_data("numberOfDecryptionFailedVote")
            self.vote_results['fraudData'] += 1

            last_block = last_block.prev_block
            continue

        # voter's chosen candidate is not an eligible candidate.
        if decrypted_vote_choice not in candidate_set:
            # update dashboard data for number of decrypted and invalid vote
            Dashboard.increase_dashboard_data("
                numberOfDecryptionSucceedAndInvalidVote")
            self.vote_results['invalidCandidate'] += 1
        else:
            # update dashboard data for number of decrypted and valid vote
            Dashboard.increase_dashboard_data("
                numberOfDecryptionSucceedAndValidVote")
            self.vote_results[str(decrypted_vote_choice)] += 1

        last_block = last_block.prev_block

    # update vote counting completed flag, so that announcement phase can start
    self.is_vote_counting_completed = True
```

4.2.6. announcement_phase.py

classAnnouncementPhase in the *announcement_phase.py* is shown below.

Code Snippet 4.25. Announcement Phase - Announce the Results

```
class AnnouncementPhase:
    def __init__(self, vote_results=None, dashboard_data=None):
        self.dashboard_data = dashboard_data
        self.vote_results = vote_results
        self.is_results_announced = False

    def init_dashboard_data(self, dashboard_data):
        self.dashboard_data = dashboard_data

    def init_vote_results(self, vote_results):
        self.vote_results = vote_results

    def plurality_voting_counting(self):
```

```
        winner_candidate_option = "0"     # as key
        winner_candidate_vote_count = 0       # as value
        for key, value in self.vote_results.items():
            if key == "numberOfUnauthorizedRequests" or key == "
                numberOfInternalError":
                continue

            if value > winner_candidate_vote_count:
                winner_candidate_option = key
                winner_candidate_vote_count = value

        self.is_results_announced = True
        return winner_candidate_option, winner_candidate_vote_count

    def get_is_results_announced(self):
        return self.is_results_announced
```

4.2.7. dashboard.py

The following code snippet shows the $classDashboard$.

Code Snippet 4.26. Dashboard Data

```
class Dashboard:
    dashboard_data = {
        'setupPhase': {
            'numberOfCandidate': 0,
            'numberOfEligibleVoter': 0
        },
        'registerPhase': {
            'numberOfVoterPrivateKey': 0,
            'totalNumberOfRegisterAttempt': 0,
            'numberOfSucceedRegisterAttempt': 0,
            'numberOfFailedRegisterAttemptCase1': 0,
            'numberOfFailedRegisterAttemptCase2': 0,
            'avgTransactionTimeToRegisterVoter': 0
        },
        'votingPhase': {
            'totalNumberOfVoteUsed': 0,
            'numberOfValidVoteUsed': 0,
            'numberOfInvalidVoteCase1': 0,
            'numberOfInvalidVoteCase2': 0,
            'numberOfInvalidVoteCase3': 0,
            'avgTransactionTimeToEncryptVoteData': 0,
```

```
            'avgTransactionTimeToVote': 0
        },
        'countingPhase': {
            'totalDecryptedVoteCount': 0,
            'numberOfDecryptionSucceedAndValidVote': 0,
            'numberOfDecryptionSucceedAndInvalidVote': 0,
            'numberOfDecryptionFailedVote': 0,
            'avgTransactionTimeToDecryptVoteData': 0
        },
        'announcementPhase': {
            'vote_results': {}
        },
        'systemData': {
            'numberOfUnauthorizedRequests': 0,
            'numberOfInternalError': 0
        }
    }
```

4.3. Simulation of a Vote Event

The introduced endpoints that accept GET HTTP requests are implemented to manage a simulation of a vote event for simplicity. The following endpoint triggers all the necessary endpoints and simulates the life cycle of a sample vote event. The simulation tool uses pre-defined dummy data to simulate each election phase. Dummy data stored under the cloned GitHub repository as JSON files, *blockchain-based-e-voting-system/simulation-data/dummy-data-name.json*

Code Snippet 4.27. Simulate of a Vote Event

