Blockchain Based Election Architecture Using XRPL

by

Arnab Ghosh

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Sciences
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2021

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Blockchain is one of the most attractive emerging technologies in the 21st century. After its introduction in the field of cryptocurrency in the Bitcoin white paper by Satoshi Nakomoto in 2008, blockchain technology has found applicability in a wide range of fields including health, finance and inventory management.

Blockchain provides some attractive properties such as immutability and distributed consensus, as well as transparency, which, otherwise, is difficult to achieve. A large number of firms have developed their own version of blockchain, e.g., XRPL, Ethereum and Hyperledger. These systems have different properties and different consensus methods, protocols, etc. However, all of them have the same basic property of being a distributed immutable ledger.

Elections are the backbone of democracy and over time, election systems have evolved to meet our needs. Over the last two decades, there has been increasing research in the area of online election systems which are secure, trustworthy and affordable. Online election systems, besides being easily accessible to the voters, also reduce costs and environmental footprint.

The properties of blockchain, as mentioned earlier, make it an attractive choice for application in elections. Distributed consensus would improve public trust in the process and immutability would ensure that results are not tampered with.

Since all transactions in a blockchain are public and traceable, there are also certain concerns such as fairness of the election process and anonymity of the voter. However, including suitable cryptographic primitives, it is possible to overcome these challenges.

In this thesis, we analyze and evaluate existing blockchain based election architectures and then proceed to propose an architecture which meets most of the critical requirements for a secure and fair election. Thereafter, we implement a small test system as a proof of concept.

Acknowledgements

First, I would like to thank my supervisor, Prof. Anwar Hasan, who provided me unflinching support throughout the course of my thesis. He guided me through the analysis and proposal and encouraged me to go ahead with the implementation of this scheme. I am also thankful to Nik Bourgalis of Ripple, who helped me implement this scheme using the XRPL ledger.

I am also thankful to all professors who taught me the crytographic and coding concepts which were invaluable for the completion of this thesis. I would especially like to thank Prof. Mahesh Tripunitara, Prof. David Jao, Prof. Alfred Menezes and Prof. Guang Gong for teaching me concepts which were extremely useful during analysis and implementation of the system.

Apart from this, I am thankful to all Prof. Lucasz Golab, Prof. Mark Aagard, Prof. Anwar Hasan and Prof. Alfred Menezes for providing me insightful feedback on my thesis.

I would also like to thank my parents for their support and encouragement from a long distance. Last, but not the least, I would like to thank all my friends and colleagues who encouraged and supported me over the course of these two years.

Dedication

This is dedicated to my parents and also my teachers and friends who helped me complete this work.

Table of Contents

List of Figures

List of Tables

Acronyms

- AES - Advanced Encryption Standard
- AWS - Amazon Web Services
- DoS - Denial-of-Service
- EVM - Electronic Voting Machine
- FMV - Follow My Vote
- HSM - Hardware Secure Module
- MITM - Man-In-The-Middle
- NIST - National Institute of Standards and Technology (USA)
- PKE - Public Key Encryption
- PoW - Proof of Work
- PoS - Proof of Stake
- PBFT - Practical Byzantine Fault Tolerance
- RSA - Rivest-Shamir-Adleman
- SIKE - Supersingular Isogeny Key Encapsulation
- SSD - Solid State Drive
- SSS - Shamir's Secret Sharing Scheme
- VYV - Verify Your Vote
- XRPL - XRP Ledger
- ZKP - Zero Knowledge Proof
- zkSNARK - Zero Knowledge Succinct Non-Interactive Argument of Knowledge

Chapter 1

Introduction

1.1 Motivation

Elections are the pillar of any democratic system. Democracy is paramount to fairness of decision making in countries as well as organizations. The voters must have trust in the voting system and the voting process should be transparent, timely and easy to access.

To cater to these needs, a variety of voting systems have been developed over time for different settings and requirements. For example, in a small setting like a boardroom meeting, a voice vote or hand raise might be the quickest and most effective solution. For larger settings such as municipal elections, parliamentary elections and presidential elections, paper or machine based voting systems are used.

Paper ballot system is one of the oldest and widely used systems in democratic elections. The process is quite simple as people indicate their preferences on a ballot, deposit it in a box and the box is opened after the election is over and the votes are counted. However, it has certain disadvantages. The voters have to be physically present at the booth and their credentials need to be manually verified. There is also the option of mail-in votes but it requires access to the postal system and also secure transport of the ballot. Then there is the cost of printing the ballots and manpower costs for polling officers at the booth. Apart from this, there is the cost of security at the booth as well as at the ballot box storage centers, and also the cost of transporting the ballots securely. The manual counting of votes is also costly and time-consuming. Additionally, after the voter has voted, they have no way to know that their vote was correctly counted. They have to rely on the people handling the ballot boxes and ones who tally the votes. In paper based

elections, often, there have been allegations of forged identity, fake or misplaced ballots, booth takeovers, faulty counting, etc., all of which erode people's trust in democracy. In recent years, there has been a lot of media scrutiny on election turnouts and fairness of the election process. With the advent of better technologies, increase in population and busier lifestyle, governments in the 21st century have been looking at alternatives to replace the legacy process.

The first digital alternative to ballot based voting system was the use of EVM in the late 20th century. For EVM based voting, the authentication and voting procedure is similar to ballot voting, except that instead of indicating their choice on a ballot, voters have to press a button on the EVM to indicate their candidate choice. EVMs reduce the cost of printing ballots and also ease the counting process as each EVM internally tallies the votes. Many large democracies such as USA and India switched to using EVMs in general elections. However, there were numerous security concerns over EVMs. The EVMs which were connected to the internet could be hacked and tallies could be altered. Internal software errors and malfunctions were another concern. Besides, EVMs still had the same accessibility and security costs associated with it like paper based systems.

In 2000, the European Commission launched the CyberVote project for designing online election systems. There were also online voting systems which were piloted in various countries like France, Italy, Germany and more recently in Estonia. Apart from initial technical issues which plagued these systems, there was no mechanism for verifiability and the voting operations were controlled by a central authority.

With the advent of blockchain technology in 2008, there have been many suggestions about some of its unique properties being useful for conducting elections. Blockchain is a distributed ledger technology with properties such as immutability and transparency which are attractive for electoral purposes. However, there have also been concerns regarding privacy and coercion resistance, as well as fairness in terms of election results, which must be kept secret until the voting is over. There have also been general concerns about the security of machines which would be storing cryptographic keys or taking part in encryption and decryption. There is also the issue of scalability. The initial blockchains had very slow transaction speeds and confirmation time was as high as 10 minutes or more. However, over the years, blockchains like XRPL and Ethereum have enhanced hope that we could achieve transaction speeds and confirmation times which are feasible for purposes such as general elections.

Blockchain has already achieved a foothold in areas such as finance and healthcare and is poised to play a key role in transforming these sectors. If we can address the concerns regarding anonymity, scalability and fairness, blockchain can usher in a new era of easily

accessible, trustworthy and environment friendly election process for the 21st century.

1.2 Contributions

In this thesis, we propose a new XRPL blockchain based architecture for use in general elections. We provide detailed reasoning to show that the architecture achieves key properties for a secure and trustworthy general election process. We implement a small model of the system and perform a proof of concept experiment and evaluate our implementation in terms of desirable properties and security. We also provide some timing estimates to perceive feasibility of use.

1.3 Outline

First, we give an overview of the existing body of research pertaining to blockchain and blockchain based election schemes in Chapter 2. Then, in Chapter 3, we take a look at some useful cryptographic primitives which we use in our implementation. Our proposed new XRPL blockchain based voting architecture is explained in Chapter 4. Details of implementation including hardware requirements and algorithms implemented are provided in Chapter 5. In Chapter 6, we provide an analysis of system properties and results of timing experiments. Finally, in the concluding Chapter 7, we propose future research in relation to our new scheme and blockchain based election systems in general.

Chapter 2

Literature review

In this chapter, we will browse through some of the existing body of work regarding online blockchain based elections. In section 2.1, we review some works on formulating the requirements and expectations from such blockchain based systems. In section 2.2, we briefly discuss the concept of blockchain and its properties with special reference to application in an election architecture. In sections 2.3 and 2.4, we briefly summarize some proposed and existing online blockchain based voting schemes and analyze their advantages and shortcomings.

2.1 Requirements of blockchain based election system

Since elections are the backbone of any democratic system, it is essential for the process to be foolproof and the stakeholders to have confidence in the system. After the introduction of online election systems, there has been extensive research on the minimum requirements which must be met by such systems so that they are considered a significant improvement over the legacy counterparts. In this section, we review and analyze some of this literature.

InnoVote published a white paper [27] in 2016 where they analyze key requirements of any election software such as security, transparency, verifiability, accessibility and price. The paper also elaborates that maintaining ballot secrecy and protecting voters against coercion is essential for any voting scheme. There should also be efficient methods of detecting failures and corruption.

In their 2016 paper on Zerocoin based voting system, Tabatake et al. [32] mention some essential requirements such as anonymity, fairness and accuracy of counting votes.

Apart from this, coercion resistance and efficiency of the system are important parameters to judge the suitability of such a system. In addition, there can be other properties such as cancelling or modification of votes.

Agora [1] was one of the first companies to invest in developing a blockchain based election system that meets the needs of the future. In their white paper released in 2017, they provide detailed analysis of expectations from such a voting architecture. These include transparency, privacy, integrity, affordability and accessibility. They proceed to mention that the current voting systems consisting of EVM and ballot paper based systems do not meet some of these expectations and hence if we are able to take advantage of blockchain to meet these requirements, it could help usher in a new era of election technology. As an example, they mention the various cost, security and transparency issues with EVMs and their source code, and manpower and accessibility issues as well as corruption vulnerability of ballot paper based legacy voting systems. Apart from this, centralized control is a disadvantage of most legacy based voting systems, which we can remedy by using blockchain, which is inherently decentralized.

Horizon State published a white paper [29] in 2018 in which they visualize the implementation of a blockchain based election solution and discuss the major outstanding issues to be addressed while implementing such a system. First, the system needs to have trust and confidence of the voters and security against hackers. Ensuring anonymity, easy access, transparency and reduced cost are paramount while considering any new online election solution. The system should also have flexibility to adapt to different jurisdictions and different election procedures such as preferential voting and first past the post systems. To facilitate trust in the system, there must audit mechanisms and traceability of votes.

Voatz, the company which designed the first online blockchain based voting system [12] widely adopted in USA presidential elections, published a white paper about the challenges of such online systems and areas of improvement. One of the most important requirements is to ensure security of the device at the voter end as well as prevent any MITM attacks during the authentication process. Apart from this, identity proofing and binding as well as viability of transparent audits after the election are key requirements. The system should also ensure *perfect forward secrecy* so that if one voting session key is compromised, other votes will not be affected. In addition, the system should be robust to guard against DoS attacks.

After the overview of literature regarding necessary properties of blockchain based election systems, we can summarize them as follows:

1. **Authentication**: This means that only the voters who are registered in the electoral rolls and are eligible to vote shall be able to cast their votes. A person who is ineligible

to vote must not be able to cast a vote and a person who is eligible must be able to cast their vote following due process. In addition, an eligible voter should be able to cast only one valid vote. Apart from this, authentication process must prevent impersonation of the voter by any entity.

2. **Anonymity**: Anonymity ensures that voters are not tied to their votes. Except the voter herself, no one should be able to know which candidate the voter voted for. This shall be ensured during and after the election process. Anonymity is one of the key properties of any election system to protect the privacy of the voter.

3. **Accessibility**: For one, accessibility relates to the notion of voters being able to cast their votes seamlessly from any location. The system should also be easily accessible to voters with disabilities. However, in case of a blockchain based election system, this also entails that the voting procedure shall be easily accessible to the voters who may not be well acquainted with cryptographic concepts. Hence, the architecture must be designed in a way such that people with minimal knowledge should be able to not only cast their votes but also have trust in the system.

4. **Coercion Resistance**: A voting system should be designed such that a coercer should not be able to force the voter to vote for a certain candidate. In other words, there shall be no mechanism using which the voter can prove to a coercer that she voted for a particular candidate. This shall apply both during and after the election process.

5. **Affordability**: There are two aspects to affordability in a voting system. One is the affordability in terms of cost. The cost of implementation shall be lower or at least comparable to the cost of legacy voting implementations. The second aspect is the affordability in terms of time taken to conduct the election. The election process involves processing millions of votes. In a blockchain based system, the transaction rate is an important parameter which determines the viability of the system. The consensus process should be such that the votes are confirmed in reasonably quick time. The voter must not have to wait for along time to cast her vote and the voting process should be completed in a days time.

6. **Security**: Since the voting process involves processing of sensitive information such as voter credentials and candidate preferences, it is imperative that each communication, i.e., between the voter and the authentication server and between the voter and the voting network shall be secure, i.e., at least have a cryptographic security level of 128 bits or more (i.e. the same security level as RSA 3072). It is estimated

that with current computing power, breaking 128 bit security would take around a billion years, and as such, it should be secure in the medium term (next decade). There should be mechanisms to guard against MITM attacks so that eavesdroppers cannot get hold of any sensitive information.

7. **Integrity**: Integrity requires that a third party shall not be able to change the vote once it is cast by the voter. For voters to have trust in an election system, integrity must be assured.

8. **Verifiability**: The verifiability property is somewhat related to the integrity of the voting process. Voters should be able to verify that their vote was counted as cast. This property, however, has to be balanced with the coercion resistance property. Thus, even though a voter should be able to verify that her vote was counted unaltered, she should not be able to prove to a third party that she voted for a particular candidate.

9. **Fairness**: For an election process to be fair, the results of the election must not be known before the voting process is completed. Fairness is important because if running election results are available, it might influence the voters to vote for a particular candidate or it might influence a candidate to coerce the voters who have not yet voted.

Apart from this, there may be other desirable properties such as adaptability to different election systems such as first past the post or preferential voting system, and forgiveness, which essentially means the voter can change their vote before the election ends.

2.2 Blockchain: concept and properties

In this section, we discuss the concept of blockchain and review its properties especially in relation to applicability in voting systems.

2.2.1 Blockchain concept

Blockchain was first introduced by Satoshi Nakomoto in the Bitcoin white paper [17]. Blockchain is essentially a ledger which is maintained by a peer to peer network. There is no central authority and hence the ledger is maintained by a system of consensus.

Blockchain was initially conceptualized for payment transactions. The nodes in the network can initiate transactions which are sent over the network. Each node signs the transactions initiated by them. The transactions must contain a sender address, a receiver address and the amount transferred. Apart from this, a transaction might contain various other data such as transaction fee, type of transaction or even some miscellaneous data embedded in a string which can be stored in the ledger.

The details of every transaction is meant to be received by all nodes in the network. Then there is a consensus process, during the course of which a block of transactions is proposed to be added to the ledger, which is the blockchain. Once a consensus criteria is reached, the block is said to added to the blockchain. Each node maintains a copy of the ledger. If any node tries to add an invalid block to the ledger, other nodes will not build on it. During the consensus process, every node checks the validity of each transaction added in the proposed block. Hence, invalid transactions or double spends are not added to the ledger. The consensus mechanism should ensure that the power to propose blocks is distributed fairly among participating nodes.

The ledger can be conceptualized as a chain of data structures (blocks), each linked to the previous one by means of a hash. Each new block includes a hash of the previous block. This imparts the property of immutability to the blockchain. If an entity tries to modify an old block, it has to also modify all the succeeding blocks in the chain. In a distributed consensus system, the next proposer would not build on this modified chain. Thus, the invalid chain is *orphaned* as the succeeding proposers then build on the longest available chain (which is valid).

Inside a block, the transactions are stored in the form of hashes. These hashes are arranged in a Merkle tree structure. A Merkle tree is a binary tree which has a root node, several branch nodes and leaf nodes. Except the leaf nodes, every other node is derived from its two children. Since the transactions are stored in a binary tree structure rather than a linear structure, it reduces the complexity of searching a transaction in a block to $O(\log n)$, where n represents the number of transactions stored in the block. However, the cost of insertion into a Merkle tree is also $O(\log n)$, compared to constant time for a linear structure.

