

# **A Blockchain Based Approach For Secure E-Voting system**

M.D. Sanjaya



# **A Blockchain Based Approach for Secure E-Voting System**

**M.D. Sanjaya**

**Index No: 14001284**

**Supervisor: Dr. T.N.K De Zoysa**

**January 2019**

Submitted in partial fulfillment of the requirements of the  
B.Sc in Computer Science Final Year Project (SCS4124)



# Declaration

I certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate Name: Mr. M.D. Sanjaya

.....  
Signature of Candidate

Date:

This is to certify that this dissertation is based on the work of

Mr. M.D. Sanjaya

under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Principle/Co- Supervisor's Name: Dr. T.N.K. De Zoysa

.....  
Signature of Supervisor

Date:

# Abstract

Voting is the process of representation of democracy in a country, to select a person for parliament, approve a bill in a parliament and decision making in board meetings etc. Voting systems have been around of hundreds of years and but they were evolving very slowly. Many solutions were proposed in the history, but most of them were rejected because of some security issues and limitations. Finally at the 21<sup>th</sup> century, e-voting systems started to bloom with the development of the web technologies

With the development of the blockchain 2.0, the researchers started to go towards a new destination by applying blockchain to software engineering applications. E-voting systems were developed based on Ethereum as well as Zcash and bitcoin. But they were not full e-voting frameworks. Due to the limitations of proposed solution, those were unable to exist with the modern world.

In order to prove feasibility of developing a secure e-voting system by combining some concepts of Zcash with the ethereum platform, this research is a proof of concept to implement a full e-voting framework with voter registration, voter verification, voting, tallying and end to end verification. To maintain user privacy, zk-SNARK which is a concept used in Zcash for maintaining private transactions, was used. To write Immutable codes, a concept called smart contract which is used in ethereum, was used. Due to the experiment and testing done, it is very clear that this system is a practical solution and works well by protecting voter privacy.

# Preface

Some of novel ideas were proposed in this research. The whole research is a novel idea, because of the previous related works targeted to solve some specific particular problem in e-voting scenarios. But this research consists of full e-voting framework consists of voter registration methodology, candidate registration methodology, candidate verification methodology, voting methodology, votes tallying methodology and votes verification methodology with less number of limitations. The concept used for the voter registration by using secret phrase, was solely my own idea and has not been proposed in any other study related to e-voting. The concept of applying zero knowledge proof and smart contracts with the e-voting scenarios in this approach was my own idea and implementation is my own work. The ethereum tool kit called as Zokrates is used to generate proof and verifier solidity contract. However, the development of the arithmetic circuit code was done by myself and has not been proposed in any other study related to the domain of e-voting. The evaluation model introduced in here was a novel idea for this domain and has not been used in any other research with domain of “e-voting with blockchain”. The experiment done with the ten voter accounts is my own work and has not been used in any other study related to e-voting.

# Acknowledgement

I would like to express my sincere gratitude to my research supervisor, Dr. T.N.K. De Zoysa, senior lecturer of University of Colombo School of computing and my research co-supervisor, Mr. T.G.A.S.M. De Silva, instructor of University of Colombo School of computing for providing me continuous guidance and supervision throughout the research. Both of them listened to my thoughts and made suggestions and improvements to this e-voting framework and finally finished the dissertation.

I also take the opportunity to acknowledge the assistance provided by Dr. H.E.M.H.B. Ekanayake as the final year computer science project coordinator.

I would like to express my gratitude to my friend Nandika Herath who gave me suggestions and encouragement to finish this dissertation and the software.

Finally I would like to thank my parents who were always supporting me with their best wishes.

# Table of Contents

.....	ii
<b>Declaration.....</b>	<b>iii</b>
<b>Abstract .....</b>	<b>iv</b>
<b>Preface .....</b>	<b>v</b>
<b>Acknowledgement .....</b>	<b>vi</b>
<b>Table of Contents .....</b>	<b>vii</b>
<b>List of Figures .....</b>	<b>x</b>
<b>List of Tables .....</b>	<b>xi</b>
<b>List of Acronyms.....</b>	<b>xii</b>
<b>Chapter 1 - Introduction .....</b>	<b>1</b>
1.1 Background to the Research .....	2
1.2 Research Problem and Research Questions .....	3
1.3 Justification for the research .....	4
1.4 Methodology .....	5
1.5 Outline of the Dissertation .....	6
1.6 Definitions.....	6
1.7 Delimitations of Scope.....	7
1.8 Conclusion .....	7
<b>Chapter 2 - Literature Review .....</b>	<b>8</b>
2.1 Introduction.....	8
2.2 Voting systems.....	8
2.3 Key concepts of Blockchain based Voting .....	11
2.4 Voting and privacy solutions based on Blockchains .....	13

2.5 Summary .....	15
<b>Chapter 3 - Design.....</b>	<b>16</b>
3.1. Introduction.....	16
3.2. Research Design.....	16
3.3 High-level architecture of off-chain and on-chain components .....	19
3.4. Registration phase .....	22
3.5. Registration verification phase .....	23
3.6. Election preparation phase .....	24
3.7. Voting phase .....	24
3.8. Tallying Phase.....	26
3.9. Verification phase .....	26
3.10. Summary .....	26
<b>Chapter 4 - Implementation .....</b>	<b>27</b>
4.1. Introduction.....	27
4.2. Software tools .....	27
4.3. Secret phrase generation .....	28
4.4. Sha256 hash calculation.....	30
4.5. Generate the proof.....	31
4.6. Smart Contracts.....	33
4.7. Summary .....	36
<b>Chapter 5 - Results and Evaluation.....</b>	<b>37</b>
5.1. Introduction.....	37
5.2. Cost analysis .....	37
5.3. Experiment.....	38
5.4. Evaluation properties .....	45
5.5. Evaluation criteria .....	46



5.5. Summary .....	47
<b>Chapter 6 - Conclusions .....</b>	<b>48</b>
6.1 Introduction.....	48
6.2 Conclusions about research questions (aims/objectives) .....	48
6.3 Conclusions about research problem .....	49
6.4 Limitations .....	50
6.5 Implications for further research.....	51
<b>Appendix A: Diagrams.....</b>	<b>56</b>
<b>Appendix B: Code Listings .....</b>	<b>57</b>

# List of Figures

Figure 1.1: The research approach.....	5
Figure 3.1: Research Design Steps .....	16
Figure 3.2: High level architecture of On-chain and off-chain component.....	19
Figure 3.3: Contracts in the voting system .....	21
Figure 3.4: High-level architecture of contracts and user interface components .....	21
Figure 4.1: Secret phrase generation.....	28
Figure 4.2: Base 2 to Base 10 conversion .....	30
Figure 4.3 : Sha256generate arithmetic circuit .....	31
Figure 4.4: Arithmetic circuit of proving pre-image for sha256hash.....	32
Figure 4.5: The proof.....	32
Figure 4.6 : Arithmetic circuit for proving pre-image for list of hashes .....	33
Figure 4.7 : Voter details structure .....	34
Figure 4.8 : Reset user account.....	34
Figure 4.9 : Candidate details structure.....	35
Figure 4.10 : Functions of Election contract .....	35
Figure 4.11: Sha256hashTest smart contract .....	36
Figure 5.1 : Apply for Election user interface .....	42
Figure 5.2 : user interface of generated secret phrase .....	42
Figure 5.3 : user interface of big integer representation of secret phrase .....	42
Figure 5.4 : user interface of voter profile.....	43
Figure 5.5 : User interface of acceptance/ reject a voter .....	43
Figure 5.6 : User interface of voting .....	43
Figure 5.7: User interface of election results.....	44
Figure 5.8 : User interface of Candidate management .....	44
Figure 5.9 : Success of Proof verification .....	44
Figure A.1 : Transactions listing in the election smart contract .....	56
Figure A.2: Account of smart contract.....	56
Figure A.3: Account of Voter.....	56

# List of Tables

Table 5.1: Transaction cost for voter .....	37
Table 5.2: Transaction cost for election authority in an election .....	38
Table 5.3: Voter details in the experiment .....	38
Table 5.4: Evaluation Model.....	46

# List of Acronyms

EVM	Ethereum Virtual Machine
zk-SNARK	Zero Knowledge Succinct Non interactive Argument of Knowledge
ZKP	Zero Knowledge Proof
ABI	Application Binary Interface
IDE	Integrated Development Environment
TX	Transaction
API	Application programming interfaces
GUI	Graphical User Interface
DApp	Decentralized application

# Chapter 1 - Introduction

Election, the formal process of selecting a person for public office or of accepting or rejecting a political proposition by voting [1]. Voting systems have been around of hundreds of years and but they were evolving very slowly. In Sri Lanka, still uses paper ballots based voting methodology for government elections. But current existing paper ballot based voting methodologies have lot of drawbacks. Some of them are Voter have to wait in a queue, Results are not trustworthy because of voting process is not visible to the public, Cost is very high because, have to pay for all the officers who work at polling locations as well as counting locations, Voter participation is less [2], Takes some time to release the results, results depend on the physical security, have to trust the officers in the polling locations as well as tallying locations, people who live in abroad are not able to cast their vote etc.

E voting, a system supports online voting by mobile phones, Desktop or laptop computers and Tablets. And the voter registration, voting, vote verification and vote counting will be done via the system. In E-voting scenarios Security and Trustworthiness is a must thing. So there should be a way to store e-voting transactions in a secure environment. Since traditional database storage is a centralized server based solution, it cannot be used to store e-voting transactions.

Blockchain is a distributed decentralized public ledger which can be used to store e-voting transactions securely. So it can be used as a substitution to a database approach. Transactions that are stored in a blockchain is publicly visible. But in e-voting scenarios all the voting transactions should be anonymous. [3] That means “To whom the voter voted should be publicly visible” but the details of the person who voted should not be publicly visible.

There are some blockchains that support for anonymous transactions. But there are other disadvantages as well in those blockchains. In this paper the solution is based on Ethereum blockchain which is not supported for anonymous transactions. The reason

for using that kind of blockchain was based on literature review. The concept called zero knowledge proof (ZKSnarks) was used to allow for the authentication of transactions without giving any personal information to the contract.

## **1.1 Background to the Research**

Since the e-voting have to run top of the network, the main challenge is to reduce the risk, it might cause. In the history, many solutions have been proposed based on the homomorphic encryption, blind signature, public key cryptography, visual cryptography and ring signature etc. As well as nowadays researchers are doing experiments to apply blockchain to the e voting application.

The topic of electronic voting has been discarded before the concept called blockchain [4] coming in to the industry. There were significant number of people who argues that electronic voting systems cannot be trusted enough to be used in government elections due to authentication problems and integrity problems of the votes. But everyone agreed on there is a need to introduce e-voting system due to paper based systems are outdated.

Many researchers were trying to propose an e-voting protocols and a set of people have proposed good solutions using blockchain based solutions. But they were not up to the level which can be used to fill the current research gap.

“Internet voting using Zcash” [5] , a paper provides Zcash [6] based solution to the e voting problem. The solution was suggested by T. Pavel and H. Tewari. In Zcash, There is a special kind of transaction called as private transactions. They have developed private transactions based e-voting solution. That means receiver and transaction details will not be available to the public. Hence it is easy to handle e voting transactions via private addresses by keeping the anonymity feature of the voters. But it is not capable of writing custom logic in Zcash [6] like smart contracts [7] in ethereum [8]. Hence this solutions doesn't provide good logic to handle this problem. And also under the future works section of this paper, they have mentioned that “The ethereum protocol has been established early on in the work as a potential

candidate to become the platform for our voting protocol”. According to these papers, we can make a conclusion like “Ethereum [8] will be the future of e voting problem”. According to these papers it is very clear that still there is a research gap in this area

## **1.2 Research Problem and Research Questions**

After suggesting many solutions also, the director boards, government still uses paper based voting methodologies for their elections. It seems like current proposed solutions still don't fulfill the requirements of the top level managers of the companies and presidents in countries. But normal people (people who voting) don't like the current existing voting methodologies. It can be easily understood by looking at the less participation of elections in recent years [2]. The traditional paper based voting methodologies requires voters to cast in appointed polling stations, which usually very time costly. So it is very clear that there is a requirement of secure e-voting system which can fulfill all the requirements in e-voting scenarios. And it is very clear that still there is a research gap in this area.

In summary, there is a need for a trustworthy, transparent, privacy protected, e2e verifiability [9] [10] , decentralized and multi-party secure e-voting system. More specifically the following sub research questions need to be addressed

1. What are the current industry practice, to make the e-voting transactions decentralized by using blockchains [4]?
2. How to make the e-voting transactions transparent?
3. How to protect the privacy of the voters who participate in the e-voting transaction?
4. How to make the E2E verifiability [9] in an e-voting transaction?

Nowadays many researchers trying to apply blockchain for many applications. [11] Among those one of their intention is to give a solution to e-voting problem using blockchain based solution. Some reasonable attempts were made by some researchers using ethereum [8] based solutions, and Zcash based solutions. But still there is no proper solution to cover all the needs of e voting system.

One of the needed feature of e voting is the transparency. When some voter voted for some candidate it should be transparent to the audience. That means there should be a mechanism to show that someone has voted for a particular candidate.

The voting details “who voted” should not be available to the public. Only thing that should be available to the public is “there is a transaction happening now and it is pointed to that particular candidate”. That means proposed solution should have a mechanism to handle privacy of the voter.

Verifiability [9]- the voter could be able to verify that their own vote has been casted as intended. The proposed solution should have a mechanism to view this verifiability [9] feature. With use of blockchain [4] , when there is a transaction, it returns the transaction address. By using that address the voter can check whether their vote has been casted as intended. So that problem will be automatically solved by using a blockchain to store e-voting transactions.

By looking at the sub research questions, a final research question can be generated like this.

- How to take a blockchain based approach, to develop a trustworthy, transparent, privacy protected, e2e verifiability, decentralized and multi-party secure e-voting system?

### **1.3 Justification for the research**

Previous researchers failed to produce a complete e-voting solution to this field. But this research contains e-voting framework that fulfils the requirement of voting scenarios. Mainly this research contains a solution to the voter privacy protection problem. According to the solution there is no interaction between the election authority and voter. That means only one set of information is sent to the verifier for verification, therefore there is no back and forth communication between the prover and verifier. [12] Here prover means the voter and verifier means the election authority. So any person can't detect who the voter was. This was done using zk-



SNARK [12]. And the tallying is not done by a person. It will be calculated by self-executing contract called smart contract [7]. There is no third parties involved in this process and this is more secure compared to the current existing paper ballot based solution in Sri Lanka.

## 1.4 Methodology

This study was first review various type of existing solutions to the secure e-voting problem. [13] At the second step, based on this previous works, suitable existing protocols were evaluated. At the third step some of suitable existing protocols were selected among existing solution pool. At the fourth step, it was taken a try to combine existing protocols. At finally a new protocol was developed by combining existing protocols to accomplish the above objectives of the Research.

According to the literature review which is explained in chapter 2, it is clear that future solution to the e-voting problem will be based on combination of Zcash [6] [5] and Ethereum [8]. Figure 1.1 represent the research approach used in this research.

