北京理工大学

本科生毕业设计(论文)

基于以太坊私有链的投票系统设计与实现

Design and Implementation of Voting System Based on Ethereum Private Blockchain

学　　　院：	计算机学院
专　　　业：	软件工程
学 生 姓 名：	马睿
学　　　号：	1820181084
指 导 教 师：	孙建伟
校外指导教师：	无

2022 年　6 月 6 日

Design and Implementation of Voting System Based on Ethereum Private Blockchain

Abstract

With the advancement of the internet of things (IoT), voting has become more and more digital. The benefits of electronic voting systems are vast, from being more convenient than conventional paper voting to being more cost-efficient and eco-friendlier. However, with all this in mind, most current e-voting systems are centralized with zero to no transparency in the voting process and result tallying. So, in this paper, I propose a new voting system based on Ethereum private blockchain to resolve these problems.

This system uses the Ethereum protocol to create and deploy smart contracts on a private blockchain to ensure the process's decentralization and transparency. However, because of the Ethereum way of data-flow using transactions that everyone can witness, such a system has zero privacy of user data and voting choice. Knowing this flaw, I also propose to use blind signatures with a combination of the Ethereum protocol to ensure both privacy and transparency at the same time, making it suitable for boardroom voting, school elections, and even community referendums.

Keywords: e-voting system; blockchain; Ethereum; smart contract; decentralization; private blockchain; blind signatures

Table of contents

1. Introduction

The proposed system is trusted, meaning the voters must trust that the operator will honestly sign and process their ballots. The only way to ensure the operator was honest is for every voter to verify his vote after the results are out. This, in combination with the smart contract, can prove the vote is honest.

1.1 Research background and significance

No matter the day or age, people's opinions always have mattered. Nevertheless, with the development of technologies, acquiring this opinion has changed.

For thousands of years, the typical voting process consists of casting a ballot into a box. This procedure has not changed that much with time. The current offline paper voting method does not differ much from the one Romans and Greeks used[1] two thousand years ago. However, such voting is inefficient, time, space, and resource-consuming[2]. Also, the voter cannot verify if his vote was tallied because there is no way to trace the ballot to the voter without deanonymizing him.

The paper voting solution is electronic voting (e-voting)[3]. E-voting solves many of the problems. However, Web2-based voting systems are not transparent and centralized.

A solution to Web2 is the blockchain[4] technology and the new Web3. With the creation of Ethereum[5], a new way of systems was made possible. Decentralized, transparent systems using smart contracts[6] to execute code on the Ethereum virtual machine (EVM).

Blockchain voting systems have several flaws[19], depending on how they are built and operated. However, if built suitable, such systems can be more transparent and fairer than their Web2 counterparts.

1.2 Research overview

Since the beginning of the informational age, people have been trying to improve and develop new ways how people vote.

David Chaum plays a massive part in this research. In 1979 he was the person who first proposed[7] every element of the blockchain protocol found in Bitcoin except the proof of work. Later Chaum was the person to develop the first e-voting system[8] in 1981 using untraceable electronic mail, return addresses, and digital pseudonyms. He was also the person who introduced blind signatures[9] in 1983 and later in his career developed several different types of e-voting systems[10][11].

Since the release of Bitcoin, blockchain research has boomed, leading to the creation of the Ethereum protocol and its EVM. With EVM, code can be executed on the blockchain using smart contracts, making activities such as voting transparent and more decentralized. This has led developers to research different types of blockchain-based e-voting systems.

There are many different propositions. Simple ones are straightforward implementation of voting smart contracts without any voter privacy or anonymity. Then some systems provide privacy and anonymity despite the Ethereum protocol's nature of data visibility. Such systems use zero-knowledge proof[12][17], intelligent agents[15], blind signatures[13][14][16], Shamir's secret sharing, or other types of cryptography to protect voters' identities and choices.

This paper proposes a private Ethereum blockchain-based voting system implementation that uses smart contracts with a combination of an elliptic curve blind signatures scheme[18].

1.3 Thesis Structure

The thesis aims to utilize the Ethereum protocol and blind signatures and create a verifiable voting system.

This article has six parts, organized as follows:

Chapter 1: Introduction. This chapter has explained the types of voting systems. Establishes the research done on blockchain-based e-voting systems and introduces the thesis structure.

Chapter 2: Private Blockchain Network. This chapter establishes the type of private network the system will use. Also, the chapter is explained how to set the network and its parameters and run it.

Chapter 3: Theoretical System Analysis. Establish the requirements a voting system should meet. Explains blind signatures and why they use them. Clarify the periods of the voting procedure.

Chapter 4: Overall System Design. Here the smart contract design used in the system is described. Next, it explains how Web2 and Web3 systems differ and finally explains the system's process flow.

Chapter 5: System Implementation. The chapter presents extensive documentation of smart contracts. Explains the signer CLI application purpose and use. It, also shows the code used from the front end to interact with the contracts.

Chapter 6: Deployment and Testing. Explains the system's architecture and show the results of the tests performed on the system.

2. Private Blockchain Network

An e-voting system has to be secure and protect the integrity of the process at any cost possible. In a system based on a private blockchain, the best decision is to use a Proof of Authority (PoA) blockchain, where all the nodes in the network are trusted, and the dangers from attack are minimized.

This chapter explains why the system uses a PoA private Ethereum network. It also shows how to set up such a network, its options, and finally, how to run it.

2.1 Proof of Authority

Proof of Authority was proposed by the co-founder and CTO, at the moment, of Ethereum, Gavin Wood, after a Denial of Service (DoS) attack was launched against the Ropsten test-net on February 24, 2017.

This attack proves that a Proof of Work consensus algorithm is unsuitable for small networks, which do not incentivize their validators with any financial rewards. The same type of network a voting system is going to use. Without incentives, there will not be enough honest miners providing a hash rate, leading to DoS attacks and 51% attacks, where bad actors fork the chain with their own.

A PoA validator node operator can be an already known and trusted person, meaning the operator puts his reputation at stake every time a new block is mined. With that in mind, the number of validators in PoA is substantially low, giving up on decentralization but gaining sustainability by eliminating the need for high-end mining hardware and security by eliminating unknown validators. by eliminating the need for high hardware and security by eliminating unknown validators.

2.2 Geth

Go Ethereum (Geth) is an open-source implementation of the Ethereum protocol written in Go's programming language. Geth provides users to contribute to the Ethereum protocol and connect to the Ethereum's main net or any other network using the Ethereum protocol. It also allows developers to create private networks and develop decentralized applications (dApps) that run on the Ethereum Virtual Machine (EVM).

Geth provides two consensus algorithms: Ethash (PoW) and Clique (PoA). Ethash is the same algorithm used by the Ethereum main net itself. However, the PoA Clique algorithm is better suited for a blockchain on which an e-voting system will operate.

2. 3 Creating private PoA network

Every blockchain network has to start from somewhere. A new blockchain needs a genesis block. In the genesis block are the network specifications: chain id, consensus algorithm, block period, and gas limit.

Creating a genesis block by hand is not an easy task. That is why Geth includes a module called puppeth for creating custom genesis blocks automatically. After running puppeth, this is the genesis block that came out:

```
{
  "config": {
    "chainId": 1234,
    "homesteadBlock": 0,
    "eip150Block": 0,
    "eip150Hash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
    "eip155Block": 0,
    "eip158Block": 0,
    "byzantiumBlock": 0,
    "constantinopleBlock": 0,
    "petersburgBlock": 0,
    "istanbulBlock": 0,
    "clique": {
      "period": 0,
      "epoch": 30000
    }
  },
  "nonce": "0x0",
  "timestamp": "0x625fce7e",
  "extraData":
"0x0000000000000000000000000000000000000000000000000000000000000000b022517f75b90a22
e6e7da738bb9a98f1342c24bc5e6c9f8893cb397c3a49ad2257cf6543e55afbf0000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000",
  "gasLimit": "0x47b760",
  "difficulty": "0x1",
  "mixHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "coinbase": "0x0000000000000000000000000000000000000000",
  "alloc": {
    "b022517f75b90a22e6e7da738bb9a98f1342c24b": {
      "balance": "0x200000000000000000000000000000000000000000000000000000000000000"
    },
    "c5e6c9f8893cb397c3a49ad2257cf6543e55afbf": {
      "balance": "0x200000000000000000000000000000000000000000000000000000000000000"
    }
  },
  "number": "0x0",
  "gasUsed": "0x0",
  "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "baseFeePerGas": null
}
```

The vital information to take out from the genesis block is:

A. The chain id is 1234. This value is required from Truffle to deploy the smart contracts to the correct network. Also, this value is an indicator of the network in any web browser wallet, such as MetaMask;

B. The period of blocks mined by the Clique algorithm is 0. A new block will be mined not at a scheduled period but when a new transaction is submitted. This way is better than aimlessly mining empty blocks without any reason for them to exist;

C. Extra data containing the addresses of the signing nodes:

a) node1: 0xc5e6c9f8893cb397c3a49ad2257cf6543e55afbf

b) node2: 0xb022517f75b90a22e6e7da738bb9a98f1342c24b

D. The gas used by each transaction in the network is 0. Therefore, transactions can be sent without any ETH inside the EOA. No pre-funding is required.

With the genesis block all set up, the only thing to create a new network is to initialize it using geth:

```
geth --datadir node1/ init genesis.json
geth --datadir node2/ init genesis.json
```

2. 4 Running the network

Running a blockchain network means running the nodes that validate the transactions. In the case of this network, there are only two nodes for demonstration purposes, but even in cases with many more nodes, the process is the same. The BASH script used to start the node1 is as follows:

```
#!/bin/bash
geth --networkid 1234 --nousb --datadir=$pwd --syncmode full --port 30310 --miner.gasprice 0
--miner.gastarget 470000000000 --http --http.addr '127.0.0.1' --http.port 8545 --http.corsdomain "*"
--http.vhosts "*" --http.api admin,eth,miner,net,txpool,personal,web3 --ws --ws.port 8547 --ws.api
admin,eth,miner,net,txpool,personal,web3 --mine --allow-insecure-unlock --unlock
"0xC5e6c9f8893Cb397C3A49Ad2257cF6543E55aFBF" --password password.txt
```

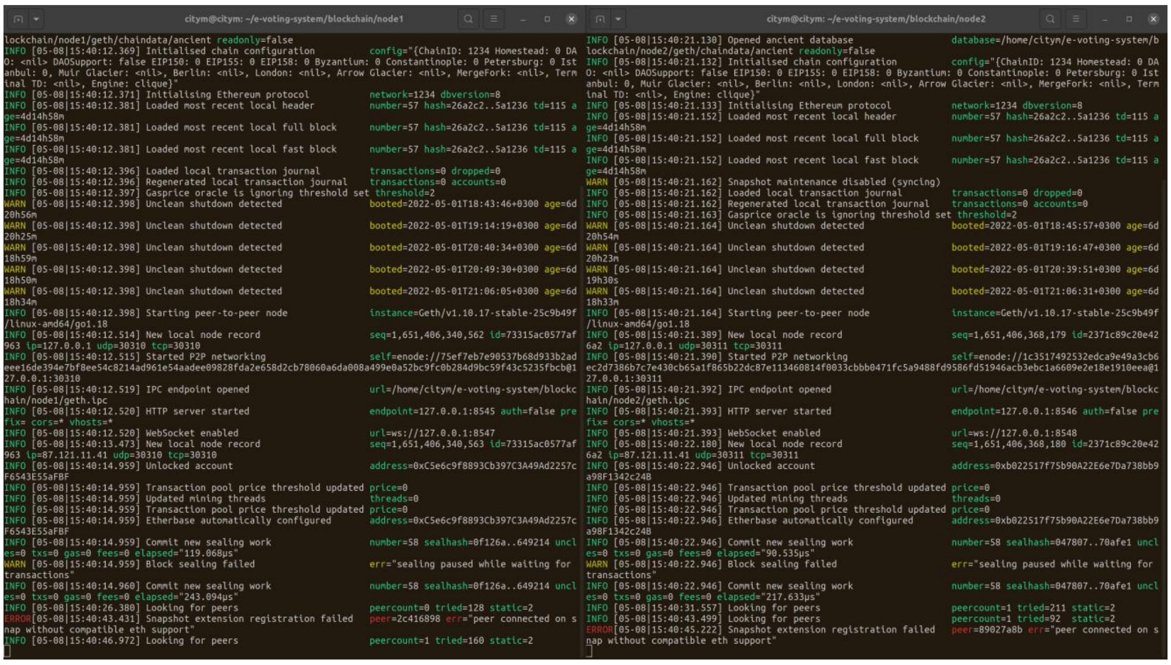
To make things clear, the options of the above Geth command are explained in Table 2-1.

Table 2- 1 Geth options

Option	Description
--networkid 1234	The network id specified here is in order to ensure frameworks such as Truffle compile the deployment artifacts with the correct network id inside.
--nousb	Disables monitoring for and managing USB hardware wallets.
--datadir=$pwd	The directory of the databases and Keystore. It is set to $pwd, meaning the script starting the node has to be executed from the node's directory.
-syncmode 'full'	Fully synchronizes the node with the rest of the nodes.
--port 30310	Assign port for geth processes.
--miner.gasprice 0	Gas price. In a private network, there is no need to pay gas fees.
--miner.gastarget 470000000000	Gas target/ limit. The exact value is the gas limit from the genesis block, but in practice, the gas limit is dynamic, based on the previous block.
--http	Enables the HTTP-RPC server.
--http.addr '127.0.0.1'	HTTP-RPC server listening interface.
--http.port 8545	HTTP-RPC server listening port.
--http.corsdomain "*"	Enables access to the API from web pages.
--http.vhosts "*"	Can accept request from any hostname.
--http.api	Enables useful APIs used over RPC calls.