In a typical blockchain Merkle tree, the hashes of the transactions form the leaf nodes [18]. The next level of branch nodes consist of the hashes of the concatenation of hashes of two transactions. For example, let there be four transactions in a block- T_1, T_2, T_3 and T_4. The bottom level of the Merkle tree then consists of four nodes of value $H(T_1)$, $H(T_2)$, $H(T_3)$ and $H(T_4)$. The next level consists of two branch nodes having value $H_{b1} = H(H(T_1)||H(T_2))$ and $H_{b2} = H(H(T_3)||H(T_4))$. Finally, the Merkle root has the value

$H(H_{b1}||H_{b2})$, which is the hash of the combined values of the second level. The structure of blocks in a blockchain is illustrated in Figure 2.1.

Apart from the Merkle root and the hash of the previous block, the block might contain other data such as a timestamp, a nonce (a unique number which is used for mining in case of Bitcoin) and some other data.

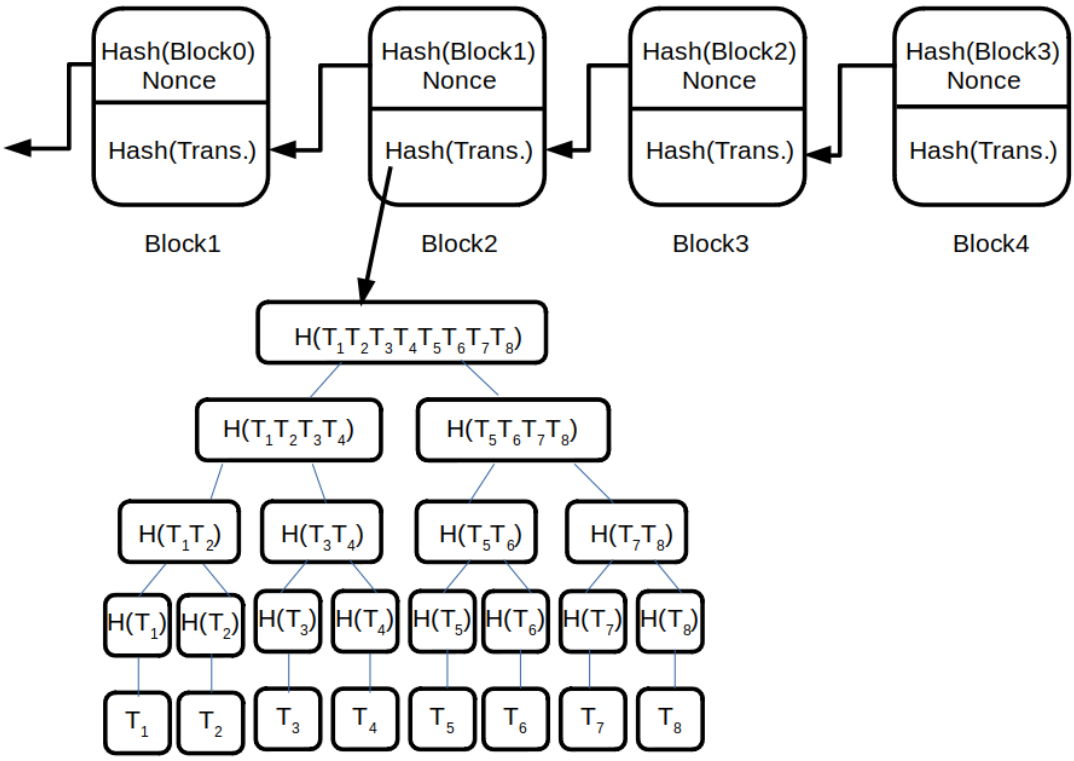

Figure 2.1: Structure of blocks in a blockchain

2.2.2 Types of Consensus

The process of adding new blocks to the blockchain is known as mining. Among the several nodes in the peer-to-peer network, one will have the privilege of proposing the next block. The process by which this is decided is known as the consensus mechanism. There are various consensus mechanisms [4] used in different blockchain networks. We will briefly

discuss some of them.

1. **Proof of Work**: Proof of Work is the consensus model which is used in the Bitcoin Blockchain. This method relies on the miner solving a difficult puzzle. The node which is the first to solve the puzzle has the privilege of proposing the next block. This difficult puzzle might be in the form of calculating the pre-image of a hash function where the digest has a certain pattern. Once the miner node finds the pre-image, it has to include the same in its proposed block so that it can be verified by other nodes in the peer to peer network. Since PoW model involves solving a difficult puzzle, which involves cost in the form of energy and hardware, the miners must have some incentive for solving this puzzle. In Bitcoin, this incentive is in the form of block reward and also transaction fee.

   We note that this system works as long as a majority of the nodes (in terms of computation power) in the network are honest. If a corrupt node has more than 50% of computation power in the network, it can easily add invalid blocks at will and thus break this system. This system also has the disadvantage of increased transaction confirmation time because it takes some time for the miners to solve the difficult hash puzzle before proposing the block.

2. **Proof of Stake**: Proof of Stake (PoS) consensus mechanism was first proposed in the Peercoin white paper in 2012 as an alternative to PoW. This essentially means that the nodes which have more stake in the system have the privilege of proposing more new blocks. Stake is measured in terms of the currency value held in the account and the time for which the currency has been held. For example, if an account holds 500 coins for 20 days, its stake value might be calculated as $500 * 20 = 10000$. In some networks, stake is defined as simply the percentage of total coins in the network held by the node. Thus, an account which holds 2% of all available cryptocoins in a network would be able to mine 2% of the transactions in the network.

   As with PoW, this system is also susceptible to a 51% attack. However, here, the attack works only if the attackers have majority stake in the system. This deters such attacks because attackers with majority stake have no incentive to break the system, which would deprecate the value of the currency.

   A popular modification of PoS is Delegated PoS (DPoS), where the nodes in the network elect a set of nodes known as delegates for validating a block. For validating the next block, voting is held again to choose another set of delegates. The value of

the vote of a node is determined by its stake, i.e., the number of coins it holds. DPoS was first implemented in BitShares blockchain in 2015.

3. **Proof of Authority**: Proof of Authority is a consensus mechanism normally used for non-cryptocurrency blockchain applications, e.g., in bank funds transfer or voting applications. This consensus mechanism is somewhat centralized, in the sense that only the nodes that have been authorized to be verifier nodes can propose new blocks to the ledger. Since there is no puzzle to solve, it is one of the fastest and energy efficient consensus mechanism. Of course, since the ledger is still public, every node in the network can still verify if the transactions included in the ledger are valid.

   The incentive to the validators is the value of their status as validators. If the validators mine invalid transactions in the blocks, their status as validators may be revoked or the network itself would lose legitimacy.

4. **Practical Byzantine Fault Tolerance (PBFT)**: PBFT is one of the consensus algorithms supported in Hyperledger blockchain. A primary node is randomly selected among the nodes in the network. When a node sends a transaction, it sends it for validation to the primary. The primary node then sends this to the other (secondary) nodes and receives a reply from each one of them. If 2/3rds of the secondaries agree to this, then a final commit message is sent by the primary node and the other nodes vote on it to reach consensus. There are also mechanisms for the secondary nodes to replace the primary in case of a failure or disconnection.

In addition to the above consensus mechanisms, there are other methods of consensus such as Proof of Capacity, Proof of Elapsed Time, Proof of Activity, etc., which are less commonly used.

2.2.3 Types of Blockchain

Blockchain is used for a wide variety of applications, which may entail different requirements. In this subsection, we give a brief overview into types of blockchain [20], depending on access and decentralization.

1. **Public Blockchain**: Public blockchain is used for most cryptocurrencies including Bitcoin and Ether. Anyone who has an internet connection can run a node on their machine and access the network. All the transaction and blockchain data is public and any node can take part in the consensus process. Thus, it is a fully decentralized

system, where each node maintains a copy of the ledger. The consensus mechanism in most of these blockchains are PoW or PoS. In such public networks, miners typically have to be given some incentive which might be in the form of a block reward or transaction fees.

2. **Private Blockchain**: A private or permissioned blockchain network is generally used for the requirements of a particular organization, e.g., inventory management, accounts management, etc. To join a permissioned blockchain, a node must have a valid permission token. Internally, a permissioned blockchain operates similar to a public blockchain. All the transactions are visible only to the nodes in the network. Generally, the consensus model used in these blockchains is some form of Proof of Authority. Since these blockchains are controlled by an organization, they are not distributed ledgers in the true sense.

3. **Hybrid Blockchain**: It is a mixture of private and public blockchain. Some of the data in these blockchains are public while other data may be visible only to certain designated nodes. The contents of some transactions may not be public, but the transaction itself can be verified. To view all the transactions or participate fully in the network, a user might need a permission token. Typically, these blockchains are used for government applications or voting. Hyperledger is an example of hybrid blockchain.

4. **Consortium Blockchain**: This is similar to a private blockchain but is managed by a group rather than a single entity. For example, a blockchain may be managed by a consortium of banks or real estate companies.

In the above types of blockchain, we note that speed and decentralization are a trade-off. The blockchains which have lesser decentralization have better speed and vice versa.

2.3 Notable election systems based on blockchain

In this section, we give an overview of blockchain based election systems proposed by various entities and analyze their properties, advantages and drawbacks.

First, we discuss the Agora [1] voting architecture. This implementation is based on the Bitcoin network. Their implementation uses five layers which communicate with each other. There is a permissioned blockchain called Bulletin Board which uses Skipchain architecture. Skipchain makes use of multiple hop forward and backward hash links in

the ledger, compared to only one hop backward hash links in legacy blockchains. This enables low power clients (such as mobile devices) to efficiently search transactions on the blockchain. The multi-hop links enable transaction search without the requirement to download all the blocks. A Byzantine consensus mechanism is used. The nodes involved in achieving the consensus are called cothority. These nodes are managed by Agora. The block proposer role is rotated among the cothority and once a block is proposed, two-thirds of the cothority need to reach consensus for it to be added to the blockchain. This permissioned blockchain layer interacts with the Catena schema which uses Bitcoin blockchain for logging snapshots of the bulletin board. These snapshots are logged using Bitcoin transactions. Apart from this arrangement, they have a Valeda layer, which are citizen auditor nodes, which validate the bulletin board by computing cryptographic proofs. The voters interact with the bulletin board blockchain using an application called Votapp. The ballots are embedded in the transactions.

The Innovote [27] system uses the BallotChain concept where the voting public key is included in the first block of the chain. Three copies of the ledger are maintained, one by election authority, one by external authorized servers and the other one by general public. The votes encrypted with the election public key are sent as transactions which are stored in the blockchain. The vote is valid only if it contains a signed token which is distributed to eligible voters by the election authority before the election. At the end, the election authority publishes the private key of the election on the blockchain and anyone can tally the results.

Votereum [33], as the name suggests, uses Ethereum blockchain for implementing an election system. The information about an election such as candidates, time frame, etc. are stored in a Smart Contract on Ethereum. This contract is then funded with Ether. Voters are given accounts in the system after verification by the authority. Once voters have an account, it is funded with ether to provide gas for the vote transaction. The voters send their vote as a transaction to the Smart Contract. After the end of the election, the Smart Contract tallies the votes and publishes the result. It also publishes a list of the blockchain addresses and the candidate who they voted for.

BitCongress [5] uses a Smart Contract Blockchain and a counterparty blockchain along with Bitcoin and an application named *AXIOMITY* to implement their online voting system. The Smart Contract contains information about the election such as name of candidates, timespan, rules, etc. Each new voter account has to submit a Bitcoin address to receive a 'vote' token from the BitCongress website. Each candidate has an address on the counterparty blockchain. If the voter wants to vote for a particular candidate, she can send the vote token to the candidate's account address on the counterparty blockchain. The election Smart Contract records the votes and tallies them at the end of the election.

The votes are then send back to the voters.

Murtaza et al. [16] propose an election scheme based on zkSNARKs, which are a form of non interactive ZKPs. In this system, the election authority, after verifying voter credentials, assigns an ID and public key to each voter, which is published in the blockchain. The voter then goes to the polling station and obtains a secret PIN which is used to derive a private key. Once the voter casts her vote, a burn address for her ID is created using her private key and some random numbers. This ID cannot be used again. A zero knowledge proof of the burn address, signed with a random private key, is sent to the address of the candidate, along with the corresponding public key. At the end of the election, the total transactions made to a particular address, or the total coins held by a particular address constitutes the vote tally for that address.

The proposed FollowMyVote (FMV) [8] voting system uses the BitShares blockchain, using DPoS consensus. Voters are provided with tokens which can be used for voting. Once verified by the authority, the voter sends a blinded token which is signed by the authority. The signed token is required to cast a valid vote. The vote is embedded in transactions on the BitShares blockchain.

DABSTERS [14] voting architecture uses a blockchain with PBFT and a Blind signature consensus mechanism. After due verification of credentials, the names of eligible voters and ID numbers are posted on the blockchain and verified. During voting, the voter sends a signed encrypted vote to the authority which checks eligibility and then blinds this signed vote. The voter then sends this vote to a peer node in the network and the transaction is recorded in the blockchain. At the end of the election process, the tallying authority decrypts and counts the votes to announce the election result.

The Verify Your Vote (VYV) [15] model suggests an online voting system on the Ethereum blockchain. There is a public bulletin board where the election authority publishes some cryptographic parameters and also sets some timings such as begin and end time of voting and counting. The authority then constructs the ballots and generates a ballot number which is the combination of a germ and a random number encrypted with a public key. This is then sent to the tallying authority. The voter obtains her authentication parameters after due verification by the registering authority. She encrypts this with the election authority's public key, and signs it and sends it to the authority which authenticates it. Now the tallying authority sends a ballot encrypted with the voter's public key to the voter. The voter then decrypts the ballot, encrypts it with a public key meant for the candidate of her choice and sends it to the tallying authority by means of a blockchain transaction. The tallying authority decrypts the ballot and counts the votes at the end of the election.

TIVI [26] is a blockchain based e-voting system developed by Smartmatic. The verification of voter is done by using facial images from smartphones and comparing it with photo IDs stored in the database. The election body generates a private and public key for the election. The voters encrypt their votes using the public key. The vote is signed by the voter and sent as a transaction in the network. A blockchain based network is used so that the transactions are time stamped. The decryption is done using the election private key, which is held in a HSM until the end of the election.

Hardwick et al. [24] propose an online voting system using blockchain as the ballot box. Each voter has a cryptographic public key. They also have a voting client installed on their device which is used to access the network. The central authority, which has access to a database storing voter identification data, verifies the voter credentials and assigns them a signed token. This signed token is essentially the authority's signature on the voter's public key and commitment to candidate choice. A blinding signature scheme is used for this purpose. The authority also publishes its signature verification key, candidate names and other election parameters in the first block of the blockchain. The ballot is a combination of the valid token, a commitment towards the choice of candidate and the voter public key. Each ballot has a unique ID. There is also provision for an alteration ballot in case the voter wants to change their vote. During counting, the voters broadcast a message consisting of their ballot ID, commitment opening value and a signed version of both these values combined. Using this opening value, the vote is revealed.

After having a brief overview of these proposed blockchain based election schemes, we are now ready to tabulate the desirable and necessary features present or absent in these schemes. This information is presented in Table 2.1.

2.4 Implementation of blockchain based voting schemes in elections

As we analyzed in the previous section, in the past decade, several entities and individuals have some up with a variety of blockchain based schemes, aiming to achieve most of the required and desirable properties required for conducting democratic elections. Some of those schemes have been implemented as pilot projects. In this section, we give an overview of some such blockchain based election schemes which have been implemented for elections in various countries.

Scheme	Anonymity	Integrity	Accessibility	Coercion resist.	Verifiability	Fairness
Agora	Yes	Yes	Yes	No	Yes	Yes
Innovote	Yes	Yes	Yes	No	Yes	Yes
Votereum	Yes	Yes	Yes	No	Unclear	Yes
BitCongress	Yes	Yes	Yes	No	Yes	No
FMV	Yes	Unclear	Yes	No	Unclear	No
VYV	Yes	Unclear	Yes	No	Yes	Yes
Murtaza	Yes	Yes	Yes	Yes	No	No
DABSTERS	Yes	Yes	Yes	No	Yes	Yes
TIVI	Yes	Yes	Yes	No	No	Unclear
Hardwick	Unclear	Yes	Yes	Unclear	Yes	Yes

Table 2.1: Properties of proposed blockchain based voting schemes

The Agora election architecture was run as a proof of concept in the Western District of Sierra Leone in presidential elections in 2018. Although the official voting did not take place on the Agora architecture, the company did a parallel record of votes on its blockchain and matched the tally with the official outcome.