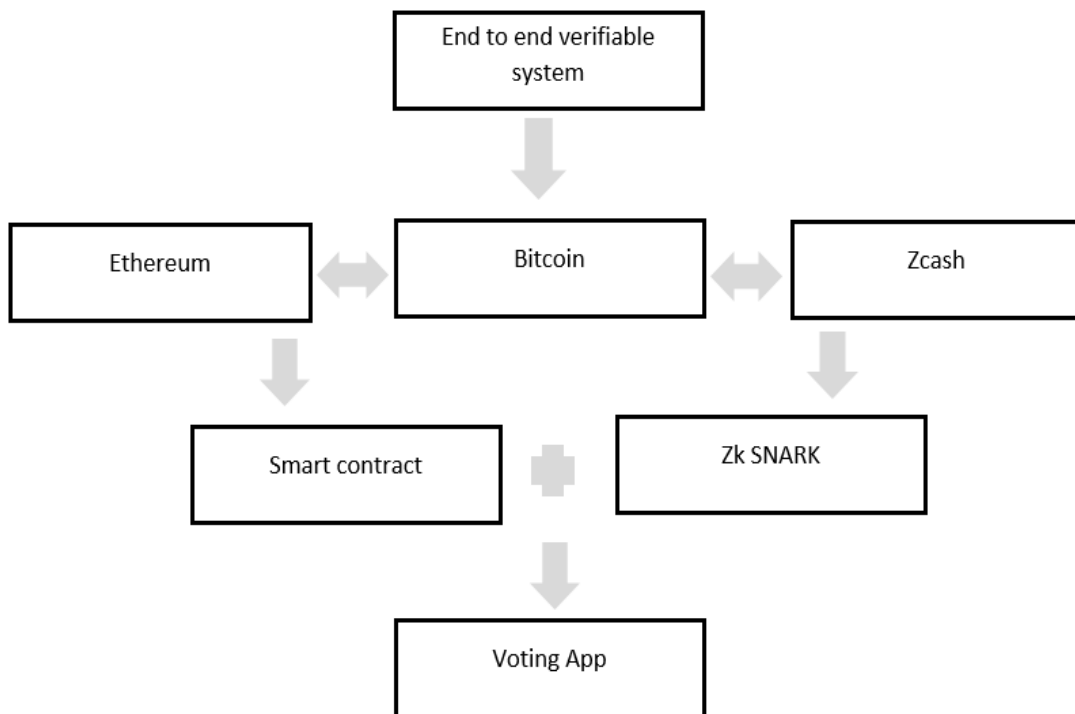


Figure 1.1: The research approach

There were some properties that cannot be addressed solely using the Blockchain, for example authentication of the voters requires additional mechanism to be integrated. For the authentication, Ethereum addresses and automatically generated secret phrase based solution was proposed. Then hash value of the secret phrase was stored at the smart contract for authenticating the voters. After the registration and voter registration verification, an arithmetic circuit need to be created by the election authority using hash values of all the voters. For the voter privacy protecting part, zk-SNARK based solution was proposed. According to the solution voters need to generate a proof which proves that they know a pre-image for a hash value which are stored at the arithmetic circuit. If a voter generate valid proof then he must be a valid voter. Verification of that proof is done without any interaction between two parties.

## **1.5 Outline of the Dissertation**

The thesis is structured as follows. Chapter two describes the voting problem we solve in this report as well as current existing methodologies regards to the domain and their problems and limitations. Chapter three describes the research design and the methodology more deeply. Chapter four demonstrate the implementation details of the blockchain based e-voting solution. Chapter five discuss the evaluation method and results obtained from the evaluation method. The chapter 6, which is last chapter, describes the conclusion and future works.

## **1.6 Definitions**

E-Voting: Electronic system that allows a voter to record his/her ballot electronically

Voter: Person who cast the vote

Candidate: A person being considered for some kind of position by an election

Ether: Programmable tokens that are used as currency on ethereum network

Gwei: Gwei is a denomination of ether ( $1 \text{ Ether} = 1 \times 10^9 \text{ Gwei}$ )

dApps: Decentralized applications

## **1.7 Delimitations of Scope**

All the transactions were done using ethereum test net called Rinkeby [14] and assumed that those test net blockchain is very similar to the Ethereum blockchain. All the e-voting transactions were tested using fake ethers and assumed the real transaction which can be done using real ethers is very similar to this process. The proposed solution is mainly focused on government elections, bill approval elections in the parliament and director board kind of multi-party elections. According to the solution only one candidate are allowed to represent one party. The solution is not focused on provincial elections kind of elections. Assume that all the voters have proper computer literacy to use the system as well as generate proofs.

## **1.8 Conclusion**

This chapter laid the foundations for the dissertation. It introduced the research problem, main research question and sub research questions. Then the research was justified, definitions were presented, the methodology was briefly described and justified, the dissertation was outlined, and the limitations were given. On these foundations, the dissertation can proceed with a detailed description of the research.

# **Chapter 2 - Literature Review**

## **2.1 Introduction**

Importance of developing a secure decentralized e-voting system has long been recognized by the industry. However, numerous electronic systems have been proposed and implemented, they were not used in the real environment. Many researchers were trying to propose an e-voting protocols and a set of researchers have proposed good solutions using public key cryptography, Ring signature, visual cryptography etc. But all the e-voting requirements were not satisfied by those solutions. In this chapter, describes the related works on e-voting and their limitations as well as drawbacks. Section 2.2 explains the related works which were done without using blockchain. Section 2.3 mainly focusing on explaining theories with regards to blockchain based e-voting. Section 2.4 describes related works which were proposed based on blockchain and their limitations and drawbacks.

## **2.2 Voting systems**

A preliminary literature review shows that past studies are primarily focused on particular type of solution such as homomorphic encryption [15] , End to end system [9], Ethereum based solutions [8] [16] and Zcash based solutions [6] [5] etc. Below it's able to see some previous solutions and their problems.

In 2000, e-voting has been used in US election, although it is an experiment in some area of Florida, it was a milestone in the development of e-voting. [17]

E2E systems: True voter- Verifiable elections. [9] In 2004, David chaum proposed a solution with E2E verifiability [9]. E2E verifiability means that the voter can verify that their own vote has been casted as intended and that the vote is accurately counted. Here he introduced a new form of receipt using visual cryptography. The receipt

contains two parts and when one part is laid top on other, then only can see the result. Only In the voting booth, the voter can see his or her choices clearly printed on the receipt. After taking one part of the receipt out from the booth, the voter can use it to ensure that the vote it contains are correctly included in the final tally. But, because the choices are safely encrypted, before it is removed from the booth, the receipt cannot be used to show others how the voter voted. According to the chaum's solution, after you input your choices using a touch screen or other input method, a small device that looks like a cash register printer generate printout (part of which will become your receipt).The printouts lists the name of the candidates you chose along with their party affiliations. If your receipt is correctly posted, you can be sure that your vote will be included correctly in the tally. A receipt that is not properly posted is physical evidence of a failure of the election system. According to the paper this approach is still some form of paper based approach.

E-voting using homomorphic encryption scheme. [15] Homomorphic encryption is a form of encryption that allows computations to be carried out on cipher text, thus generating an encrypted result which, when decrypted, matches the results of operations performed on the plain text. Paillier cryptosystem is asymmetric algorithm for public key cryptography. An important feature of the Paillier cryptosystem is homomorphic property. It allows the user to register itself so that it can cast a vote. Voter registration is done at the registration system where he enters all his required details and is given a unique voter id. Using this unique voter id and password entered by him, will be able to login at the client side voting page. Since this is a centralized server based application, it is hard to guarantee the security of this homomorphic encryption scheme application also.

E-voting protocol based on public key cryptography. [18] This solution comprises of 3 stages. Those are system access control process, voting process and collecting data process. At the first phase user will be given a public key via a text message by using the mobile details which was given at the user registration. At the registration phase the voter data will be saved in a special election server then pass that data to mobile phone Company for advance process. At the second phase users should enter the received public key and submit his selected candidate. Then the voter data will be

encrypted by implementing RSA encryption algorithm with the received public key. Then the cipher text will be sent to the government election server. At the final phase received cipher text will be decrypted by using RSA private key and the final result of the election will be announced. This solution is also a centralized server based solution. So we cannot make 100% trust on a single server based solution.

Anonymous voting by two-round public discussion [19]. This solution is bit different from above other solutions. Because it requires no trusted third parties or private channels. It is able to execute the protocol by sending two round public messages. This solution has self-tallying functionality. Compared with the other solution it is very efficient solution.

Efficient maximal privacy in boardroom voting and anonymous broadcast [20]. In this solution Kiayias and young introduced the concept of elections with perfect ballot secrecy. That paper consists of two contributions to the e-voting research area. First contribution is a new voting scheme. Second contribution is to construct and anonymous broadcast channel with perfect message secrecy.

Self-tallying elections and perfect ballot secrecy. [21]. The primary objective of this solution was to protect the privacy of users. This paper introduced three new contributions to the e-voting domain. Those are Perfect ballot secrecy, Self-tallying, Dispute freeness. According to this solution final vote counting process is allowed only after voting is done by all the remaining voters. The implementation behind this was a special method called multiparty computation. According to the solution votes are stored as part wise with many nodes and when the election ended up, the final results will be formed using a function which is focused on adding those multi parties. After the election anyone can check whether the tallying phase is correct or not.

## 2.3 Key concepts of Blockchain based Voting

**Blockchain:** Blockchain [4] is a public distributed immutable ledger that is capable of storing e-money based transactions. But nowadays it is used for storing application specific transactions also. Most important feature of the blockchain is, it allows value exchange without the need for trust or central authority. There is a special method to check validation as well. It is needed to order the transaction (timestamp), to validate them and get prevented from the double spending problem. Hence the network select some random transactions from the pool of transaction (unconfirmed transaction) and order them by putting together into groups called blocks. Blocks are therefore organized in to one after the other in time related chain. That gives the name to this data structure: Blockchain. Each block contains the hash value of previous block to maintaining the immutable property of the blockchain. To add a new block to the blockchain, every node have to find the answer to the cryptography non reversible hash function. It could take about a year for a typical computer to guess the right number. Since there are lot of nodes in a blockchain network, it is possible to solve a block averagely once every 10 minutes. Then it broadcast that block to the whole network saying that this is a valid block and containing valid transactions.

**Transactions:** In a cryptocurrency transaction, it is needed to sign the transaction by private key of the sender. And the new transaction should contains previous transaction id. That means hash value of the previous transaction and public key of the new owner. And also signature of the previous owner. Hence looking at new transaction, it is able to detect sender and receiver. In our case it is capable of storing e voting transactions in a blockchain since it provides immutable property. Especially in e voting scenarios it is not good to trust any third parties. Since blockchain is a decentralized thing, the most suitable data structure to store e voting transactions is a blockchain.

**Proof of work** [4] is a requirement to define an expensive computer calculation , also called mining , that need to be performed in order to create new group of trustworthy transactions (the so called block) on a distributed ledger called blockchain

Networks: There are different kind of networks in ethereum and other blockchains. Main network is the ethereum public blockchain where the production applications running and real transactions happening. It is accessible by anyone. Mining process is going on the main net. Testnet is also a public blockchain but for testing purposes only, not for the productions. Ex: Ropsten, Rinkeby and Kovan. A mining process is going on the test net also. An ethereum network is a private network if the nodes are not connected to the main network nodes. Mining process is not going on private networks. Private deployment might use different consensus, typically does not need incentives for participants.

Accounts: To communicate with the blockchain and to write a smart contract we need to have a valid accounts. There are two types of accounts in ethereum. Those are externally owned accounts (EOAs) and contract accounts. EOAs has an ether balance and is capable of sending transactions (Ether transfer). These accounts are controlled by private keys and has no associated code. Contract account is type of account has an ether balance and associated code. Code execution is triggered by transactions or messages (calls) received from other contracts. Every time a contract account receives a transaction, its code is executed as instructed by the input parameters sent as part of the transaction.

Public key cryptography: Public key cryptography [18] is a way of encrypting and decrypting messages which uses two keys, one public and one private. Public key is used to encrypt the data. Data encrypted with the public key can only be decrypted by using the private key. Private Key is used to sign the data. Data signed with private key can be verified using the public key. The private key cannot be derived from the public key. But the public key can be derived from the private key.

Smart contracts [8]: Ethereum has some special feature called smart contract [20]. Smart contracts are self-executing with the terms of the agreement between buyer and seller being directly written in to lines of code. Contracts are written in high level scripting language like solidity, serpent or LLL. Every contract that reside on the ethereum blockchain is stored in a specific format called EVM bytecode which is an ethereum specific binary format



Metamask: [22] is a browser extension that lets ethereum users run dApps without being part of the ethereum network as an ethereum Node. Metamask manage the ethereum wallet, which contains your ethers, and allows you to send and receive ethers a dApp of interest. It is easy to use ethereum network via the metamask browser extension.

zk-SNARK: [12] A zero knowledge proof allows one party, the prover, to convince another party, the verifier, that a given statement is true, without revealing any information beyond the validity of the statement itself. A zk-SNARK is a variant of zero knowledge proof that enables a prover to succinctly convince the any verifier of the validity of a given statement and achieves computational zero knowledge without requiring interaction between the prover and the verifier.

## **2.4 Voting and privacy solutions based on Blockchains**

Zcash based approach with zk-SNARKS. [5]. Zcash support both anonymous and transparent transactions as it has two types of addresses differs from the bitcoin single address. Those anonymous transactions are called as private transactions. One of the biggest difference between Zcash and ethereum is the proof of work system, where Zcash relies on zero knowledge proofs. The private transactions of Zcash are based on the zero knowledge proving system. To facilitate these private transactions without disclosing to the others, Zcash implements zk-SNARKS (Zero Knowledge Succinct Non-Interactive Argument of Knowledge). Zcash contribute our e-voting problem by protecting the anonymity of users who participate for a particular transaction. But Ethereum growing very fast and challenging to the Zcash. Ethereum supports creation of contracts which are operated by the EVM. Contracts are the agents that bring about the generic functionality of Ethereum and allow one to create custom behavior for one's blockchain application.

The Ethereum blockchain [8] works under concept called smart contract. Smart contracts are meaningful pieces of codes, to be integrated with the blockchain [4] and executed as scheduled in every step of blockchain updates. Developers started to apply the concept behind the Ethereum platform to the other applications because it was very easy to integrate the custom logic to the Ethereum blockchain [8]. But in our case Ethereum doesn't support to protect the privacy of the voter who voted for a particular candidate. There are many researches that use ethereum to e-voting system. But they failed to protect the privacy of the voter. The future ethereum aims to make use of zk-SNARK to add privacy and anonymity of transactions. [23] [24] So the Future of the e-voting solution will be combination of Zcash [6] and ethereum.

Decentralizing privacy: using blockchain to protect personal data. [25] In this paper their main target was to keep user data themselves without giving it to third party storage service. So their main target was to protect the user privacy. They mainly focused this research on mobile app services and mobile users. According to their solution when someone install new application, permissions given to the application service will be stored in the blockchain. All the data exist in off chain. When the service need to access particular data from the user, then they have to query the blockchain. If particular service has access to particular data, then only they can access the user data. User can change that permissions any time with making new transactions on the blockchain.