--ws	Enables the WS-RPC server.
--ws.port 8547	WS-RPC server listening port.
--ws.api	Enables useful APIs used over RPC calls.
--mine	Enable mining.
--allow-insecure-unlock	Allow insecure account unlocking when account-related RPCs are exposed by HTTP.
--unlock address	Unlocks the node1 Keystore file.
--password password.txt	The file containing the password for the Keystore file.

After executing the above command for nodes, the private Ethereum network is up and running as shown in Image 2-1.


citym@citym: ~/e-voting-system/blockchain/node1
citym@citym: ~/e-voting-system/blockchain/node2

```
lockchain/node1/geth/chaindata/ancient readonly=false
INFO [05-08|15:40:12.369] Initialised chain configuration           config="{ChainID: 1234 Homestead: 0 DA
O: <nil> DAOSupport: false EIP150: 0 EIP155: 0 EIP158: 0 Byzantium: 0 Constantinople: 0 Petersburg: 0 Ist
anbul: 0, Muir Glacier: <nil>, Berlin: <nil>, London: <nil>, Arrow Glacier: <nil>, MergeFork: <nil>, Term
inal TD: <nil>, Engine: clique}"
INFO [05-08|15:40:12.371] Initialising Ethereum protocol            network=1234 dbversion=8
INFO [05-08|15:40:12.381] Loaded most recent local header           number=57 hash=26a2c2..5a1236 td=115 a
ge=4d14h58m
INFO [05-08|15:40:12.381] Loaded most recent local full block       number=57 hash=26a2c2..5a1236 td=115 a
ge=4d14h58m
INFO [05-08|15:40:12.381] Loaded most recent local fast block       number=57 hash=26a2c2..5a1236 td=115 a
ge=4d14h58m
INFO [05-08|15:40:12.396] Loaded local transaction journal          transactions=0 dropped=0
INFO [05-08|15:40:12.396] Regenerated local transaction journal     transactions=0 accounts=0
INFO [05-08|15:40:12.397] Gasprice oracle is ignoring threshold set threshold=2
WARN [05-08|15:40:12.398] Unclean shutdown detected                 booted=2022-05-01T18:43:46+0300 age=6d
20h56m
WARN [05-08|15:40:12.398] Unclean shutdown detected                 booted=2022-05-01T19:14:19+0300 age=6d
20h25m
WARN [05-08|15:40:12.398] Unclean shutdown detected                 booted=2022-05-01T20:40:34+0300 age=6d
18h59m
WARN [05-08|15:40:12.398] Unclean shutdown detected                 booted=2022-05-01T20:49:30+0300 age=6d
18h50m
WARN [05-08|15:40:12.398] Unclean shutdown detected                 booted=2022-05-01T21:06:05+0300 age=6d
18h34m
INFO [05-08|15:40:12.398] Starting peer-to-peer node                instance=Geth/v1.10.17-stable-25c9b49f
/linux-amd64/go1.18
INFO [05-08|15:40:12.514] New local node record                     seq=1,651,406,340,562 id=73315ac0577af
963 ip=127.0.0.1 udp=30310 tcp=30310
INFO [05-08|15:40:12.515] Started P2P networking                    self=enode://75ef7eb7e90537b68d933b2ad
eee16de394e7bf8ee54c8214ad961e54aadee09828fda2e658d2cb78060a6da008a499e0a52bc9fc0b284d9bc59f43c5235fbcb@1
27.0.0.1:30310
INFO [05-08|15:40:12.519] IPC endpoint opened                       url=/home/citym/e-voting-system/blockc
hain/node1/geth.ipc
INFO [05-08|15:40:12.520] HTTP server started                       endpoint=127.0.0.1:8545 auth=false pre
fix= cors=* vhosts=*
INFO [05-08|15:40:12.520] WebSocket enabled                         url=ws://127.0.0.1:8547
INFO [05-08|15:40:13.473] New local node record                     seq=1,651,406,340,563 id=73315ac0577af
963 ip=87.121.11.41 udp=30310 tcp=30310
INFO [05-08|15:40:14.959] Unlocked account                          address=0xC5e6c9f8893Cb397C3A49Ad2257c
F6543E55aFBF
INFO [05-08|15:40:14.959] Transaction pool price threshold updated price=0
INFO [05-08|15:40:14.959] Updated mining threads                    threads=0
INFO [05-08|15:40:14.959] Transaction pool price threshold updated price=0
INFO [05-08|15:40:14.959] Etherbase automatically configured        address=0xC5e6c9f8893Cb397C3A49Ad2257c
F6543E55aFBF
INFO [05-08|15:40:14.959] Commit new sealing work                   number=58 sealhash=0f126a..649214 uncl
es=0 txs=0 gas=0 fees=0 elapsed="119.068µs"
WARN [05-08|15:40:14.959] Block sealing failed                      err="sealing paused while waiting for
transactions"
INFO [05-08|15:40:14.960] Commit new sealing work                   number=58 sealhash=0f126a..649214 uncl
es=0 txs=0 gas=0 fees=0 elapsed="243.094µs"
INFO [05-08|15:40:26.380] Looking for peers                         peercount=0 tried=128 static=2
ERROR[05-08|15:40:43.431] Snapshot extension registration failed   peer=2c416898 err="peer connected on s
nap without compatible eth support"
INFO [05-08|15:40:46.972] Looking for peers                         peercount=1 tried=160 static=2
```

```
INFO [05-08|15:40:21.130] Opened ancient database                   database=/home/citym/e-voting-system/b
lockchain/node2/geth/chaindata/ancient readonly=false
INFO [05-08|15:40:21.132] Initialised chain configuration           config="{ChainID: 1234 Homestead: 0 DA
O: <nil> DAOSupport: false EIP150: 0 EIP155: 0 EIP158: 0 Byzantium: 0 Constantinople: 0 Petersburg: 0 Ist
anbul: 0, Muir Glacier: <nil>, Berlin: <nil>, London: <nil>, Arrow Glacier: <nil>, MergeFork: <nil>, Term
inal TD: <nil>, Engine: clique}"
INFO [05-08|15:40:21.133] Initialising Ethereum protocol            network=1234 dbversion=8
INFO [05-08|15:40:21.152] Loaded most recent local header           number=57 hash=26a2c2..5a1236 td=115 a
ge=4d14h58m
INFO [05-08|15:40:21.152] Loaded most recent local full block       number=57 hash=26a2c2..5a1236 td=115 a
ge=4d14h58m
INFO [05-08|15:40:21.152] Loaded most recent local fast block       number=57 hash=26a2c2..5a1236 td=115 a
ge=4d14h58m
WARN [05-08|15:40:21.162] Snapshot maintenance disabled (syncing)
INFO [05-08|15:40:21.162] Loaded local transaction journal          transactions=0 dropped=0
INFO [05-08|15:40:21.162] Regenerated local transaction journal     transactions=0 accounts=0
INFO [05-08|15:40:21.163] Gasprice oracle is ignoring threshold set threshold=2
WARN [05-08|15:40:21.164] Unclean shutdown detected                 booted=2022-05-01T18:45:57+0300 age=6d
20h54m
WARN [05-08|15:40:21.164] Unclean shutdown detected                 booted=2022-05-01T19:16:47+0300 age=6d
20h23m
WARN [05-08|15:40:21.164] Unclean shutdown detected                 booted=2022-05-01T20:39:51+0300 age=6d
19h30s
WARN [05-08|15:40:21.164] Unclean shutdown detected                 booted=2022-05-01T21:06:31+0300 age=6d
18h33m
INFO [05-08|15:40:21.164] Starting peer-to-peer node                instance=Geth/v1.10.17-stable-25c9b49f
/linux-amd64/go1.18
INFO [05-08|15:40:21.389] New local node record                     seq=1,651,406,368,179 id=2371c89c20e42
6a2 ip=127.0.0.1 udp=30311 tcp=30311
INFO [05-08|15:40:21.390] Started P2P networking                    self=enode://1c3517492532edca9e49a3cb6
ec2d7386b7c7e430cb65a1f865b22dc87e113460814f0033cbbb0471fc5a9488fd9586fd51946acb3ebc1a6609e2e18e1910eea@1
27.0.0.1:30311
INFO [05-08|15:40:21.392] IPC endpoint opened                       url=/home/citym/e-voting-system/blockc
hain/node2/geth.ipc
INFO [05-08|15:40:21.393] HTTP server started                       endpoint=127.0.0.1:8546 auth=false pre
fix= cors=* vhosts=*
INFO [05-08|15:40:21.393] WebSocket enabled                         url=ws://127.0.0.1:8548
INFO [05-08|15:40:22.180] New local node record                     seq=1,651,406,368,180 id=2371c89c20e42
6a2 ip=87.121.11.41 udp=30311 tcp=30311
INFO [05-08|15:40:22.946] Unlocked account                          address=0xb022517f75b90A22E6e7Da738bb9
a98F1342c24B
INFO [05-08|15:40:22.946] Transaction pool price threshold updated price=0
INFO [05-08|15:40:22.946] Updated mining threads                    threads=0
INFO [05-08|15:40:22.946] Transaction pool price threshold updated price=0
INFO [05-08|15:40:22.946] Etherbase automatically configured        address=0xb022517f75b90A22E6e7Da738bb9
a98F1342c24B
INFO [05-08|15:40:22.946] Commit new sealing work                   number=58 sealhash=047807..70afe1 uncl
es=0 txs=0 gas=0 fees=0 elapsed="90.535µs"
WARN [05-08|15:40:22.946] Block sealing failed                      err="sealing paused while waiting for
transactions"
INFO [05-08|15:40:22.946] Commit new sealing work                   number=58 sealhash=047807..70afe1 uncl
es=0 txs=0 gas=0 fees=0 elapsed="217.633µs"
INFO [05-08|15:40:31.557] Looking for peers                         peercount=1 tried=211 static=2
INFO [05-08|15:40:43.499] Looking for peers                         peercount=1 tried=92  static=2
ERROR[05-08|15:40:45.222] Snapshot extension registration failed   peer=89027a8b err="peer connected on s
nap without compatible eth support"
```

Image 2-1 Running private blockchain

3. Theoretical System Analysis

Although there are different voting systems, they all have to meet exact requirements. However, meeting some of these requirements in a blockchain-based electronic voting system proved more complicated than traditional paper voting methods. So with smart contacts and cryptography, we intend to make the system fair and end-to-end verifiable.

This chapter explains the requirements that a voting system needs to meet. Also, here is an introduction to what blind signatures are and how to use them for blind voting.

3. 1 Overall system requirements

A blockchain-based voting system should inherit all the properties of a conventional voting system and only build on top. The following list is the requirements a fair, end-to-end verifiable voting system must have:

A) Eligibility, where only people with the right to vote can participate in the process;

B) Integrity, where the system must not allow bad actors to temper whit the voters' data or with the vote itself;

C) Privacy, where no voter can be traced back to the choice They made;

D) Verifiable, where everyone can verify their vote, hence verifying the integrity of the final result;

E) Decentralization, where multiple Ethereum nodes are running at any given time to keep the private blockchain synchronized and alive if any of the nodes fail.

3. 2 Blind Signatures

Blind signatures//add references// ensure the privacy of the voter's choice. Like a digital signature, a blind signature is used to validate the authenticity of a message. The difference is that the message is blinded and encrypted, meaning that the signed data differs from the original message. Furthermore, the method ensures that the receiver of the signed message can unblind and decrypt it, obtaining the original message.

All of the above makes the message's authenticity verifiable without reviewing the sender's identity, making the method suitable for use in systems requiring privacy.

3. 3 Voting procedure

Due to the nature of the Ethereum protocol, every transaction is public. Therefore, anyone with a connection to the blockchain can see them. All of this is good for transparency but bad for privacy. Not having privacy is a significant flaw in a voting system. For this reason, the system uses blind signatures to create blind voting (Figure 3-2). So, to apply blind voting, the system is divided into four periods.

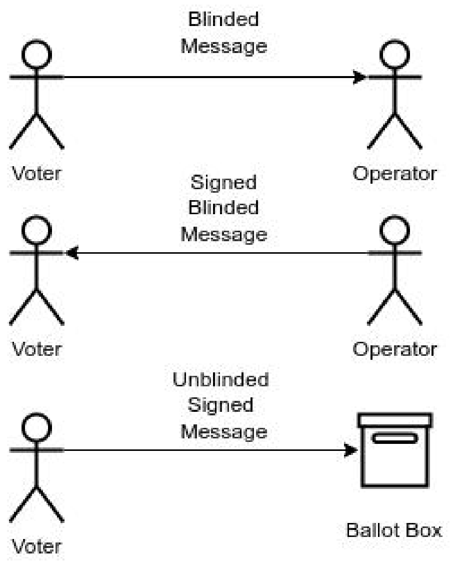


Figure 3- 1 Blind Voting

3. 3. 1 Pre-Registration period

The organizer creates a new vote during this period by deploying a new smart contract. The contract’s constructor takes the time of both registration and voting, the name of the vote, the choices, and the operator’s public keys for the blind-signature procedure. After the contract creation, all these variables are immutable to prevent bad actors from tampering with them.