The blockchain based election system by Voatz was used in West Virginia in the USA for remote voting availed by troops in the federal elections. Voatz outsourced the Mobile Device Security check to a third party, Zimperium. The messages between the smartphone and the election network is encrypted using AES/GCM. The backbone of the Voatz voting system is the Hyperledger permissioned blockchain. The encrypted votes are sent as transactions on the network and stored in the blockchain. For authorization, Voatz uses a third party app. However, the paper by Specter et al. [28] provided a critique of the Voatz application. The authors claimed that they could disable malware detection on the client side and also that the storage of the private user PIN was insecure. Apart from this, they also claimed that the system was prone to MITM attacks. These claims were, however, refuted by Voatz.

Luxoft [13] designed a blockchain based system called e-vote for the Swiss city of Zug. The platform is based on Hyperledger permissioned blockchain. The ID registration system was implemented through Ethereum blockchain. The system provides for anonymous voting and also third party audits. A feedback survey after the election suggested that around 80% of voters were satisfied with the election process.

VOTEFOR developed a blockchain based solution for e-voting in the city of Tsukuba [2] in Japan. This was based on the Ethereum blockchain and implemented on the AWS

cloud. The authentication data of the voters is also stored in the AWS cloud and the authorization is done by a local government representative.

In 2019, parliament elections in Moscow was held using a blockchain based e-voting system. It was based on the Ethereum blockchain and used the El-Gamal algorithm for encryption of votes. However, the system did not work as intended because it used 256 bit El-Gamal keys 3 times in succession, which did not provide the required security level. Later, the organizers updated the key to 1024 bits. However, a paper by Gaudry et al. [10] showed how the system still had security loopholes.

Recently, South Korea has approved a project to explore a blockchain based election solution by running a pilot trial with around 10 million people. Apart from this, the election commission in India, in collaboration with technological universities have started on a project for implementing blockchain voting for general election applications.

2.5 Remarks

Overall, the response to implementation of blockchain based elections has been mixed. The obvious advantage of such systems is the ease of voting because voters can vote from anywhere and their physical presence at the booth is not required. The immutability of blockchain also provides an attractive advantage as votes stored in a blockchain cannot be tampered with. However, there have been several recent papers, one of them famously authored by Rivest [19], suggesting that it is still too early for the adoption of blockchain based systems for general elections. The main concerns regarding blockchain based, or any form of online election, is the security of the voter's device. There are also concerns that software errors may not be accounted for in such implementations and also that the manufacturers of such systems may take advantage of their technical expertise to manipulate election results. These concerns need to be addressed adequately before we can envisage the widespread adoption of blockchain based election networks.

Chapter 3

Cryptographic concepts

In this chapter, we review relevant cryptographic concepts which will help us understand their implementation in the voting system architecture. These concepts are important for obtaining the desirable traits of a voting system such as anonymity, fairness and verifiability. We will be visiting short mathematical explanations of these concepts.

3.1 Groups, rings and finite fields

In this section, we briefly go through some number theory concepts of groups, rings, fields and finite fields [25] [11]. The concept of finite fields is important for understanding many cryptographic algorithms.

A group includes a set of elements along with an arithmetic operation such as addition (+) or multiplication (*) such that the set is closed under this operation, and also possesses the properties of associativity, inverse and identity. The definition of addition or multiplication with respect to a group of elements might differ from the conventional notion of these operations. For our description, let us assume that the group operation is denoted by * (where * does not necessarily mean multiplication). Then the properties of a group are defined as follows:

1. Associativity: For elements a, b and c in the group, $a * (b * c) = (a * b) * c$.
2. Identity: There must exist an identity element i such that for every element a in G, $a * i = i * a = a$.
3. Inverse: For every element a in G, there exists an inverse a^{-1}, such that $a * a^{-1} = a^{-1} * a = i$.

If, for group elements a and b, $a * b = b * a$, then that group is called *abelian group*. For example, the group of integers under addition forms an abelian group.

A ring is an abelian group which is closed under two operations. Let us say these operations are $\cdot$ and $+$. Then in addition to the properties of a group, a ring must also possess the distributive property. Thus, for elements a, b and c in the ring,

$$a \cdot (b + c) = a \cdot b + a \cdot c$$

Also, we must note that multiplicative inverses may not exist in a ring. So, we notice from our previous example, that the group of integers with the operations $+$ and $\cdot$ also form a ring.

Now, we explore the concept of fields. A field F is an abelian group under an operation (say $+$), such that it is also closed under another operation (say *), and $F \setminus \{i\}$ forms an abelian group under *, where i is the identity element of F under the $+$ operation. Here, we notice that the group of integers under the operations $+$ and * do not form a field. This is because multiplicative inverses of any integer greater than 1 or less than -1 is not contained in the set of integers. However, the set of integers modulo N, where N is a prime number, forms a field under $+$ and *. Also, since this field contains a finite number of elements, it is a finite field. This finite field of integers modulo N is used in many cryptographic algorithms as we shall see later.

3.2 Shamir's secret sharing scheme

The SSS algorithm was proposed by Adi Shamir [23] in 1979. In this scheme, there is a secret and there are a number of parties who are interested in knowing this secret. Each party is provided with a part of the secret such that it is infeasible to derive the original secret just from this one part. However, once a certain number of parties share their secrets with each other, it is easy to compute the original secret.

For an initial visualization, we may recall the concept of obtaining the equation of a curve from given points. To derive the equation of an n-degree curve, we require the coordinates of $n + 1$ points on the curve. The equation of an n-degree curve is, essentially, an n-degree polynomial. We use a similar concept for SSS.

Let us assume that there is a secret s. We need to divide this secret into n parts with the condition that if k out of n parties (or more) collaborate, the secret s can be easily recovered. However, it is infeasible to recover the secret if fewer than k parties collaborate.

For this, we need to encode the secret into a polynomial of degree $k-1$. To do this, we uniformly choose $k-1$ random numbers from a finite field. We choose this finite field to be a field of integers modulo a random prime number, say N.

Let us assume that the $k-1$ random numbers are denoted by $d_1, d_2, \cdots, d_{k-1}$. We use these numbers as the coefficients of the polynomial. Thus,

$$D(x) = s + d_1 x + d_2 x^2 + \cdots + d_{k-1} x^{k-1}$$

Now, we need n portions of the secret to distribute among n stakeholders. So, we put in the value of x as $1, 2, 3, \cdots, n$ and the resulting $D(x)$ values are the portions of the secret to be shared. Thus:

$$\begin{aligned} S_1 &= D(1) \bmod N \\ S_2 &= D(2) \bmod N \\ &\cdot \\ &\cdot \\ S_n &= D(n) \bmod N. \end{aligned}$$

Each stakeholder receives their portion of the secret as (X, Y) where X is a number between 1 to n and $Y{=}S_X$.

Now, we describe the reconstruction of the secret from the parts. Let us assume k stakeholders shared their secret among themselves. The secrets can be denoted by (x_i, S_i) where i varies from 1 to k. We can reconstruct the original polynomial using the Lagrange interpolation. Using this method, the Lagrange basis polynomials can be calculated as follows:

$$l_i(x) = \prod_{i=1}^{k} \frac{x - x_j}{x_i - x_j},$$

where i varies from 1 to k.

Finally, we have the reconstructed polynomial as follows:

$$D(x) = \sum_{i=1}^{k} S_i * l_i$$

The constant term for this polynomial is the secret that was originally shared. Clearly, this reconstruction cannot be done if fewer than k stakeholders collaborate. This completes the short description of SSS scheme.

3.3 RSA and blind signatures

3.3.1 RSA signature

RSA [22] is one of the oldest asymmetric key cryptographic scheme. RSA is used for both encryption and signature purposes. In this section, we limit our discussion to a short description of the RSA signature scheme. The RSA public key encryption scheme uses a similar concept.

RSA relies on the hardness of the factorization problem in number theory. Thus, a number N is chosen such that it is a product of two primes p and q. The security depends on number of bits in N. These days, N is typically 2048 or 3096 bits for a secure application.

Now, we choose two numbers d and e in the following manner:

- Compute $\phi(N) = (p-1)(q-1)$.
- Choose e between 1 and $\phi(N)$ such that $\gcd(e, \phi(N)) = 1$.
- Calculate $d = e^{-1} \bmod \phi(N)$. This can be done using the Extended Euclidean algorithm.

Now, for any number, say m, we notice that:

$$(m^d)^e \equiv m \bmod N.$$

Here, d is the private key and (e, N) is the public key.

Suppose we have a plaintext message m. To sign this message, we first compute the hash of this message $h(m)$. Then the RSA signature of this message is

$$s(m) = (h(m))^d \bmod N.$$

Now we proceed to the verification side. Given message m and signature $s(m)$, we need to verify that it is indeed the signature of m. Like the signing side, we compute the hash of the message $h(m)$. We also have the public key (e, N). Hence, we can now verify that

$$(s(m))^e \bmod N = h(m).$$

3.3.2 Blind signatures

In this section, we explore blind signatures [35] with special reference to the RSA cryptosystem. This is especially useful for maintaining anonymity in a token based system. The blind signature is nothing but a routine RSA signature on a blinded message.

Let us say a client requires the signature of the server on a message m; however, the server should not know the value m. The algorithm proceeds as follows:

The client obtains the RSA public key of the server (e, N). Then, she uniformly chooses a value r between 0 and N such that r and N are co-prime. The client then calculates $mr^e \bmod N = m'$ and sends m' to the server. The server routinely signs the message and returns $(m')^d \bmod N = s'$ to the client. Now, the client computes the value of $s = s'r^{-1} \bmod N$. s is the signature of the server on the message m. We notice that the server does not have a feasible way to know the value m.

3.4 Elliptic curves

Elliptic curves over finite fields are used widely in public key cryptography. Some cryptographic schemes based on elliptic curve isogenies are known to be quantum computer secure, unlike other widely used public key cryptosystems such as RSA. In this section, we briefly explore elliptic curves, Montgomery curves and isogenies.

An elliptic curve is described using an equation:

$$Y^2 = X^3 + AX + B.$$

The equation is defined over a finite field. Here, the polynomial on the right hand side must have distinct roots. Hence, we have the condition:

$$4A^3 + 27B^2 \neq 0.$$

The elliptic curve [25] is also said to include a point at infinity. Since this curve is a polynomial, it can, if defined over real numbers, be plotted on a graph (except the point at infinity).

The points on an elliptic curve form a group. The point at infinity forms the identity element in this group under the addition operation. Two points on an elliptic curve can be added using some well defined rules.

To add two points P and Q on an elliptic curve, we join the two points using a straight line. This line will intersect the curve at another point, say R'. The reflection of this point R' along the x axis yields the point R, which is the addition of points P and Q. Algebraically, the point $R(x_R, y_R)$ translates to the following:

$$x_R = m^2 - x_P - x_Q,$$
$$y_R = m(x_P - x_Q) - y_P,$$

where,

$$m = \frac{y_P - y_Q}{x_P - x_Q}.$$

Also, negative of P is defined as the reflection of P along the X axis. Thus, we notice that when we add P to $-P$, the straight line does not intersect the elliptic curve at any point on the graph. Hence $P + (-P)$ is nothing but the point at infinity, which, indeed, is the identity element.

The multiplication of a point on a elliptic curve can be defined as repeated addition. Thus,

$$kP = \underbrace{P + P + ... + P}_{k\,times}$$

Given two points P and Q on the elliptic curve, and given $P = kQ$, where k is some integer, the problem of finding k is known to be hard. This is the basis of security for many elliptic curve based cryptosystems.

A Montogomery curve [7] is a type of elliptic curve. It is defined as follows: We take two elements A and B in a finite field, such that $B(A^2 - 4) \neq 0$. Then the Montgomery curve equation is given as:

$$By^2 = x^3 + Ax^2 + x.$$

The j-invariant of this curve is defined as:

$$j(C) = \frac{256(A^2 - 3)^3}{A^2 - 4}.$$

We also define m-torsion set of the elliptic curve $C[m]$ as the set of points on the curve such that $mP = O$ where O is the point at infinity.

Now, we briefly visit the topic of elliptic curve isogenies [31]. Suppose C_1 and C_2 are two elliptic curves over the same finite field. An isogeny, defined over the finite field, is a function which rationally maps each point in C_1 to a point in C_2. Thus, if there is an isogeny between two elliptic curves, they must have the same number of points. The point at infinity of C_1 is mapped to the point at infinity of C_2.

The isogeny is generally expressed as a ratio of polynomials.

$$\phi = \left(\frac{p(x)}{q(x)}, \frac{r(x)}{t(x)} * y\right),$$

where $p(x)$ and $q(x)$ as well as $r(x)$ and $t(x)$ are relatively prime polynomials and ϕ is a mapping from C_1 to C_2.The maximum of the degrees of the polynomials is also referred to as the degree of the isogeny.

For example, let there be two elliptic curves [9]:

$$E_1 : y^2 = x^3 + 1132x + 278$$
$$E_2 = y^2 = x^3 + 500x + 1005$$

defined over F_{2003}. Both these curves have 1956 points. Then the following mapping ϕ from E_1 to E_2 is a degree-2 isogeny:

$$\phi(x, y) = \left(\frac{x^2 + 301x + 527}{x + 301}, \frac{x^2 + 602x + 1942}{x^2 + 602x + 466}y\right)$$

3.5 SIKE public key algorithm

In this section, we delve briefly into the SIKE public key encryption scheme [7]. We do not aim to go into the detail of how it works, but gain a conceptual understanding of the primitives used to construct the cryptosystem.

SIKE is an elliptic curve isogeny based cryptosystem that uses a Montogomery supersingular elliptic curve as the public starting curve. The equation of this curve is given by:

$$C/F_{p^2} : y^2 = x^3 + 6x^2 + x.$$

Here, p is a prime. We also choose two positive integers a and b such that $p = 2^a 3^b - 1$. Apart from this, we also choose three points A_1, A_2 and A_3 in the torsion set $C[2^a]$ and

three points B_1, B_2 and B_3 in the torsion set $C[3^b]$. Here, $B_3 = B_1 - B_2$ and $A_3 = A_1 - A_2$. The x coordinates of these points, along with a, b and the equation of the elliptic curve form the public parameters.

The secret keys are integers. The public key is computed using the secret key and the public parameters using l-degree isogeny where $l \in \{2, 3\}$. For the SIKE public key encryption system, there are two secret keys corresponding to $l = 2$ and $l = 3$. The one corresponding to $l = 2$ is used for randomness.

The SIKE scheme uses three functions:

1. Generate: Generates pk and sk_3 corresponding to $l = 3$.
2. Encrypt: The inputs are pk and message to be encrypted, say m. Generates sk_2 corresponding to $l = 2$ and computes ciphertext (c_0, c_1).
3. Decrypt: The inputs are sk_3 and the ciphertext. Computes the plaintext m.

3.6 Cryptographic accumulators

Cryptographic accumulators [3] [6], first proposed by Benaloh and De Mare, are structures which are used to establish proof of membership in a particular set. An accumulator is essentially a value obtained by accumulating the members of a set. Accumulators can be static or dynamic. Dynamic accumulators allow us to add or delete values from the accumulated set.

Every accumulated value has a witness, which can be used to prove membership. We will illustrate this using an RSA accumulator.

Let g be the generator of a group of unknown order. An RSA accumulator is a value formed by raising g to the power of the product of unique primes which represent the members of the set. Thus, we need a hash function $H(x)$ to map each member to unique primes. Now let us add a number n to the accumulator a. The following procedure is followed:

1. $m = H(n)$.
2. New accumulator value $a' = a^m$.
3. Return witness $w = a'^{1/m}$.

Here, we see that given the number n and witness w, along with the accumulation value a', which is public, anyone can verify that n belongs to the accumulation set by ensuring that $a' = w^m$, where $m = H(n)$. This is the basic concept of accumulator.