On November 15, 2017, the first digital zug id was officially registered on the Ethereum blockchain in front of a live press audience. Uport [26] launched a pilot program to register residents' IDs on the blockchain to unlock access to government e-services like online voting and proof of residency [26].

## **2.5 Summary**

This chapter mainly focused on reviewing previous similar works which was done without using blockchain and with using blockchain. Their advantages, disadvantages as well as the limitations were discussed in this chapter. Background theories which is needed to understand the blockchain also were discussed in this chapter. Privacy protection methods which were taken by previous researchers and how will be the future of this domain also were discussed.

# Chapter 3 - Design

## 3.1. Introduction

This chapter covers proposed solutions for the research questions and step by step design phases towards the final research aim. Research design steps consists of six phases named as preparation phase, Registration phase, Registration verification, Voting phase, tallying phase, Verification phase. And research approach will be deeply explained in this chapter and research design and final product architecture also will be explained.

## 3.2. Research Design

To reach the final step, several steps were passed and some steps were dropped since it is not the most suitable way of doing it. Some steps were deeply analyzed since it contributed to take the research towards the aim. The Figure 3.1 displays the research design steps.

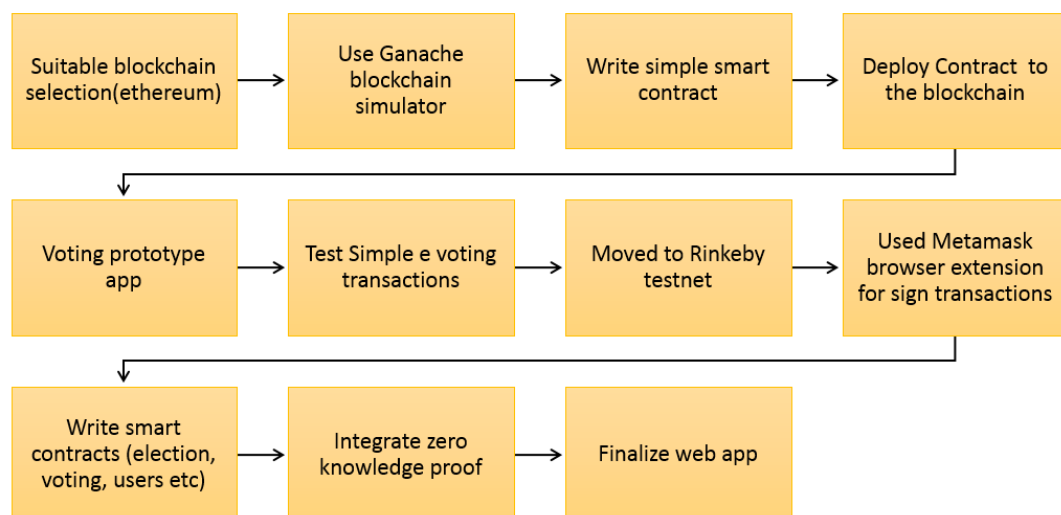


Figure 3.1: Research Design Steps

The one of primary requirement of e voting system is end to end verifiability [9]. End to end verifiability means voter should be able to check whether their vote has been casted as intended. According to the literature review, to achieve end to end verifiability previously the developers have used public bulletin boards to publish voter details. Bulletin board is some form of notice boards. But in large scale elections, this is not a practical solution since it cannot be publish large amount of details in a bulletin board.

Then they have used authenticated web page to publish voter details rather than using bulletin board. Since it is a single server base solution, it is not able to guarantee the consistency of data. Web page based solution is not the exactly what they want.

In 2009, satoshi Nakamoto implemented the first blockchain database. [4] Then with the era of blockchain 2.0, researchers tried to check whether is it possible to store e voting transactions in a blockchain rather than publishing it in a web page.

Since blockchain is decentralized thing and it's safe and secure eco system, took developers and researchers towards developing blockchain based applications for e-voting systems.

The research approach started with above background. Then the problem exist at that time was to check whether it is possible to use Bitcoin blockchain for e-voting systems. Then they realized that, with the Bitcoin network, it is possible to assume that Bitcoin sender is the voter and Bitcoin receiver is the Candidate. Then it is possible to calculate number of transactions for bitcoin address of Particular candidate and take it as a voting count. But the ecosystem of bitcoin blockchain doesn't provide best environment for developing applications. Bitcoin blockchain doesn't allow to write custom logics. Hence, main problem in bitcoin blockchain is logic resides in off-chain.

Then researchers were moving towards Ethereum [8] based solution. The ethereum blockchain focuses on running any program written in something called smart contracts [7], in Ethereum virtual machine. The main strength of ethereum blockchain is the smart contracts. It facilitates to write contracts between the sender and receiver.

In our case Voter and candidate. But the problem in ethereum is it doesn't provide private transactions. Because of that the details like who voted, to whom the voter voted for etc. will be publicly visible.

Then researchers were moving towards Zcash [6] like coins which provide private transaction and enables to hide transaction details from the public. But the problem in Zcash is, it also doesn't provide smart contract like concepts to write custom logics.

In this paper we decided to take the logic behind private transactions of Zcash and apply it to ethereum. Since the logic behind Zcash [6] is zk-SNARK to protect the privacy of users and logic behind ethereum was to write immutable contracts, the combination of both of this will work best for e-voting scenarios. Then our intention was to integrate zk-SNARK and smart contract together and apply that into e-voting problem.

According to that solution, securely vote storing problem, verification problem etc can be solved using blockchain but it's unable to use blockchain directly for user registration and privacy protection. If the voters were asked to download wallet or do some configuration to communicate with the blockchain then it will not be practical. But if it can be easily distribute ethereum addresses using browser extension application called metamask, [22] that browser extension is the most suitable thing for this component. Our intention was to give digital identity [27] to each and every voter using ethereum addresses. With use of the digital identity, it is able to identify each voter uniquely. Because according to the proposed solution identity will be hidden to the smart contract. But smart contract can check whether this is a valid identity or not. The logic that trying to implement that kind of privacy protected method is zero knowledge proof.

### 3.3 High-level architecture of off-chain and on-chain components

Here, the connection to the blockchain from user interfaces were handled via web3.js (Ethereum JavaScript API). It's a collection of libraries which allows to interact with a local or remote ethereum node using HTTP or IPC connection. The figure 3.2 displays the high level architecture of on-chain and off-chain components.

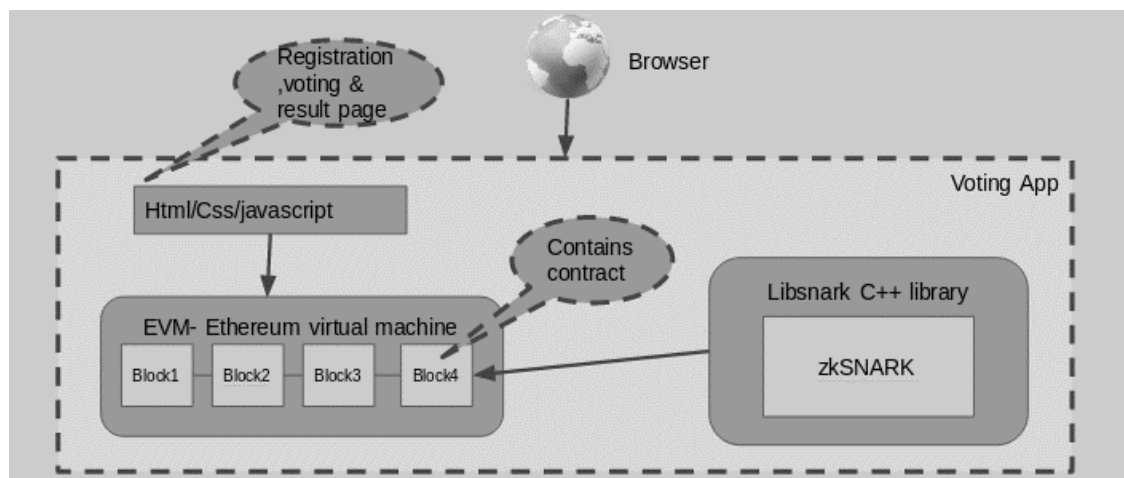


Figure 3.2: High level architecture of On-chain and off-chain component

The solution consists of six contracts and those are running in ethereum virtual machine. To communicate with the blockchain, it should contain with pre specified interfaces called as ABI definitions. Libsnark c++ library is an implementation of zkSNARK. It was used to protect voter privacy in smart contracts [8]. The proof was generated using an ethereum tool kit called as Zokrates. Html,css react js files are running as off-chain components.

The contract called as “Authentication.sol” is used to store initial user details and maintain logic for sign-in to the system and maintain the user profile. The ethereum address of logged in account and address of the default metamask account should be same for every request to the blockchain except the request which is used for voting. Because in the voting request, different address will be used.

The contract named as “voter.sol” consists of the logic for voter registration and voter state update functions. After the registration for an election, they can track their registration requests. Their registered ethereum addresses will be converted to a verified accounts after the accounts are verified by required legislation authorities. With every New Year the user details will be updated and verified for maintaining the consistency. For verification process voters have to go for the grama nildhari and have to handover valid documents within defined time period from their temporary registration. Before registration starts at each year the verified status of all the voters will be set to null. The grama niladhari maintains two states of voters called as “new names recommended for acceptance” and “names recommended for rejection”. Then the account verification/rejection process will be done via the district office. Voter.sol smart contract will be used for these kind of state update things. In this report detailed description for each phase will be provided.

The contract called as “candidate.sol” consists of the logic to manage candidates. Candidate registration is done via this contract. The verification process of the candidates is done same as the voter verification. Only the verified candidate will be used to create Election smart contract.

The contract named “Election.sol” is used to create the election. To create an election, only the valid candidate list will be needed. No need of valid voter list. Because authentication process of voters will be done via zero knowledge proof. The election contracts consist of the voting functions as well as tallying functions

The contract named as “sha256hashTest.sol” consists of the verification function to take the parameters of the zero knowledge proof. After taking those parameters those parameters will be passed to the verifier.sol smart contract.

The figure 3.3 displays the contracts which were used in the e-voting solution and they are categorized according to their functionality.



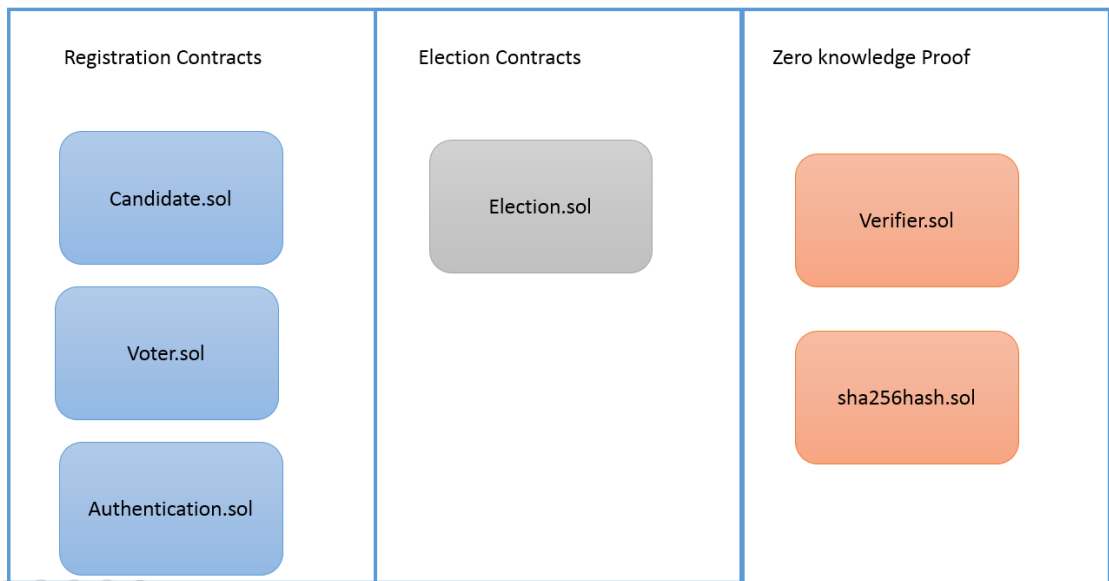


Figure 3.3: Contracts in the voting system

The contract named as “verifier.sol” is used to verify the zero knowledge proof. This contract is generated by the Zokrates ethereum tool kit. If the arithmetic circuit is changed, the verifier contract also should be changed. Since, at a new election registered voters changes, the arithmetic circuit should also be changed. Hence verifier contract should be changed. The figure 3.4 displays the high level architecture of contracts and user interface components.

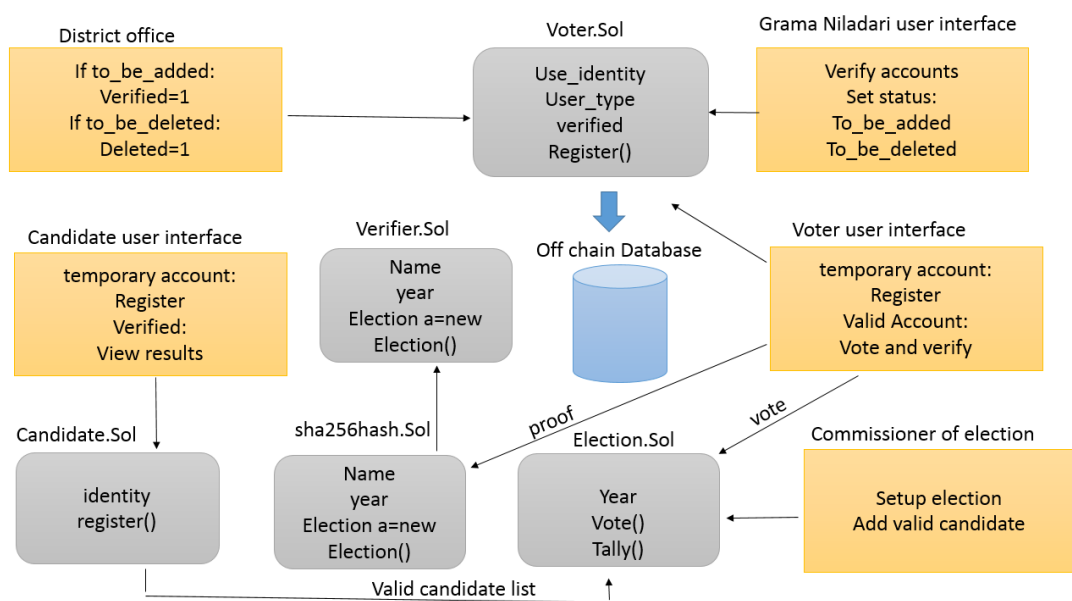


Figure 3.4: High-level architecture of contracts and user interface components

The proposed system consists with following entities. Voters ( $V_i$ ), candidates ( $C_i$ ), Registration Authority (RA), Election Authority (EA).

**Voters ( $V_i$ ):** List of eligible users that is selected via Registration Authority

**Candidates ( $C_i$ ):** List of candidates that is selected for election.

**Registration Authority (Grama niladhari and district offices):** verify user details of users who were registered metamask [22] ethereum address.