```
@app.route('/simulate/newVotingLifecycle', methods=['GET'])
def simulate_new_voting():
    global config_data
    try:
        config_data = Helper.get_config("config.ini")
    except Exception as err:
        sys.exit(0)

    base_url = "http://" + config_data["IP"] + ":" + str(config_data["PORT"])
    init_endpoints = [
        '/setupPhase/setStartEndTime',
        '/setupPhase/initCandidates',
        '/setupPhase/generateKey',
```

```
        '/setupPhase/initEligibleVotersSsn',
        '/registerPhase/initEligibleVotersInfo',
        '/votingPhase/voteAttempt',
        '/votingPhase/individualVerify',
        '/countingPhase/countVotes',
        '/announcementPhase/announceResults',
        '/announcementPhase/individualVerify'
    ]

    try:
        for init_endpoint in init_endpoints:
            url = base_url + init_endpoint
            req = requests.get(url=url)
    except Exception as error:
        error_message, error_code = Helper.text_parser(str(error))
        response = {"error_description": error_message}
        return jsonify(response), error_code

    response = {"description": "New voting event is simulated successfully"}
    return jsonify(response), 200
```

In this case, you just need to trigger this special $'/simulate/newVotingLifecycle'$ endpoint. The straightest way to do that open a terminal and run the following command.

```
$ python3
```

Right now, we can execute python codes in the terminal. Type the following code piece into the python terminal.