After deployment, the operator inserts the external owner accounts (EOA) of all the eligible voters who can participate in the vote. For this, voters’ EOA must be known beforehand by the organizer to prevent voters from accessing the personal information of other voters, hence keeping the privacy and integrity of the system.

3. 3. 2 Registration period

All eligible voters can register to vote by locally generating a universally unique identifier (UUID) and a set of public and private keys. Because of the nature of UUID, the collision probability is close to zero. So, the UUID will be the message the voter blinds during the registration and sends to the contract for signing.

The registration act notifies the operator that there is a blinded message to be signed. After the blinded message is signed, the operator uploads it to the contract. Then the voter can use it during the voting period.

3. 3. 3 Voting period

All registered voters who want to vote must unblind the blinded UUID, signed by the operator. Then, compose a Ballot consisting of the unblinded signed UUID, the UUID itself, the public key, and the choice. After that, the voter sends the ballot with a different EOA from the one used to register if he supposedly wants to be anonymous. The UUID and unblinded signed UUID values cannot be traced back to the original EOA. Also, the public key used for blinding and unblinding the UUID was never shared, so it cannot be traced. By casting a vote this way, the voter's identity is kept secret even from the operator. Hence the system provides privacy for the vote.

3. 3. 4 Tally period

After the voting period ends, the organizer must tally the vote results. Off-chain, with the blind signature schema[18], the operator uses the unblinded signed UUID and the public key from a cast Ballot to verify its integrity. Then if the Ballot is valid, the vote is added to the contract. After the verification procedure is complete, the results are made public. Voters then can verify their choice by providing their UUID.

4. Overall System Design

4. 1 Web2 and Web3 comparison

In Web2, the application is divided into client-side and server-side, but in Web3, this is no more the case. Instead, server and database applications run on the blockchain (Figures 4-1 and 4-2).

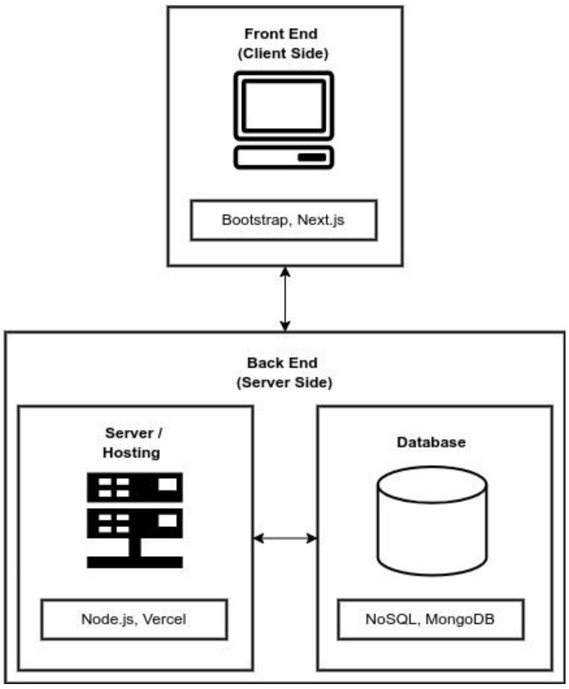


Figure 4- 1 Conventional System

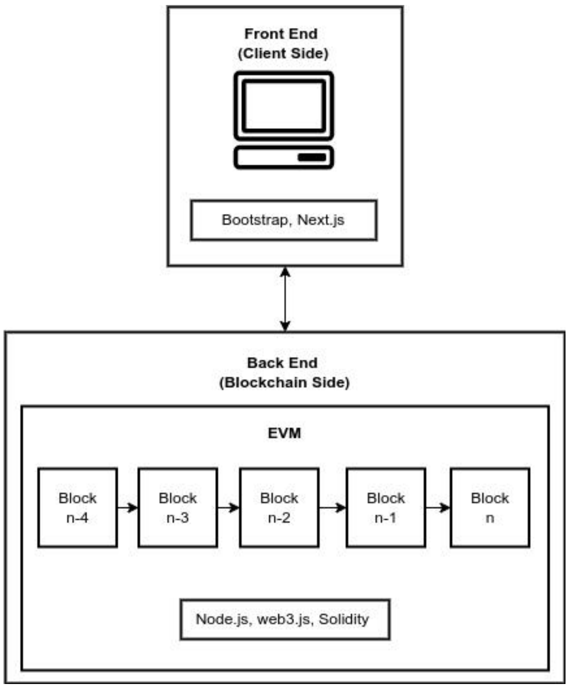


Figure 4- 2 dApp System

Decentralized applications (dApps) use the blockchain as a server to run code and as a database to store data. Building apps this way gives up on speed and cost efficiency but gains process transparency, user trust, and data security guaranteed by the blockchain. In the case of a voting system, the benefits a dApp provides over a conventional web2 app far exceed the disadvantages.

4. 2 Smart Contract Design

The paper's proposed voting system is envisioned to be used in boardroom votes, close community referendums, and other similar small-scale polls. Deriving from that, once the system is set, it will be used multiple times for different polls. It makes the factory design pattern the most suited for the smart contracts on which the voting will take place. For every new voting instance, a new smart contract, VotingSystem, will be deployed by the operator, making each vote independent from the others, as can be seen from the UML (Figure 4-3).

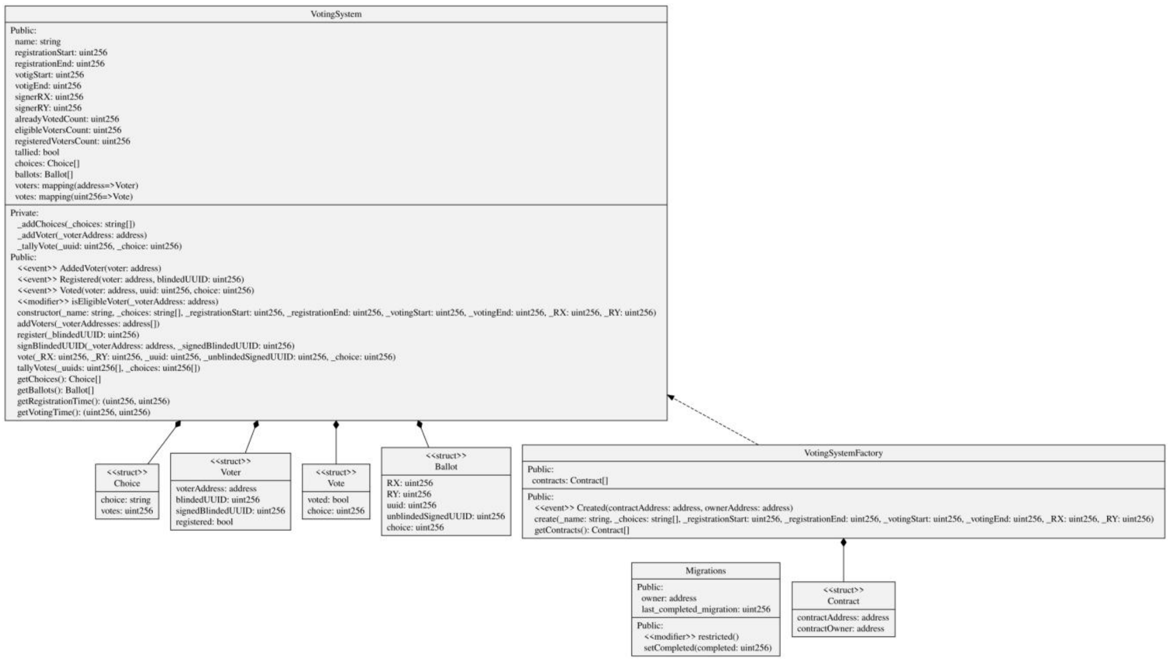

VotingSystem
Public:
name: string
registrationStart: uint256
registrationEnd: uint256
votigStart: uint256
votigEnd: uint256
signerRX: uint256
signerRY: uint256
alreadyVotedCount: uint256
eligibleVotersCount: uint256
registeredVotersCount: uint256
tallied: bool
choices: Choice[]
ballots: Ballot[]
voters: mapping(address=>Voter)
votes: mapping(uint256=>Vote)
Private:
_addChoices(_choices: string[])
_addVoter(_voterAddress: address)
_tallyVote(_uuid: uint256, _choice: uint256)
Public:
<<event>> AddedVoter(voter: address)
<<event>> Registered(voter: address, blindedUUID: uint256)
<<event>> Voted(voter: address, uuid: uint256, choice: uint256)
<<modifier>> isEligibleVoter(_voterAddress: address)
constructor(_name: string, _choices: string[], _registrationStart: uint256, _registrationEnd: uint256, _votingStart: uint256, _votingEnd: uint256, _RX: uint256, _RY: uint256)
addVoters(_voterAddresses: address[])
register(_blindedUUID: uint256)
signBlindedUUID(_voterAddress: address, _signedBlindedUUID: uint256)
vote(_RX: uint256, _RY: uint256, _uuid: uint256, _unblindedSignedUUID: uint256, _choice: uint256)
tallyVotes(_uuids: uint256[], _choices: uint256[])
getChoices(): Choice[]
getBallots(): Ballot[]
getRegistrationTime(): (uint256, uint256)
getVotingTime(): (uint256, uint256)
<<struct>>
Choice
choice: string
votes: uint256
<<struct>>
Voter
voterAddress: address
blindedUUID: uint256
signedBlindedUUID: uint256
registered: bool
<<struct>>
Vote
voted: bool
choice: uint256
<<struct>>
Ballot
RX: uint256
RY: uint256
uuid: uint256
unblindedSignedUUID: uint256
choice: uint256
VotingSystemFactory
Public:
contracts: Contract[]
Public:
<<event>> Created(contractAddress: address, ownerAddress: address)
create(_name: string, _choices: string[], _registrationStart: uint256, _registrationEnd: uint256, _votingStart: uint256, _votingEnd: uint256, _RX: uint256, _RY: uint256)
getContracts(): Contract[]
Migrations
Public:
owner: address
last_completed_migration: uint256
Public:
<<modifier>> restricted()
setCompleted(completed: uint256)
<<struct>>
Contract
contractAddress: address
contractOwner: address

Figure 4- 3 Smart Contract UML

A problem that occurred during the early stages of the development of the system using this pattern was that the system was 100% decentralized, with a PoW type of consensus algorithm, and open for everyone to join the network and deploy a new voting instance. Unfortunately, an open system like this allows for spam. Bad actors can drive it unusable by flooding it with invalid polls or creating voting instances with names similar to valid ones, making the system prone to phishing.

The whole system was reworked to solve a massive problem like this one and fix the overall security without compensating for usability. The blockchain validation consensus algorithm was changed to PoA, where only trusted people can run a block signing node.

Combined within code restrictions to whom can deploy contracts by using the already tested OpenZeppelin's Ownable smart contract. With these improvements VotingSystem smart contract can be deployed only from the signing nodes, using locally created accounts, giving up on freedom and decentralization but dramatically improving security.

4. 3 System flow

Voting is a time-locked type of process. Therefore, the voting flow in the system is time-locked as well. Because of time-lock and division of voter and operator roles in the system, there is not a single start-to-end process flow but a cross-functional one (Figure 5).

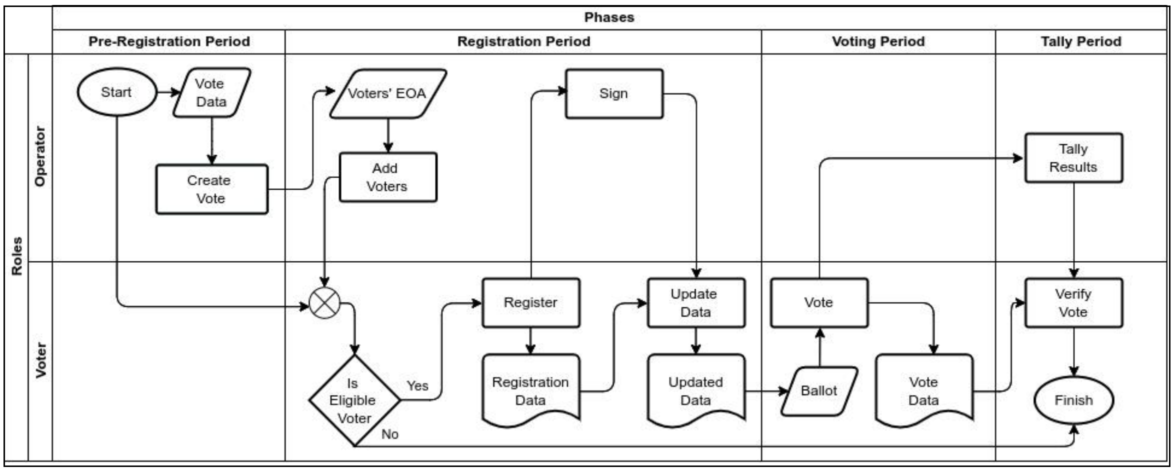


Figure 4-4 System Cross Functional Flow Chart

4. 3. 1 **Operator Pre-Registration Period flowchart**

An operator must create a vote by deploying a new VotingSystem smart contract.