**Election Authority (commissioner of election):** setup a new election

The e voting system can be divided into 6 phases

1. Registration phase
2. Registration verification phase
3. Election Preparation phase
4. Voting phase
5. Tallying phase
6. Verification phase

### 3.4. Registration phase

1. Users who hope to participate the election need to download the metamask browser extension. By downloading and creating account there, citizens will get an ethereum address (id).
2. To register their id in e-voting voter.sol contract, the citizens sign-in to the e voting portal by creating temporary account with their ethereum address.
3. The e voting web portal is requesting to register for the election with their ethereum address. The address which is used to sign the election registration transaction and address which is used to logged-in to portal should be same.
4. After clicking registration button users will get a secret phrase. Voters need to get the hash value of the 'big integer representation of the secret phrase'. The

secret phrase and the big interger representation of secret phrase should be stored in a secure environment for future use.

5. At the next interface users need to submit the personal details and hash values of the secret phrase.
6. Since the citizen's personal information is submitted from their still unverified ethereum address, this submission requires in-person registration verification. Users have to produce valid documents to the grama niladhari within limited time period.

Note:the registration of candidate process is similar to this. But verification process and document collecting process is not done by the grama niladhari and district offices. It is done by the commissioner of election

### **3.5. Registration verification phase**

1. After registration, citizens will be given limited time period to verify their details by registration authority. It is able to train grama niladhari to do this registration review task and collect required document from the voters. To do the verification task, citizen should provide one of the official government document (ex: NIC). Then grama niladhari verify the documents and send it to the district office via the system using scanned copies. Then grama niladhari login in to their portal and update the registered voter ethereum address to "recommended for acceptance" state or "recommended for rejection" state.
2. District office registration authority sign in to the web portal admin panel using their ethereum address which exist in the metamask. This enables the officers to review the citizen submission with the information they provided against their documents in-person. At this step voter acceptance or rejection will be done and the smart contract will be updated with new states of the voters
3. After registration closes, registered voters need to wait till the election to start.

### **3.6. Election preparation phase**

1. Election authority need to collect the hash values of the verified voters and need to create the arithmetic circuit using them with the Zokrates ethereum tool kit.
2. Then need to compile the arithmetic circuit which is written in Zokrates high level language to machine readable code
3. Then need to generate the verification and proving key which is used to prove and verify zero knowledge proof.
4. At next step verifier.sol smart contract will be generated with verification key exist inside of the smart contract.
5. Then the proving key and machine readable arithmetic circuit will be uploaded to the web portal to download it by voters.
6. At finally election authority should update the election contract by giving it a meaningful election name, year and candidate list to the contract and should start the election.

### **3.7. Voting phase**

After login to the web portal via valid ethereum address exist in the metamask, registered voters can cast their vote. The design of voting phase facilitates to protect the privacy of voters by hiding the voter details and identity. The zero knowledge proof helps to do that.

A zk-SNARK is a variant of zero knowledge proof that enables a prover to succinctly convince the any verifier of the validity of a given statement and achieves computational zero knowledge without requiring interaction between the prover and the verifier. So in this paper the aim is to notify the validity of user details in an e-voting transaction to the smart contract without revealing user identity.

There are three parts in zk-SNARK. G, P and V. G is called as generator, P is called as prover and V is called as verifier. Thirds party should run the generator G by giving

Program C and random number lambda as input parameters. It will output the proving key  $p_k$  and verification key  $v_k$ .

$$(p_k, v_k) = G(c, \text{lambda})$$

Then the generator will share  $p_k$  and  $v_k$  with sender and receiver. That means the prover and verifier. Then the prover will generate the proof by giving  $x$ ,  $w$  and  $p_k$  as the input. “ $x$ ” is the publicly available input . “ $w$ ” is the witness.

$$\text{prf} = P(p_k, x, w)$$

Then that proof will be sent to the receiver that means to the verifier. At the verifier, it gives input as  $v_k$ ,  $x$  and  $\text{prf}$ . It will return output as true if the proof is true.

$$V(v_k, x, \text{prf}) \rightarrow \text{return true/false}$$

So in this case prover will be the client side code and verifier will be the smart contract [7]. Witness value created by giving correct secret phrase to the arithmetic circuit. The public variables are output from the arithmetic circuit which is 0 or 1. Those details will be used to generate the proof. So the proof doesn't contain any information about the secret phrase. But It proves that, this proof contains the knowledge for pre-image which exist in the arithmetic circuit.

The client side need to convince the verifier by generating a proof like “the prover know the secret phrase of one of the vote without revealing any information of the voter to the verifier except that statement is true. Following are the steps that need to adhere in this phase.

1. Download the machine readable arithmetic circuit and the proving key.
2. Generate a valid witness by giving correct secret phrase that matches the sha256 hash values which exist inside the arithmetic circuit.
3. Create a proof by giving witness and the public inputs.
4. Submit the proof via the web portal.

5. If and only if the proof is correct voters will be redirected to voting interface.
6. In this step voters need to change their metamask account to another account which is not similar to registered ethereum address. Because to do a transaction an ethereum address is a must thing to sign the transactions. Hence in here need to use address which is unknown to the smart contract. In this step the system know this is valid user because of the submitted valid proof. So to cast their vote, the voters can use any account they have.

### **3.8. Tallying Phase**

The tallying process is not manual thing. It is self-executing in the Election smart contract. Hence it is trustworthy process. When voters cast their votes, results will not be displayed to them until the election ends. After the election ended up, the calculated votes for each candidate will be displayed.

### **3.9. Verification phase**

After casting their votes, public people also can check whether the casted votes are casted as intended by verifying. To verify their votes they need to have transaction hash which they got at the voting phase. Here the verification process checks whether their transaction is included in a block or not.

### **3.10. Summary**

This chapter provided detailed description on the research design, high level architecture of the on chain and off chain components of the system as well as design of the smart contracts. And the 6 phases of the voting system was explained step by step in this chapter.

# Chapter 4 - Implementation

## 4.1. Introduction

This chapter elaborates the implementation details of the proposed solutions. Section 4.2 describes the software tools which were used in this research. Section 4.3 describes the secret phrase generation process and section 4.4 describes sha256hashcalculation process and the section 4.5 describes the proof generation process and the section 4.6 describes the smart contracts that is used for the registration, voting, verification etc.

## 4.2. Software tools

At the very beginning of this research, a software called Ganache was used as alternative to ethereum homestead blockchain. The ganache is a blockchain simulator and it comes with ten test accounts integrated which has ten private and public key pairs. Each account consist with hundred fake ethers for testing purposes. Then the ethereum testnet called Rinkeby is used. Since ganache is running in our local machine and it is not running on distributed environment, Ganache is replaced by Rinkeby testnet. Rinkeby testnet is more similar to ethereum homestead blockchain and can easily view the blockchain transaction details and smart contract details via [28]. To download the Rinkeby testnet another software called “Geth” is used. Geth is implemented in Go language and it enables to run the ethereum testnet in local machine after downloading.

At the early stages of the development, web3.js is used to communicate with the blockchain and deploy smart contracts. To compile the smart contracts a compiler called “solc” is used. Then the framework named as truffle was used. Because it is very easy to handle smart contracts using truffle. To deploy smart contract with downloaded version of Rinkeby testnet, it has to be deployed smart contract locally and then need to replicate. That process takes some time and our machine also will be

part of mining. Hence an IDE called “remix” is used. With that solution no need of downloading the whole blockchain to deploy smart contracts. Remix is a web based IDE which is used to deploy smart contracts to the blockchain via online web interface and for testing purposes of function written in smart contract. Smart contracts were written in programming language called “solidity”. Rinkeby testnet doesn’t come with pre-defined accounts like it was in Ganache blockchain simulator. So the person who use the blockchain have to add accounts himself. So in this research, at very early stage, a browser extension called metamask is used to communicate with the blockchain.

To generate the zero knowledge proof [12] an ethereum tool kit called as Zokrates [29] was used. It run on docker container and helps to calculate the sha256hash generation process as well as the zero knowledge proof generation process. There were many off chain components in this system. Those were developed using react js javascript framework and jsx was used with react to render the html components.

### 4.3. Secret phrase generation

At the registration phase of the voter, voter will be given automatically generated secret phrase. It’s a combination of the upper and lower case letters which has 64 characters. Figure 4.1 displays the secret phrase generation code block.

```
randomString() {  
    var charSet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz';  
    var randomString = '';  
    for (var i = 0; i < 64; i++) {  
        var randomPoz = Math.floor(Math.random() * charSet.length);  
        randomString += charSet.substring(randomPoz,randomPoz+1);  
    }  
    return randomString;  
}
```

Figure 4.1: Secret phrase generation



As an example, the following phrase is a generated random secret phrase.

dPwIxicthNYvNgThmWZqNSyWtKeImZjGIpRgERrAryiMlxLuLmbjxgbwudnbXYU  
S

This phrase has 64 characters and need to encode it to 512 bit binary representation since the voter need to calculate the sha256 hash value of the secret phrase. The input to the sha256 hash is 512 bit and output from the sha256 hash is 256 bits. This is the encoded 512bit version interpretation of the above secret phrase.

011001000101000001110111010010010111100001101001011000110111010001101  
00001001110010110010111011001001110011001110101010001101000

011011010101011101011010011100010100111001010011011110010101011101110  
10001001011011001010100100101101101010110100110101001000111

010010010111000001010010011001110100010101010010011100100100000101110  
01001111001011010010100110101101100011110000100110001110101

010011000110110101100010011010100111100001100111011000100111011101110  
10101100100011011100110001001011000010110010101010101010011

It is represented as 4 parts with each part is interpreted with 128 bit numbers. To calculate the sha256 hash via Zokrates [29] ethereum tool kit, input parameters should be 254 bit maximum length. That's the reason for dividing 512 bits version of the secret phrase to four parts. To calculate the sha256 hash, those four parameters should be given as big integers. The following is the big integer representation of the secret phrase.

133340602754838679641981343934269838440  
145339415761135797541089420193811819079  
97616852279997134974522228037740153973  
101589284148823472461439080844888724819

The following function in figure 4.2 is used to convert the 512bit interpretation of base 2 binary number to base 10 big integer number

```
convertBase(bigint, inputBase, outputBase) {
    var inputValues = this.parseBigInt(bigint, inputBase),
        outputValues = [], //output array, little-endian/lsd order
        remainder,
        len = inputValues.length,
        pos = 0,
        i;
    while (pos < len) { //while digits left in input array
        remainder = 0; //set remainder to 0
        for (i = pos; i < len; i++) {
            //long integer division of input values divided by output base
            //remainder is added to output array
            remainder = inputValues[i] + remainder * inputBase;
            inputValues[i] = Math.floor(remainder / outputBase);
            remainder -= inputValues[i] * outputBase;
            if (inputValues[i] == 0 && i == pos) {
                pos++;
            }
        }
        outputValues.push(remainder);
    }
    outputValues.reverse(); //transform to big-endian/msd order
    return this.formatBigInt(outputValues, outputBase);
}
```

Figure 4.2: Base 2 to Base 10 conversion

## 4.4. Sha256 hash calculation

To calculate the sha256 hash of the secret phrase, voters will be given an arithmetic circuit called as sha256generate.code. Figure 4.3 displays the source code of arithmetic circuit which was written to calculate the sha256 hash values of the secret phrases.

```

import "LIBSNARK/sha256packed"

def main(private field a, private field b, private field c, private field d) -> (field, field):
    h0, h1 = sha256packed(a, b, c, d)
    return h0, h1

```

Figure 4.3 : Sha256generate arithmetic circuit

Using the arithmetic circuit voters will compute the 256 bit hash. Since return parameter maximum length is 254 bits, the output will be divided in to two parts and will be returned as big integers.

The following phrase is the output from the above circuit

```

~out_1 327485948427890063163657922405583843277
~out_0 73098509705847516637526624146845665259

```

## 4.5. Generate the proof

Voters need to generate a proof to convince the verifier that they know the secret phrase for above hash values which is existing in the second arithmetic circuit called as sha256hash.code

This is the arithmetic circuit which is created using submitted hash values of the voters. Here it contains only one hash. But if there are three voters then need to write the arithmetic circuit with three hashes by implementing logic with ‘OR’ gates. Since there are no OR gates in the zokrates high level language, it will be needed to write another logic to do that. Figure 4.4 displays the arithmetic circuit which was used to generate the witness.

```

import "LIBSNARK/sha256packed"

def main(private field a, private field b, private field c, private field d) -> (field):
    h0, h1 = sha256packed(a, b, c, d)
    h0 == 73098509705847516637526624146845665259
    h1 == 327485948427890063163657922405583843277
    return 1

```

Figure 4.4: Arithmetic circuit of proving pre-image for sha256hash

If the preimage for the above function is correct, it will generate the witness file. Then using the witness file and the public outputs, the generation of the proof will be done. The following is the proof generated for the above circuit. Figure 4.5 displays the generated proof.

```

{
  "proof":
  {
    "A": ["0x2f6a568985b4b6c879780b454ab1ab5d5c2b9ff8f8e0dbcac29e18b333d2db65",
    "0x152edca682582683410a8e06ca99bca32a466cb210f2fcdac8cacc0b3bc07fa6"],
    "A_p": ["0x24ba60e4da990fa5316c26431379f391544d6eff7d6978a518e02a1dbd388a9d",
    "0x1c41f06294dce2c445a7a7a0c1c8b79b9214ef069061b42cc594bddd178cd645"],
    "B":
    [
      ["0x9f5a2850efffa29f833527b4cc28cf385bc08b36d3c18e5a0aac583e517eb15",
      "0xd0f7c0a117a9c86e7a7063d5172c7e8c198cad7dd5292b56d06f16f96c06067"],
      ["0x216b16f387e3093f35a1e3e366a6e78b025189e7bdbb1a801282c8e5b5b6a2ea",
      "0x25ea4d278118362dd1683f21b2f299b42ac59a1bcd76476f7bd6d9a145e8b856"]],
    "B_p": ["0x2492b12772269abdd3b146e6ba805963b97db6acde1fec0a1d0e842b4688e866",
    "0x18f99452069d75bf61c304954f8d34063bd80d613cc1b2ecabe6bc9b4282e33"],
    "C": ["0x230a0b17b211c4f338895640264778dcecb8dfeccaa58b6849fe56b7897a376f",
    "0x2cba59f9fb37fccb17e7441cdba787b841a65ac17edcb1fc19ca68ea8eabb380"],
    "C_p": ["0x210b4b47c7b3cffe40d64b2bc759218008d102a5f361fe987b5e7c281d294097",
    "0x3053732fc5b3028e2295a3a920536c4665e54b0755b9c82d44b1ac0b5392108b"],
    "H": ["0x22b8cb4166a4b37638e93b6b7e1ce3c113364f0298c97e9bc83ef66ff1bd2074",
    "0x29c062865e212ed6ddb71c8265bf47848491cd6fe6ed569297fab9d5b3bd635"],
    "K": ["0x1e2abcfacfee992d0c7a7cda315662c7b947ba1bd867208acf26d276002c296e",
    "0x23af9abecd494c8b35847912af1b7e0a132c67aec6968cef94f1ea4e3d70974"]
  ],
  "input": [1]
}