```
>>> url = "http://127.0.0.1:5000/simulate/newVotingLifecycle"
>>> req = requests.get(url=url)
```

In addition to those $.json$ dummy data files, $postman_collection.json$ file is also exported using Postman. This file contains the sample request for each endpoint separately including $'/simulate/newVotingLifecycle'$ endpoint. You can directly import it into Postman and use the GUI to trigger the simulation tool/feature.

In addition to the election simulation tool, the system ensures the *xiii. Election replay* requirement thanks to the architectural design of the proposed solution. The user can replay formal/official elections using shared data such as anonymous voters' Paillier public key, shuffled keyring, and VoteBlockchain. One is a helper tool; the other is a system requirement.

CHAPTER 5

RELATED WORKS

5.1. Academic Researches

To solve any problem, we first need to understand what the requirements of the solution are. We can only propose a solution after we have identified the requirements. Identifying the requirements for a well-designed and secure e-voting system is naturally one of the most critical steps for our proposed thesis as well. The following paper ”EDU-VOTING: An Educational Homomorphic e-Voting System” [28] is the source of the requirement list of this thesis. Since that paper already identifies the requirements and fully covers them in detail. The selected paper has a very similar design with the our solution and the study has been published in the recent history. In addition to these reasons, other factors are listed as follows concerning common points.

- Paillier Homomorphic Cryptosystem is used for encryption.

In both studies, Paillier is preferred. This encryption mechanism provides for all votes to be kept in encrypted form until the voting process is complete. The way in which the data is kept in encrypted form is chosen to preserve manipulation of the current election results, and of course the security issues.

- VSS is used to distribute private key between authorities.

To make the system more secure and reliable, the key is distributed between authorities in both study. The proposed system distributes the system’s RSA private key. However the read vote data, an attacker need to access voter’s Paillier private key as well.

- Mixing the data approach is applied.

In the selected work, extra features exist, like shuffling the votes to protect the user’s privacy. The question here is, why do we need to mix the votes? Suppose an attacker somehow decrypts the vote data and accesses the database activities. In that case, the attacker can detect which decrypted data belongs to which user by comparing the vote time value and the login operation’s database activity log. This issue does not occur with

the proposed solution since different blockchains are used, and no log file is kept for traversing. However, the proposed thesis also uses shuffling for a similar reason. Voter's Paillier private keys are added into a keyring, and then the system shuffles the keyring before sharing it with the World Wide Web.

- All requirements are covered with different but similar approaches.

Using blockchain technology provides extra features to the proposed solution. And these features are used to cover identified requirements in new ways. In addition to these similarities, the proposed solution directly publishes all necessary information/data, including public-private key pairs, hash algorithms, blockchains itself, etc. to the World Wide Web after the election is completed to cover requirement *xiii. Election replay*.

The main difference between the two studies are as follows:

- Blockchain-based technology is not used in the background.

The selected paper offers an e-voting system without using blockchain technology. The proposed solution uses various blockchains, Merkle root hash usage, extra encryption layer, etc. And these features provide an extra security level, immutability, and transparency to the system.

The following paper "A Survey on Feasibility and Suitability of Blockchain Techniques for the E- Voting Systems" [51] is selected as the second related work because a well-designed and secure e-voting system is clearly held without blockchain technology in the first related work. However, this paper analyzes the feasibility and suitability of using blockchain technology together with e-voting systems, regarding both technical and non-technical aspects.

Below major challenges of the e-voting system are taken from the selected survey:

- Secure digital identity management: All the voters should sign up before the election and their information should be kept private in a format that is digitally processable in any database.
- Anonymous vote-casting: Voters should be able to choose any candidate or non anonymously. The votes should be kept anonymous during the whole process.
- Individualized ballot processes: Representation of vote data in web application or databases is still subject to argument. As a straightforward approach, a clear text message would be the worst idea in terms of voter anonymity. On the other hand,

a hashed token can solve the anonymity problem and provide integrity. Yet such a solution is not non-repodiable.