The data required for the vote is validated during the voter creation, and then a new voting instance is deployed. In (Figure 4-5) is shown the process flow of a new vote creation.

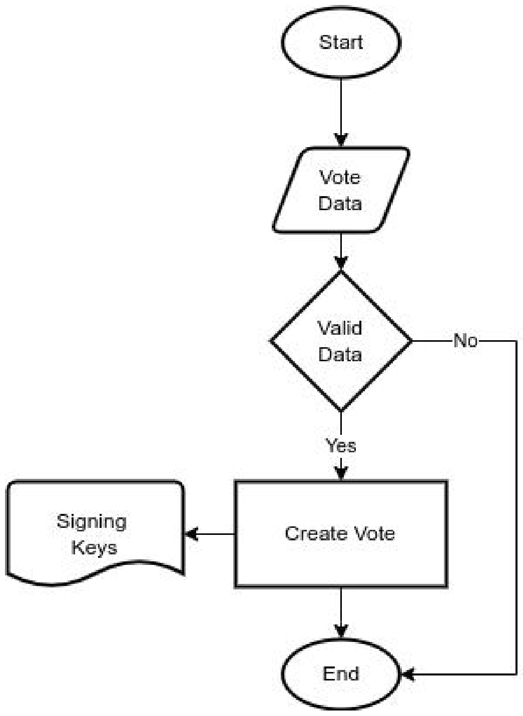


Figure 4-5 Operator Pre-Registration Period flowchart

4. 3. 2 **Operator Registration Period flowchart**

During this period (Figure 4-6), the operator plays a crucial role. First, he adds eligible EOAs to the smart contract. Then, the operator has to make sure that the blinded UUID of every eligible voter who has requested a signature is signed.

The whole vote integrity relies upon the operator’s ability to sign the incoming requests. If somewhere during this process something breaks, it must be fixed immediately if possible. If a critical error during the signing process cannot be fixed, the operator should create a new voting instance and start the vote all over again.

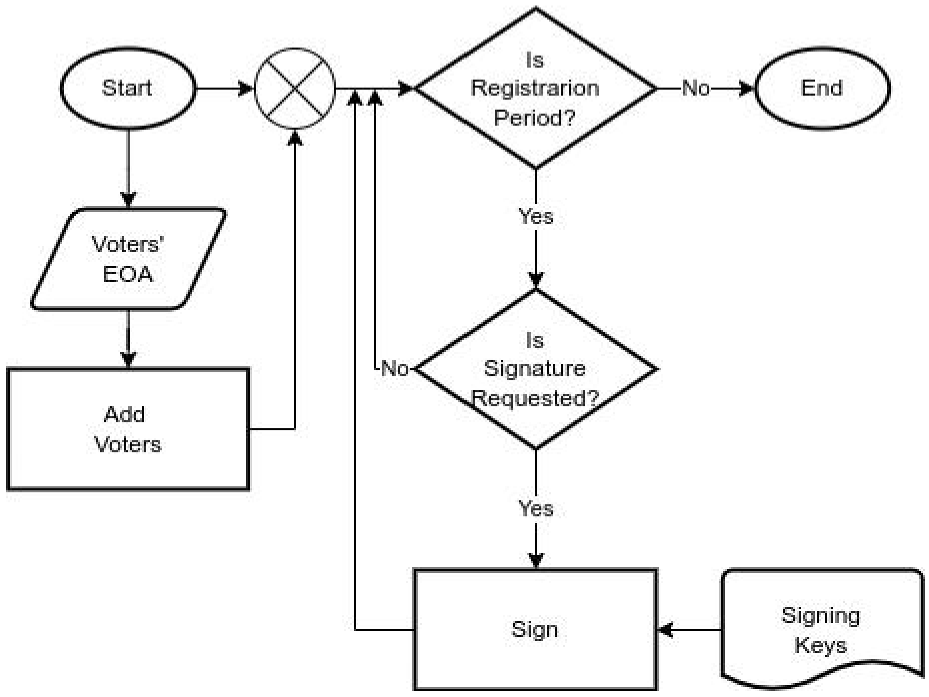


Figure 4-6 Operator Registration Period flowchart

4. 3. 3 Voter Registration Period flowchart

The voter must register to vote during the registration period (Figure 4-5). When registering, the voter generates locally UUID and keys required by the blind signature schema. Then he blinds the UUID and uploads it to the contract, requesting a signature from the operator. After the request is made, the voter must download his keys. Downloading the keys is required because they are not stored anywhere else to prevent other people from seeing them. When the operator signs the blinded UUID, the voter must upload his registration keys and update them to voting keys. To prevent deanonymization, the voter must never share any of the keys with anyone.

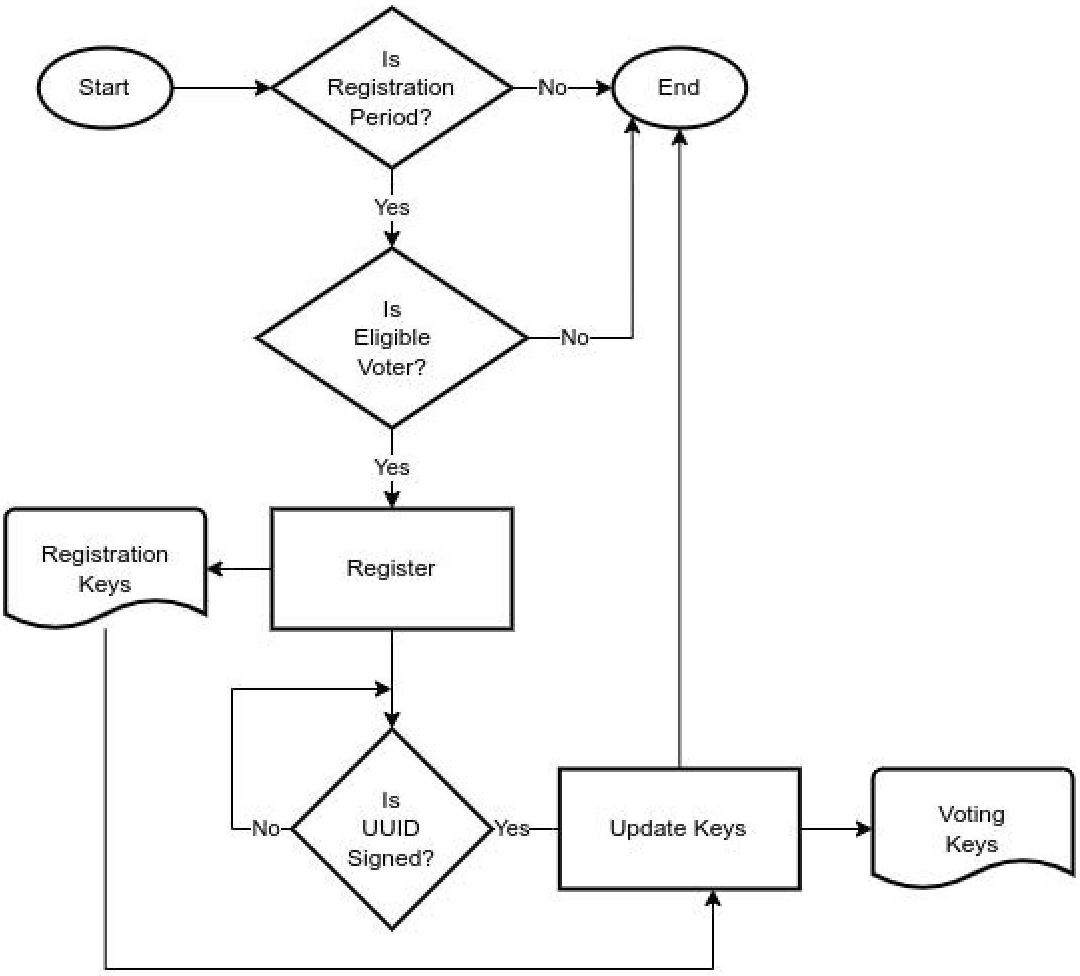


Figure 4-7 Voter Registration Period flowchart

4. 3. 4 Voter Voting Period flowchart

All the registered voters are eligible to vote during the voting period (Figure 4-6). First, a registered voter must upload his voting keys. Then, the voter with the data from the keys creates a Ballot that sends to the smart contract. If the voter wants to verify his vote after the tallied results, he must download his verification keys. To prevent deanonymization, the voter must never share these keys with anyone.

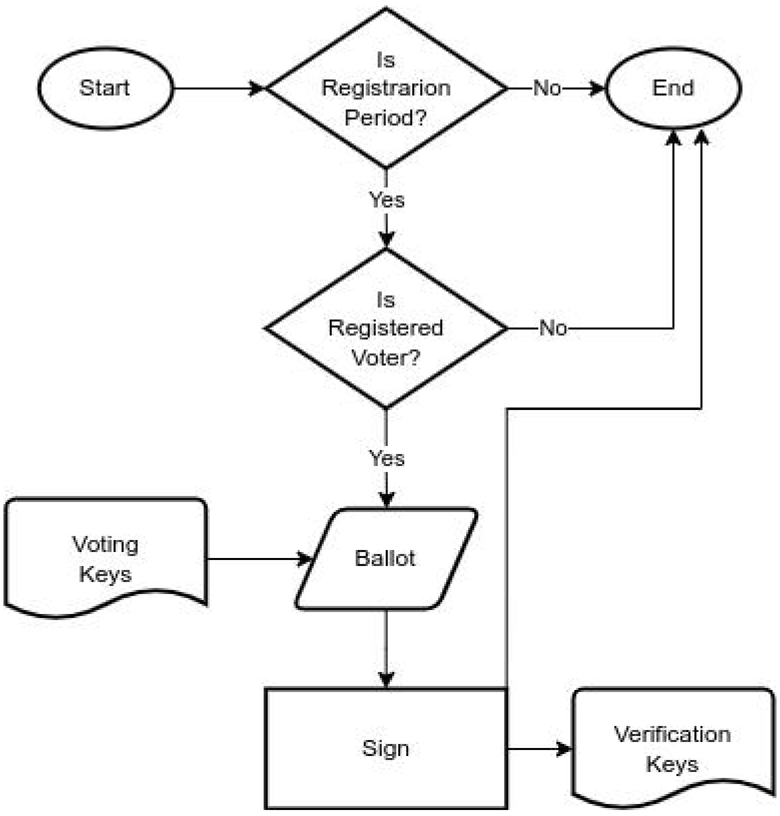


Figure 4-8 Voter Voting Period flowchart

4. 3. 5 Operator Tally Period flowchart

After the voting period has ended, the operator must tally the results (Figure 4-7). He gets the Ballots from the contract and verify them using the blind signature schema and the signing keys. After the tally procedure is finished, the operator updates the final results.

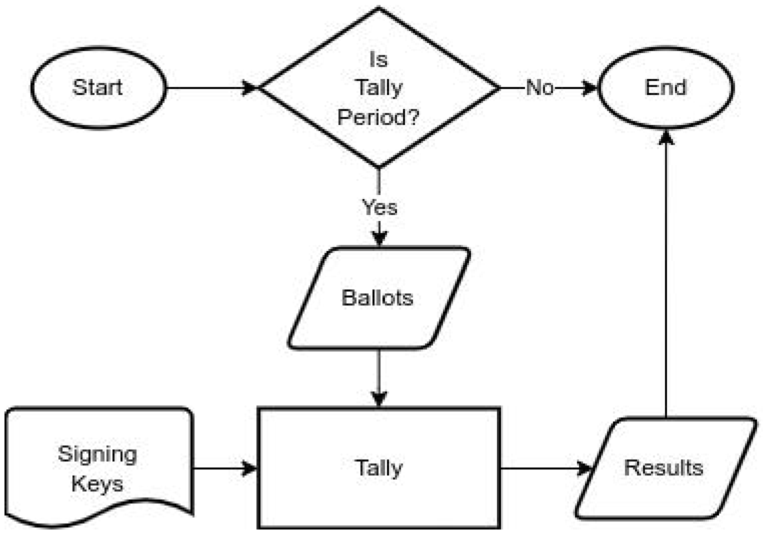


Figure 4-9 Operator Tally Period flowchart

4. 3. 6 Voter Tally Period flowchart

The final stage of the whole voting process is when the voter verifies his vote (Figure 4-8). The voter must wait for the operator to tally the results to verify his vote. Then with the verification keys, he can see his choice. To prevent deanonymization, the voter must delete the keys if he will not use them anymore.