```

Figure 4.5: The proof

If there are more than one voter in the system, the following circuit was used by using array of hash values of the voters. In this example code, there are two voters. If there are more than two voters, can write the arithmetic circuit by adding voter's hash values to the array.

```
import "LIBSNARK/sha256packed"

def main(private field a, private field b, private field c, private field d) -> (field):
    field[2] hash1 = [73098509705847516637526624146845665259,113738447700479173714816867410493084445]
    field[2] hash2 = [327485948427890063163657922405583843277,38770939290833947590945745017321709757]
    field result1 = 0
    field result2 = 0
    field z = 0
    field t = 0
    h0, h1 = sha256packed(a, b, c, d)
    for field i in 0..2 do
        result1 = if hash1[i] == h0 then 1 else 0 fi
        result2 = if hash2[i] == h1 then 1 else 0 fi
        z = if result1 == 1 then result2 else 0 fi
        t = if z == 1 then 1 else 0 fi
    endfor
    return t
```

Figure 4.6 : Arithmetic circuit for proving pre-image for list of hashes

## 4.6. Smart Contracts

Voter.sol is the contract which is used to register voters. Initially the verified status of voters are false. That mean when they register, they will get temporary account. After providing the required document, the grama nildhari updates the to\_be\_added status to true. Then district office review it and set to\_be\_added status to false as well as verifies status to true. Then the voter will get verified account for particular election. After the election, again their verified status is set to false. This is the structure which is used to store voter details. Figure 4.7 displays the voter details structure in voter solidity contract.

```

struct VoterDetails {
    bytes32 name;
    bytes32 nic;
    bytes32 hashOfSecret;
    bool submitted_to_review;
    bool to_be_deleted;
    bool to_be_added;
    bool deleted;
    bool verified;
    bool temp_registered;
    bool voted;
}

```

Figure 4.7 : Voter details structure

This is how the reset is done for a voter. In here all the parameters will be set to default values. Figure 4.8 displays the function for resetting a user account to default state.

```

function reset(address voterAddress) public{
    voters[voterAddress].submitted_to_review=false;
    voters[voterAddress].to_be_added=false;
    voters[voterAddress].to_be_deleted=false;
    voters[voterAddress].deleted=false;
    voters[voterAddress].verified=false;
    voters[voterAddress].temp_registered=true;
    voters[voterAddress].voted=false;
}

```

Figure 4.8 : Reset user account

Candidate.sol is the contract which is used to register candidates. At initially, the accepted status of the candidate will be false. The commissioner of election review these list and update their status. When creating election contract, only accepted candidates are passed as constructor parameters. This is the structure which is used to store candidate details. Figure 4.9 displays the structure which was used to store candidate details in the candidate smart contract.

```

struct CandidateDetails {
    bytes32 name;
    address addr;
    bytes32 nic;
    bytes32 party;
    bool doesExist;
    bool accepted;
}

```

Figure 4.9 : Candidate details structure

Election contract consist of the voting functions and total votes calculate functions. When particular voter casted his vote, the vote count of the candidate will be increased by one. The voters are given only one chance. Hence they can't vote more than one in a single election. Figure 4.10 displays the some functions which are exist in the election smart contract.

```

function totalVotesFor(uint candidate) view public returns (uint8){
    require(validCandidate(candidate));
    return votesReceived[candidate];
}

function voteForCandidate(uint candidate) public {
    require(validCandidate(candidate));
    votesReceived[candidate] += 1;
}

function validCandidate(uint candidate) view public returns (bool){
    for(uint i = 0; i<candidateList.length; i++){
        if(candidateList[i] == candidate){
            return true;
        }
    }
    return false;
}

```

Figure 4.10 : Functions of Election contract

The sha256hash contract is used to pass the parameters for the verification of zero knowledge proof. From this function, verifyTx function of the verifier contract will be called and return whether the proof is true or not. Figure 4.11 displays the smart contract which was used to pass the proof parameters for verify.

```
pragma solidity ^0.4.23;
import './verifier.sol';
contract sha256hash is Verifier {
    bool public success = false;
    function sha256hashTest(
        uint[2] a,
        uint[2] a_p,
        uint[2][2] b,
        uint[2] b_p,
        uint[2] c,
        uint[2] c_p,
        uint[2] h,
        uint[2] k,
        uint[3] input) public {
        // Verify the proof
        success = verifyTx(a, a_p, b, b_p, c, c_p, h, k, input);
    }
    function get() public view returns (bool) {
        return success;
    }
}
```

Figure 4.11: Sha256hashTest smart contract

## 4.7. Summary

In this chapter software tools utilized to implement the proposed solution was discussed followed by the important functionalities of the proposed solution. The main functionalities which are secret phrase generation, sha256hash calculation, proof generation and smart contracts used in the solution were discussed in the above chapter.



# Chapter 5 - Results and Evaluation

## 5.1. Introduction

This chapter elaborates how results are evaluated and the success level of the proposed solution. New evaluation method was used to compare with other solutions as well has the practically an election with 10 voters was held to measure how succeed the project. Section 5.2 describes the cost needed for transactions. Section 5.3 describes the experiment done with 10 voters. Section 5.4 describes set of evaluation properties. Section 5.5. elaborates on this proposed evaluation model.

## 5.2. Cost analysis

The table 5.1 contains the transaction cost for a voter in an election

Table 5.1: Transaction cost for voter

<b>Tx Hash</b>	<b>TX name</b>	<b>Gas</b>	<b>TX Cost(Ether)</b>
0xa273ba8c4f680302f97e87a0e48f55734f3b905d468509de1d16183382b56309	Web portal sign up	23208	0.000023208
0xfd1e590a5ecf41dc9a0790e2831f08be743118fe53ff6915d0964d501f0f5e35	Voter Registration	185547	0.000185547
0x4586952337f9053f6924fb654c143667b22d6d8a63f69d1f4e2e92eeb591bdc4	Proof verification	1625259	0.001625259
0x8244e7cda8a5831982c05a09a8ad62a60db8d154327881c2c2c8db3b51114398	Mark as voted	28422	0.000028422
0xb8650057caf7effaf17ecaa9be29783f8a11dd126043ce3f511b14f2720cf49	Voting	28108	0.000028108
	Total		0.001890544

According to the table 5.1 approximated Minimum cost a voter has to spend in an election is 0.001890544 Ethers.

The table 5.2 contains the cost for the election authority in an election.

Table 5.2: Transaction cost for election authority in an election

<b>Tx hash</b>	<b>Tx Name</b>	<b>Gas</b>	<b>Tx Cost(Ether)</b>
0xccf45e0f201516e63855e8c50e75c7a48b ac09a5f41c6747aaf8c616ead25619	Add to Acceptance list	39110	0.00003911
0x2447dd0bafb09386d6792244830fd9687ee 2269afcb349a13461bb7276ce521	Verify by district office	49914	0.000049914
0x854591ffc2fd9f0cc918e3bb4bb35769f eed93aac19cf7abffbd77f53a6df495	Reset Account	60828	0.000060828
0x7a7c1e6393281c012a3573e0fc113410a71 a7d7fdb7dc4530605eadf8a95a8ec	Sha256 contract deploy	1903793	0.001903793

According to the above table approximated cost for an election =

$$N \times (0.00003911 + 0.000049914 + 0.000060828) + 0.001903793$$

Note: here N= number of voters

### 5.3. Experiment

The table 5.3 contains the voter details of the voters who participate the experiment voting process.

Table 5.3: Voter details in the experiment

<b>Voter</b>	<b>Voter details</b>
Voter 1	<b>Secret Phrase</b> dPwIxicthNYvNgThmWZqNSyWtKeImZjGIpRgERrAryiMlxLuLmbjxgbw udnbXYUS <b>Hash value of secret phrase</b>

	~out_1 327485948427890063163657922405583843277 ~out_0 73098509705847516637526624146845665259
Voter 2	<b>Secret Phrase</b> kvQemEgJIiCEuqOVIvCPkDRJccoJqKVtNSrLbCzcQCeUNbUdAVKMmhJsQ JzIvWUy <b>Hash values of secret phrase</b> ~out_1 38770939290833947590945745017321709757 ~out_0 113738447700479173714816867410493084445
Voter 3	<b>Secret Phrase</b> PfxaODuOeVSMqUMPHElhYHznwZAIYWzkUAHAnOUhXRqijpOmXGhEp cAuDObkLPcN <b>Hash values of Secret Phrase</b> ~out_1 304128538971998420361452597614819735649 ~out_0 242473399342478884090178563963057785197
Voter 4	<b>Secret Phrase</b> xOkIAZVsCOziikRmpESDMJcghkunderFIHjudAWLMkCKTRLHrmkRLbrcymY JueXFp <b>Hash Values of Secret Phrase</b> ~out_1 93416859694490417389814587885682230767 ~out_0 205464505716366980569582860983562116290
Voter 5	<b>Secret Phrase</b> lZVpxWTOkzUFsCBZBzntTbeluLokEmMmvSqUjQkMucbTSKRIBmnNgPOJj mjzufIP <b>Hash values of secret phrase</b> ~out_1 75542475852409913654231317221826347569 ~out_0 329095829093775114943339581131415684233
Voter 6	<b>Secret Phrase</b> DSfGGGfCXpWyXDPUkaitbHWxOOGeclYwQRTxGVaxHrUwfcCKvfQrucG TTiiZdJMJ <b>Hash values of secret phrase</b> ~out_1 15340325397175966268399056258233591362 ~out_0 137533077572602957241513332529167108327
Voter 7	<b>Secret Phrase</b> UVkAYRIgjpAscNkEbKSyAmTvJmUmxSscLmNGQDOmMrOmiKwqdjRMp FwGdlzoUAMI <b>Hash values of secret Phrase</b>

	~out_1 223446833854513802361614611908663895249 ~out_0 244518891250802074099079021886282118318
Voter 8	<b>Secret Phrase</b> EeRKmNrgzimCMbjLHxUhUVgAwoToJiFPRyegZvVDGGOsTgvViHhVPDlkH LBKlmgO <b>Hash values for secret phrase</b> ~out_1 266539457247887759025322348548525715464 ~out_0 309317082009254409317484108622675315600
Voter 9	<b>Secret Phrase</b> AnDHLSUEBLHXePLneXWhSseJvjwzVbOnKtvGBRkYcXBdoVJKlxqbBuEvvg rYQkgR <b>Hash values for secret phrase</b> ~out_1 169714464883728830521182276408705619794 ~out_0 298125511474834766228494084275969289962
Voter 10	<b>Secret Phrase</b> VKCzmhBtKApCmOczQBGKMqomnLyqLMzKTHckroEXhHjxBuKrhvhdCV dbziiCYQj <b>Hash values for secret phrase</b> ~out_1 323084469091209566631827681793714177130 ~out_0 255441356545460746997402010315979326891

At this experiment ten accounts created and registered for voting. The hash values of the secret phrase that they submitted can be seen at above table. An arithmetic circuit was developed using above hash values.

```
import "LIBSNARK/sha256packed"
```

```
def main(private field a, private field b, private field c, private field d) -> (field):
```

```

    field[10] hash1 =
[73098509705847516637526624146845665259,1137384477004791737148168674104930844
45,242473399342478884090178563963057785197,2054645057163669805695828609835621
16290,329095829093775114943339581131415684233,1375330775726029572415133325291
67108327,244518891250802074099079021886282118318,3093170820092544093174841086
22675315600,298125511474834766228494084275969289962,2554413565454607469974020
10315979326891]
```

```

    field[10] hash2 =
[327485948427890063163657922405583843277,387709392908339475909457450173217097
57,304128538971998420361452597614819735649,9341685969449041738981458788568223
0767,75542475852409913654231317221826347569,153403253971759662683990562582335
91362,223446833854513802361614611908663895249,2665394572478877590253223485485
25715464,169714464883728830521182276408705619794,3230844690912095666318276817
93714177130]

    field result1 = 0
    field result2 = 0
    field z = 0
    field t = 0
    h0, h1 = sha256packed(a, b, c, d)
    for field i in 0..10 do
        result1 = if hash1[i] == h0 then 1 else 0 fi
        result2 = if hash2[i] == h1 then 1 else 0 fi
        z = if result1 == 1 then result2 else 0 fi
        t = if z == 1 then 1 else 0 fi
    endfor
    return t

```

Then the proving key and verification key was generated by the election authority. The proving key and compiled arithmetic circuit was distributed among the verified voters. Then the verifier smart contract was generated and deployed it and started the election.

In this experiment the account of the voter 10 was used for testing. After downloading the arithmetic circuit, the 10<sup>th</sup> voter created a valid witness by giving big integer interpretation of the correct secret phrase as arguments. Then a proof was generated using public inputs and the generated witness. Then voted for a candidate, by submitting valid proof via the web portal.

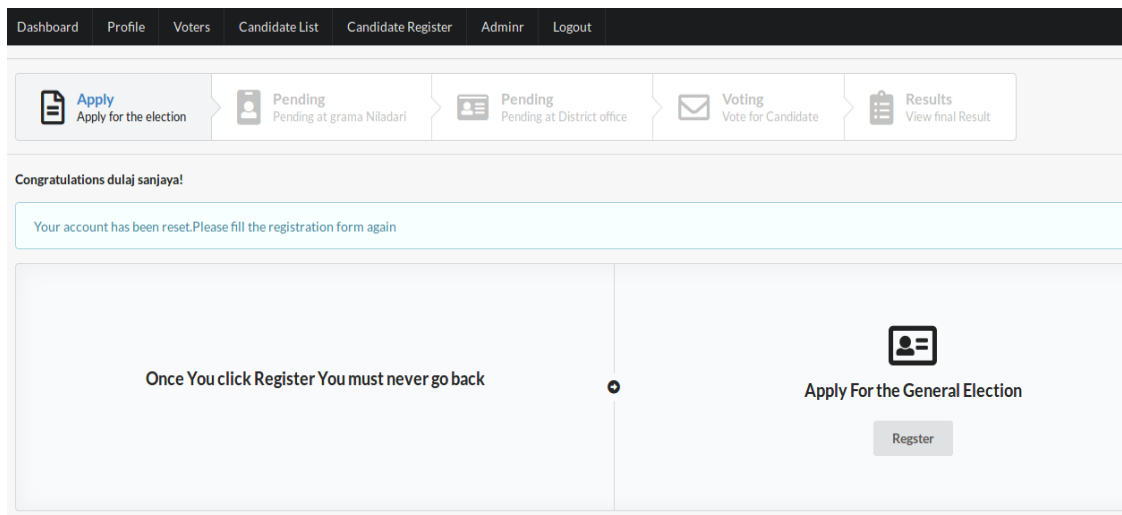


Figure 5.1 : Apply for Election user interface

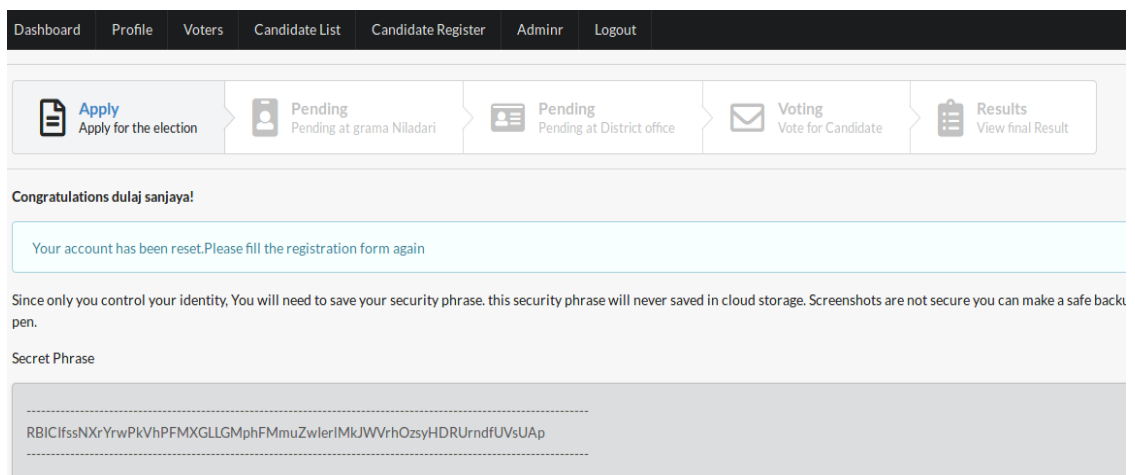



Figure 5.2 : user interface of generated secret phrase



Figure 5.3 : user interface of big integer representation of secret phrase

## Profile

Edit your account details here.