- Ballot casting verifiability by (and only by) the voter: For the sake of trust of the voters, vote data should be accessible to and verifiable by the voters themselves. With this feature, the system can prevent/detect possible suspicious situations if any [51].

These technical challenges can be considered as a subset of the first related work's requirements. In the proposed solution, first related work's requirements are preferred because they perfectly identify and cover the use case of e-voting.

The second related work indicates the reasons why we need blockchain technology for e-voting systems. The paper also gives a feasibility flowchart that explains when blockchain technology is suitable and useful for different use cases. There are various use cases that are explained and analyzed in detail in the paper.

Since this research is a survey, it aims to analyze the feasibility and suitability of a blockchain-based e-voting system. The authors do not specify a concrete system design and architecture.

The feasibility analysis includes the following topics: Blockchain Fundamentals, Social Aspects, Financial Aspects, Security, and Reliability, Comparison with Alternatives and SWOT Analysis. Traditional Elections, Naive e-voting Systems, and Blockchain-based e-voting are compared with respect to strengths, weaknesses, opportunities, and threats. This data is valuable for the proposed solution because the data is used for designing the architecture.

5.2. Current Blockchain-Based E-Voting Systems

A couple of projects use blockchain technology under the hood and let people vote transparently. These projects are listed below:

- Follow My Vote [21]
- Voatz [53]
- Polyas [42]
- Polys [43]
- Agora [2]

All these projects use different consensus algorithms, cryptography mechanisms, etc. However, all of them suffer from scalability issues. The details of consensus algorithms and scalability issues will be discussed in the following chapter *6. Discussion*.

CHAPTER 6

DISCUSSION

As mentioned in *3.3.1 Setup Phase*, the system prefers RSA encryption. After the RSA public-private key pair is generated for the system itself, the system will use its own RSA public key to encrypt eligible voters' sensitive data, such as SSN. Furthermore, we need to ensure that no one can read or edit this data. The system's RSA private key is the only way to decrypt this data. As explained earlier, the system uses VSS to distribute the system's RSA private key between trusted authorities. So our key has to be generated using asymmetric encryption such as RSA. Symmetric encryption such as AES, is not suitable for the proposed solution's use case. It is also important to note that small key sizes are one of the main reasons for choosing RSA over other asymmetric encryption mechanisms [6] [18].

In chapter *5. Related Works*, we described a potential weakness related to voter privacy. Instead of traditional databases, blockchains are used to store any data. That is why the system protects itself automatically. However, there may still be insecure parts, such as the creation order of RegisteredVoterBlockchain and PrivateKeyBlockchain. An attacker may create a relationship between the vote data and the registered voter. That is why the system shuffles the voters' Paillier private key in the keyring.

As explained in chapter *2. Background*, there are three different blockchain types. And, both Public and Private Blockchains have their own advantages and disadvantages. The proposed thesis prefers to use a hybrid approach. Since the system will work under the cloud and be accessible to anyone, we may assume it is a permissionless blockchain. On the other hand, the system uses a unique proof-based authentication as a consensus algorithm. It restricts access to the blockchain for several reasons, so we may assume that a part of our system is permissioned. Therefore, the thesis is designed to take advantage of both blockchain types.

The introduced consensus algorithm takes advantage of both blockchain types as well. To ensure consensus, we do not need to consume huge amounts of electricity, we do not need to solve a complex problem, or we do not need to wait a long time to validate

a block. Since a consensus algorithm that is proper for the problem domain can increase the performance of the blockchain.

The Figure 3.4 and Figure 3.5 shows the details of our blockchains.

In the system, the blockchains are chained to each other. Each blockchain relies on the other, and that makes the system fully reliable and secure. DDOS attacks are easy to detect thanks to the chain between blockchains and proof-based authentication using hash values. Hash of the username-password pair will be used to detect whether the request is coming from a real voter or an automated script.

In section *3.3. Proposed Solution and Design*, we reviewed the overall processes clearly. The system uses different blockchains instead of the shared database.

- to store eligible voter list in *3.3.1. Setup Phase*, EligibleVoterBlockchain
- to verify the voter's eligibility to register in *3.3.2. Registration Phase*, RegisteredVoterBlockchain
- to verify the voter's eligibility to vote in *3.3.3. Voting Phase*, RegisteredVoterBlockchain
- to store Paillier public key of voter in *3.3.2. Registration Phase*, RegisteredVoterBlockchain