Chapter 4

Architecture

In this chapter, we present the architecture for our blockchain based voting scheme. While designing this architecture, we keep in mind the various requirements of an online voting scheme. We also take into consideration the feasibility of the implementation in terms of cost and time, as well as the crucial aspect of developing public trust in the system, which is essential for any new scheme to be widely adopted. Apart from this, we ensure that the system is accessible to all voters, even those who may not be familiar with blockchain and cryptographic concepts.

As with any voting system, we have different stakeholders including voters, registration authority, counting authority, etc. In section 4.1, we give a brief overview of how these roles are allocated in the blockchain based voting system. In addition, we have some actions which will be performed before the voting period starts. These are summarized in section 4.2. Then we enter the vote casting phase which is explained in section 4.3 and finally, we have the counting of votes and declaration of the winner, which is discussed in section 4.4. To ensure a smooth voting experience, our aim is to reduce the time consumed during the vote casting and counting period and accommodate most of the time consuming work in the pre-voting phase. We briefly discuss these three phases of our scheme later in this chapter. Besides, our architecture is implemented on top of Ripple's XRPL blockchain. Hence, our architecture also requires a good understanding of the architecture of the XRPL blockchain, which is discussed briefly in the final section.

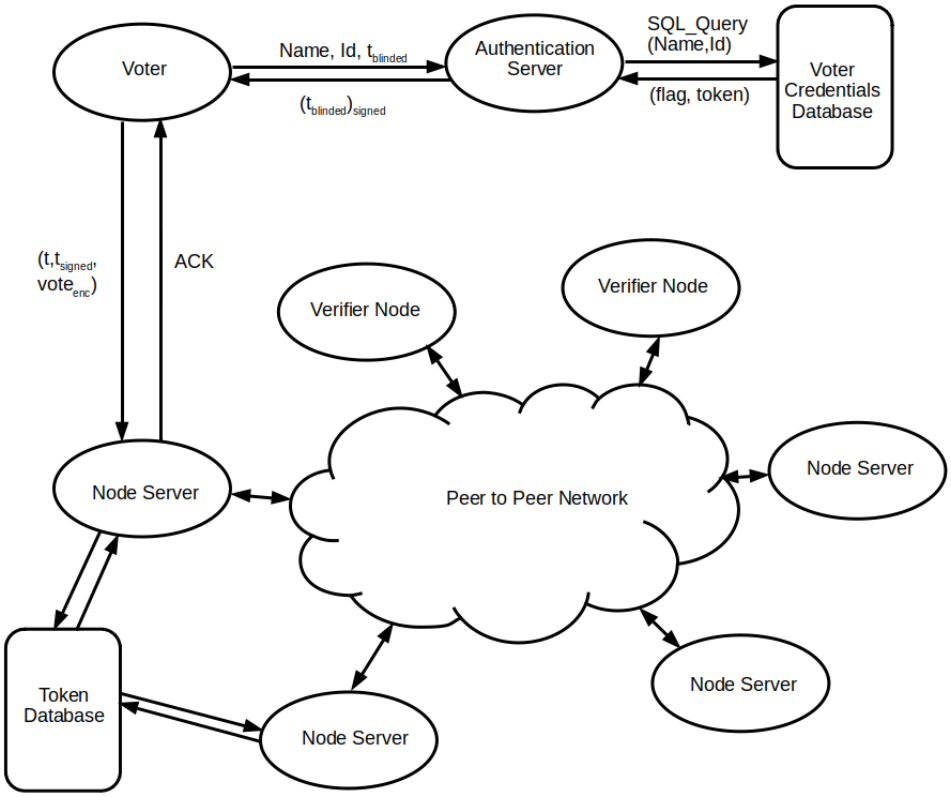

Figure 4.1: Architecture schematic

4.1 Stakeholders

In this section, we get an insight into the various stakeholders in our blockchain based election scheme. Just as in a legacy election system, we have voters, candidates and some independent parties for the smooth conduct of elections.

First, we have the voter, who is the most important stakeholder. The voter, as shown in Figure 4.1, is represented by a device, which sends an encrypted vote to a node in the blockchain. Before performing this, the voter also has to obtain authentication in the form of a signed token from the authentication server.

Next, we discuss the various neutral stakeholders. Our proposed architecture includes an authentication server which sends signed tokens to the voters after verifying their pres-

ence in the electoral rolls. Apart from this, the architecture includes *nodes*, each of which receives encrypted votes from voters and transmits it in the form of a transaction which is recorded in the blockchain. The nodes maintain a token database to ensure that the same token is not used more than once. The architecture also has neutral verifiers who share the private key of the vote and release it once the elections are over.

Finally, we have the candidates, who perhaps, have the highest stake in the result of the elections. To maintain the neutrality of the election process, the candidates play no role in the verification process. However, they do participate in the election process as voters, and are notified about the results after the election is over.

4.2 Pre-voting phase

We know that in legacy election process, certain important formalities are completed during the pre-voting phase. The authentication part includes registration of new voters in a constituency, verification of electoral rolls, issue of voter ID cards, etc. Apart from this, the polling and returning officers are appointed and the voting machines or ballot boxes are kept ready. We follow a similar mechanism for our online voting scheme. The following subsections shed light on the pre-election procedures in our voting scheme. The authentication message sequence is displayed in Figure 4.2.

4.2.1 Registration

The registration phase is somewhat similar to the legacy election process. The voters register themselves with the authority who assigns them a voter ID. The voter ID along with name and other particulars can be then stored in a SQL database. Here, we must mention that a disadvantage of using this scheme is the reliance on the central authority, which is somewhat in contradiction to what blockchain stands for, which is distributed consensus. An alternative here is to use a cryptographic accumulator scheme. The voters can register themselves with the accumulator and after all voters are registered, they can be issued witnesses using which they can authenticate themselves during the voting process.

The authentication database stores four variables – the name of the voter (say v_name), an ID number (say v_id), a flag (say v_flag) which indicates whether this voter has already asked for authentication or not, and a token number (say v_token). v_token is set to 0 initially and updated when the voter requests for a token.

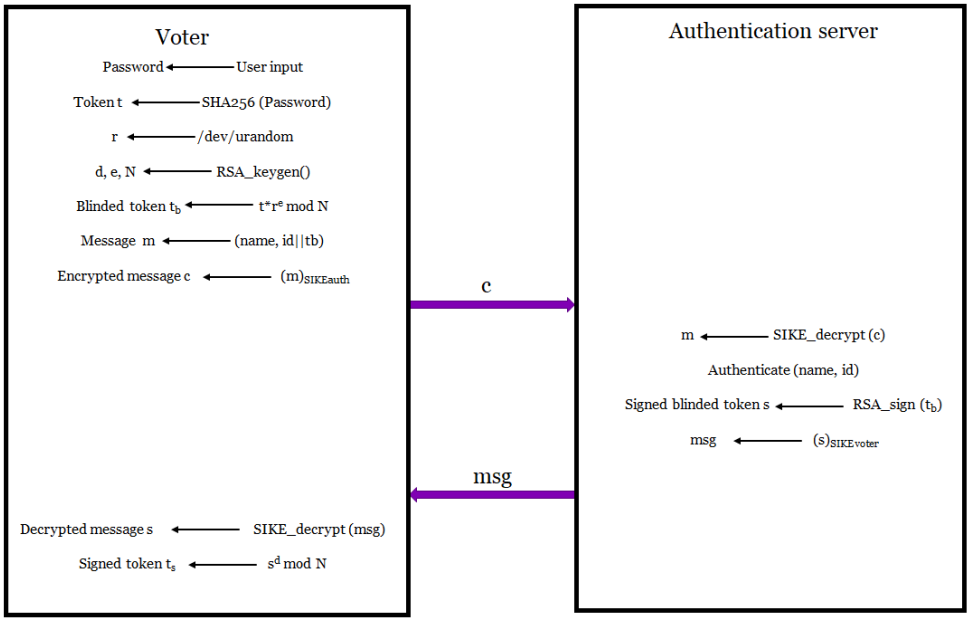

Figure 4.2: Authentication message sequence

4.2.2 Authentication

In this section, we briefly discuss about authentication of voters before they can cast their votes. An important consideration is that each voter should be able to cast her vote only once. To ensure this, we implement the concept of authentication tokens. The voter must prove to the token issuing server that she is on the electoral rolls and has not asked for a token already, to prevent double vote.

Thus, the voter sends her credentials to the server which queries the database. She also sends a blinded token to be signed by the server. This blinded token is a number selected by the voter, blinded using the RSA blinding scheme. The blinding ensures that the token cannot be tracked to the voter.

The server queries the SQL database to ensure that the voter is registered in the electoral rolls and that she has not received a signed token already. The server then proceeds to sign the blinded token and send it back to the voter. It also includes a note of the blinded token in the SQL database. In case the voter accidentally loses her signed token, she can send another authentication request to the server and the server will send

back the blinded token along with its signed version to the voter. The communication between the authentication server and the voter is shown in Figure 4.1

4.2.3 Key generation

Now, we consider the different public and private keys which are to be generated before the start of the voting phase. It is essential that all communication between the voter and the server, as well as between the voter and the node in the blockchain network, are encrypted. Apart from this, we use SSS scheme at the counting side, so it is imperative to ensure that the key sharing also proceeds in a secure environment.

First, the authentication server generates a keypair for communication with the voter. Here, we use an elliptic curve isogeny based PKE scheme called *SIKE* due to its small key size. We can also use other PKE schemes such as RSA. The public key is used by the voter to encrypt the authentication information and the blinded token which she sends to the server. Apart from this, each voter device generates a PKE keypair. The public key of the voter is used by the server to encrypt the signed blinded token which it sends to the voter.

In addition, we have the PKE keypair generated by the node. The node receives the vote and after due verification transmits it as a transaction in the network. The communication between the voter and the node is achieved through a secure and authenticated channel. This requires the voter to encrypt the message it sends to the node with the node public key.

Next, we discuss the key generation at the counting side. We need to ensure that the results of the election are not known before the voting is over, i.e., we must prevent any party from knowing how many votes has been cast for which candidate before the end of the election. This is essential for maintaining the fairness property of the election process. Since the blockchain is public to ensure transparency and verifiability, we need an encryption mechanism to safeguard the fairness of results. For this, we use a PKE scheme with SSS. Ideally, the keypair for this PKE should be generated in a HSM as the secrecy of the private key is paramount. The public key is released and this is the key which the voters use to encrypt their votes. We can also call this the public key of the vote. The private key is given as input to SSS algorithm. The parts of the secret are then distributed to the verifiers. We use a k out of n secret sharing scheme for this purpose. This ensures that even if some of the verifiers ($< n - k$) fail to turn up and share their secrets, the votes

can still be counted smoothly.

$$votesk = SIKE\_keygen()$$
$$k = 3, n = 5$$
$$r_1, r_2, r_3 \xleftarrow{\$} \mathbf{Z}$$
$$p = votesk + r_1 x + r_2 x^2 + r_3 x^3$$
$$S_1 = (1, p(1))$$
$$S_2 = (2, p(2))$$
$$S_3 = (3, p(3))$$
$$S_4 = (4, p(4))$$
$$S_5 = (5, p(5))$$

Here, we also note that the shares of the secret should be distributed using a secure channel. To realize this, each verifier has their own PKE keypair and the shares are encrypted using their respective public keys before distribution. The system is secure assuming that at least $n - k - 1$ parties out of n parties are honest. Thus, if we choose $k = \frac{2k}{3}$ then the scheme remains secure assuming at least one-third of the verifiers are honest, which is a reasonable assumption.

4.3 Voting day

The voting day is the most important day in any election process. Before voting begins, each voter has a valid token for casting their vote. Let us assume that a voter has a token t and a signed version of the token s. Each candidate is assigned a unique number, which has a fixed number of bits. For example, in an election with 20 candidates in the fray, each candidate is represented by a unique number which has exactly $\lceil \log_2 20 \rceil = 5$ bits. Suppose a voter wants to vote for a candidate who has been assigned the unique number c. The voter then chooses a random number r. This is achieved by implementing a hash function on a password selected by the voter. The hash function here is SHA-256. The voter device then encrypts $c||r$ with the public key of the vote to get a ciphertext v_{enc}.

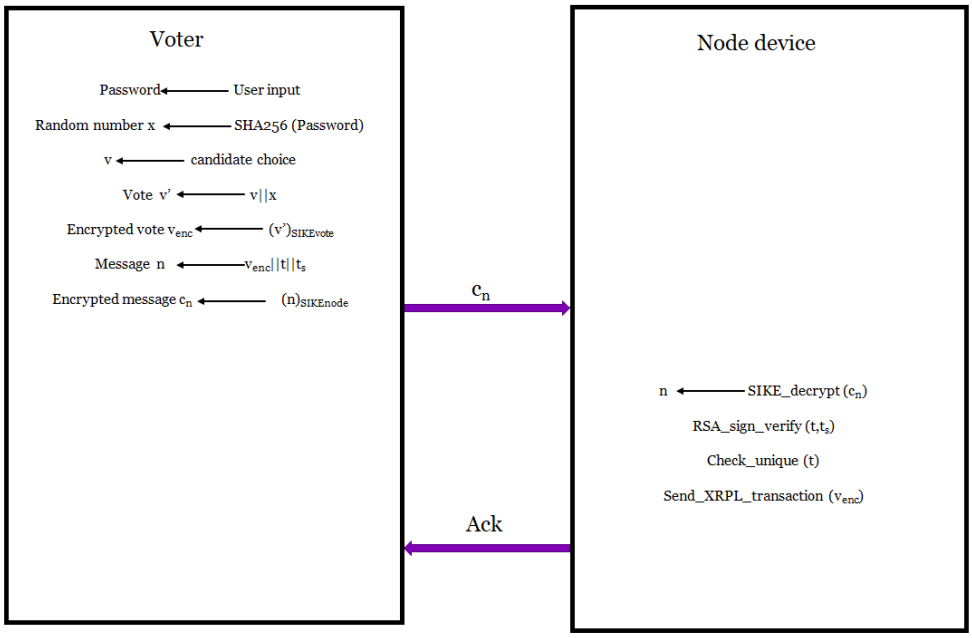

Figure 4.3: Voter-node interaction message sequence

Further, it encrypts t, s and v_{enc} with the node public key and sends it to the node. The message sequence between a voter and a node is depicted in Figure 4.3

$$r = H(seed)$$
$$vote = r||c$$
$$v_{enc} = (r||c)_{votepk}$$
$$c_n = v_{enc}||t||s$$

Using the public key of the authentication server, the node verifies that s is indeed the signed version of t. Further, it queries the token database to ensure that the token has not already been used. It then adds the token to the database and then sends the encrypted vote v_{enc} in the form of a transaction to a fixed account. The encrypted vote is included in the *Memos* field of the transaction. This transaction is stored in the blockchain. The node also sends an acknowledgement to the voter once the vote is successfully sent. The communication between the voter and the node is shown in Figure 4.1

4.4 Post vote procedure

After the election period is over, we have two major activities - the counting of votes and verification. We give a brief overview of these activities in the forthcoming subsection.

4.4.1 Counting

Once the voting time is declared over, the verifiers share their portion of the secret. These are then combined to obtain the vote private key. This key is made public. Each verifier then evaluates the result by decrypting and accumulating the votes stored in the blockchain. Here, we note that anyone including the voters or the candidates can verify the count since they know the vote private key.

$$coordinates = [(sx_1, sy_1), (sx_2, sy_2), (sx_3, sy_3), ...(sx_k, sy_k)]$$

$$l_1 = \frac{x - sx_2}{sx_1 - sx_2} * \frac{x - sx_3}{sx_1 - sx_3} * ... * \frac{x - sx_k}{sx_1 - sx_k}$$

$$\vdots$$

$$l_k = \frac{x - sx_1}{sx_k - sx_1} * \frac{x - sx_2}{sx_k - sx_2} * ... * \frac{x - sx_{k-1}}{sx_k - sx_{k-1}}$$

$$p = l_1 + l_2 + \cdots l_k$$

$$votesk = constant(p)$$

$$r||c = SIKE\_decrypt(v_{enc}, votesk)$$

4.4.2 Verification

In addition to count verification, the voters should be able to verify that their votes were correctly counted. Since the blockchain is public, it suffices to verify that their vote was correctly stored in the blockchain. We can recall from the voting phase that the encrypted vote was nothing but $(c||r)_{votepk}$ where c was the unique number representing the candidate and r was a random number which was the hash of a password chosen by the voter. Hence, with high probability, $c||r$ is indeed a unique number and the membership of this unique encrypted vote $(c||r)_{votepk}$ in the blockchain serves to verify that the vote was correctly stored and counted.