**dulaj sanjaya**

Joined in 2015

Matthew is a musician living in Nashville.

22 Friends

Name


This is a required field.

Cancel or Save

**User Identity**

0x4432ec4e9378f08e6fbac81b168c461cffd6d47

Figure 5.4 : user interface of voter profile



**dulaj sanjaya**

Voter

**Voter Details**

Account status

Voted

NIC

931510406V

Hash of Secret

123

345

**Actions**

Identity

0x4432Ec4E9378F08E6fbacE81B168c461cffd6D47

To be Added List To be Deleted List

Delete Verify Reset

View Documents

Figure 5.5 : User interface of acceptance/ reject a voter

Dashboard Profile Voters Candidate List Candidate Register Adminr Logout

Apply  
Apply for the election

Pending  
Pending at grama Niladari

Pending  
Pending at District office


Voting  
Vote for Candidate


Results  
View final Result

**Congratulations dulaj sanjaya!**

Your Voting Account has been verified. wait for the election to start

**Candidates**

 kasun lakmal ☐ x1

 ishika godage ☐ x2

Submit

Figure 5.6 : User interface of voting

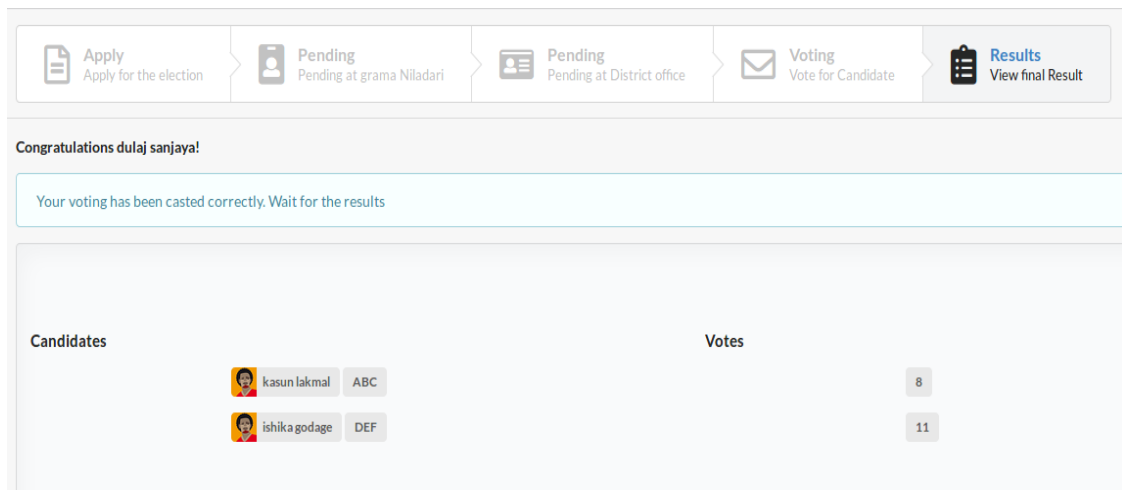


Figure 5.7: User interface of election results

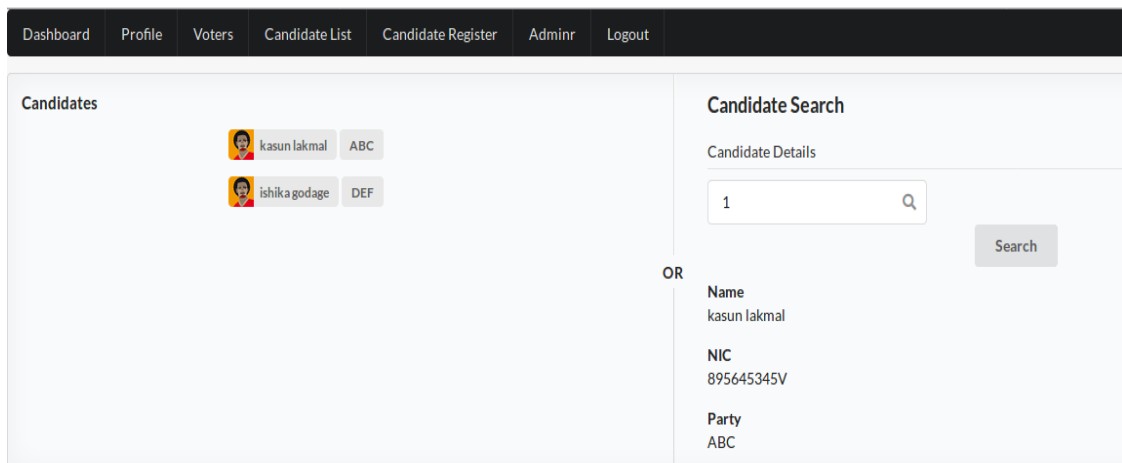


Figure 5.8 : User interface of Candidate management



Figure 5.9 : Success of Proof verification



## 5.4. Evaluation properties

**Privacy:** anyone doesn't know whom the voter voted for. Normally in cryptocurrency, the blockchain account address for each user is a public thing. That means anyone can publish their address saying that deposit money to their accounts. In the previous solutions for e voting system, they have suggested that it is capable of handle this problem using random public key generation and distribute them among voters randomly and voter should keep it as a secret thing. But if someone get in to know that specific person's account address is this, then he will be able to view the voting details including whom the voter voted for etc. using etherscan like web api. But in our solution, it is capable of handle this problem using zero knowledge proof.

**E2E verifiability:** The voter should be able to verify that him/her vote has been casted as intended. It will be able to handle this problem using a blockchain. When we do transaction in a blockchain we get a transaction id. Actually it is the hash value of that transaction. Voters can easily view their transaction details via Etherscan like web API by inserting their transaction id.

**Transparent:** Every vote details should be available to the public except, by whom the voter voted. Final tally for each candidate should be available to the public. Finally the sum of rejected votes and valid votes should be equal to the sum of votes for every candidates.

**Eligibility:** Only eligible user are able to cast their vote. Voters need to prove their eligibility by submitting valid proof via the web portal. If he/she is unable to submit valid proof, then they are considered as un eligible users.

**Decentralized:** Especially in e voting scenarios it is hard to trust a third party. The server is a third party as well as it is a single point of failure. Hence it is not secure to store voting details in mysql like database. Since blockchain is a distributed that means decentralized, the blockchain will be the most suitable data structure to store voting details.

**Scheduling:** After the registration date, citizens should not be able to register as a voter. And also after the voting period or before the voting period voters should not be allowed to vote for anyone.

**Vote limit:** Every registered voter can only vote once for a particular election when they submit a correct proof the voted status of the voter will be updated. So in the voting process no backward paths. If the voter start the voting process then they have to definitely vote. If not their voting chance will be destroyed.

## 5.5. Evaluation criteria

In the evaluation phase, evaluation properties are compared between our solutions and existing solutions. Suggested criterion method is a score criterion.

Ex:

Score 1        -        Yes  
Score 0        -        No

Then total marks were calculated for each solution. Then total marks of our solution was compared with others and ranked solutions according to the total marks. The solution which takes the highest score will be the best one. Our intention was to take our solution towards the best solution.

Table 5.4: Evaluation Model

	Privacy	Verifiability	Transparent	Eligibility	No cost	Decentralized	Scheduled	Total marks
[9]	No	Yes	No	Yes	Yes	No	No	2
[18]	No	No	No	Yes	Yes	No	No	2
[15]	Yes	No	No	Yes	Yes	No	No	3
[10]	No	Yes	Yes	Yes	Yes	No	No	4

[16]	No	No	Yes	Yes	No	Yes	No	3
[5]	Yes	No	Yes	Yes	No	Yes	No	4
Our solution	yes	yes	yes	yes	No	yes	yes	6

According to the evaluation model, our solution was able to get 6 points out of 7. Hence it is very clear that our solution is better than the existing solutions.

## 5.5. Summary

This chapter elaborated the cost analysis for voter and election authority separately. At section 5.2 the cost for both parties were calculated using ether. At section 5.3 voting experiment was done and the results of experiment were displayed. At section 5.4 evaluation properties were discussed. And at finally the evaluation criteria and evaluation model were discussed. Then the retrieved results were explained in this chapter

# **Chapter 6 - Conclusions**

## **6.1 Introduction**

This chapter includes a review of the research aim and objectives, research problem, limitation of the current work and implication for further research.

## **6.2 Conclusions about research questions (aims/objectives)**

The long term goal of the research was to develop a secure e-voting system. One of the goal is to fill the research gap in this area. To do that needed to review the current existing e-voting systems and needed to check whether they are providing the minimal properties in a voting system.

The sub research question 1 was to analyze about the current industry practices, to make the e-voting transaction decentralized by using blockchain. To answer that question, literature review was done. And according to the literature review, found out that still there is a research gap in this area. There were solutions based on zcash blockchain and ethereum blockchain. The main strength of ethereum was the smart contracts. It facilitates to write the contracts between sender and receiver. In our case voter and candidate. But the problem in ethereum is, it doesn't provide private transactions. Then, moved towards zcash like coins which provide private transactions and enables to hide transaction details from the public. But the problem in zcash is it doesn't provide smart contract like concepts to write custom logic. By doing those things solution to the sub research question 1 was provided.

The sub research question 2 was to make the e-voting transaction transparent. At the very beginning of the e-voting problem, solution to this problem was to publish voting details in a website. But it's not a practical thing and it's not secure although. So our solution to this problem was the blockchain. The ethereum blockchain was selected as

a solution since it provide smart contract to write logics. According to this solution all the e-voting transactions were stored in a blockchain.

The third sub research question was about protecting the privacy of the voters. Since ethereum doesn't provide private transactions, the voter details will be available to the public. Then solution to this problem was take the logic behind private transactions of Zcash and apply it to ethereum. The logic behind Zcash is zk-SNARK to protect the privacy of users. Then our solution was integrate zk-SNARK and smart contract together and apply that in to e-voting problem

The last sub research question was about E2E verifiability. That means there should be a way to verify that the voters' vote has been casted as intended. The blockchain itself contains the solution to this problem. When a voter vote for a candidate, the transaction hash is given to voter. Voter need to save it on somewhere secure place. After their voting process they can check whether their transaction is included in a block or not via the web portal.

Finally the main research question was to analyze, How to take a blockchain based approach to develop a trustworthy, transparent, privacy protected, e2e verifiability, decentralized and multiparty secure e-voting system. A proper solution was provided to this problem with 6 phases. Those are Registration phase, Registration verification phase, election preparation phase, voting phase, tallying phase, verification phase.

### **6.3 Conclusions about research problem**

According to the experiment we done with 10 voter accounts, it is very clear that final objective of this research was explored and good solution to the research problem was provided. One of our main objective of this research was to protect the privacy of the voters who participate for the voting process. A very strength solution was provided to this problem and proved the practical aspect of it with the experiment.

According to the experiment, the voters who participate for the election, create account in the web portal provided. Before doing that, they need to have metamask browser extension installed. After clicking registration they will get secret phrase. Then, they

need to calculate sha256hash of the secret phrase with the help of provided arithmetic circuit which developed to calculate sha256 hashes. Then they need to submit the hash values via web portal. The voter verification process is done by the grama niladhari and district office. After the verification, the election authority collects the hashes of verified accounts and create arithmetic circuit using those hash values. According to that arithmetic circuit, when someone insert a correct secret phrase which maps to one of hashes in the arithmetic circuit, it return 1. If not it returns 0. By providing a correct secret phrase voters will get a witness file. Then using that witness file and public inputs, the voter need to create the proof.

Then at the voting web portal voters need to submit their proof. Only if they have provided a valid proof, the user interface will change to voting interface. If not they will not get voting interface. If someone has reached to the voting interface, we can conclude that this is a valid voter. With submitting of the proof, the voted state of the voter will be updated as true. So he is unable to vote again if he cancel this one way process. At the voting step he need to have another separate ethereum address which is not used for the election previously. It's must to have an ethereum address to do an ethereum state update transaction. That's the reason for using ethereum address here. If not, no need of any ethereum address here. Because reaching to this interface verifies that he is a valid user.

The proof doesn't provide any information about the voter. But it helps to verify that this is a valid user. Hence the privacy of the voter is protected at its best. When we tested with registered account in our experiment, we got the voting interface only if we submit the valid proof. So the proof verification part of our research is successful. And At voting phase, since we voted using another account, the election authority or the developers also can't track who the voter was. So it is very clear that according to our experiment, main objective is covered.

## **6.4 Limitations**

This approach can be used to any number of voters as well as any number of candidates. But the time takes to generate the witness and proof will be increased with

the increasing of number of voters since it needs to check the correct secret phrase with lot of hashes. But at the proof verification phase, the time takes to verify the proof will not be increased with the number of voters since the proof size doesn't change with the number of voters. The verification process depend only on the proof and not on the number of voters.

To sign the transactions voters need to download the metamask browser extension and need to create account there. The voters should have computer literacy to do that process.

The ethereum blockchain is based on proof of work. Hence it provide inception to the block verifiers. Hence the voters need to provide some gas to do their transactions. To provide gas the voters should have ether in their ethereum accounts. That means voters have to pay some amount of money for maintaining the consistency and validity of the transactions.

At every election, the election authority has to write a new arithmetic circuit and need to publish the verifier smart contract again if the voter details has changed. And the voters also should have knowledge to run the zokrates tool kit in docker, in order to generate witness, proof as well as hash for their secret phrase.

In the current solution if the user insert hash values which has been inserted previously by another voter, the voter will be informed by this is not a valid hash value. So the voter will get to know that this is the secret phrase for existing hash value.

This is not a completely decentralized solution since the verification of the voter is done by district office. And the arithmetic circuit generation also should be done by election authority.