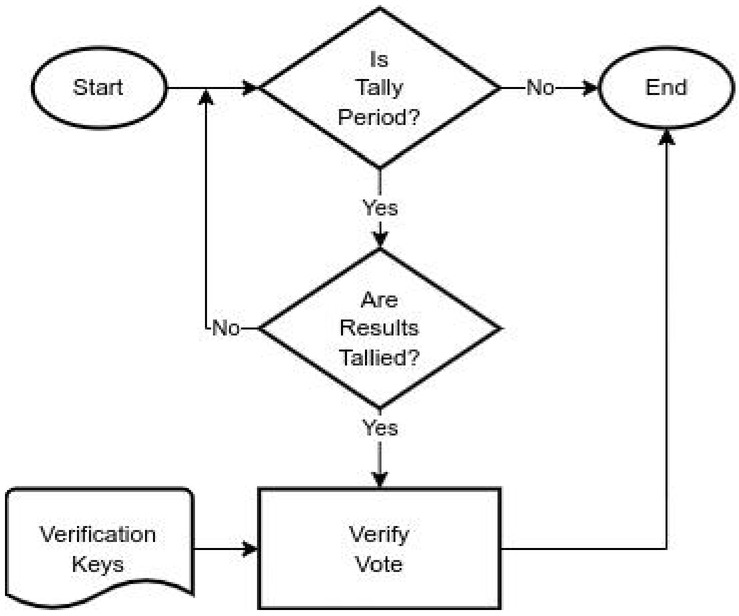


Figure 4-10 Voter Tally Period flowchart

5. System Implementation

The system's software is divided into three parts. Solidity smart contracts with Truffle as a development framework. A front-end Next.js web application. And a Node.js command-line interface (CLI) application, which listens to and logs events, and signs voters' blind UUID.

5. 1 VotingSystem smart contract documentation

A new voting instance is created when a smart contract VotingSystem is deployed by the factory VotingSystemFactory contract to the blockchain.

5. 1. 1 Structs

Several struct data types were initialized and used trough out the whole contract. The main reason behind these structures is for developer ease and code readability.

5. 1. 1. 1 Choice

Name: Choice

Description: The Choice struct contains a single choice and the number of votes the given choice had received during the voting process.

Code:

```
struct Choice {
    string choice;
    uint256 votes;
}
```

5. 1. 1. 2 Voter

Name: Voter

Description: The Voter struct is used to store the voter’s data. Because the system is designed to protect the voter’s privacy, the only data connected to with a voter is his address, the blinded UUID, the signed blinded UUID and whether he has registered to vote or not.

Code:

```
struct Voter {
     address voterAddress;
```

```
    uint256 blindedUUID;
    uint256 signedBlindedUUID;
    bool registered;
}
```

5.1.1.3 **Vote**

Name: Vote

Description: Every voter who voted has a Vote structure corresponding to his UUID stored in a map on the smart contract. This struct aims to check if a given UUID has voted when the votes are tallied and keep a voter's choice.

Code:

```
struct Vote {
    bool voted;
    uint256 choice;
}
```

5.1.1.4 **Ballot**

Name: Ballot

Description: When a voter cast a vote a Ballot is created and stored in an array on the smart contract. A Ballot object has all the data needed to verify if a vote is valid or not. It contains the voter’s public key X and Y value, UUID, the unblinded signed UUID, and the choice. This data was never shared with anyone during the whole process of registration and blind signature procedure. Therefore, casting votes using a struct such as this makes the voting anonymous and ensures voter privacy.

Code:

```
struct Ballot {
   uint256 RX;
   uint256 RY;
   uint256 uuid;
   uint256 unblindedSignedUUID;
   uint256 choice;
}
```

5.1.2 **Events**

Events are used for logging data and flagging the need for blinded UUID signing.

5. 1. 2. 1 **Add voter**

Name: AddedVoter

Description: When a voter is added the event AddedVoter is emitted. Its purpose is for the signer-cli to log the eligible voters.

Table 5- 1 AddedVoter event parameters

Name	**Type**	**Description**
voter	address	Voter’s address.

Code:

```
event AddedVoter(address indexed voter);
```

5. 1. 2. 2 **Register**

Name: Registered

Description: This event is emitted when a voter has registered successfully. Its purpose is to notify the signer-cli application to sign the registered voter blinded UUID.

Table 5- 2 Registered event’s parameters

Name	**Type**	**Description**
voter	address	Voter’s address.
blindedUUID	uint256	Voter’s blinded UUID.

Code:

```
event Registered(address indexed voter, uint256 indexed blindedUUID);
```

5. 1. 2. 3 **Voted**

Name: Voted

Description: This event is emitted when a voter casts his vote. The purpose of the event is so the ballots are logged.

Table 5- 3 Voted event's parameters

Name	**Type**	**Description**
voter	address	Voter’s address.

Name	Type	Description
uuid	uint256	Voter’s UUID.
choice	uint256	Voter’s choice.

Code:

```
event Voted(address indexed voter, uint256 indexed uuid, uint256 indexed choice);
```

5. 1. 3 Constructor

Name: constructor;

Description: Initialize a vote. All the parameters are immutable once set. The name of the vote must not be an empty string. The array of choices must not be empty or contain empty strings. The value of the registration start timestamp must be lesser than the registration end timestamp. The value of the voting start timestamp must be lesser than the voting end timestamp but greater than or equal to the registration end time. Finally, the operator’s public key values must not be zero;

Table 5- 4 Constructor's parameters

Name	Type	Description
_name	string	Name of the vote.
_choices	string[]	An array of choices.
_registrationStart	uint256	The start of the registration period.
_registrationEnd	uint256	The end of the registration period.
_votingStart	uint256	The start of the voting period.
_votingEnd	uint256	The end of the voting period.
_RX	uint256	The X value of the operators signing public key.
_RY	uint256	The Y value of the operators signing public key.

5. 1. 3. 1 Add choices

Name: _addChoices;

Description: A private function executed in the constructor to validate the choices.

Table 5- 5 _addChoices function's parameters

Name	Type	Description
_choices	string[]	An array of choices.

5. 1. 4Add Voters

Name: addVoters;

Description: The operator adds eligible voter addresses to the contract.

Table 5- 6 addVoter function's parameters

Name	Type	Description
_voterAddresses	address[]	An array of voter addresses.

The function addVoters invoke a private function _addVoter.

5. 1. 4. 1 Add voter

Name: _addVoter

Description: This function is executed for every address in the array _voterAddresses. The function ensures that the given address is valid. Also, it ensures the address was not been added previously, therefore preventing the chance of double vote casting. When an address is added successfully the function emits AddedVoter event.

Table 5- 7 _addVoter function's parameters

Name	Type	Description
_voterAddress	address	A voter addresses.

5. 1. 5 Get ballots

Name: getBallots.

Table 5- 8 getBallots function's returns

Name	Type	Description

Name	Type	Description
	tuple[]	Array of type Ballot with all the ballots.

5.1.6 Get choices

Name: getChoices.

Table 5- 9 getChoices function's returns

Name	Type	Description
	tuple[]	Array of type Ballot with all the ballots.

5.1.7 Get registration time

Name: getRegistrationTime.

Table 5- 10 getRegistrationTime function's returns

Name	Type	Description
	uint256	Starting time of registration period.
	uint256	Ending time of registration period.

5.1.8 Get voting time

Name: getVotingTime.

Table 5- 11 getVotingTime function's parameters

Name	Type	Description
	uint256	Starting time of voting period.
	uint256	Ending time of voting period.

5.1.9 Register

Name: register;

Description: The function is called by the voter himself in order to upload the blinded UUID to the contract to be signed. Marking the voter as register, giving him ability to vote in the voting period. The function can be executed only during the registration period by only eligible voters.

Table 5- 12 register function's parameters

Name	Type	Description
_blindedUUID	uint256	The blinded value of the voters UUID.

5. 1. 10 Set voter's signed blinded UUID

Name: signBlindedUUID;

Description: The operator signs the voter’s blinded UUID offchain and uploads it to the contract using this function.

Table 5- 13 signBlindedUUID function's parameters

Name	Type	Description
_voterAddress	address	The address of the given voter.
_signedBlindedUUID	uint256	The address of the given voter.

5. 1. 11 Tally votes

Name: tallyVotes;

Description: The function tally the results after they have been verified as valid by the operator. Requires the time of execution to be past the voting end time. Can be called only once by the operator.

Table 5- 14 tallyVotes function's parameters

Name	Type	Description
_uuids	uint256[]	An array of voters' UUIDs.
_choices	uint256[]	An array of voters' corresponding choices.

5. 1. 11. 1 Tally vote

Name: _tallyVote;

Description: A private function executed inside tallyVotes for every UUID and choice pair. It validates the UUID and the choice. If both are valid then the about of votes the given choice has received is incremented by one.

Table 5- 15 _tallyVote function's parameters

Name	**Type**	**Description**
_uuid	uint256	Voter's UUID.
_choice	uint256	Voter's corresponding choice.

5. 1. 12 Vote

Name: vote;

Description: The voter cast his vote using this function. It takes all the values required to compose a Ballot and cast it. The voter’s public key values must not be zero. Same goes for the UUID and the unblinded signed UUID. This function can be executed only during voting period.

Table 5- 16 vote function's parameters

Name	**Type**	**Description**
_RX	uint256	The X value of the voter's public key.
_RY	uint256	The Y value of the voter's public key.
_uuid	uint256	Voter's UUID.
_unblindedSignedUUID	uint256	Voter's unblinded signed UUID.
_choice	uint256	Voter's choice.

5. 1. 13 Voters

Name: voters;

Description: A mapping with voter’s address as a key and a struct Voter as value.

Table 5- 17 voters mapping key value

Name	**Type**	**Description**
	address	Voter’s address.

Table 5- 18 voters mapping return

Name	Type	Description
voterAddress	address	Voter’s address.
blindedUUID	uint256	The voter’s UUID blinded by the voter.
signedBlindedUUID	uint256	The blinded UUID signed by the operator.
registered	bool	A flag noting if the voter has registered or not.

5. 1. 14 **Votes**

Name: votes

Description: A mapping with voter’s UUID as a key and a struct Vote as value.

Table 5- 19 votes mapping key value

Name	Type	Description
	uint256	Voter’s UUID.

Table 5- 20 votes mapping return

Name	Type	Description
voted	bool	A flag noting if the voter has voted or not.
choice	uint256	Voter’s choice.

5. 2 **VotingSystemFactory smart contract documentation**

The system is developed with the idea of multiple uses with only one setup. In such a system, VotingSystem smart contract must be deployed every time a new vote occurs. The VotingSystemFactory smart contract does this task.

5. 2. 1 **Create**

Name: create;

Description: Deploys a VotingSystem smart contract. All the parameters are immutable once set. The name of the vote must not be an empty string. The array of choices must not be empty or contain empty strings. The value of the registration start timestamp must be lesser than the registration end timestamp. The value of the voting start timestamp must be lesser than the voting end timestamp but greater than or equal to the registration end time. Finally, the operator’s public key values must not be zero.

Table 5- 21 create function's parameters

Name	Type	Description
_name	string	Name of the vote.
_choices	string[]	An array of choices.
_registrationStart	uint256	The start of the registration period.
_registrationEnd	uint256	The end of the registration period.
_votingStart	uint256	The start of the voting period.
_votingEnd	uint256	The end of the voting period.
_RX	uint256	The X value of the operators signing public key.
_RY	uint256	The Y value of the operators signing public key.

5. 3 Blind signatures[18] implementation

The system uses a JavaScript implementation of a elliptic curve blind signature scheme[18]. The scheme uses the secp256k1 curve. This curve is used by Ethereum to sign transactions.

To work the signer (vote operator) has to generate a signing key pair and verification key pair. The person requesting signing (voter) generates two secret values and a public key.

The scheme is implemented in file blind-secp256k1.js. The following documentation describes the functions in the file.

5. 3. 1 New key pair

Name: newKeyPair;

Description: Generates a new elliptic curve key pair for verification. These keys are kept secret locally by the operator.

Table 5- 22 newKeyPair function's returns

Name	Type	Description
d	string	Private key.

Name	Type	Description
QX	string	Signer's public key X value.
QY	string	Signer's public key Y value.

5.3.2 New request parameters

Name: newRequestPair;

Description: Generates a new elliptic curve key pair for signing. The public keys generated by this function are the ones uploaded to the smart contract. The voter uses them when blinding the UUID.

Table 5- 23 newRequestPair function's returns

Name	Type	Description
k	string	Private key.
signerRX	string	Signer's public key X value.
signerRY	string	Signer's public key Y value.

5.3.3 Blind message

Name: blindMessage;

Description: Blinds the message. In case of the system the voter blinds his UUID with this function.

Table 5- 24 blindMessage function's parameters

Name	Type	Description
message	string	Message to bind.
signerRX	BN	Signer's public key X value.
signerRY	BN	Signer's public key Y value.

Table 5- 25 blindMessage function's returns

Name	Type	Description
a	string	First blinding secret value.
b	string	Second blinding secret value.

Name	Type	Description
RX	string	User’s public key X value.
RY	string	User’s public key Y value.
hm	string	Message hash.
blindedMessage	BN	Blinded message.

5. 3. 4 Sign blinded message

Name: signBlindedMessage;

Description: Signs a blinded the message. The operator gets the voter’s blinded UUID from the contract and signs it with this function.

Table 5- 26 signBlindedMessage function's parameters

Name	Type	Description
blindedMessage	BN	Blinded message.
d	string	Signing private key.
k	string	Verification private key.

Table 5- 27 signBlindedMessage function's returns

Name	Type	Description
signedBlindedMessage	BN	A signed blinded message.

5. 3. 5 Unblind signed blinded message

Name: unblindSignedBlindedMessage;

Description: Unblinds the signed blinded message. The voter uses this function to get the unblinded signed message required for the vote.

Table 5- 28 unblindSignedBlindedMessage function's parameters

Name	Type	Description
signedBlindedMessage	BN	Signed blinded message.
a	string	First secret blinding value.
b	string	Second secret blinding value.