4.5 Design choices

In this section, we briefly explain our design decisions and mention some potential alternatives.

4.5.1 Authentication

The authentication process is done using SQL based client server interaction. The alternative choice was using an accumulator based proof of membership. However, here, the bottleneck was to find an efficient hash function which hashes every unique identity to a prime. We chose the SQL based interaction because of its simplicity and time efficiency. However, an accumulator based authentication system adds transparency to the process and hence can be explored in the future.

4.5.2 Encryption

For encryption, we have a number of design choices including group based schemes such as El-Gamal or RSA, Elliptic Curve Cryptography based schemes, Lattice based and Code based schemes. Here, we used SIKE PKE scheme for our implementation due to its small key size and quantum resistant property. However, SIKE is a slower encryption scheme than other ECC based schemes and we can switch to a faster ECC based scheme if quantum resistance is not required.

4.5.3 Fairness

For implementation of fairness, we can use a commitment scheme, a time locked encryption scheme or a secret sharing scheme. In our implementation, we used Shamir's secret sharing scheme, which is simple and time efficient. It also involves only one time interaction between the voter and the election architecture, whereas a commitment scheme might involve two corresponding interactions. The downside is the assumption that a minimum number of talliers should be honest, for the scheme to work as intended. In future, the scope of using an efficient time-locked encryption scheme can be explored.

4.5.4 Blockchain

For the blockchain backbone of our election system, we can use other blockchains such as Ethereum or Hyperledger. However, XRPL has the highest transaction speed among all these blockchains, around 1500 transactions per second, compared to around 30 transactions per second for Ethereum, and hence provides scalability to our system. We also note that we can use Smart Contracts to store election rules and parameters for a more transparent implementation of our architecture. Ethereum has supported a wide variety of Smart Contracts, whereas XRPL is planning to bring in Smart Contract functionality in the near future.

4.6 XRPL Blockchain

XRPL is the blockchain designed by Ripple, which is primarily used for financial transactions. For initial testing of our voting system, we used the test blockchain of XRPL, which is a replica of the main XRPL blockchain. The XRPL network is represented by the peer to peer cloud shown in Figure 4.1. In this section, we briefly describe the architecture of this blockchain with reference to our voting scheme.

4.6.1 XRPL network and transactions

Like any other blockchain, XRPL is a ledger which is maintained by a peer to peer network. Anyone can connect to this network as a node. To simplify our architecture and to add authentication, we separated the roles of voter and node in the blockchain. In our architecture, several voters can connect to a neutral device, which connects as a node in the XRPL network.

Any node can send a transaction to another node in the XRPL network and this transaction is recorded in the XRPL blockchain. There are mechanisms to control who can send money to an account. There are also mechanisms to convert different currencies from and to XRP, which is the currency of transaction in the XRPL network. However, in our case, we use the transaction fields to piggyback our encrypted vote. We use the *Memos* field of the transaction, which allows upto 1 kB data to be piggybacked with the transaction. The transaction amount, in our case, is just a token amount sent to a fixed account. The objective here is that when the transaction gets stored in the ledger, so does the *Memos* field and hence, the encrypted vote gets stored in the ledger.

4.6.2 Consensus

As with any blockchain, there exists a consensus mechanism. Here, we have an advantage using the XRPL blockchain as its consensus mechanism does not involve expenditure of too much time and energy, unlike in other blockchains, such as PoW in the Bitcoin or Ethereum blockchains, which require miners to solve a hard mathematical puzzle. The XRPL network comprises independent servers called 'validating servers'. These servers communicate among themselves and share and evaluate transaction proposals. This process is quite fast and takes only 4-5 seconds. However, the assumption here is that the validating servers do not collude to add invalid transactions to the chain. Once there is consensus among the validators, the transaction is added to the blockchain. It is expected that only well-formed and valid transactions are accepted in the ledger.

When a block of transactions are added to the ledger, it attains a new state. For a ledger state to be validated, there must be a consensus among a supermajority of the validators, which is currently defined as 80% of the validator nodes.

Chapter 5

Implementation

In order to evaluate the architecture proposed in the previous chapter, we have implemented a small scale version of the system. The first part involves the proof of concept where the implementation of back-end coding is on a personal laptop which is connected to a node on the XRPL test network, and the second part involves an isolated implementation using some hardware, which is essentially a prototype of a practical voting system, on which we aim to do some timing analysis.

5.1 Proof of concept

In this section, we review the proof of concept implementation of our blockchain based election system. For this implementation, all the code for authentication as well as tallying resides on a single machine.

5.1.1 Hardware requirements

For this experiment, we require a minimum of 4 CPU. For this, we used one laptop (say L1) and three Raspberry Pi sets (including accessories such as power adaptors, heat sink, etc). All the Raspberry Pi have 32 GB memory card and an extension SSD of 120 GB capacity. All the devices are connected using a switch and ethernet patch cables (this can also be done using a local hotspot). One of the Raspberry Pis (say R1) is used as the authentication server. The other two Raspberry Pi (R2 and R3) are used as voter clients. The Laptop is used as a device which connects to the XRPL Test network. The Laptop is

also used to simulate the SSS scheme and for the tallying of votes. In a practical system, the SSS scheme should be implemented either in a Smart Contract or in a HSM. R1, R2 and R3 have Ubuntu 20.0 installed and also assigned static IP addresses in the local network. We can simulate more voters by running multiple instances of the client program on R2 or by adding more Raspberry devices in the network. All the devices are connected using socket implementation (C/Python). The schematic is shown in Figure 5.1.

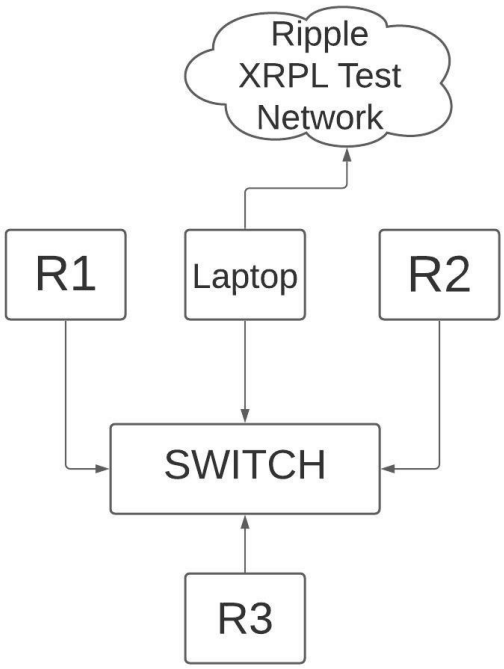

Figure 5.1: Implementation schematic

5.1.2 Authentication

In this section, we visit the implementation of the authentication server and its interaction with the voters. The database storing the voter name and id card number is implemented using MySQL. For security reasons, this implementation must be done in a HSM. An example snapshot of the table storing the voter credentials is provided in Figure 5.2. We note that for simplicity of representation in the diagram, the token size is 20 characters in the snapshot. The original token size in the scheme is 256 characters corresponding

to RSA-2048. The tag "NOTOKEN" indicates that the voter has not yet requested for a token. In that case, the token field stores "NONE". In case the voter has already requested a token earlier, the token is stored in the token field and the flag field is set to "TOKENRECD". In case the voter loses her token, she can send a token request to the server and the server can sign and send back the token. Note that the token stored here is actually the blinded token which is sent by the voter. Hence, the server does not actually know the value of the token and cannot link the voter with the unblinded token.

The authentication server (i.e., Raspberry device R1) as well as the other devices (L1, R2 and R3) also generate their SIKE434 key pair using the SIKE key generator function in C. This key pair will be used for secure communication among the devices.

Now, we come to the voter machine. The token generator function implemented in R2 takes in a password from the user input and generates a random token. The user input password should have a minimum length of 12 characters (for password security). An example token might be "UWaterloo@2020vote".

Figure 5.2: SQL Database example printout

The RSA based blinding algorithm was explained in Chapter 3. However, there is a security loophole in the raw version of the scheme. We notice that the signature is valid on both the blinded and unblinded versions of the token. Thus, if we follow the raw blind signature scheme, the voter essentially has two valid tokens after one interaction with the authentication server. To plug this loophole, we require that the voter should blind the SHA-256 digest of the token (say $H(t)$). Here, we notice that a signature on the blinded version of $H(t)$ does not constitute a valid token. The message generator function on the voter side gets the SHA-256 digest of the message using the OpenSSL library in C.

For the RSA signature portion of the scheme, the OpenSSL library cannot be used

because it uses SHA-256 hash of the digest for signing. For this, we implement RSA algorithm using GMP library in C. The RSA key generator function generates two random primes using seeds from user input (the /dev/random can also be used as a source of entropy here). Thereafter the private and public keys are generated. The blinder function takes in the random token generated earlier along with the RSA keypair. It then hashes the token to get $H(t)$ and then blinds it using a random number sourced from /dev/random.

The message generator function calls the RSA key generator and encryption function with the SHA-256 digest as input. It receives the resulting blinded digest, along with the name and credentials of the voter and concatenates the three. The message generator then calls the SIKE encryption function using the public key of R1 as input and receives the output which is the encrypted packet. This packet is sent to the authentication server, i.e., R1 using socket.

At R1, the handler function calls the SIKE decryption function with the R1's secret key and the received packet as input and parses the output to get the name, credentials and the blinded token. Then it performs a query on the SQL database using the name and id number. Here, three results are possible. First, the voter's name along with the id number is not on the list. In this case, the server just sends back a rejection message using the same socket to the voter. If the voter credentials are valid, then there might be two cases:

1. Case 1: The voter does not have a token. In this case, the handler function calls the RSA sign function with the blinded token, i.e., $(H(t))_{blinded}$ as the input. The RSA sign algorithm has been implemented using the GMP library in C.

2. Case 2: The voter already has a token. In this case, the handler function retrieves the stored token from the SQL database, and calls the RSA sign function with the retrieved token as input.

Before sending the signed token back to the voter, the handler function calls the SIKE encryption function is encrypted using the voter's SIKE public key to ensure the security of the communication. The communication between the voter (say R2) and the authentication server (say R1) is illustrated in Figure 5.3.

The receiver function on device R2 receives the encrypted packet from R1 and calls SIKE decryption function to get the signed blinded token. RSA decrypt function is then called with signed blinded token and ransom number r as inputs to get the signed token. Algorithms 1, 2, 3 and 4 illustrate the implementation of the authentication process.

Algorithm 1: R2 Key generator

input : None
output: SIKE private key sk_{R2_3}, SIKE public key pk_{R2_3}
SIKE_key_gen(&pk_{R2_3},sk_{R2_3});

Algorithm 2: R2 Message generator

input : SIKE public key pk_{R1_3}, name v_name, credential v_id, SIKE public key pk_{R2_3}
output: RSA unblinding key d, token t, random number r
t=token_generate();
H_t=get_SHA_256(t);
d, e=RSA_keypair_generate();
r=get_random();
t_b=blind_token(H_t,e);
f_pk=SIKE_key_flatten(pk_{R2_3});
msg=concat_string(t_b, v_name, v_id,f_pk);
enc_msg=SIKE_enc_alg(msg, &pk_{R1_3});
msg_send(enc_msg, ip_{R1});

Algorithm 3: R1 handler

g **input :** SIKE private key sk_{R1_3}, RSA signing key sk
output: None
$m1$= receive_packet();
c_0, c_1=parse_SIKE_msg($m1$);
$m1\_dec$= SIKE_dec_algo(&c_0, c_1, sk_{R1_3});
$v\_name, v\_id, t_b, pk_{R2_3}$=parse_msg($m1\_dec$);
$flag$,$token$= SQL_Query(v_name, v_id);
if *flag==0* **then**
  msg="NOT_EXIST_IN_DB";
  enc_msg=SIKE_enc_alg(msg,&pk_{R2_3});
  msg_send(enc_msg);
  conn.close();
else if *flag==1* **then**
  t=t_b;
else
  t=$token$;
end
$signed_t$=RSA_sign(t,sk);
enc_msg=SIKE_enc_alg($signed_t$,&pk_{R2_3});
msg_send(enc_msg);
conn.close();

Algorithm 4: R1 receiver

input : RSA unblinding key d, random number r
output: Signed token t_s
enc_token=receive_packet();
c_0, c_1=parse_SIKE_msg(enc_token);
t_dec= SIKE_dec_algo(&c_0, c_1, sk_{R2_3});
t_s= RSA_unblind(d, r);

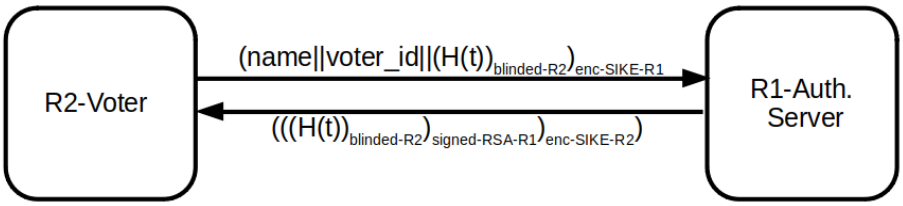

Figure 5.3: Communication between voter and authentication server

5.1.3 Voting

Voting consists of three steps: the initial voting setup, communication between R2 and L1 and the communication between L1 and the XRPL test network. L1 has access to a SQL database which stores all the tokens used for voting so far.

Initial voting setup

First we briefly discuss the voting key pair setup. The code for the SSS algorithm is implemented on L1 using GMP library in C. In a full scale implementation, this code should be implemented on a HSM since it generates the voting private key the secrecy of which is

Algorithm 5: Voting Key Management

input : Verifier ips $ip_{v1}, ip_{v2}, ip_{v3}, ip_{v4}, ip_{v5}$,
Verifier public keys $pk_{v1_3}, pk_{v2_3} pk_{v3_3}, pk_{v4_3}, pk_{v5_3}$
output: vote public key pk_{e_3}
SIKE_key_gen(&pk_{e_3},sk_{e_3});
s_1, s_2, s_3, s_4, s_5=Shamir_secret_partition(sk_{e_3},3,5);
m_1=SIKE_enc_alg(s_1,&pk_{v1_3});
m_2=SIKE_enc_alg(s_2,&pk_{v2_3});
m_3=SIKE_enc_alg(s_3,&pk_{v3_3});
m_4=SIKE_enc_alg(s_4,&pk_{v4_3});
m_5=SIKE_enc_alg(s_5,&pk_{v5_3});
msg_send_client(m_1, ip_{v1});
msg_send_client(m_2, ip_{v2});
msg_send_client(m_3, ip_{v3});
msg_send_client(m_4, ip_{v4});
msg_send_client(m_5, ip_{v5});

essential for the fairness of the system. The random secret key is generated by calling the SIKE key generator function. The Shamir secret partition function is then called which partitions the secret into five parts. The scheme used is three out of five shamir secret sharing. The implementation is done using GMP library, since the secret to be shared, i.e., the SIKE private key is of type mpz_t (GMP integer). The source of entropy for the random coefficients used in the Shamir secret partition function is the /dev/random in linux. The SIKE encryption function is then called with the public key of each of the verifiers to encrypt their portions of the secret. In the proof of concept implementation, only 3 parts of the secret are sent to R1, R2 and R3. In a full scale implementation, these 5 portions of the secret shall be sent to independent verifiers who must be different from voters or candidates.

Communication between R2 and L1

In this subsection, we review the communication between the voter R2 and the node in the blockchain network represented by L1. The R2 vote generator function takes in a user input c indicating the candidate to vote for. The the length of this is 3 bytes. It also takes in a user input password and generates a number h which is the hash of the password. The other inputs to this function are the token generated by the message generator function and the signed token decrypted by the R2 handler function. The vote SIKE public key which was generated by the vote key management setup is also taken in as input. The candidate choice c and the random number h are concatenated and used as input to the SIKE encryption function with the vote public key as the other input. The output from this function is then concatenated with the token and the signed token and input to the SIKE encryption function with the L1 SIKE public key as the other input. Finally, this encrypted message is sent to the node L1 using socket.