- to store Paillier private key of voter in *3.3.2. Registration Phase*, PrivateKeyBlockchain

The proposed thesis ensures that nothing is changed and nothing will change during the whole election thanks to the most basic feature, immutable, of blockchain. Let us examine the list one by one:

EligibleVoterBlockchain: This blockchain stores the unique identifiers of eligible voters. The voters must verify their eligibility. Since the data is stored in blockchain and finalized version of Merkle root hash is shared to the World Wide Web, no one can add ineligible voters to our blockchain.

In *3.3.2. Registration Phase*, the users had to register to the system and then had to create real username, fake username, and password. While registering a new account, the system uses EligibleVoterBlockchain to verify the user's eligibility. When a new possible voter tries to register and types the SSN, the system will calculate the hash of SSN and ask this value to EligibleVoterBlockchain. The system will traverse from the last block to

the genesis block and check whether hashes are matched or not. If one of the hash values is matched, then it means that possible voter is actually an eligible voter, and he/she can create real username, fake username, and password.

RegisteredVoterBlockchain: This blockchain stores the hash values of username password pairs and can use to detect that if a voter is eligible to vote or not. The Paillier public keys of voters are stored in this blockchain.

In section *3.3.3. Voting Phase*, after the vote and encryption process is completed, the user has to log in to the system. The system uses RegisteredVoterBlockchain to do that. Instead of asking shared database, the verification is handled by blockchain, just like in the Registration Phase.

PrivateKeyBlockchain: This blockchain stores the voter's Paillier private key and can use to generate a shuffled keyring. The keyring will be used to decrypt each vote data. If the decryption process fails, then the system understands that the vote data is encrypted by an ineligible voter, and this makes it fraud data. So, the choice will not be evaluated during the counting phase.

VoteBlockchain: This blockchain stores the encrypted choice of the voter and the hash value of the vote data. Vote data's hash value will be used during *3.3.5. Individual Verifiability During Voting Phase*. Before adding a new block into the VoteBlockchain, the system verifies the voter's eligibility by using hash value of username-password. So that the system can easily detect DDOS attacks and rejects the vote data. Therefore, we ensure that our blockchain will not become garbage.

Besides technical aspects, accessibility and availability is also an important topic. The system will be available and accessible during the voting for anyone; personal mobile phone or computer can be used to register, vote, and check/verify the results. The proposed system is reachable to older people or people with disabilities, unlike kiosk machines or classic paper-based elections. We live in the age of technology, so they should be able to vote from anywhere.

Thanks to the advantage of the hybrid approach, a large-scale election is supported as explained in section *3.1. System Assumptions*. The voters can access the proposed system using any browser with a proper internet connection. In these types of elections, different countries can choose different voting systems. The proposed system is designed

for the plurality voting system. In this type of voting system, the winner is the one that is mostly chosen regardless of whether it has the majority of votes or not [50].

There are many different voting systems in the world, like First Past the Post, Single Transferable Vote, Additional Member System, Two-Round System, etc. First Past the Post is the name for the electoral system used to elect Members of Parliament, and it is used in the United States, India, etc. [50]. To adopt this system for various voting systems, the only thing to be updated is the setup phase. The remaining parts are about the system's security, expressly or implicitly.

In addition, this system can also be used for a referendum which is a vote on a question. A referendum can be started by a citizen or by the government. Let us clarify this feature with an example; Lawmakers can use this system to vote for a law. Or university students can vote on a list of specific questions at the same time. The candidates/choices will be simple 'Yes' and 'No' instead of candidates. However, a national election is chosen in this study, and the system is designed and implemented according to it.

CHAPTER 7

FUTURE WORK

To improve the efficiency and reliability of the system following options can be integrated into the current system:

- Better authentication methods can be used as Two-FA rather than email or SMS code verification, such as QR-Code, fingerprint, face recognition, etc.

Ordinary username-password pairs do not provide a sufficient security level. Authentication Technology, one of the most developed areas in the recent period, has great capabilities and various techniques to authenticate user. Instead of emal/SMS code verification, we may consider fingerprint, iris, or face recognition: Bio-metric technologies offer more reliable and efficient ways. As an alternative, QR-Code can provide more simple and effective way for authentication.