Table 5- 29 unblindSignedBlindedMessage function's returns

Name	Type	Description
unblindedSignedMessage	BN	A unblinded signed message.

5. 3. 6 Verify

Name: verify;

Description: This function verifies that the provided message hash is the one used in the process of acquiring the unblinded signed message.

Table 5- 30 verify function's parameters

Name	Type	Description
hm	BN	Message hash.
unblindedSignedMessage	BN	A unblinded signed message.
RX	BN	User’s public key X value.
RX	BN	User’s public key X value.
QX	string	Signer’s verification public key X value.
QY	string	Signer’s verification public key Y value.

Table 5- 31 verify function's returns

Name	Type	Description
	bool	True if the hash is valid, false if not.

5. 4 Next.js web application

The front end of the application is built using the Next.js framework. Next.js is a React.js based type of framework. The UI is built using JSX, HTML, and CSS inside JavaScript files instead of plain HTML and CSS.

A front-end app’s purpose is to interact with the back-end. Here the back end is the blockchain and the smart contracts.

In order to interact with the smart contracts, I created wrapper classes for both VotingSystem and VotingSystemFactory. Classes are used because the code when using the web3.js library is messy, and the return data from the contract functions is messy and needs transformation.

The following code of the functions that interact with the blockchain through the front-end:

5. 4. 1 Create vote

```
const create = async (
  factoryContract,
  name,
  choices,
  registrationStartDate,
  registrationEndDate,
  votingStartDate,
  votingEndDate
) => {
  const ethRegistrationStartDate = (registrationStartDate / 1000) | 0;
  const ethRegistrationEndDate = (registrationEndDate / 1000) | 0;
  const ethVotingStartDate = (votingStartDate / 1000) | 0;
  const ethVotingEndDate = (votingEndDate / 1000) | 0;
  const choicesArray = [];
  choices.forEach((choice) => choicesArray.push(choice.choice));
  // Generate keys using the blind-secp256k1 implementation
  const { d, QX, QY } = newKeyPair();
  const { k, signerRX, signerRY } = newRequestParameters();
  const keys = {
    d,
    QX,
    QY,
    k,
    signerRX: signerRX.toString(16),
    signerRY: signerRY.toString(16),
  };
  // Uses the "VotingSystemFactory" contract wrapper class to create new voting instance
  await factoryContract.create(
    name,
    choicesArray,
      ethRegistrationStartDate,
      ethRegistrationEndDate,
      ethVotingStartDate,
      ethVotingEndDate,
      signerRX,
      signerRY
  );
  // Downloads the signing keys so they can be used by the signer-cli app
  download(keys, "signing-keys");
};
```

5. 4. 2 Add voters

```
const add = async (contract, voters) => {
    const addressArray = [];
    voters.forEach((voter) => addressArray.push(voter.address));
    // Uses the “VotingSystem” contract wrapper class to add voters
    await contract.object.addVoters(addressArray);
};
```

5. 4. 3 Register

```
const register = async (contract) => {
    // Generates UUID
    const uuid = v4();
    // Blinds the UUID using the blind-secp256k1 implementation
    const { a, b, RX, RY, hm, blindedMessage } = blindMessage(
        uuid,
        contract.signerR.RX,
        contract.signerR.RY
    );
    const keys = {
        a,
        b,
        RX,
        RY,
        uuid: hm,
    };
    // Uses the “VotingSystem” contract wrapper class to register
    await contract.object.register(blindedMessage);
    // Downloads the registration keys so they can be updated later
    download(keys, "registration-keys");
};
```

5. 4. 4 Sign

```
const sign = async (contract, keys, voters) => {
  for (const v of voters) {
    // Uses the “VotingSystem” contract wrapper class to get a voter data
    const voter = await contract.object.voter(v.address);
    // Signs the blinded UUID using the blind-secp256k1 implementation
    const signedBlindedMessage = signBlindedMessage(
      voter.blindedUUID,
      keys.d,
      keys.k
    );
    // Uses the “VotingSystem” contract wrapper class to set voters signed blinded UUID
    await contract.object.signBlindedUUID(v.address, signedBlindedMessage);
  }
};
```

5. 4. 5 Update

```
const update = (contract, keys) => {
  // Update keys
  keys.signedBlindedUUID = contract.voter.signedBlindedUUID;
  // Downloads the updated keys so they can be used during voting
  download(keys, "voting-keys");
};
```

5. 4. 6 Vote

```
const vote = async (contract, keys, choice) => {
  // Unblinds the signed blinded UUID using the blind-secp256k1 implementation
  const unblindedSignedUUID = unblindSignedBlindedMessage(
    keys.signedBlindedUUID,
    keys.a,
    keys.b
  );
  // Update keys
  keys.unblindedSignedUUID = unblindedSignedUUID;
  // Downloads the keys so they can be used for vote verification later
  download(keys, "verification-keys");
  // Uses the “VotingSystem” contract wrapper class to cast a vote
  await contract.object.vote(
    keys.RX,
    keys.RY,
    keys.uuid,
    unblindedSignedUUID,
    choice
  );
};
```

5.4.7 **Tally**

```
const tally = async (contract, keys) => {
  // Uses the “VotingSystem” contract wrapper class to get the ballots
  const ballots = await contract.object.ballots();
  const uuids = [];
  const choices = [];
  ballots.forEach((ballot) => {
    // Verify the ballot integrity using the blind-secp256k1 implementation
    const isValid = verify(
      ballot.uuid,
      ballot.unblindedSignedUUID,
      ballot.RX,
      ballot.RY,
      keys.QX,
      keys.QY
    );
    if (isValid) {
      uuids.push(ballot.uuid);
      choices.push(ballot.choice);
    }
  });
  if (uuids.length === choices.length && uuids.length !== 0)
    // Uses the “VotingSystem” contract wrapper class to tally the votes
    await contract.object.tallyVotes(uuids, choices);
  else alert("No ballots to tally");
};
```

5.4.8 **Verify vote**

```
const verifyVote = async (contract, keys) => {
  const vote = await contract.object.votes(keys.uuid);
  alert(`You voter for: ${contract.choices[vote.choice].choice}`);
};
```

5. 5 Signer command-line interface application

signer-cli is a Node.js module. It's main purpose it to sign the voters' blinded UUIDs automatically instead of the operator manually check if a voter has requested a signature and then sign the UUID from the user interface (UI).

While the application is running, it listens to event using a WebSocket connected to the blockchain and the web3.js API, and makes logs.

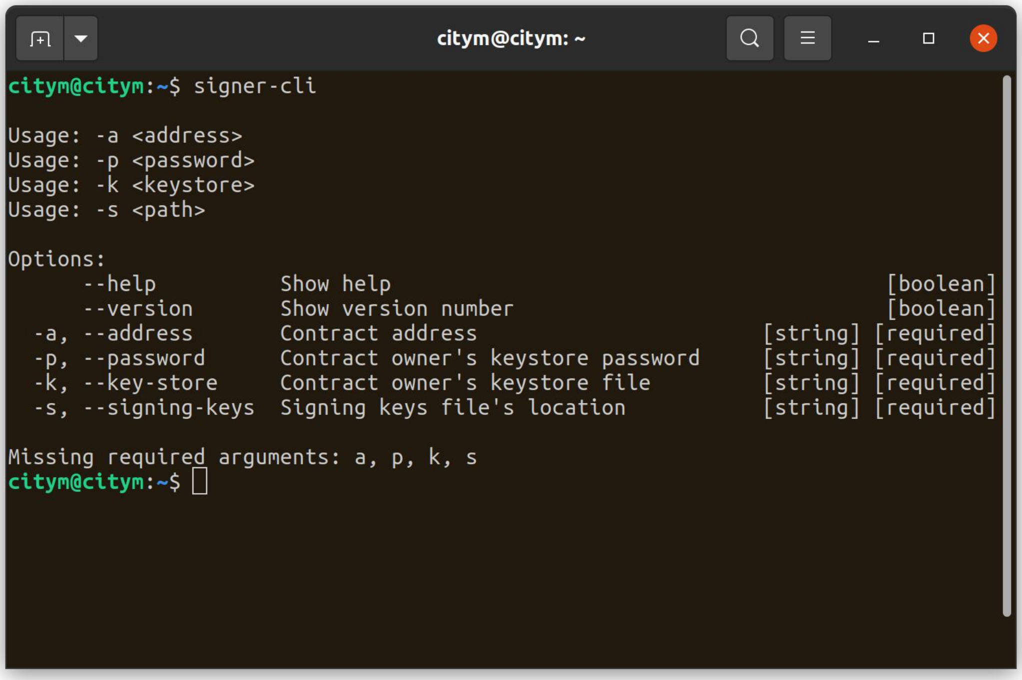
The CLI application is an Node.js module so it can be installed locally using npm. This is done by going to the module's directory in the terminal and running the following command:

```
npm i -g .
```

Once the module is installed, it can be called from anywhere. For example, the following command is used to start the signer-cli module:

```
signer-cli
```

The command by itself without any options will return the following message describing the required options and their description:



```
citym@citym:~$ signer-cli

Usage: -a <address>
Usage: -p <password>
Usage: -k <keystore>
Usage: -s <path>

Options:
      --help          Show help                                        [boolean]
      --version       Show version number                              [boolean]
  -a, --address       Contract address                       [string] [required]
  -p, --password      Contract owner's keystore password     [string] [required]
  -k, --key-store     Contract owner's keystore file         [string] [required]
  -s, --signing-keys  Signing keys file's location           [string] [required]

Missing required arguments: a, p, k, s
citym@citym:~$ 
```

Figure 5- 1 signer-cli Description

6. Deployment and Testing

In the last chapter, the system is deployed and tested.

6. 1 System Deployment

6. 1. 1 Development environment

The hardware and software environments are as follows:

Table 6- 1 Development environment

Type	Specifications
Hardware	RAM: 2x8 GB SODIMM DDR4 Synchronous 2667 MHz
Environment	CPU: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
Software	Operation System: Ubuntu 20.04 LTS
Environment	Development Environment: Go-Ethereum client, VS Code IDE
	JavaScript frameworks: Node.js, Next.js
	Deployment frameworks: Truffle

6. 1. 2 System deployment architecture

All Web2 applications are front-end and back-end, which serve data to the front-end. Likewise, Web3 dApps require the same. The only difference is in the back-end, where Web2 apps will connect to databases, where Web3 dApps will connect to the blockchain.

The system connects to the blockchain using an RPC connection provided by Geth. The Truffle framework deploys the VotingSystemFactory contract through this connection before the system is operational. Later, when the system runs via that RPC connection, the operator will deploy voting instances from the front-end.

Using the RPC, the users who participate in the vote connect their wallets to the private blockchain connection and interact with smart contracts through the front-end.

The architecture of the deployed system can be seen in (Figure 6-1).

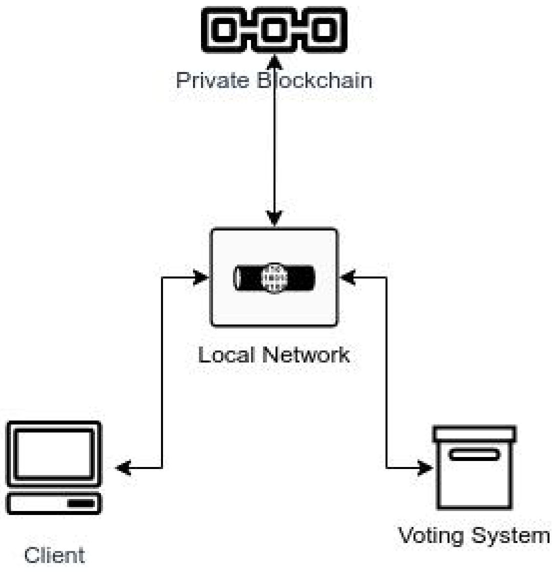


Figure 6- 1 Deployed System

6. 2 System module function test

This section displays the black-box, functional, integration and system tests.

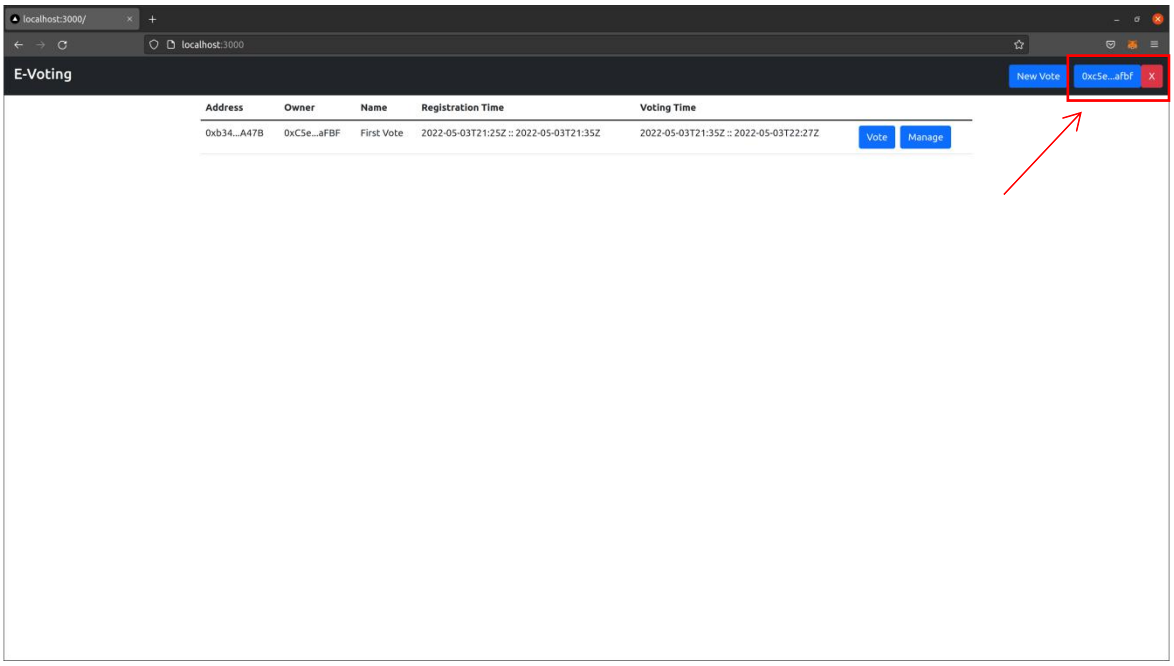
6. 2. 1 Connect with MetaMask

Operators and voters must connect to the private blockchain with a Metamask wallet to interact with the system.