We note that the SIKE encryption function in the original SIKE module available on NIST website outputs a GMP integer and a structure containing 3 points on the elliptic curve, both of which constitute the ciphertext. Hence, we have implemented a wrapper function inside which we have a flattening procedure which converts the two outputs c_0 and c_1 into a single cipher string which is then sent as the message. Thus, in this chapter, a reference to SIKE encryption function implies this wrapper function which includes SIKE encryption along with the flattening procedure. At the decryption end, we parse the string the get back c_0 and c_1 which can then be input to the decryption function.

The L1 handler function receives the message sent by R2. The RSA signature public key of R1 (authentication server) is also taken in as an input. The SIKE decryption function is called and the output is parsed to get the token, signed token and the encrypted vote.

Thereafter, the RSA signature verification function is called which checks the validity of the signature on the token. Once the signature is verified, the handler calls the SQL Query function which queries an SQL database to determine whether the token has already been used. If the token has not been used, the token is added to the database. If the token has already been used, the handler sends back an error message to R2. Once it is determined that the token is valid and is not being re-used, the handler hands over the encrypted vote to the vote send function.

Algorithms 6 and 7 illustrate the vote generation and node handler algorithms.

Algorithm 6: R2 Vote generator

input : vote public key pk_{e_3}, token t, signed token t_s, candidate preference c, user password pwd, L1 sike public key pk_{l_3}, laptop ip ip_l
output: None
h=get_SHA_256(pwd);
$vote$=concat_string(c, h);
v_enc=SIKE_enc_alg($vote$,&pk_{e_3});
msg=concatenate_string(v_enc, t, t_s);
msg_e=SIKE_enc_alg(msg,&pk_{l_3});
msg_send(msg_e, ip_l);

Algorithm 7: L1 handler

input : RSA signature public key pk, SIKE private key sk_{l_3}
output: , SIKE public key $pkR2_3$
$m1$=receive_packet();
c_0, c_1=parse_msg(m);
$vote\_msg$= SIKE_dec_algo(&c_0, c_1, sk_{l_3});
$t, t_s, vote$ = parse_vote_msg($vote_msg$);
if *RSA_sign_verify(t, t_s, pk)==0* **then**
  $m2$="INVALID TOKEN";
  msg_send($m2$);
  conn.close();
if *SQL_Query_db(t)==0* **then**
  $m2$="TOKEN ALREADY USED";
  msg_send($m2$);
  conn.close();
vote_send($vote$);

Communication between L1 and XRPL test network

In this subsection, we consider the process of sending the vote on the network. For this purpose, we use the python interface of XRPL. Once the voting process starts, L1 connects

to the XRPL test network. Two accounts are created, one for from which the transaction is to be sent and another account to which the transactions will be sent. The transactions can be of any amount, but for our demo, we used the amount 1 XRP. Since a new account in the XRPL test network is credited with 1000 XRP, we can send 1000 votes uninterrupted. We note that in a full scale implementation, the sender account would be initiated with a large amount of XRP so that the voting process can continue uninterrupted.

The encrypted vote is embedded in the 'memos' field of the XRPL transaction. The function waits until the sent transaction is confirmed on the XRPL test ledger. In this test case, we do not choose any verifier nodes and hence the verification is done by the nodes which are automatically set in the XRPL test network. In a full scale implementation, we would have a set of verifier nodes for confirming the transactions in the ledger.

An example of an XRPL transaction with encrypted data embedded in the 'memos' field is shown in Figures 5.4 and 5.5.

```
m1=Memo(memo_data=enc_vote)
my_tx_payment = Payment(
    account=test_account,
    memos=[m1],
    amount='100000',
    destination='rpJJFUeG7UpV4VRvDDqJRGhEYtZHVT2u1r',
)
```

Figure 5.4: XRPL transaction structure

```
Payment object: Payment(account='rDgGGonjsFap6L6W1D38yXR8YDyq1c962', transactio
n type=<TransactionType.PAYMENT: 'Payment'>, fee=None, sequence=None, account t
xn id=None, flags=0, last ledger sequence=None, memos=[Memo(memo data='CDE47678
382947659365890745215678900989924827482974720492492842834029849023409327402374 9
237482748023742093409284089088023740734094109849184097137408134781374093478374 8
937803740137490174147910841903470931874380174137409173471034791374901374018347 8
013741374081374031740813741734763247092398748329748927492378429749279723640293 4
0982234243232374809723', memo format=None, memo type=None)], signers=None, sour
ce tag=None, signing pub key='', txn signature=None, amount='100000', destinati
on='rpJJFUeG7UpV4VRvDDqJRGhEYtZHVT2u1r', destination tag=None, invoice id=None,
 paths=None, send max=None, deliver min=None)
```

Figure 5.5: Payment object in XRPL generated from transaction

5.1.4 Tallying

Once all the votes are cast, the secret-holders come together and use the Shamir secret construction algorithm to reconstruct the SIKE private key to decrypt the votes. The Shamir secret construction algorithm is implemented using the GMP library in C.

Anyone can retrieve the votes cast by collecting the ledger history from the XRPL test ledger. In this case, since all the votes were sent to one account, we can query the history of transactions in that account to collect the votes. This is implemented using the python interface to XRPL.

```
The list of transactions:  [{'meta': {'AffectedNodes': [{'ModifiedNode': {'FinalFields': {'Account':
'rpJJFUeG7UpV4VRvDDqJRGhEYtZHVT2u1r', 'Balance': '1004500000', 'Flags': 0, 'OwnerCount': 0, 'Sequence
': 18950042}, 'LedgerEntryType': 'AccountRoot', 'LedgerIndex': '9A3A1589B01909C35470BDAB2A375104DB6C6
9087FAE0A0C870B24AB35824DA3', 'PreviousFields': {'Balance': '1004400000'}, 'PreviousTxnID': '664C662E
188A36E89060352D0D776C54294F9BF37B01339ADB3592AA0F618CD2', 'PreviousTxnLgrSeq': 18950072}}, {'Modifie
dNode': {'FinalFields': {'Account': 'rDgGGonjsFap6L6W1D38yXR8YDyq1c962', 'Balance': '999899990', 'Fla
gs': 0, 'OwnerCount': 0, 'Sequence': 19159825}, 'LedgerEntryType': 'AccountRoot', 'LedgerIndex': 'AFC
E84D2F37D57025D07C29ED13DE5C7DBCED626BA373C7026DBD1C21D92C3FA', 'PreviousFields': {'Balance': '100000
0000', 'Sequence': 19159824}, 'PreviousTxnID': '25F9B04A0E08F0A33B4EF3C587B4D133A4A8FDC7AB20CD432ECB5
C1695BE793E', 'PreviousTxnLgrSeq': 19159824}}], 'TransactionIndex': 0, 'TransactionResult': 'tesSUCCE
SS', 'delivered_amount': '100000'}, 'tx': {'Account': 'rDgGGonjsFap6L6W1D38yXR8YDyq1c962', 'Amount':
'100000', 'Destination': 'rpJJFUeG7UpV4VRvDDqJRGhEYtZHVT2u1r', 'Fee': '10', 'Flags': 0, 'LastLedgerSe
quence': 19159845, 'Memos': [{'Memo': {'MemoData': 'CDE476783829476593658907452156789009892482748297
472049249284283402984902340932740237492374827480237420934092840890880237407340941098491840971374081348
781374093478374893780374013749017414791084190347093187438017413740917347103479137490137401834780137413
374081374031740813741734763247092398748329748927492378429749279723640293409822342432323748097237}}],
'Sequence': 19159824, 'SigningPubKey': 'ED1243CBF75C926D3CFE6CF43CCB6CCD87B7069B06CC8123EC1D6FF239A42
7D0C4', 'TransactionType': 'Payment', 'TxnSignature': '8968686C010E23CABDAEA6DBAA1BBF079620A328F8A667
79691432BB73083A4C1A51B324B21FD94F7E87D05BA2A2D6C1E5E6A4F7288A8D4F7F11D17DADF5840C', 'date': 67936737
0, 'hash': '801AE14866650919CB1D2402667C50ED9E4BAA5967E6F760DCFFFCF442678D55', 'inLedger': 19159827,
'ledger_index': 19159827}, 'validated': True}, {'meta': {'AffectedNodes': [{'ModifiedNode': {'FinalFi
```


Figure 5.6: Note that the memo field holds the encrypted data

An example of retrieved transactions from an account is shown in Figure 5.6. All the retrieved transactions are stored in a text file.

If the SIKE private key is now publicly posted, anyone can decrypt the votes to tally the final numbers. For tallying the votes, the verifiers run the vote decryption function which parses through the retrieved history to extract each individual vote and call the SIKE decryption algorithm to to decrypt and tally the votes.

The tallying algorithm is illustrated in Algorithm 8.

Algorithm 8: Tallying Algorithm

input : SIKE private key sk_{e_3}
output: None
fp=fopen("retrived_data.txt","r");
while *True* **do**
    tran=get_transaction(fp);//gets details of one transaction from the file in a string(buffer)
    if *tran==EmptyString* **then**
        break; //end of file
    enc_vote=parse_transaction(tran);//gets encrypted vote from memo field
    c_0, c_1=parse_vote(enc_vote);//gets ciphers c0 and c1 from encrypted vote
    vote_msg=SIKE_dec_algo(&c_0, c_1, sk_{e_3});
    count_vote(vote_msg);//converts first 3 chars of vote_msg into int and
    //increments corresponding count
end

5.1.5 Verification

Since the details of the transactions to the account can be retrieved by anyone, the voter can easily verify that her vote was correctly stored in the ledger and counted. For this, she has to download the history in a file and run the same parse transaction algorithm used for tallying to get each of the encrypted votes. Then, instead of the parse vote function, she just calls a string matching function for every vote retrieved to verify that one of the encrypted votes matches with the encrypted vote that she sent. We note that even during this verification process, the encrypted vote reveals nothing about the person whom the voter voted for. The voter knows the content of the vote she sent and she can verify that her vote is recorded in the blockchain. However, she does not have any proof that she actually cast that vote. She cannot conclusively prove to a third party that she voted for a particular candidate. Hence the system is coercion resistant.

5.2 Prototype network

For the prototype network, we use 10 RaspBerry Pi, one laptop and one unmanaged switch. The laptop and RaspBerry devices are connected using patch cables to the unmanaged

switch. The network is isolated and none of the devices are connected to the internet. We build the XRPL package on each of the devices. Thus, the whole network acts as an isolated private blockchain. There are five verifier nodes and five nodes which send the vote transactions.

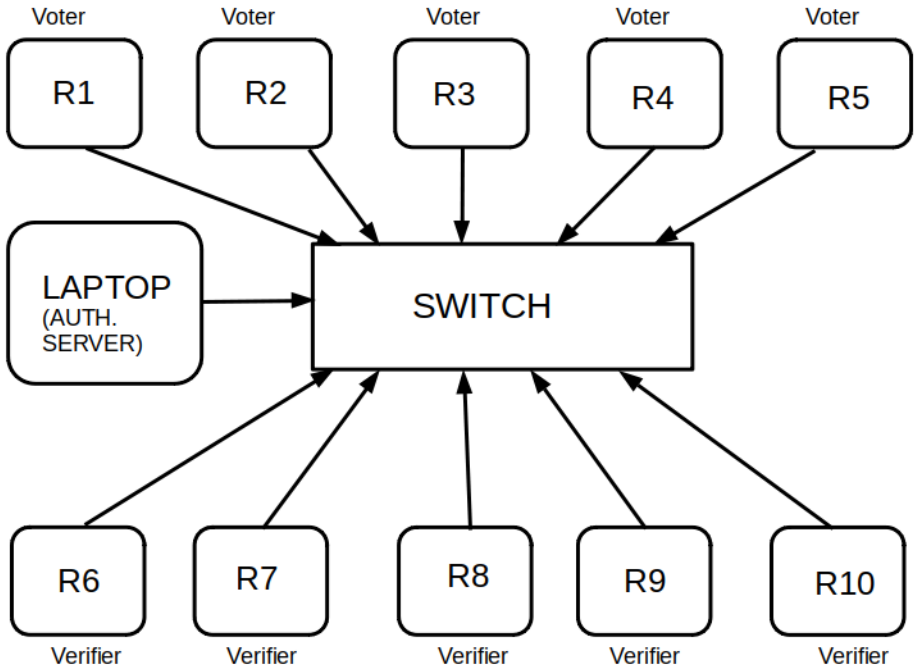

Figure 5.7: Prototype network for voting using RaspBerry Devices

The laptop acts as the authentication server and also generates the vote key pair using the SSS algorithm. The rest of the algorithms used are the same as those used for the experiment using the XRPL test network. This network closely resembles a practical implementation since it is an isolated network with only the voting transactions and involves separate verifier and voting nodes. A diagrammatic representation of the prototype network is given in Figure 5.7. A picture of the laboratory setup is given in Figure 5.8.

Here, we note that in terms of network performance, the prototype network gives slightly optimistic results because of low number of votes and absence of any other network traffic. In terms of performance of algorithms, the results are slightly pessimistic because they are implemented on low end RaspBerry Pi devices.

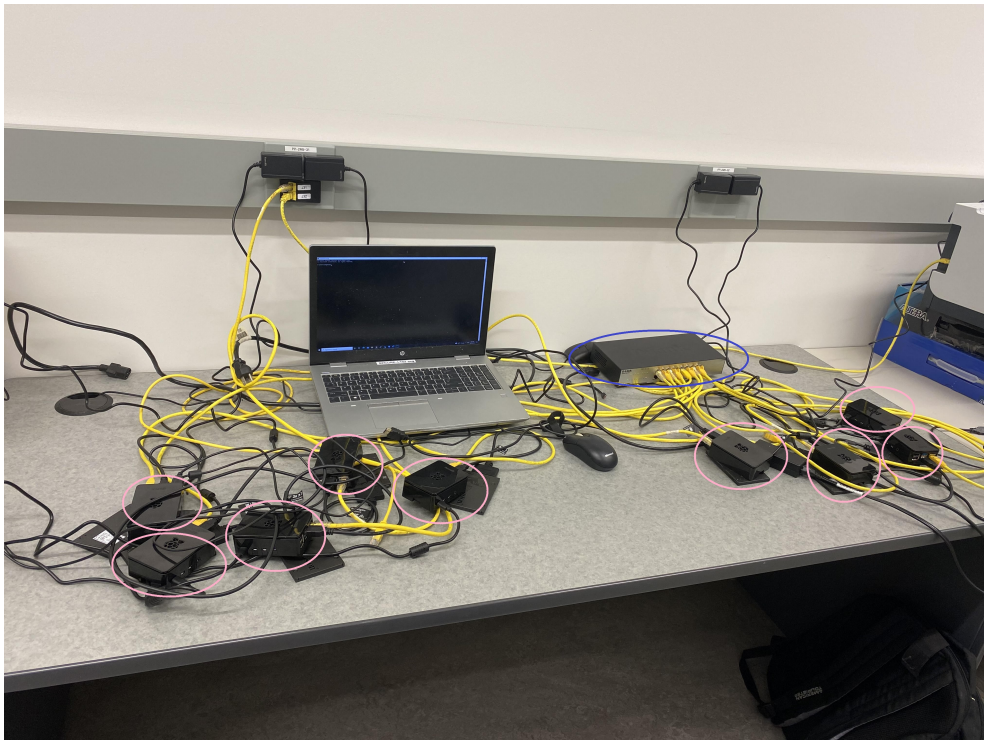

Figure 5.8: The RaspBerry devices are marked with pink circles and the switch is marked with a blue circle.

Chapter 6

Evaluation

In this chapter, we analyze the proposed voting system in terms of security, desirable properties and also provide some timing results obtained from our proof of concept implementation.

6.1 System security overview

In this section, we analyze the security of the system in terms of classical bit level security and also security against possible attacks on the system.