Since the main objective is not the authentication technology in this thesis, it will be enough to know that the system's security can be improved using these new techniques, and vulnerability to fake vote risks can be minimized.

- Different emergency protocols can be applied when the voter attempts to login with fake username, or when the system detect an anomaly using gesture recognition such as capture a photo, start a video record, start a voice record, share the location of the voter, etc.

As explained in the previous sections, the system is capable of detecting whether a vote is cast with free will or under pressure. If the voter login with a fake username, we can design an emergency protocol for such cases.

The listed recognition technologies in the first paragraph can also be used for different purposes. Let us consider the face recognition used in the proposed system as Two-Factor Authentication. Nowadays, gesture recognition is another popular research area, and this cool feature can be integrated into the proposed system. Assume that the voter accepted the terms of use and that the system can access the device's location, camera, and audio. When the voter performs a vote attempt if the system detects that the

voter looks nervous, frightened, or anxious using Gesture Recognition Algorithms. Then, the camera will immediately capture a photo, start a video record, or start a voice record to catch any valuable evidence against the attacker. Since the system receives the vote data from the voter via HTTPS connection, the system can access the IP address of the voter's device, and the system can share this IP address together with the exact location to the proper authority, such as The General Directorate of Security. These data can directly transfer into the proposed system, and can trigger the related emergency protocol easily.

- Two-Factor Authentication can be expanded to N-Factor Authentication using different methods.

As explained in the previous paragraphs, there are different types of recognition algorithms. Except from these options, the system can increase the performance and security level using the voter's expected location, Geolocation, during the voting phase.

The voter has to provide some information to register to the system, such as fake username, real username, password, contact information, etc. in the setup phase. Assume that there is another field called vote_geolocation. The voter can specify two different state/city/town and this data will be used for the n-th authentication method. The system will check the IP address of the request. And then check whether the request is coming from the specified location or not. If the location does not match, the system will not append a new block for this vote attempt. The first obvious advantage of such a system is that the system can protect the voter's login data against Brute-Force attacks. The second remarkable advantage of this feature is that the system can detect DDOS attacks easily and skips these malicious vote-casting requests.

- The encryption mechanisms can be reconsidered regarding the possible Quantum cyber-attacks.

Along with classical security issues, there is also an upcoming quantum computer problem for most cryptosystems considering many ongoing researches on the field [47]. As mentioned before, cryptosystems such as Paillier, RSA, and ElGamal are vulnerable to quantum attacks such as Shor's algorithm that makes it possible to execute prime number factorization, and, find discrete logarithms in polynomial time using a quantum computer [48].

Since security is obviously essential for an election, it is best if the idea is reinforced by using post-quantum (quantum-resistant) cryptosystems. For example, lattice-based cryptography algorithms such as NTRU and GGH are relatively strong against quantum attacks [47] [36] [15].

CHAPTER 8

CONCLUSION

In this thesis, the requirements of an e-voting system such as inalterability, non-reusability, eligibility, fairness, etc. are clearly introduced and deeply analyzed. After that, the depth of the functionality of the proposed solution is discussed based on: RSA and Paillier Homomorphic Cryptosystem for encryption-decryption operations and blockchain technology. However, any asymmetric encryption mechanism can be preferred instead of Paillier. The details of vote encryption are introduced in *3.3.4. Blockchain Phase*.

Blockchain technology is adopted to the proposed e-voting system with the help of Proof of Authentication Consensus Algorithm. Different consensus algorithms are well-suited for different use cases. These use cases are directly related to the type of network which can be either public or private network. This triangle (type of network - type of consensus algorithm - type of blockchain) should always be considered as a whole, and the system architecture should be designed for that. The reason why Proof of Authentication Consensus Algorithm was preferred for this system is briefly discussed in the thesis.