Table 6- 2 Connect MetaMask wallet test case and result

Item	Description
Use case id	Case 1
Use case name	Connect MetaMask wallet
Conditions	MetaMask is already installed.
Testing purposes	Test whether voters and operators can connect to the blockchain.
Steps	(1) Open the application in the browser;
	(2) Click the "Connect" button.
Group 1 Expected Results	Operators can connect using the node address.

Group 1 Actual Results	Operators can connect using the node address.
Group 2 Expected Results	Voters can connect using their own addresses.
Group 2 Actual Results	Voters can connect using their own addresses.
Group 3 Expected Results	An alert window pops up when trying to connect to the wrong network.
Group 3 Actual Results	An alert window pops up when trying to connect to the wrong network.



Address	Owner	Name	Registration Time	Voting Time
0xb34...A47B	0xC5e...aFBF	First Vote	2022-05-03T21:25Z :: 2022-05-03T21:35Z	2022-05-03T21:35Z :: 2022-05-03T22:27Z

Figure 6- 2 Operators can connect using the node address

6. 2. 2 Vote creation

The operator must create a new voting instance. Creating a new vote deploys a new VotingSystem smart contract on the blockchain. This is done through the UI. The following tests (Table 6-3) show whether this feature works properly.

Table 6- 3 Create a new polling test case and result

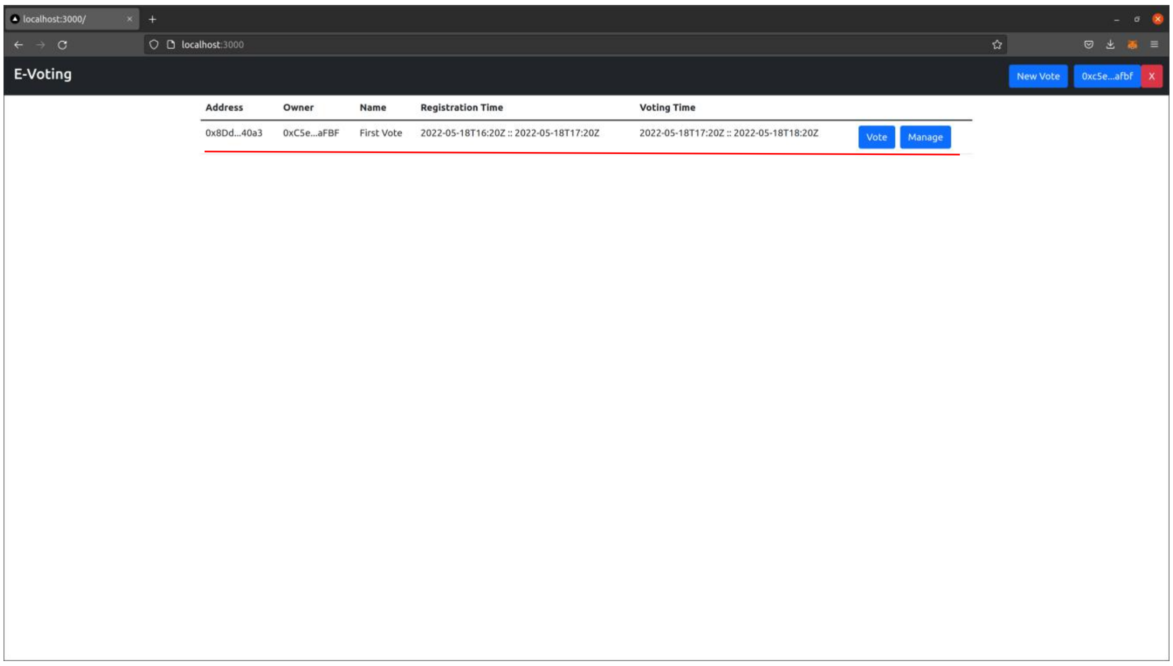
Item	**Description**
Use case id	Case 2
Use case name	Create a new poll

Condition	The operator is connected to his node address.
Testing purposes	Test if operators can create new polls.

Steps	（1） Open the application in the browser;
	（2） Click the "New Vote" button;
	（3） Input voting data;
	（4） Click the "Submit" button.

Group 1 test data	Vote Name: Fist Vote Choice 0: A Choice 1: B Choice 2: C Registration Start Date: 05/18/2022, 07:20 PM Registration End Date: 05/18/2022, 08:20 PM Voting Start Date: 05/18/2022, 08:20 PM Voting End Date: 05/18/2022, 09:20 PM
Group 1 Expected Results	When the correct data is entered, a new vote is created and the operator can download the signing key for the specific vote.
Group 1 Actual Results	When the correct data is entered, a new vote is created and the operator can download the signing key for the specific vote.

Figure 6-3 shows the successful creation of the poll for the first test. Times are converted to UTC for people from different time zones. They can connect to the private network via VPN and vote. UTC is popular and used globally, especially in the blockchain community.



Address	Owner	Name	Registration Time	Voting Time
0x8Dd...40a3	0xC5e...aFBF	First Vote	2022-05-18T16:20Z :: 2022-05-18T17:20Z	2022-05-18T17:20Z :: 2022-05-18T18:20Z

Figure 6- 3 Created a vote successfully

6. 2. 3 **Add voters**

After creating the vote, the operator must add the voter's address to the smart contract. Table 6-4 is the test performed for this function.

Table 6- 4 Add voters test cases and results

Item	**Description**
Use case id	Case 3
Use case name	Add voters
Condition	The operator is connected to his node address.
Testing purposes	Test if the operator can add voters.
Steps	（1） Open the application in the browser;
	（2） Click the "Manage" button on this poll;
	（3） Select the "Add Voters" tab;
	（4） Add voters;
	（5） Click the "Add" button.
Group 1 test data	Voter 0: 0x871ac0c271Bf817517b05F52A540Bf317525Ba2e
	Voter 1: 0xe4BACd4F44Ddc422c7CefB7c90C8a2f02f346078

	Voter 2: 0xDc47CDdB2f71564e0b7cA1d40a730E6352125B04
Group 1 Expected Results	Voter added successfully
Group 1 Actual Results	Voter added successfully
Group 2 test data	Voter 0:
Group 2 Expected Results	Display an alert with an error message.
Group 2 Actual Results	Display an alert with an error message.

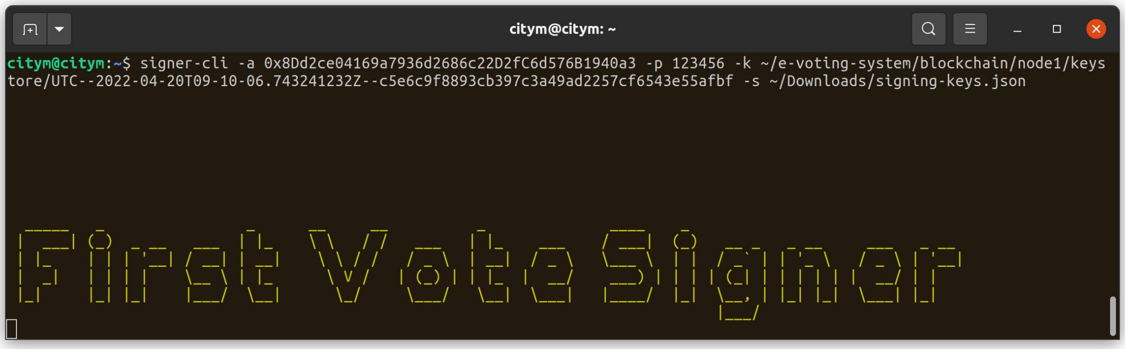
6. 2. 4 **Running signer-cli**

The signer-cli app is used to sign voters' UUIDs and make logs. Operators use it for specific polls. The application must be running on the node machine. The tests performed on signer-cli are shown in Table 6-5.

Table 6- 5 signer-cli black box testing

Item	**Description**
Use case id	Case 4
Use case name	run signer-cli
Condition	signer-cli must be started on the node's machine.
Testing purposes	Tests if the operator can start the signer-cli for the given vote.
Steps	（1） Open a terminal window （2） Enter the signer-cli command with the desired options in the terminal.
Group 1 test data	-a: 0x8Dd2ce04169a7936d2686c22D2fC6d576B1940a3 -p: 123456 -k: ~/e-voting-system/blockchain/node1/ keystore/UTC--2022-04-20T09-10-06. 743241232Z--c5e6c9f8893cb397c3a49ad2257cf6543e55afbf -s: ~/Downloads/signing-keys.json
Group 1 Expected Results	Successfully run signer-cli.
Group 1 Actual Results	Successfully run signer-cli.

Group 2 test data	-a: <invalid address>
Group 2 Expected Results	The rest is the same as Test 1.
Group 2 Actual Results	signer-cli doesn't start and throws an error.
Group 3 test data	signer-cli doesn't start and throws an error.
Group 3 Expected Results	-p: <invalid address>
Group 3 Actual Results	The rest is the same as Test 1.
Group 4 test data	signer-cli doesn't start and throws an error.
Group 4 Expected Results	signer-cli doesn't start and throws an error.
Group 4 Actual Results	-k: <invalid address>
Group 5 test data	The rest is the same as Test 1.
Group 5 Expected Results	signer-cli doesn't start and throws an error.
Group 5 Actual Results	signer-cli doesn't start and throws an error.



```
citym@citym:~$ signer-cli -a 0x8Dd2ce04169a7936d2686c22D2fC6d576B1940a3 -p 123456 -k ~/e-voting-system/blockchain/node1/keys
tore/UTC--2022-04-20T09-10-06.743241232Z--c5e6c9f8893cb397c3a49ad2257cf6543e55afbf -s ~/Downloads/signing-keys.json
```

Figure 6- 4 singer-cli running

6. 2. 5 **Register**

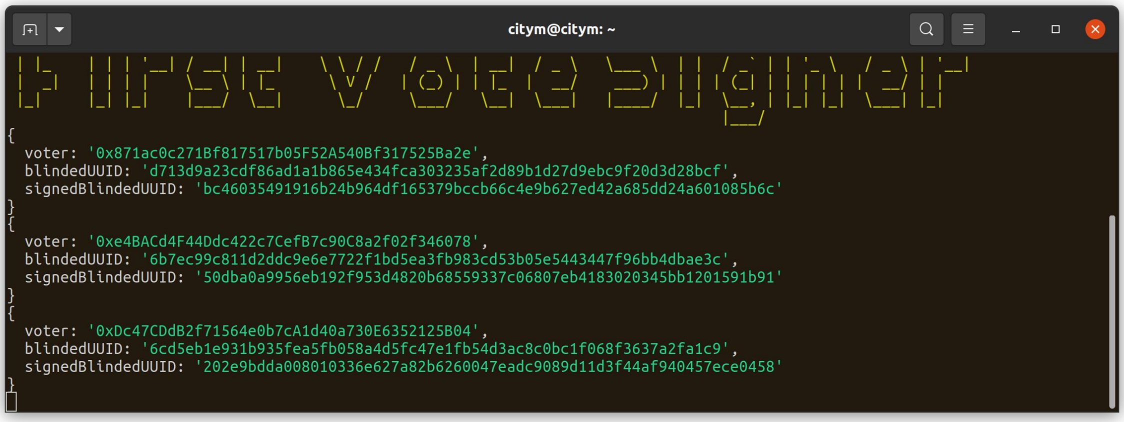
After the operator adds the addresses, voters must register to vote. They must go to the voting page by pressing the button and register from the registration tab. The test results are shown in Table 6-6.

Table 6- 6 Voter registration test

Item	**Description**
Use case id	Case 5

Use case name	voter registration
Condition	Connect to the blockchain and be an eligible voter.
Testing purposes	Test if voters can register to vote.
Steps	（1） Open the application in the browser; （2） Click the "Vote" button on this poll; （3） Select the "Register" tab; （4） Click the "Register" button.
Group 1 Expected Results	Eligible voters successfully registered and downloaded the registration key.
Group 1 Actual Results	Eligible voters successfully registered and downloaded the registration key.
Group 1 Expected Results	MetaMask displayed an error when trying to register outside the registration period.
Group 1 Actual Results	MetaMask displayed an error when trying to register outside the registration period.
Group 2 Expected Results	Voters who are not eligible will see a "Not Eligible For Registration" message.
Group 2 Actual Results	Voters who are not eligible will see a "Not Eligible For Registration" message.