6.1.1 Authentication

For authentication, the voter first generates a random token. The source of the token is a user input. Then it is hashed using SHA-256. The collision resistance level of SHA 256 is 128 bits, which is reasonable for today's secure applications. Thereafter, a RSA keypair is generated to blind the token. The RSA implementation used is one using GMP library. The size of modulus is 2048 bits (which can be upgraded to 4096 bits if necessary). This offers a security level of 112 bits. Here, we also mention that the primes for our RSA implementation have been chosen using two random seeds from user input and then using the next prime function in GMP library. Thereafter, a check is done to determine whether it is a safe prime. If not, the next prime is chosen. The voter machine then obtains a random number (in our case we used 521 bits) using the GMP random number function. The seed for this random number is /dev/random. In Linux, this fetches a random number

from the Kernel's random number generator. This is further seeded with entropy from hardware values, and hence generates numbers that are considered truly random as they rely on intrinsic variations in physical variables. The token is then blinded using the RSA key and the random number. Thereafter, the voter machine appends the blinded string with the credentials and their own SIKE public key and encrypts it using the servers SIKE public key. For our experiments, we used SIKE434 and SIKE751, which provide classical security level of 117 and 235 bits respectively as well as quantum security of 124 and 219 bits respectively.

The encrypted message is received by the authentication server which holds voter data is an SQL database. The implementation of this authentication server is recommended on HSM so that the voter credentials are secure. The authentication server again used RSA signature using 2048 bit RSA scheme, which has 112 bit security level. The authentication server then sends back the signed token to the voter using SIKE encryption.

Here, we must mention that the security of the voter device is important in context of security and privacy of the scheme. The SIKE keys and credential storage on the voter device should be secure. This was also one of the critical issues which was pointed out with respect to the Voatz scheme which was piloted in the US elections.

6.1.2 Voting

The vote is encrypted using the election public key. The election keypair is generated in a secure HSM (in our prototype, we use a laptop, but an HSM is recommended in a full scale implementation). SIKE751 encryption is used for this purpose. Apart from this, the communication between the voter and the node is also encrypted using another layer of SIKE encryption. Thus, even the token or the signed token is not revealed to anyone other than the voter.

Only valid transactions are recorded in the blockchain. This is ensured because the sending nodes as well as the verifying nodes belong to the election authority. A voter can only send her vote to the node machine and vote is sent as a transaction only once the token is verified.

6.1.3 Verification and tallying

After the election, the SIKE secret key is retrieved and votes are decrypted. The SSS, which is used for sharing parts of the secret is known to be information theoretically secure. For

our implementation of SSS, we use 521 bit coefficients. For the attacker to guess the coefficients she would have to make $2^{521} * k$ guesses where $k = 3$ in our experimental case. The secret sharing with the verifiers is also achieved using SIKE secured channels.

Since all communication in the scheme is secured using SIKE, the only possible vulnerabilities of the scheme are side channel attacks during the RSA encryption and signing, security of private key storage in respective devices and the case that k out of n verifiers are corrupt. Apart from this, there are also some Cold-Boot side channel attacks which have been proposed on SIKE scheme [34].

6.2 Evaluation of desirable properties achieved

In this section, we evaluate our proposed architecture in terms of achievement of desirable properties as enumerated earlier in chapter 2 and compare it with properties of other proposed blockchain based voting systems.

1. **Authentication**: In the proposed architecture, only the voters whose names are registered in the SQL database available to the authentication server can cast their votes. This is ensured because the authentication server signs on the blind token only if it can retrieve the voter's credentials from the database. Also, if that voter had already requested for a token, the authentication server does not sign any new token sent by the voter. Instead, it signs the old blinded token stored in the database and sends it back to the voter. This is just to guard against the situation when the voter, due to any network issues, does not receive the signed token. The hash also ensures that only the unblinded signed hash is a valid signature. The blinded signed hash along with the token does not form a valid signature pair and manufacturing a second token from this information would require the voter to calculate a pre-image of SHA-256.

2. **Anonymity**: Our system achieves full anonymity as except the voter herself, no one can link the voter with the vote. When the voter sends the token to the authentication server, it is hashed and blinded. So the authentication server, cannot, in any way, link the token to the voter. When the node machine receives the token along with the encrypted vote, it cannot decipher the identity of the voter. It can only check the validity of signature of the authentication server on the token. The encrypted vote itself contains only the candidate number padded with some random number chosen by the voter machine. Hence, even after decryption of the vote, the identity of the voter is never revealed.

3. **Accessibility**: The proposed model is easily accessible because it does not require the voter to be physically present during authentication or voting. Also, the voter does not need to have any know-how about cryptographic operations to cast her vote. All she has to do is enter some random passwords which are used as seeds for generating random numbers.

4. **Coercion Resistance**: The voter cannot prove to a third party the she voted for a particular candidate. The vote is a combination of the candidate number and a random number chosen by the voter. But the voter cannot, after voting, conclusively prove to a third party that she chose that particular random number.

5. **Affordability**: The device requirements for the voter is not high end as it just has to support the voting app which generates keys and sends and receives messages to and from the authentication server. However, the device should have provision for secure storage so that the private keys are not compromised.

   Regarding the infrastructure requirement for the election, we only require node devices with few TB of memory (in our small scale experiment, we used only 120 GB SSD). However, for implementation of the authentication server and the vote key management server, we require two HSM, which do carry significant cost. However, the cost of implementation would still be affordable in comparison to legacy election systems, which require the cost of hundreds of EVMs, as well as significant man-power costs.

6. **Security**: The security of the system has already been evaluated in section 6.1.

7. **Integrity**: The sending and verification nodes are owned by the election authority. Hence, invalid messages or votes are not stored on the blockchain. Further, there is acknowledgement from the node to the voter device once the vote is successfully sent. While MITM attacker can tamper the message from the voter to the node, the node will not send the invalid vote message on the network and also the voter will receive a message stating that the vote has not been sent. Even if the MITM attacker tampers with the acknowledgement message, the voter will know that her vote has not been sent because she did not receive a positive acknowledgement from the node machine. Once the vote is sent as a transaction and stored on the blockchain, it is rendered immutable due to the inherent property of blockchain.

8. **Verifiability**: Our XRPL based voting system is both individually and universally verifiable. The system is universally verifiable because anyone can tally the final vote count once the election private key is publicly posted after conclusion of voting. It

is also individually verifiable as the voter can verify that her vote is stored in the blockchain as cast. The voter knows the encrypted vote message she sent to the node machine. To verify the presence of her vote on the blockchain, she just has to download the transaction history and search for the presence of her encrypted vote in the 'memos' field in one of the transactions.

9. **Fairness**: The proposed architecture is fair provided that at least $n - k + 1$ verifiers are honest. Here, n is the total number of verifiers and k is the minimum number of shares required to decode the election private key. In a practical scenario, if we consider 20 verifiers and a 15 out of 20 SSS scheme, fairness is guaranteed provided at least 6 verifiers are honest. However, we must also mention that increasing k has its own disadvantages because we need at least k verifiers to decode the private key without which the votes cannot be decrypted. Since verifiers are supposed to be owned by the election authority and independent observers, it is reasonable to assume that a majority of them will be honest.

   Apart from this, the random number padding by the voter to the vote before encryption also ensures fairness. Even if two voters vote for the same candidate, their encrypted vote would not be same because of the random number padding.

6.2.1 Possible improvements to the system

Here, we point out some areas of improvement which can possibly increase the security and feasibility of the proposed architecture.

On the authentication side, there is a possible weakness to side channel attack because we use RSA for blinding and signing the token. To guard against this, we have the communications between the voter and the authentication server through secure channel using SIKE751 encryption. Apart from this, the authentication server is recommended on a HSM, which virtually rules out side channel attacks. However, this increases cost of implementation. Recently, there have been some works relating to blind elliptic curve based signature schemes, which can be explored.

There is also the requirement of non-corruption of verifiers. To mitigate this, the private key can be stored in a smart contract and the contract will reveal the public key only once the election time is over. The start and end time of election can also be set in the smart contract. As of now, the XRPL ledger does not have implementation of smart contracts. However, recently, there has been a proposal by Ripple to include smart contract functionality in XRPL without affecting transaction rate by using side chains.

Finally, there is the remote possibility of two voters generating the same token, in which case the second voter cannot cast her vote. Here, we must mention that this possibility is negligible since the token is sourced from user input, which is of variable length. However, to make the system immune to this situation, one possible solution is to have a unique number database. Anyone can request numbers from this database and every number is unique. A person can anonymously request as many numbers as she wants. Then she combines some user input with one of these unique numbers and uses it as token. However, this system would be vulnerable to DoS attacks because attackers could possibly flood the server with requests.

6.3 Experiment timing results

In this section, we present some timing results from our experimental implementation which was detailed in Chapter 5. First we sent votes in batches of 10 to note the time required for encrypting and confirming a vote. There is also the time required by the node to decrypt the packet from the voter and check for the validity of the token. Apart from this, we also made a note of the time required for the SSS scheme as well as the authentication process, which, however, does not affect the voting throughput since these activities can be done before or in parallel to the voting process.

The time taken by voter to create the vote packet to be sent to the node machine is measured to be around 3.1 seconds. The time taken by the node to decrypt and parse the packet is 2 seconds. For comparison and verification of RSA signature, the time taken is negligible, measured around 0.025 seconds for 10 operations. The SQL database search takes negligible time (around 0.005 seconds). The time for confirmation of the transaction message on the XRPL blockchain is around 8-10 seconds. We note that the node machine which acts as a server creates multiple threads to deal with voter clients, so the processing is parallel. We also note that the time taken for encryption and decryption did not change much when we switched from SIKE434 to higher levels of security using SIKE751.

Since the confirmation of transaction takes around 8-10 seconds and the XRPL ledger can handle around 1500 transactions per second, we could possibly process more than 100k votes per hour.

For the authentication process, the total time taken for token generation, blinding, encryption and then the whole communication process with the server takes around 6 seconds. We note that these results were obtained in a completely traffic free private network with only four devices. So in reality, the time taken might vary depending on network traffic.

The SSS encryption takes only around 0.02 milliseconds and the decryption takes around 0.05 milliseconds. The download of history takes around 1-1.5 second. This does not change significantly with increasing number of votes in our small scale experiment. However, for a large scale experiment, the download history time is expected to be higher. Here, we must note that this is a one time process and we can afford it even if the download time increases with a large number of votes. For parsing and decryption of vote, the time taken is around 1.5 seconds per vote. Decrypting a hundred thousand votes would take around 70 machine hours. However, since this operation can be parallelized, the actual time taken can be reduced to a few hours.

The system can easily be scaled up using more node devices. For each constituency, there will be a cluster of nodes and an authentication server. This would ensure that the voter can vote only in her assigned constituency. Each constituency will have a separate network and a separate blockchain, which will speed up the vote throughput and also the counting process.

6.3.1 Scalability

In this section, we discuss the scalability of our proposed blockchain based election architecture. Historically, scalability has been one of the major challenges faced by blockchain based systems. However, using XRPL, one of the fastest blockchains in terms of transaction speed, we show that our system is capable of hosting a large scale election.

Let us assume that there is an election with 1,000,000 voters. Let us say that the total time taken by the node to process a vote is 15 seconds (10 seconds for verification and 2 seconds for vote processing, plus some buffer allowance). The XRPL system can process 1500 vote transactions per second. The number of node machines we need to use depends on the number of threads we can spawn on a single machine. High end server machines can handle 128 threads with one thread per core (we can also run two threads per core).

Number of transactions XRPL can process in 15 seconds $= 1500 * 15 = 22500$.

Let us say that we want around 10,000 votes be processed per 15 seconds. The number of node server machines required $= 10,000/128 = 79$. We take a 20% buffer and round it off to 100. 10,000 votes per 15 seconds would scale up to about a million votes an hour even at 50% efficiency.

We also need verifier nodes, which should preferably be the same number as non-verifier nodes (which is 100). We can also implement the system using less number of server nodes for a time-cost trade-off.

Apart from this, we note that due to the authentication structure of our architecture, we need to implement one blockchain for each voting precinct/constituency.

Chapter 7

Conclusion and future work

7.1 Conclusion

In this thesis, we have designed a blockchain based election architecture looking at the the requirements such as transparency, fairness, security, verifiability, etc. Since blockchain is a distributed and transparent entity, the biggest challenges were implementing the properties of anonymity, fairness and coercion resistance. We tried to balance these properties using the SSS scheme to achieve fairness and the random number padding to achieve verifiability while maintaining anonymity and coercion resistance. To maintain security of the system, we used SIKE encryption scheme, which is known to be quantum resistant. Our system meets most of the requirements which have been enumerated in previous work about blockchain based election systems. We also implemented a test system for proof of concept and the initial timing results indicate that the system is feasible for scaling up and implementation in a pilot scheme.

7.2 Future work

The system can be scaled up and implemented as a pilot project on a private XRPL network. For this, two platforms can be considered - cloud based and device based. A device based platform may consist of a number of devices serving as nodes on the private blockchain along with devices for the authentication server and the election key generator server. The private blockchain ensures security and also control of network traffic. There is also the option of implementing the nodes and the servers on cloud. This system is

cheaper and provides easier access but there is less control on the overall security of the network.

While implementing the pilot project, we must also implement a certificate authority for issuing public key certificates. This is essential for authentication of public keys. For simplicity, we have not implemented it in our small scale network.

In the long run, some shortcomings of the system can be resolved. Once XRPL incorporates Smart Contract functionality in its system, then the requirement of using the SSS scheme can be done away with and a Smart Contract can be used to store the election private key for the specified time. The Shamir Secret sharing can also be replaced by a distributed key generation scheme. The SQL database can also be replaced with a cryptographic accumulator based system to address the security concerns regarding voter credentials. Further, in our system, if the node server changes the vote, the voter knows about the change, but cannot do anything to prevent the changed vote from getting registered in the blockchain. This problem can be resolved if the voter is a verifier of the block which contains her transaction. This would involve some changes to the code of the XRPL blockchain. The blockchain, in that case, needs to be a permissioned one so that only authenticated voters can join the blockchain.

In addition, there can be more flexibility incorporated into the system. Currently, it is only suitable for a first past the post system. However, it can be slightly modified to make it suitable for a preferential system as well. This can be done by including a string indicating all preferences of the voter in the *vote* instead of including only one preference as is the case in our test implementation. Apart from this, another direction of improvement is to add the feature of the voter being able to change her vote, i.e., forgiveness property. This would require allowing the voter to vote more than once, while ensuring the previous vote is cancelled. A connection between the new and cancelled vote, which is known only to the voter, can be the random number r chosen by the voter. The possibility of using this to introduce the forgiveness property without compromising on integrity and anonymity, can be explored.

All the above mentioned changes would make the system more robust and attractive for large scale implementation in general elections.

References

[1] AGORA. Bringing our voting systems into the 1st century. Technical report, 2017. Available at https://static1.squarespace.com/static/5b0be2f4e2ccd12e7e8a9be9/t/5f37eed8cedac41642edb534/1597501378925/Agora_Whitepaper.pdf, accessed on 12.07.2021.

[2] Amazon. Tsukuba city: AWS deployment case study. Available at https://aws.amazon.com/solutions/case-studies/tukuba/, accessed on 12.07.2021.

[3] Josh Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital signatures (extended abstract). In *Workshop on the Theory and Application of Cryptographic Techniques, EUROCRYPT*, 1993. pp. 480-494.

[4] Daily Bit. 9 types of consensus mechanisms that you didn't know about. Available at https://medium.com/the-daily-bit/9-types-of-consensus-mechanisms-that-you-didnt-know-about-49ec365179da, accessed on 12.07.2021.

[5] BitCongress. Bitcongress whitepaper. Technical report, 2016. Available at https://cryptochainuni.com/wp-content/uploads/BitCongress-Whitepaper.pdf, accessed on 12.07.2021.

[6] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '02, Berlin, Heidelberg, 2002. Springer-Verlag. pp. 61-76.

[7] David Jao et al. Supersingular isogeny key encapsulation. Post-Quantum Cryptography Standardization, 2020. Available at https://sike.org/files/SIDH-spec.pdf, accessed on 12.07.2021.

[8] FollowMyVote. Blockchain voting: The end to end process. Available at https://followmyvote.com/blockchain-voting-the-end-to-end-process/, accessed on 12.07.2021.