Event-driven Process Chain notation is used to model processes, analyze them, and identify potential improvements for the system. The EPC notation expresses the business processes through events and activities and is presented with Figure 3.2. In addition to this notation, each process is explained in detail in section *3.3. Proposed Solution and Design*.

The details of how the system satisfies the requirement *xiii. Election replay* are introduced in *3.3.8. Individual and Universal Verifiability After Results Announced*.

In the proposed thesis, a novel and unique consensus algorithm is designed and implemented considering the voter's privacy and the system's security. The details of our consensus algorithm are introduced in *3.3.4. Blockchain Phase*.

As we identified in chapter *7. Future Work*, Post-Quantum cryptography is gaining popularity, and Quantum computers are becoming accessible daily. So, we may consider

preferring lattice-based cryptography algorithms. Since asymmetrical cryptosystems such as RSA, ElGamal, or ECC are vulnerable to Post-Quantum cryptographic attacks.

In chapter *6. Discussion*, advantages of using a private network, different types of consensus algorithms, and performance criteria are discussed.

The following table indicates the relationship between the satisfied requirements and the system phases.

Table 8.1. An Overview of the System's Phases in terms of Requirements.

	i. Inalterability	ii. Non-reusability	iii. Eligibility	iv. Fairness	v. Individual verifiability	vi. Universal verifiability	vii. Privacy	viii. Authentication	ix. Integrity	x. Coercion-resistance	xi. Receipt-freeness	xii. Secrecy	xiii. Election replay
Setup Phase		+	+	+					+		+	+	
Registration Phase		+	+				+	+	+	+		+	
Voting Phase					+		+	+	+		+	+	
Blockchain Phase	+								+	+	+	+	
Individual Verifiability During Voting Phase					+								
Counting Phase	+			+			+		+				
Announcement Phase							+						
Individual and Universal Verifiability After Results Announced					+	+							+

These requirements are the indicators of the security, reliability, transparency, and privacy of the system. The table shows which requirements are covered in each step.

In conclusion to this study, a well-designed, secure, transparent, and reliable voting system, such as: inalterable, non-reusable, eligible, fair, etc., can be designed with the help of blockchain technology.

REFERENCES

[1] Academy Binance. What is a blockchain consensus algorithm? https://academy.binance.com/en/articles/what-is-a-blockchain-consensus-algorithm. Accessed: 2022-11-09.

[2] Agora Vote. Bringing voting systems into the digital age. https://www.agora.vote/. Accessed: 2022-11-09.

[3] Al-Kaabi, S. and S. Brahim Belhaouari (2019, 05). Methods toward enhancing rsa algorithm : A survey. *International Journal of Network Security Its Applications 11*, 53–70.

[4] Al-Saqqa, S. and S. Almajali (2020, 09). Blockchain technology consensus algorithms and applications: A survey. *International Journal of Interactive Mobile Technologies (iJIM) 14*.

[5] Bach, L. M., B. Mihaljevic, and M. Zagar (2018, May). Comparative analysis of blockchain consensus algorithms. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 1545–1550.

[6] Barker, E. (2020, May). Recommendation for key management: Part 1 - general.

[7] Bayer, D., S. Haber, and W. S. Stornetta (1993). Improving the efficiency and reliability of digital time-stamping. In R. Capocelli, A. De Santis, and U. Vaccaro (Eds.), *Sequences II*, New York, NY, pp. 329–334. Springer New York.

[8] Bendovschi, A. (2015). Cyber-attacks – trends, patterns and security countermeasures. *Procedia Economics and Finance 28*, 24–31. 7th International Conference On Financial Criminology 2015, 7th ICFC 2015, 13-14 April 2015,Wadham College, Oxford University, United Kingdom.

[9] Bitcoin Community. Genesis block. https://en.bitcoin.it/wiki/Genesis_block. Accessed: 2022-11-28.

[10] Chaudhry, N. and M. M. Yousaf (2018, Dec). Consensus algorithms in blockchain: Comparative analysis, challenges and opportunities. In *2018 12th International Conference on Open Source Systems and Technologies (ICOSST)*, pp. 54–63.

[11] Chor, B., S. Goldwasser, S. Micali, and B. Awerbuch (1985, Oct). Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pp. 383–395.

[12] CoinMarketCap. Total cryptocurrency market cap. https://coinmarketcap.com/charts/. Accessed: 2022-11-26.

[13] David Chaum. Computer systems established, maintained, and trusted by mutually suspicious groups. nakamotoinstitute.org/literature/computer-systems-by-mutually-suspicious-groups/. Accessed: 2022-11-28.

[14] Digicert. Ssl/tls/https. https://www.websecurity.digicert.com/security-topics/what-is-ssl-tls-https. Accessed: 2023-01-20.

[15] Ding, J. and J.-P. Tillich (2020, 01). *Post-Quantum Cryptography 11th International Conference, PQCrypto 2020, Paris, France, April 15–17, 2020, Proceedings: 11th International Conference, PQCrypto 2020, Paris, France, April 15–17, 2020, Proceedings*.

[16] Dylan Yaga, Peter Mell, N. R. K. S. (2018, October). Blockchain technology overview.

[17] El Makkaoui, K., A. Ezzati, and A. Beni-Hssane (2016). Securely adapt a paillier encryption scheme to protect the data confidentiality in the cloud environment. In *Proceedings of the International Conference on Big Data and Advanced Wireless Technologies*, BDAW '16, New York, NY, USA. Association for Computing Machinery.