## **6.5 Implications for further research**

Currently, the arithmetic circuit creation process is done manually by the election authority. That means all the hash values of the voters should be included to the arithmetic circuit manually. That process should be automated. The verifier smart contract publish process also should be done manually. To generate the witness, proof and sha256 hash, voters need to download and run Zokrates on docker container. This tools should be replaced by more usable and user friendly tool. It's better to do sha256 hash generation at client side using javascript without using zokrates. And the format

of the hash should be compatible with the arithmetic circuit which is developed to generate the witness and proof. If it is done using javascript, no need to notify to the voter if they enter existing hash value. System can automatically ignore the generated secret phrase and can generate new secret phrase for that voter.



- [1] R. Gibbins, P. D. Webb and H. Eulau, "election-political-science," Encyclopaedia Britannica, [Online]. Available: <https://www.britannica.com/topic/election-political-science>. [Accessed 14 October 2018].
- [2] "Elections in Sri Lanka," Wikipedia, [Online]. Available: [https://en.wikipedia.org/wiki/Elections\\_in\\_Sri\\_Lanka](https://en.wikipedia.org/wiki/Elections_in_Sri_Lanka). [Accessed 7 January 2019].
- [3] M. Möser, "Anonymity of Bitcoin Transactions An Analysis of Mixing Services," 2013.
- [4] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," *Cryptography Mailing list at https://metzdowd.com.*, 2009.
- [5] P. Tarasov and H. Tewari, "Internet Voting Using Zcash," *IACR Cryptology ePrint Archive*, vol. 2017, p. 585, 2017.
- [6] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer and M. Virza, "Zerocash: Decentralized Anonymous Payments from Bitcoin," in *IEEE Symposium on Security and Privacy*, San Jose, CA, USA, 2014.
- [7] L. Chen, L. Xu, N. Shah, Z. Gao, Y. Lu and W. Shi, "Decentralized Execution of Smart Contracts: Agent Model Perspective and Its Implications," in *International Conference on Financial Cryptography and Data Security*, 2017.
- [8] D. G. Wood, "Ethereum: a Secure Decentralised Generalised Transaction Ledger," 12 april 2017. [Online]. Available: <https://ethereum.github.io/yellowpaper/paper.pdf>. [Accessed 14 october 2018].
- [9] D. Chaum, "Secret-ballot receipts: True voter-verifiable elections," *IEEE Security & Privacy*, vol. 2, pp. 38-47, 2004.
- [10] J. D. Cohen and M. J. Fischer, "A robust and verifiable cryptographically secure election scheme," in *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, Portland, OR, USA, USA, 1985.
- [11] X. Xu, C. Pautasso, L. Zhu, V. Gramoli, A. Ponomarev and A. B. Tran, "The Blockchain as a Software Connector," in *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, Venice, Italy, 2016.
- [12] C. Reitwiessner, "zkSNARKs in a nutshell," Ethereum, 5 December 2016. [Online].

Available: <https://blog.ethereum.org/2016/12/05/zksnarks-in-a-nutshell/>.  
[Accessed 7 Januray 2019].

- [13] D. K. Tosh, S. Shetty, X. Liang, C. A. Kamhoua, K. A. Kwiat and L. Njilla, "Security Implications of Blockchain Cloud with Analysis of Block Withholding Attack," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, Madrid, Spain, 2017.
- [14] "rinkeby test net," [Online]. Available: <https://www.rinkeby.io>. [Accessed 14 october 2018].
- [15] T. Sharma, "E-voting using homomorphic encryption scheme," *International Journal of Computer Applications*, vol. 141, 2016.
- [16] E. Y. K. K. U. C. Ç. and G. D. , "Towards secure e-voting using ethereum blockchain," in *6th International Symposium on Digital Forensic and Security (ISDFS)*, Antalya, 2018.
- [17] L. C. Schaupp and L. Carter, "E-voting: from apathy to adoption," *Journal of Enterprise Information Management*, vol. 18, no. 5, pp. 586-601, 2005.
- [18] H. K. Al-Anie, M. A. Alia and A. A. Hnaif , "E-Voting protocol based On public key cryptography," *International Journal of Network Security & Its Applications (IJNSA)*, vol. 3, 2011.
- [19] F. Hao, P. Ryan and P. Zielinski, "Anonymous voting by two-round public discussion," *IET Information Security*, vol. 4, no. 2, pp. 62-67, 2010.
- [20] J. Groth, "Efficient maximal privacy in boardroom voting," *Financial Cryptography*, pp. 90-104, 2004.
- [21] A. Kiayias and M. Yung, "Self-tallying elections and perfect ballot secrecy," *Public Key Cryptography*, pp. 141-158, 2002.
- [22] "Metamask," Metamask, [Online]. Available: <https://metamask.io/>. [Accessed 7 January 2019].
- [23] S. Bowe, "zkSNARKs in Ethereum," Zcash, 2016. [Online]. Available: <https://z.cash/blog/zksnarks-in-ethereum/>. [Accessed 7 January 2019].

- [24] Reitwiessner, "An Update on Integrating Zcash on Ethereum (ZoE)," Ethereum, 19 January 2017. [Online]. Available: <https://blog.ethereum.org/2017/01/19/update-integrating-zcash-ethereum/>. [Accessed 7 January 2019].
- [25] G. Zyskind, O. Nathan and A. Pentland, "Decentralizing Privacy: Using Blockchain to Protect Personal Data," in *IEEE Security and Privacy Workshops*, San Jose, CA, USA, 2015.
- [26] uport, "First official registration of a Zug citizen on Ethereum," 15 november 2017. [Online]. Available: <https://medium.com/uport/first-official-registration-of-a-zug-citizen-on-ethereum-3554b5c2c238>. [Accessed 15 october 2018].
- [27] P. Dunphy and F. A. P. Petitcolas, "A First Look at Identity Management Schemes on the Blockchain," *IEEE Security & Privacy*, vol. 16, pp. 20-29, 2018.
- [28] "Rinkeby Etherscan," Etherscan, [Online]. Available: <https://rinkeby.etherscan.io/>. [Accessed 18 october 2018].
- [29] "Zokrates," Zokrates, [Online]. Available: <https://github.com/Zokrates/ZoKrates>. [Accessed 7 January 2019].

# Appendix A: Diagrams

TxHash	Block	Age	From		To	Value	[TxFee]
0x37c2d6e86da328...	3651814	2 hrs 10 mins ago	0x4432ec4e9378f08...	IN	0x1716ce1d1a782a...	0 Ether	0.000028884
0x2145acd3729a72...	3644334	1 day 9 hrs ago	0x4432ec4e9378f08...	IN	0x1716ce1d1a782a...	0 Ether	0.000028108
0xbad39956cc2bed...	3641230	1 day 22 hrs ago	0x4432ec4e9378f08...	IN	0x1716ce1d1a782a...	0 Ether	0.000028884
0xb8650057caf7eff...	3638789	2 days 8 hrs ago	0x4432ec4e9378f08...	IN	0x1716ce1d1a782a...	0 Ether	0.000028108
0x63af0bea107dae5...	3624926	4 days 18 hrs ago	0x4432ec4e9378f08...	IN	0x1716ce1d1a782a...	0 Ether	0.000028108
0x3fb23d3283fc19c...	3619351	5 days 17 hrs ago	0x4432ec4e9378f08...	IN	0x1716ce1d1a782a...	0 Ether	0.000028108
0xbe55024aa1c403...	3602703	8 days 14 hrs ago	0x4432ec4e9378f08...	IN	0x1716ce1d1a782a...	0 Ether	0.000028108
0x2a55bbad20c775...	3595792	9 days 19 hrs ago	0x4432ec4e9378f08...	IN	0x1716ce1d1a782a...	0 Ether	0.000028884
0x22df3c71d30f1ce...	3595784	9 days 19 hrs ago	0x4432ec4e9378f08...	IN	0x1716ce1d1a782a...	0 Ether	0.000028884
0x76649733be98ac...	3594825	9 days 23 hrs ago	0x4432ec4e9378f08...	IN	0x1716ce1d1a782a...	0 Ether	0.000028884
0x13f8974dde80e2a...	3594810	9 days 23 hrs ago	0x4432ec4e9378f08...	IN	0x1716ce1d1a782a...	0 Ether	0.000028884
0xe7008482ccde8fa...	3594788	9 days 23 hrs ago	0x4432ec4e9378f08...	IN	0x1716ce1d1a782a...	0 Ether	0.000028884
0x4cb2f8dfc5a9a41...	3594775	9 days 23 hrs ago	0x4432ec4e9378f08...	IN	0x1716ce1d1a782a...	0 Ether	0.000028884
0x9a881653f1d35d6...	3594712	10 days 6 mins ago	0x4432ec4e9378f08...	IN	0x1716ce1d1a782a...	0 Ether	0.000028108

Figure A.1 : Transactions listing in the election smart contract

**Etherscan** RINKBY The Ethereum Block Explorer

RINKBY (CLIQUE) TESTNET Search by Address / Txhash / Block / Token / Ens GO Language

HOME BLOCKCHAIN TOKEN MISC

Contract 0x1716ce1d1A782A1591152156fE978F4aFe6878FD Home / Accounts / Address

**Contract Overview**

Balance: 0 Ether

Transactions: 21 txns

Misc: Contract Creator: 0x4432ec4e9378f08... at txn 0x5a5f1c0b62da6a9...

Figure A.2: Account of smart contract

**Etherscan** RINKBY The Ethereum Block Explorer

RINKBY (CLIQUE) TESTNET Search by Address / Txhash / Block / Token / Ens GO Language

HOME BLOCKCHAIN TOKEN MISC

Address 0x4432Ec4E9378F08E6fbacE81B168c461cfd6D47 Home / Accounts / Address

**Overview**

Balance: 9.496733272 Ether

Transactions: 269 txns

Figure A.3: Account of Voter

## Appendix B: Code Listings

A detailed implementation of all the smart contracts is provided below. The following code block is the voter smart contract which is used to manage the state levels of voters and voter registration

```
pragma solidity ^0.4.23;
contract Voter{
    //voter details
    struct VoterDetails {
        bytes32 name;
        bytes32 nic;
        uint128 hashOfSecret1;
        uint128 hashOfSecret2;
        bool submitted_to_review;
        bool to_be_deleted;
        bool to_be_added;
        bool deleted;
        bool verified;
        bool temp_registered;
        bool voted;
    }

    uint numVoters;

    mapping (address => VoterDetails) voters;

    function getNumOfVoters() public view returns(uint) {
        return numVoters;
    }

    //this should be updated by the applicant
    function addVoter(bytes32 name, bytes32 nic, uint128
hashOfSecret1,uint128 hashOfSecret2) public
returns(bool,bool,bool,bool,bool,bool,bool) {
        //if user doesn't exist
        if(voters[msg.sender].name==0x0){
            voters[msg.sender] =
VoterDetails(name,nic,hashOfSecret1,hashOfSecret2,false,false,f
alse,false,false,false,false);
            numVoters++;
            voters[msg.sender].submitted_to_review = true;
        }
    }
}
```

```

        return
        (voters[msg.sender].submitted_to_review,voters[msg.sender].to_b
        e_deleted,voters[msg.sender].to_be_added,voters[msg.sender].del
        eted,voters[msg.sender].verified,voters[msg.sender].temp_regist
        ered,voters[msg.sender].voted);
    }
    //if user exist(that means account reseted)
    voters[msg.sender].hashOfSecret1 = hashOfSecret1;
    voters[msg.sender].hashOfSecret2 = hashOfSecret2;
    voters[msg.sender].submitted_to_review = true;
    return
    (voters[msg.sender].submitted_to_review,voters[msg.sender].to_b
    e_deleted,voters[msg.sender].to_be_added,voters[msg.sender].del
    eted,voters[msg.sender].verified,voters[msg.sender].temp_regist
    ered,voters[msg.sender].voted);

}

//query specific voter details
function getVoter(address voterId) public view returns
(bytes32,bytes32,uint128,uint128,bool,bool,bool,bool,bool,
bool) {
    VoterDetails memory v = voters[voterId];
    return
    (v.name,v.nic,v.hashOfSecret1,v.hashOfSecret2,v.submitted_to_re
    view,v.to_be_deleted,v.to_be_added,v.deleted,v.verified,v.temp_
    registered,v.voted);
}

//this should be updated by the grama nildari
function toBeDeleted(address voterAddress) public{
    voters[voterAddress].submitted_to_review = false;
    voters[voterAddress].to_be_added=false;
    voters[voterAddress].to_be_deleted = true;
}

//Voted
function voted(address voterAddress) public{
    voters[voterAddress].voted = true;
}

//this should be updated by the grama nildari
function toBeAdded(address voterAddress) public{
    voters[voterAddress].submitted_to_review=false;
    voters[voterAddress].to_be_deleted=false;
    voters[voterAddress].to_be_added=true;
}

```

```

//this should be updated by the district office
function deleted(address voterAddress) public{
    voters[voterAddress].submitted_to_review=false;
    voters[voterAddress].to_be_added=false;
    voters[voterAddress].to_be_deleted=false;
    voters[voterAddress].verified=false;
    voters[voterAddress].deleted=true;
}

//this should be updated by the district office
function verified(address voterAddress) public{
    voters[voterAddress].submitted_to_review=false;
    voters[voterAddress].to_be_added=false;
    voters[voterAddress].to_be_deleted=false;
    voters[voterAddress].deleted=false;
    voters[voterAddress].verified=true;
}

function reset(address voterAddress) public{
    voters[voterAddress].submitted_to_review=false;
    voters[voterAddress].to_be_added=false;
    voters[voterAddress].to_be_deleted=false;
    voters[voterAddress].deleted=false;
    voters[voterAddress].verified=false;
    voters[voterAddress].temp_registered=true;
    voters[voterAddress].voted=false;
}

}

```

Following code block contains the source code of candidate.sol which was used to manage the candidates

```

pragma solidity ^0.4.23;
contract Candidate{

    struct CandidateDetails {
        bytes32 name;
        address addr;
        bytes32 nic;
        bytes32 party;
        bool doesExist;
        bool accepted;
    }
}

```

```

uint numCandidates;

mapping (uint => CandidateDetails) candidates;

function addCandidate(bytes32 name, bytes32 nic, bytes32
party) public {
    // Create new Candidate Struct with name and saves it
to storage.
    numCandidates++;
    candidates[numCandidates]
CandidateDetails(name,msg.sender,nic,party,true,false);
    =
}

function getNumOfCandidates() public view returns(uint) {
    return numCandidates;
}

function getCandidate(uint candidateId) public view
returns (bytes32,bytes32, bytes32) {
    CandidateDetails memory v = candidates[candidateId];
    return (v.name,v.nic,v.party);
}

}

```

Following code block contains the Election.sol smart contract which is used to manage the election

```

pragma solidity ^0.4.23;
contract Election{

    mapping(uint=>uint8) public votesReceived;

    uint[] public candidateList=[1,2];

    function totalVotesFor(uint candidate) view public returns
(uint8){
        require(validCandidate(candidate));
        return votesReceived[candidate];
    }

    function voteForCandidate(uint candidate) public {
        require(validCandidate(candidate));
        votesReceived[candidate] +=1;
    }
}

```



```

        function    validCandidate(uint    candidate)    view    public
returns (bool){
        for(uint i = 0; i<candidateList.length; i++){
                if(candidateList[i] == candidate){
                        return true;
                }

        }
        return false;
}

}