[9] H.Cohen, G.Frey, R.Avanzi, C.Doche, T.Lange, K.Nguyen, F.Vercauteren. *Handbook of elliptic and hyperelliptic curve cryptography*. Chapman and Hall/CRC, 2005. p. 277.

[10] Pierrick Gaudry and Alexander Golovnev. Breaking the encryption scheme of the Moscow internet voting system, 2019. *Financial cryptography and data security*, pp. 32-49.

[11] H.A.Priestly. Introduction to groups, rings and fields. *Lecture notes*, HT and TT 2011, available at https://people.maths.ox.ac.uk/flynn/genus2/sheets0405/grfnotes1011.pdf, accessed on 12.07.2021.

[12] Voatz Inc. Voatz mobile voting platform-an overview: Security, identity, auditability. Technical report, 2019. Available at https://new.voatz.com/wp-content/uploads/2020/07/voatz-security-whitepaper.pdf, accessed on 12.07.2021.

[13] Luxoft. Luxoft's e-voting platform enables first consultative vote based on blockchain in Switzerland. Available at https://www.luxoft.com/pr/luxofts-evoting-platform-enables-first-consultative-vote-based-on-blockchain-in-switzerland/, accessed on 12.07.2021.

[14] Chaieb Marwa, Mirko Koscina, Souheib Yousfi, Pascal Lafourcade, and Riadh Robbana. *DABSTERS: A Privacy Preserving e-Voting Protocol for Permissioned Blockchain*. 10 2019. pp. 292-312.

[15] Chaieb Marwa, Souheib Yousfi, Pascal Lafourcade, and Riadh Robbana. *Verify-Your-Vote: A Verifiable Blockchain-Based Online Voting Protocol: 15th European, Mediterranean, and Middle Eastern Conference, EMCIS 2018, Limassol, Cyprus, October 4-5, 2018, Proceedings*. 01 2019. pp. 16-30.

[16] Malik Hamza Murtaza, Zahoor Ahmed Alizai, and Zubair Iqbal. Blockchain based anonymous voting system using zksnarks. In *2019 International Conference on Applied and Engineering Mathematics (ICAEM)*, 2019. pp. 209-214.

[17] Satoshi Nakomoto. Bitcoin: A peer-to-peer electronic cash system, 2008. Available at https://bitcoin.org/bitcoin.pdf, accessed on 12.07.2021.

[18] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies*. Association of American Publishers, 2017. pp. 31-58.

[19] Sunoo Park, Michael A. Specter, Neha Narula and R. Rivest. Going from bad to worse: from internet voting to blockchain voting. *Journal of Cybersecurity.*, 7, 2021. eprint: https://academic.oup.com/cybersecurity/article-pdf/7/1/tyaa025/36276521/tyaa025.pdf.

[20] Bernard Peh. What are public, private and hybrid blockchains? Available at https://medium.com/@blockchain101/what-are-public-private-and-hybrid-blockchains-e01d6e21eb41, accessed on 12.07.2021.

[21] Ripple. XRP ledger documentation. Available at https://xrpl.org/docs.html, accessed on 12.07.2021.

[22] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems, 1978. *Communications of the ACM*, vol.21, pp. 120-126.

[23] Adi Shamir. How to share a secret, 1979. *Communications of the ACM*, vol.22, pp. 612-613.

[24] Freya Sheer Hardwick, Apostolos Gioulis, Raja Naeem Akram, and Konstantinos Markantonakis. E-voting with blockchain: An e-voting protocol with decentralisation and voter privacy. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018. pp. 1561-1567.

[25] Nigel Smart. *Cryptography Made Simple*. Springer, 2016. pp. 3-32.

[26] Smartmatic. TIVI-verifiable voting: Accessible, anytime, anywhere. Technical report. Available at https://tivi.io/, accessed on 12.07.2021.

[27] Innovote Solutions. Software that powers democracy should be free. Technical report, 2016. Available at http://inno.vote/whitepaper/Inno.vote%20%E2%80%94%20Bringing%20Democracy%20to%20Elections.pdf, accessed on 12.07.2021.

[28] Michael A. Specter, James Koppel, and D. Weitzner. The ballot is busted before the blockchain: A security analysis of voatz, the first internet voting application used in

u.s. federal elections. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, August 2020. pp. 1535-1553.

[29] Horizon State. Decentralized engagement and decision platform, 2018. Available at https://cryptorating.eu/whitepapers/Horizon-State/horizon_state_white_paper.pdf, accessed on 12.07.2021, White paper.

[30] Scott Nadal Sunny King. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. Technical report, 2012. *self-published paper*, vol.19.

[31] Andrew Sutherland. Isogeny kernels and division polynomials. *Lecture notes*, Available at https://math.mit.edu/classes/18.783/2015/LectureNotes6.pdf, accessed on 12.07.2021.

[32] Yu Takabatake and Y. Okabe. An anonymous distributed electronic voting system using zerocoin. *2021 International Conference on Information Networking (ICOIN)*, 2021. pp. 163-168.

[33] Linh Vo-Cao Thuy, Khoi Cao-Minh, Chuong Dang-Le-Bao, and Tuan A. Nguyen. Votereum: An ethereum-based e-voting system. In *2019 IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF)*, 2019. pp. 1-6.

[34] Ricardo Villanueva-Polanco and Eduardo Angulo-Madrid. Cold boot attacks on the Supersingular Isogeny Key Encapsulation (SIKE) mechanism. *Applied Sciences*, 11(1), 2021.

[35] Cathie Yun. Adventures with RSA blind signing. Available at https://cathieyun.medium.com/adventures-with-rsa-blind-signing-397035585121, accessed on 12.07.2021.

APPENDICES

Appendix A

Code Implementation of Election Architecture

A.1 Helper Functions

A.1.1 Token generate

```
#include<stdio.h>
#include<string.h>
char* token_generate(){
    char userinp[101];
    printf("Enter a password less than 100 characters");
    scanf(%s,userinp);
    return userinp;
}
```

A.1.2 Get SHA 256

```
#include <stdio.h>
#include <string.h>
#include <openssl/sha.h>

unsigned char* get_SHA_256(char* msg)
```

```
{
    unsigned char opmsg[32];
    unsigned long lenstr=strlen(msg);
    SHA256_CTX ctx_sha;
    SHA256_Init(&ctx_sha);

    SHA256_Update(&ctx_sha, (unsigned char*)msg, lenstr);
    SHA256_Final(opmsg, &ctx_sha);
    return opmsg;
}
```

A.1.3 RSA Keypair generate

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<gmp.h>
void RSA_keypair_generate(mpz_t* rsask){
    mpz_t seed1,seed2,rand1,rand2,p_1,q_1,a1,b1;
    mpz_t nmod,lcm,pneck,qneck,phi,rsa_pk,rsa_sk;
    mpz_init(a1);
    mpz_init(b1);
    mpz_init(p1);
    mpz_init(q1);
    mpz_init(seed1);
    mpz_init(seed2);
    mpz_init(rand1);
    mpz_init(rand2);
    gmp_randstate_t s1;
    gmp_randstate_t s2;
    gmp_randinit_mt(s1);
    gmp_randinit_mt(s2);
    char uinp1[11];
    char uinp2[11];
    printf("Enter a 10 digit number:");
    scanf(%s,uninp1);
    int flag=0;
```

```
while(flag==0)
{
printf("Enter another 10 digit number:");
scanf(%s,uinp2);
if(strcmp(uinp1,uinp2))
{
flag=1;
}
}
mpz_set_str(seed1, uinp1, 10);
mpz_set_str(seed2, uinp2, 10);
gmp_randseed(s1, seed1);
gmp_randseed(s2, seed2);
mp_bitcnt_t bits=1024;
mpz_rrandomb(a1, s1, bits);
mpz_rrandomb(b1, s2, bits);
mpz_nextprime(p_1,a1);
mpz_nextprime(q_1,b1);
While(!safeprime(p_1))
{
mpz_nextprime(p_1,p_1);
}
While(!safeprime(q_1))
{
mpz_nextprime(q_1,q_1);
}

mpz_init(lcm);
mpz_init(nmod);
mpz_mul(nmod,p_1,q_1);
mpz_init(pneck);
mpz_init(qneck);
mpz_sub_ui(pneck, p_1,1);
mpz_sub_ui(qneck, q_1,1);
mpz_lcm(lcm,pneck,qneck);
mpz_init(phi);
mpz_mul(phi,pneck,qneck);
mpz_init(rsa_pk);
```

```
    mpz_init(rsa_sk);
    mpz_set_ui(rsa_pk, 65537);
    mpz_invert(rsa_sk, rsa_pk,phi);
    mpz_set(rsask,rsa_sk);


}
```

A.1.4 Get random

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<gmp.h>
void get_random(mpz_t* rand1)
{
    unsigned char readrand[10];
    int fp = open("/dev/urandom", O_RDONLY);
    read(fp, readrand, 10);
    close(fp);
    char *random1 = malloc(sizeof readrand * 2 + 1);
    for (size_t i = 0; i < sizeof readrand; i++)
        { sprintf(random1 + i * 2, "%02x", readrand[i]);
             }
    mpz_set_str(rand1,random1,16);
}
```

A.1.5 Blind Token

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<gmp.h>
void blind_token(unsigned char* token, mpz_t rsapk,
mpz_t* msg_enc,mpz_t nmod,mpz_t rand1 )
{
```

```
    mpz_t msgnum;
    mpz_init(msgnum);
    get_mpz_from_str(token,msgnum)
    mpz_powm(msg_enc, rand1, rsa_pk, nmod);
    mpz_mul(msg_enc,msg_enc,msgnum);
    mpz_mod(msg_enc,msg_enc,nmod);
}
```

A.1.6 SIKE key flatten

```
#define _GNU_SOURCE

#include <unistd.h>
#include <sys/syscall.h>
#include <linux/random.h>
#include <stdio.h>
#include <ctype.h>
#include "sike_params.h"
#include "fips202.h"
#include "encoding.h"
#include <stdlib.h>
#include <string.h>
#include "rng.h"
#include "api_generic.h"
#include "api.h"
char* SIKE_key_flatten(sike_public_key_t pk)
{
    char pkstring[1356];
    char temp[227];

    mpz_get_str(pkstring,16,pk.xP.x0);
    mpz_get_str(temp,16,pk.xP.x1);
    strcat(pkstring,temp);
    mpz_get_str(temp,16,pk.xQ.x0);
    strcat(pkstring,temp);
    mpz_get_str(temp,16,pk.xQ.x1);
    strcat(pkstring,temp);
```

```
    mpz_get_str(temp,16,pk.xR.x0);
    strcat(pkstring,temp);
    mpz_get_str(temp,16,pk.xR.x1);
    strcat(pkstring,temp);
    return pkstring;

}
```

A.1.7 RSA sign

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<gmp.h>
void RSA_sign(mpz_t rsassk, mpz_t nmod, mpz_t blinded_token,mpz_t* signed_token)
{
    mpz_powm(signed_token, blinded_token, rsassk, nmod);
}
```

A.1.8 Shamir Secret Sharing

```
// credits: adapted from
//https://www.geeksforgeeks.org/shamirs-secret-sharing-algorithm-cryptography
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<gmp.h>
struct coord{
        mpz_t x1;
        mpz_t y1;
};

struct fraction{
        mpz_t num;
        mpz_t den;
};
```

```
void coord_y(mpz_t x, int k, mpz_t* p, mpz_t* Y_val)
{
        mpz_t y;
        mpz_init(y);
        mpz_set_ui(y,0);
        mpz_t temp;
        mpz_init(temp);
        mpz_set_ui(temp,1);

        for (int i=0;i<k;i++) {
               mpz_addmul(y,temp,p[i]);
               mpz_mul(temp,temp,x);
           }
        mpz_set(Y_val[0],y);
}
void share_secret(mpz_t s, struct coord* coord_array, int n, int k)
{
       mpz_t* coeff=(mpz_t*)malloc(k*(sizeof(mpz_t)));
    mpz_set(coeff[0],s);

    for (int i=1;i<k;i++)
    {
            mpz_t c,seed;
            mpz_init(c);
            mpz_init(seed);
            mp_bitcnt_t bits_=521;
            gmp_randstate_t st1;
            gmp_randinit_mt(st1);
            get_random(&seed);
            gmp_randseed(st1, seed);
            mpz_rrandomb(c, st1, bits_);
            mpz_set(coeff[i],c);
            mpz_clear(c);
    }
    mpz_t* y_val=(mpz_t*)malloc(sizeof(mpz_t));
    for(int i=1;i<=n;++i)
    {
            mpz_t a;
```

```
            mpz_init(a);
            mpz_set_ui(a,i);
            coord_y(a, k, coeff, y_val);
            mpz_set(coord_array[i-1].x1,a);
            mpz_set(coord_array[i-1].y1,*y_val);

    }
}
void reduce_frac(struct fraction* f)
{
        mpz_t gcd;
        mpz_init(gcd);
        mpz_gcd(gcd,f->num,f->den);

}
struct fraction mult_frac(struct fraction f1, struct fraction f2)
{
        struct fraction f3;
        mpz_init(f3.num);
        mpz_init(f3.den);
        mpz_mul(f3.num,f2.num,f1.num);
        mpz_mul(f3.den,f2.den,f1.den);
        return f3;

}
struct fraction add_frac(struct fraction f1,struct fraction f2)
{
        struct fraction f3;
        mpz_init(f3.num);
        mpz_init(f3.den);
    mpz_mul(f3.num,f1.num,f2.den);
    mpz_addmul(f3.num,f2.num,f1.den);
    mpz_mul(f3.den,f1.den,f2.den);
    reduce_frac(&f3);



}
```

```
void SSS_dec(struct coord* c_array, int k, mpz_t* ss)
{
        struct fraction fr;
        mpz_init(fr.num);
        mpz_init(fr.den);
        mpz_set_ui(fr.num,0);
        mpz_set_ui(fr.den,1);
        for(int i=0;i<k;++i)
        {
                struct fraction t1;
                mpz_init(t1.num);
                mpz_init(t1.den);
                mpz_set_ui(t1.den,1);
                mpz_set(t1.num,c_array[i].y1);
                for (int j=0;j<k;++j)
                {
                        if(i!=j)
                        {
                                struct fraction t2;
                                mpz_init(t2.num);
                                mpz_init(t2.den);
                                mpz_neg(t2.num,c_array[j].x1);
                                mpz_sub(t2.den,c_array[i].x1,c_array[j].x1);
                                t1=mult_frac(t1,t2);
                        }
                }
                fr=add_frac(fr,t1);
        }
        mpz_set(*ss,fr.num);
}
void SSS_enc(struct coord* coord_array, int k, int n, mpz_t st)
{
                 mpz_t num;
                 mpz_init(num);
                 mpz_set(num, st);
                 share_secret(num,coord_array,n,k);

}
```

A.1.9 XRP Transaction Send

```
#credits: https://xrpl.org/get-started-using-python.html
from xrpl.clients import JsonRpcClient
from xrpl.wallet import generate_faucet_wallet
from xrpl.core import addresscodec
from xrpl.models.requests.account_info import AccountInfo
from xrpl.models.transactions import Payment
from xrpl.models.transactions.transaction import Transaction, TransactionType
from xrpl.models.transactions.transaction import Memo
from xrpl.transaction import send_reliable_submission, safe_sign_and_autofill_transacti
from xrpl.utils import xrp_to_drops
from time import sleep
import xrpl.ledger
import json
import js2py
def sendtransaction(memostring, selfaddress):

    link = "https://s.altnet.rippletest.net:51234/"
    cl1 = JsonRpcClient(link)
    m1=Memo(memo_data=memostring)
    vote_send = Payment(
            account=selfaddress,
            memos=[m1],
            amount='100000',
            destination='rpJJFUeG7UpV4VRvDDqJRGhEYtZHVT2u1r',
        )

    vote_signed = safe_sign_and_autofill_transaction(vote_send, test_wallet, cl1)
    vote_hash = vote_signed.get_hash()
    sent_vote = xrpl.transaction.submit_transaction(vote_signed, client)

    while True:
        sleep(1)
        tr_result = xrpl.transaction.get_transaction_from_hash(vote_hash, cl1)
```

```
if tr_result.is_successful():
    if tr_result.result.get("validated"):
        break
    else:
        continue
```