[18] Federal Office for Information Security (BSI) (2022, January). Cryptographic mechanisms: Recommendations and key lengths.

[19] Feng, Q., D. He, S. Zeadally, M. K. Khan, and N. Kumar (2019). A survey on privacy protection in blockchain system. *Journal of Network and Computer Applications 126*, 45–58.

[20] Flask. Flask user guide. https://flask.palletsprojects.com/en/2.2.x/#user-s-guide. Accessed: 2022-11-26.

[21] Follow My Vote. Secure decentralized application development. https://followmyvote.com/. Accessed: 2022-11-09.

[22] Haber, S. and W. S. Stornetta (2004). How to time-stamp a digital document. *Journal of Cryptology 3*, 99–111.

[23] Howard, H., M. Schwarzkopf, A. Madhavapeddy, and J. Crowcroft (2015, jan). Raft refloated: Do we have consensus? *SIGOPS Oper. Syst. Rev. 49*(1), 12–21.

[24] Hu, V. (2021, December). Blockchain for access control systems.

[25] Huang, D., X. Ma, and S. Zhang (2020, Jan). Performance analysis of the raft consensus algorithm for private blockchains. *IEEE Transactions on Systems, Man, and Cybernetics: Systems 50*(1), 172–181.

[26] IBM. Private cloud. https://www.ibm.com/topics/private-cloud. Accessed: 2023-01-20.

[27] Lamport, L. (1998, may). The part-time parliament. *ACM Trans. Comput. Syst. 16*(2), 133–169.

[28] Leyla Tekin, Hüseyin Özgür, B. S. A. K. P. E. and S. Şahin (2018, 05). Edu-voting: An educational homomorphic e-voting system. *International Conference on Advanced Technologies, Computer Engineering and Science (ICATCES'18).*

[29] Liang, Y.-C. (2020, 01). *Blockchain for Dynamic Spectrum Management*, pp. 121–146.

[30] Lin, I.-C. and T.-C. Liao (2017, 09). A survey of blockchain security issues and challenges. *International Journal of Network Security 19*, 653–659.

[31] Lu, Y. (2018). Blockchain: A survey on functions, applications and open issues.

[32] Micciancio, D. and O. Regev (2009). *Lattice-based Cryptography*, pp. 147–191. Berlin, Heidelberg: Springer Berlin Heidelberg.

[33] Morabito, V. (2017, 01). *Business Innovation Through Blockchain*.

[34] Moraru, I., D. G. Andersen, and M. Kaminsky (2013). There is more consensus in egalitarian parliaments. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, New York, NY, USA, pp. 358–372. Association for Computing Machinery.

[35] Mustafa Karaçay. Blockchain based e-voting system. https://github.com/mustafakaracay/blockchain-based-e-voting-system. Accessed: 2023-01-20.

[36] Nejatollahi, H., N. Dutt, S. Ray, F. Regazzoni, I. Banerjee, and R. Cammarota (2019, jan). Post-quantum lattice-based cryptography implementations: A survey. *ACM Comput. Surv. 51*(6).

[37] NewsAPI. Queryparameter. https://newsapi.org/docs/authentication. Accessed: 2023-01-20.

[38] NIST (2015, August). Secure hash standard (shs).

[39] Pahlajani, S., A. Kshirsagar, and V. Pachghare (2019, April). Survey on private blockchain consensus algorithms. In *2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT)*, pp. 1–6.

[40] Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, Berlin, Heidelberg, pp. 223–238. Springer-Verlag.

[41] Pilkington, M. (2016). Chapter 11: Blockchain technology: principles and applications. In *Research Handbook on Digital Transformations*, Cheltenham, UK. Edward Elgar Publishing.

[42] Polyas. Online elections, nominations, and voting. https://polyas.com/. Accessed: 2022-11-09.

[43] Polys. Blockchain-based online voting. https://polys.me/. Accessed: 2022-11-09.

[44] Rivest, Ronald L. (Belmont, M. S. A. C. M. A. L. M. A. M. (1983, September). Cryptographic communications system and method.

[45] Rivest, R. L., A. Shamir, and D. A. Wagner (1996). Time-lock puzzles and timed-release crypto. Technical report, USA.

[46] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf. Accessed: 2022-11-26.

[47] ScienceDaily. Quantum computers news. https://www.sciencedaily.com/news/computers_math/quantum_computers/. Accessed: 2022-11-26.

[48] Shor, P. W. (1997, oct). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing 26*(5), 1484–1509.

[49] Sridokmai, T. and S. Prakancharoen (2015, Nov). The homomorphic other property of paillier cryptosystem. In *2015 International Conference on Science and Technology (TICST)*, pp. 356–359.

[50] The Electoral Reform Society. Types of voting system. www.electoral-reform.org.uk/voting-systems/types-of-voting-system. Accessed: 2022-11-09.

[51] Umut Can Çabuk, Eylül Adıgüzel, E. K. (2018, 03). A survey on feasibility and suitability of blockchain techniques for the e-voting systems. *International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE)*.

[52] VMware. Virtual machine. https://www.vmware.com/topics/glossary/content/virtual-machine.html. Accessed: 2023-01-20.

[53] Voatz. Voatz secure and convenient voting anywhere. https://voatz.com/. Accessed: 2022-11-09.

[54] Wood, D. D. (2014). Ethereum: A secure decentralised generalised transaction ledger.

[55] Yıldırım, M. and I. Mackie (2019, 12). Encouraging users to improve password security and memorability. *International Journal of Information Security 18*.

[56] Zheng, Z., S. Xie, H. Dai, X. Chen, and H. Wang (2017, June). An overview of blockchain technology: Architecture, consensus, and future trends. In *2017 IEEE International Congress on Big Data (BigData Congress)*, pp. 557–564.