```

The following code block represents the authentication contract which is used to logged in to the web portal

```

pragma solidity ^0.4.2;

import './zeppelin/lifecycle/Killable.sol';

contract Authentication is Killable {
    struct User {
        bytes32 name;
    }

    mapping (address => User) private users;

    uint private id; // Stores user id temporarily

    modifier onlyExistingUser {
        // Check if user exists or terminate

        require(!(users[msg.sender].name == 0x0));
        _;
    }

    modifier onlyValidName(bytes32 name) {
        // Only valid names allowed

        require(!(name == 0x0));
        _;
    }

    function login() constant
    public
    onlyExistingUser

```

```

returns (bytes32) {
    return (users[msg.sender].name);
}

function signup(bytes32 name)
public
payable
onlyValidName(name)
returns (bytes32) {
    // Check if user exists.
    // If yes, return user name.
    // If no, check if name was sent.
    // If yes, create and return user.

    if (users[msg.sender].name == 0x0)
    {
        users[msg.sender].name = name;

        return (users[msg.sender].name);
    }

    return (users[msg.sender].name);
}

function update(bytes32 name)
public
payable
onlyValidName(name)
onlyExistingUser
returns (bytes32) {
    // Update user name.

    if (users[msg.sender].name != 0x0)
    {
        users[msg.sender].name = name;

        return (users[msg.sender].name);
    }
}
}

```

The following code block represents the sha256hash verification contract which is used to take the argument of the proof

```

pragma solidity ^0.4.23;
import './verifier.sol';

```

```

contract sha256hash is Verifier {
    bool public success = false;
    function sha256hashTest(
        uint[2] a,
        uint[2] a_p,
        uint[2][2] b,
        uint[2] b_p,
        uint[2] c,
        uint[2] c_p,
        uint[2] h,
        uint[2] k,
        uint[1] input) public {
        // Verifiy the proof
        success = verifyTx(a, a_p, b, b_p, c, c_p, h, k,
input);
    }
    function get() public view returns (bool) {
        return success;
    }
}

```

This is the contract which is used to verify the proof. That contract is called as verifier.sol and it was generated using ethereum tool kit called as zokrates

```

// This file is MIT Licensed.
//
// Copyright 2017 Christian Reitwiessner
// Permission is hereby granted, free of charge, to any person
obtaining a copy of this software and associated documentation
files (the "Software"), to deal in the Software without
restriction, including without limitation the rights to use,
copy, modify, merge, publish, distribute, sublicense, and/or
sell copies of the Software, and to permit persons to whom the
Software is furnished to do so, subject to the following
conditions:
// The above copyright notice and this permission notice shall
be included in all copies or substantial portions of the
Software.
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.

```

```

pragma solidity ^0.4.23;
library Pairing {
    struct G1Point {
        uint X;
        uint Y;
    }
    // Encoding of field elements is: X[0] * z + X[1]
    struct G2Point {
        uint[2] X;
        uint[2] Y;
    }
    /// @return the generator of G1
    function P1() pure internal returns (G1Point) {
        return G1Point(1, 2);
    }
    /// @return the generator of G2
    function P2() pure internal returns (G2Point) {
        return G2Point(
            [11559732032986387107991004021392285783925812861821192530917403
            151452391805634,

            108570469990230571359445707622328294813707563595785180869905199
            93285655852781],

            [40823678758634336813322034031454355683168513275934012081057410
            76214120093531,

            849565392312343141760497324748927243841819058726360014877028064
            9306958101930]
        );
    }
    /// @return the negation of p, i.e. p.addition(p.negate())
    should be zero.
    function negate(G1Point p) pure internal returns (G1Point)
    {
        // The prime q in the base field F_q for G1
        uint q =
        218882428718392752222464057452572750886963111572978236626890378
        94645226208583;
        if (p.X == 0 && p.Y == 0)
            return G1Point(0, 0);
        return G1Point(p.X, q - (p.Y % q));
    }
    /// @return the sum of two points of G1
    function addition(G1Point p1, G1Point p2) internal returns
    (G1Point r) {
        uint[4] memory input;

```

```

        input[0] = p1.X;
        input[1] = p1.Y;
        input[2] = p2.X;
        input[3] = p2.Y;
        bool success;
        assembly {
            success := call(sub(gas, 2000), 6, 0, input, 0xc0,
r, 0x60)
            // Use "invalid" to make gas estimation work
            switch success case 0 { invalid() }
        }
        require(success);
    }
    /// @return the product of a point on G1 and a scalar, i.e.
    /// p == p.scalar_mul(1) and p.addition(p) ==
p.scalar_mul(2) for all points p.
    function scalar_mul(G1Point p, uint s) internal returns
(G1Point r) {
        uint[3] memory input;
        input[0] = p.X;
        input[1] = p.Y;
        input[2] = s;
        bool success;
        assembly {
            success := call(sub(gas, 2000), 7, 0, input, 0x80,
r, 0x60)
            // Use "invalid" to make gas estimation work
            switch success case 0 { invalid() }
        }
        require (success);
    }
    /// @return the result of computing the pairing check
    ///  $e(p1[0], p2[0]) * \dots * e(p1[n], p2[n]) == 1$ 
    /// For example pairing([P1(), P1().negate()], [P2(),
P2()]) should
    /// return true.
    function pairing(G1Point[] p1, G2Point[] p2) internal
returns (bool) {
        require(p1.length == p2.length);
        uint elements = p1.length;
        uint inputSize = elements * 6;
        uint[] memory input = new uint[](inputSize);
        for (uint i = 0; i < elements; i++)
        {
            input[i * 6 + 0] = p1[i].X;
            input[i * 6 + 1] = p1[i].Y;
            input[i * 6 + 2] = p2[i].X[0];
            input[i * 6 + 3] = p2[i].X[1];

```

```

        input[i * 6 + 4] = p2[i].Y[0];
        input[i * 6 + 5] = p2[i].Y[1];
    }
    uint[1] memory out;
    bool success;
    assembly {
        success := call(sub(gas, 2000), 8, 0, add(input,
0x20), mul(inputSize, 0x20), out, 0x20)
        // Use "invalid" to make gas estimation work
        switch success case 0 { invalid() }
    }
    require(success);
    return out[0] != 0;
}

/// Convenience method for a pairing check for two pairs.
function pairingProd2(G1Point a1, G2Point a2, G1Point b1,
G2Point b2) internal returns (bool) {
    G1Point[] memory p1 = new G1Point[](2);
    G2Point[] memory p2 = new G2Point[](2);
    p1[0] = a1;
    p1[1] = b1;
    p2[0] = a2;
    p2[1] = b2;
    return pairing(p1, p2);
}

/// Convenience method for a pairing check for three pairs.
function pairingProd3(
    G1Point a1, G2Point a2,
    G1Point b1, G2Point b2,
    G1Point c1, G2Point c2
) internal returns (bool) {
    G1Point[] memory p1 = new G1Point[](3);
    G2Point[] memory p2 = new G2Point[](3);
    p1[0] = a1;
    p1[1] = b1;
    p1[2] = c1;
    p2[0] = a2;
    p2[1] = b2;
    p2[2] = c2;
    return pairing(p1, p2);
}

/// Convenience method for a pairing check for four pairs.
function pairingProd4(
    G1Point a1, G2Point a2,
    G1Point b1, G2Point b2,
    G1Point c1, G2Point c2,
    G1Point d1, G2Point d2
) internal returns (bool) {

```

```

        G1Point[] memory p1 = new G1Point[](4);
        G2Point[] memory p2 = new G2Point[](4);
        p1[0] = a1;
        p1[1] = b1;
        p1[2] = c1;
        p1[3] = d1;
        p2[0] = a2;
        p2[1] = b2;
        p2[2] = c2;
        p2[3] = d2;
        return pairing(p1, p2);
    }
}

contract Verifier {
    using Pairing for *;
    struct VerifyingKey {
        Pairing.G2Point A;
        Pairing.G1Point B;
        Pairing.G2Point C;
        Pairing.G2Point gamma;
        Pairing.G1Point gammaBeta1;
        Pairing.G2Point gammaBeta2;
        Pairing.G2Point Z;
        Pairing.G1Point[] IC;
    }
    struct Proof {
        Pairing.G1Point A;
        Pairing.G1Point A_p;
        Pairing.G2Point B;
        Pairing.G1Point B_p;
        Pairing.G1Point C;
        Pairing.G1Point C_p;
        Pairing.G1Point K;
        Pairing.G1Point H;
    }
    function verifyingKey() pure internal returns (VerifyingKey
vk) {
        vk.A =
Pairing.G2Point([0x33de61db4f5934f8646980bdbb09ba19ff68983c38d0
967196b01b972a50fb5,
0x167096f00a6c7a01607c19b1351cd9b1c50bcfaec5fed77ad98e7fe876511
374],
[0x2751b1f30775dd81454dee862b4d02e91ad15756d6224756aedbe58ca06c
216,
0x26a09b5bbd5cdb912b8f09d707f1741a9ac48d6e0f73e862236f051b73d53
b55]);
        vk.B =
Pairing.G1Point(0x18f8de99ee586e675b3f9fea80f25f3283d9c03a13b26

```

```

107a6d883aee82b5c73,
0x21f33c90cd3fb9d1bc8ea131bfc7b7850846d5fffb70ce66a2dceb7799b8d
99a);

vk.C =
Pairing.G2Point([0x2f5e4b788b4a2c2174e8c4d97bbae932484eb5535aec
9fe311bf4f9387aal01,
0x2cdf2769a28bc1d11fc1de2372925cfefab1e560b08cc719cff37d99b7e87
091],
[0x18fbcdeaae46d19281d396f5f641e3b67af55b3f77cd2ec671606587ee1a
6b42,
0x239ded808e68ea3781235d66a136c938c968bca62f8aa002379db00225f1c
5ca]);

vk.gamma =
Pairing.G2Point([0x615bf5be61b937ea94d053bc81eebba8ab2effe55783
f81c39d1c95ab7a957e,
0x10c8ed52c3f5c843c968d5c09acde4840734b4c9d723d0cb6ea0a7e3a5d85
dca],
[0x640498f9bc8bb0e2336c502aad129c0a1ee9ba61e8f44ad39b1f13f36ce9
076,
0x2f698085427044680920d514f67e26ff984824f1464a27fee5b83e5d27513
c46]);

vk.gammaBeta1 =
Pairing.G1Point(0x2a99bc15b1edfa1c86a9fd7ed333d5cb0d35282b0e907
d5baa644ce6d2049851,
0x1e289523f69aacdb76ba5747367454b7594b133baf97d8eb6d8780d74e29b
9ec);

vk.gammaBeta2 =
Pairing.G2Point([0x21041f7250d4dc2b96e166d86130b6f3466d59b75658
claabe8a208275c5fe04,
0x2e11b8c7f4a97d9c2f8fccef3760b5ccf3a11ed9f9c38159a450534aa8f35
195],
[0x29e0bf2a6723a9ba6ca0f687338d7eb9450e0a4d6cceb91e54347296b18a
26da,
0x23fc872cff5f984f77e78f68039a263be7b40768518c92ae4dd81ff66b06f
a46]);

vk.Z =
Pairing.G2Point([0x235d9e4193d28bebc47c55d58ce05b8d7f2cb26f09b0
6b8c85c9dd70139dba0e,
0xd6395535f6adbc98bcfc87ae2d56950e266f445de3c1e9287b022d63585ea
87],
[0x124d45c1d43d9406886ddfa86c5be4d0a2840ce061fdfa1910c58d715b50
fbaa,
0x40395fdc52a1ca64779871db11b78e98f97ee85df07c7ebee711e3c7ca9f7
d3]);

vk.IC = new Pairing.G1Point[] (2);
vk.IC[0] =
Pairing.G1Point(0x134b88b827ad466340562ce6ae375307bfecd1fbed4dd
78a3bba7742c5eb45f1,

```



```

0x18fa4f8997558930b456921ebecefbbeadcd875b0298a27c9ba69d9eea45e8
eea);
        vk.IC[1] =
Pairing.G1Point(0x1d694a79a0570423a77d5415ffbf690443a9877b396a8
8286d54146550c3c178,
0xc4e475824a8dcf1ddc0eb5b2bcacf60401123a68892ea8dae0d69956e84c2
14);
    }
    function verify(uint[] input, Proof proof) internal returns
(uint) {
        VerifyingKey memory vk = verifyingKey();
        require(input.length + 1 == vk.IC.length);
        // Compute the linear combination vk_x
        Pairing.G1Point memory vk_x = Pairing.G1Point(0, 0);
        for (uint i = 0; i < input.length; i++)
            vk_x = Pairing.addition(vk_x,
Pairing.scalar_mul(vk.IC[i + 1], input[i]));
        vk_x = Pairing.addition(vk_x, vk.IC[0]);
        if (!Pairing.pairingProd2(proof.A, vk.A,
Pairing.negate(proof.A_p), Pairing.P2())) return 1;
        if (!Pairing.pairingProd2(vk.B, proof.B,
Pairing.negate(proof.B_p), Pairing.P2())) return 2;
        if (!Pairing.pairingProd2(proof.C, vk.C,
Pairing.negate(proof.C_p), Pairing.P2())) return 3;
        if (!Pairing.pairingProd3(
            proof.K, vk.gamma,
            Pairing.negate(Pairing.addition(vk_x,
Pairing.addition(proof.A, proof.C))), vk.gammaBeta2,
            Pairing.negate(vk.gammaBeta1), proof.B
        )) return 4;
        if (!Pairing.pairingProd3(
            Pairing.addition(vk_x, proof.A), proof.B,
            Pairing.negate(proof.H), vk.Z,
            Pairing.negate(proof.C), Pairing.P2()
        )) return 5;
        return 0;
    }
    event Verified(string s);
    function verifyTx(
        uint[2] a,
        uint[2] a_p,
        uint[2][2] b,
        uint[2] b_p,
        uint[2] c,
        uint[2] c_p,
        uint[2] h,
        uint[2] k,
        uint[1] input

```

```

    ) public returns (bool r) {
    Proof memory proof;
    proof.A = Pairing.G1Point(a[0], a[1]);
    proof.A_p = Pairing.G1Point(a_p[0], a_p[1]);
    proof.B = Pairing.G2Point([b[0][0], b[0][1]], [b[1][0],
b[1][1]]);
    proof.B_p = Pairing.G1Point(b_p[0], b_p[1]);
    proof.C = Pairing.G1Point(c[0], c[1]);
    proof.C_p = Pairing.G1Point(c_p[0], c_p[1]);
    proof.H = Pairing.G1Point(h[0], h[1]);
    proof.K = Pairing.G1Point(k[0], k[1]);
    uint[] memory inputValues = new uint[](input.length);
    for(uint i = 0; i < input.length; i++){
        inputValues[i] = input[i];
    }
    if (verify(inputValues, proof) == 0) {
        emit Verified("Transaction successfully
verified.");
        return true;
    } else {
        return false;
    }
}
}

```

**Note:** All the source codes of the project can be accessed via the following github link  
<https://github.com/orgs/Blockchain-E-Voting/>