When voters sign up, the signer-cli application is running and signing incoming requests (Figure 6-5).



```
{
  voter: '0x871ac0c271Bf817517b05F52A540Bf317525Ba2e',
  blindedUUID: 'd713d9a23cdf86ad1a1b865e434fca303235af2d89b1d27d9ebc9f20d3d28bcf',
  signedBlindedUUID: 'bc46035491916b24b964df165379bccb66c4e9b627ed42a685dd24a601085b6c'
}
{
  voter: '0xe4BACd4F44Ddc422c7CefB7c90C8a2f02f346078',
  blindedUUID: '6b7ec99c811d2ddc9e6e7722f1bd5ea3fb983cd53b05e5443447f96bb4dbae3c',
  signedBlindedUUID: '50dba0a9956eb192f953d4820b68559337c06807eb4183020345bb1201591b91'
}
{
  voter: '0xDc47CDdB2f71564e0b7cA1d40a730E6352125B04',
  blindedUUID: '6cd5eb1e931b935fea5fb058a4d5fc47e1fb54d3ac8c0bc1f068f3637a2fa1c9',
  signedBlindedUUID: '202e9bdda008010336e627a82b6260047eadc9089d11d3f44af940457ece0458'
}
```

Figure 6- 55 signer-cli sign blind UUID

6. 2. 6 Update keys

After the voter is signed, he must update his keys. The keys update test is shown in Table 6-7.

Table 6- 7 Update keys test

Item	**Description**
Use case id	Case 6
Use case name	Update keys
Condition	Voters who are connected to the blockchain and registered.
Testing purposes	Test if the operator can add voters.
Steps	（1） Open the application in the browser;
	（2） Click the "Vote" button on this poll;
	（3） Select the "Register" tab;
	（4） Upload the registration key;
	（5） Click the "Update" button.
Group 1 Expected Results	Successfully updated registration keys and downloaded voting keys.
Group 1 Actual Results	Successfully updated registration keys and downloaded voting keys.

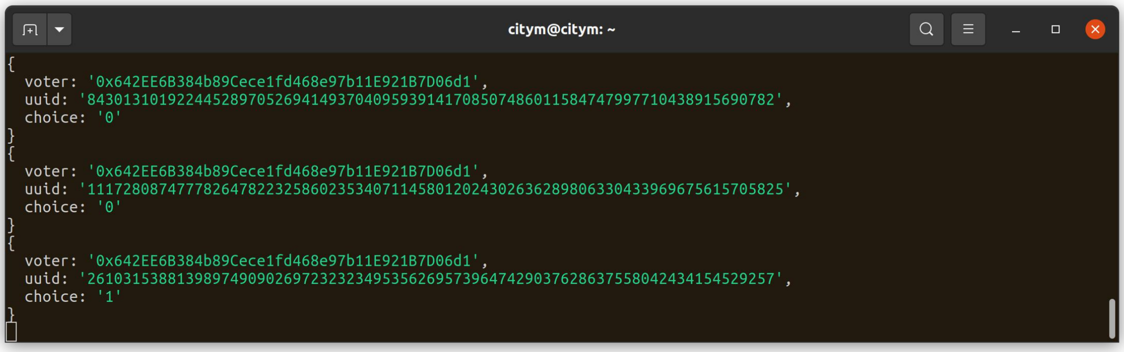
6. 2. 7 Vote

Voting can only be done during the voting period. In order to vote, the voter must upload his voting key. If a voter wants to remain anonymous, he must vote using a different address than the one he used to register.

Table 6- 8 Voting test

Item	**Description**
Use case id	Case 7
Use case name	vote
Condition	Connect to the blockchain and be an eligible voter.
Testing purposes	Test if voters can vote.
Steps	（1） Open the application in the browser; （2） Click the "Vote" button on this poll; （3） Select the "Vote" tab; （4） Upload the voting key; （5） Select an option; （6） Click the "Vote" button.
Group 1 Expected Results	Successfully voted during the voting period.
Group 1 Actual Results	Successfully voted during the voting period.
Group 1 Expected Results	Failed to vote outside the voting period.
Group 1 Actual Results	Failed to vote outside the voting period.

A log of voters' votes can be seen in Figure 6-6. All votes are from the same address to indicate that there is no link to the starting registration address. Thus, making the voting process private and anonymous.



```
{
  voter: '0x642EE6B384b89Cece1fd468e97b11E921B7D06d1',
  uuid: '84301310192244528970526941493704095939141708507486011584747997710438915690782',
  choice: '0'
}
{
  voter: '0x642EE6B384b89Cece1fd468e97b11E921B7D06d1',
  uuid: '111728087477782647822325860235340711458012024302636289806330433969675615705825',
  choice: '0'
}
{
  voter: '0x642EE6B384b89Cece1fd468e97b11E921B7D06d1',
  uuid: '26103153881398974909026972323234953562695739647429037628637558042434154529257',
  choice: '1'
}
```

Figure 6- 6 Votes log

6. 2. 8 **Tally results**

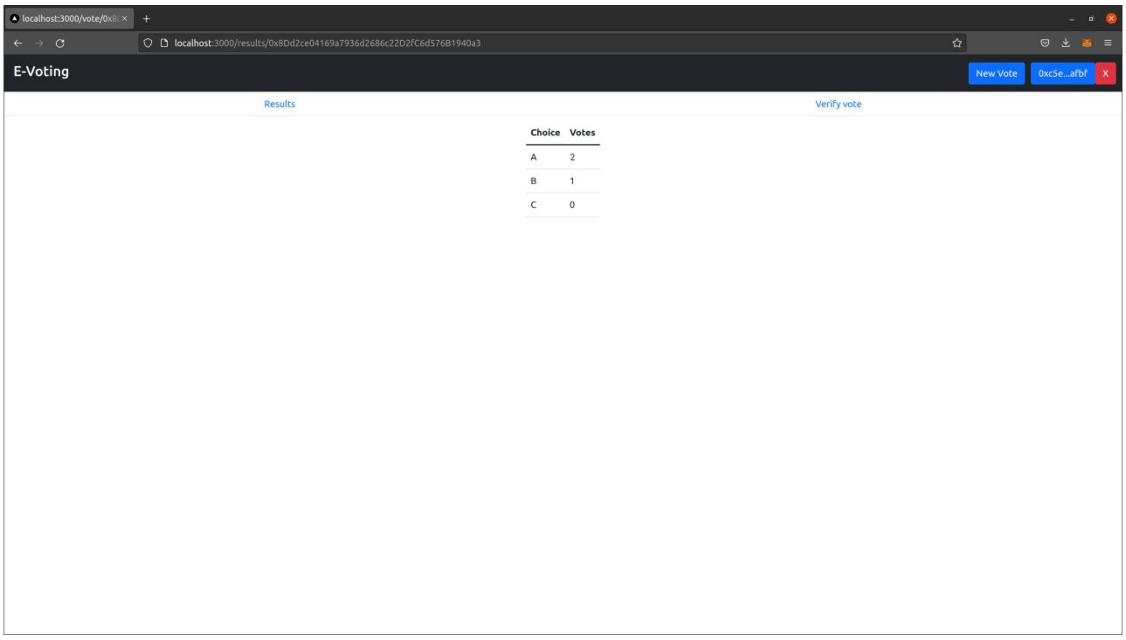
After the voting period ends, the operator must count the votes. He has to go to the Tally Votes tab, upload the signing key and calculate the result. The results of this test are shown in Tables 6-9.

表 6- 9 Tally results tests

Item	**Description**
Use case id	Case 8
Use case name	Tally results
Condition	The time is after the voting period. The operator is connected to the blockchain.
Testing purposes	The process of testing calculated results.
Steps	（1） Open the application in the browser;
	（2） Click the "Manage" button on this poll;
	（3） Select the "Tally Votes" tab;
	（4） Upload the signature key;
	（5） Click the "Tally" button.
Group 1 Expected Results	Results cannot be counted until the end of the voting period.
Group 1 Actual Results	Results cannot be counted until the end of the voting period.
Group 2 Expected Results	The result is calculated successfully.
Group 2 Actual Results	The result is calculated successfully.
Group 3 Expected Results	If it has already been counted, the result cannot be counted.

Group 3 Actual Results	If it has already been counted, the result cannot be counted.

As you can see in Figure 6-6, two votes voted 0 and one voted 1, which means that two voters chose A and one voter chose B. Same result as shown in Figure 6-7.



Choice	Votes
A	2
B	1
C	0

Figure 6- 7 Results

6. 2. 9 Vote verification

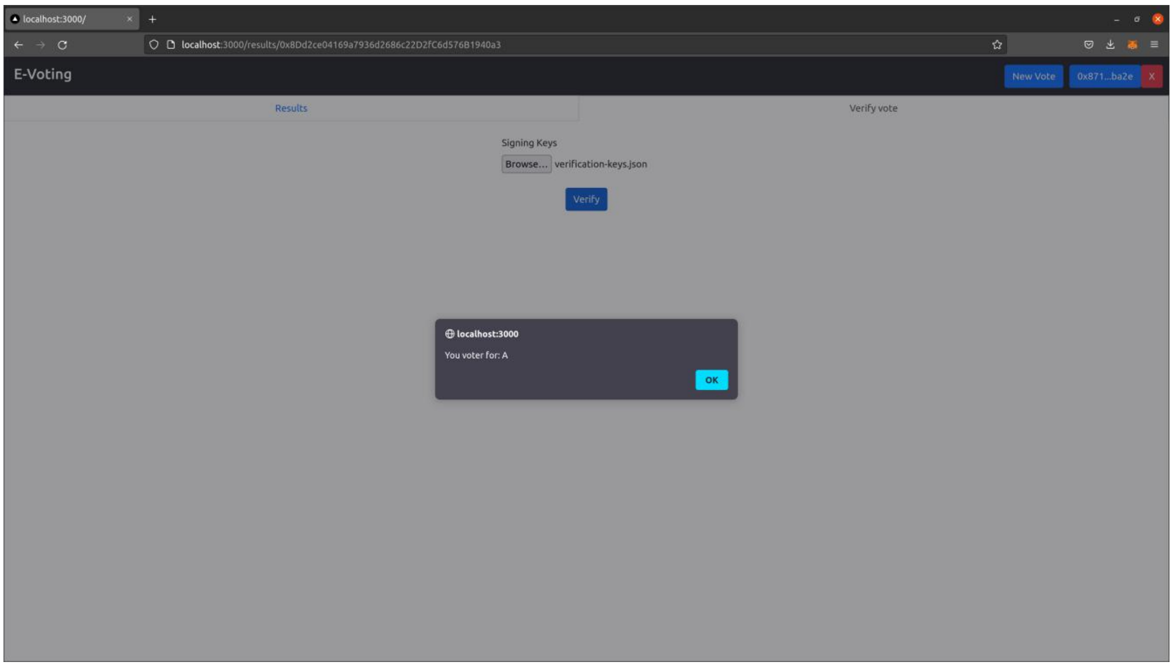
To verify his vote, the voter must upload his verification key and then compare the displayed results with the choices he made during the vote.

Table 6- 9 Vote verification test

Item	**Description**
Use case id	Case 9
Use case name	Vote verification
Condition	The time is after the voting period. Voters are connected to the blockchain.
Testing purposes	Test whether the vote can be verified.

Steps	（1） Open the application in the browser; （2） Click the "Results" button on this poll; （3） Select the "Verify vote" tab; （4） Upload the verification key; （5） Click the "Verify" button.
Group 1 Expected Results	Displays the same choices as those made during voting.
Group 1 Actual Results	Displays the same choices as those made during voting.

In Figure 6-8, you can see how authentication is displayed to the user. In this specific example, the address used in voting voted A, and the validation showed A, so the functionality worked. It is best to use a different address than the one used when registering to verify the vote. Also, after verification is complete, it's a good idea to delete the key.


localhost:3000/results/0x8Dd2ce04169a7936d2686c22D2fC6d576B1940a3

localhost:3000

You voter for: A

Figure 6- 8 Voter verifies his vote

Conclusion

The entire project summarized is a voting system based on Ethereum's private blockchain that uses blind signatures to enforce privacy. The proposed system is trusted, meaning voters must trust the operator to sign and process their ballots honestly. The only way to ensure the operator's honesty is for each voter to verify his vote after the results are out. All this, combined with smart contracts, can prove that votes are honest.

The system consists of three parts, including Solidity smart contracts and an Ethereum private chain, a Next.js front-end web application, and a Node.js CLI application for signing voters and recording vote data.

The system uses a private Ethereum blockchain and adopts the PoA consensus algorithm to ensure the security and stability of the network.

Web3.js is a library that connects the pieces by providing a way for JavaScript code to interact with the blockchain. First, the Truffle framework uses this library to deploy the VotingSystemFactory contract. Then with the help of the web3.js library and MetaMask in the Next.js application, users can easily connect to the blockchain and interact with smart contracts without prior knowledge about the blockchain. Also, the web3.js library is used in the Node.js CIL application to monitor events and automate signing.

Finally, after testing, the end result is a system that implements a credible end-to-end verifiable electronic voting that guarantees the privacy of the voting process.

References

[1] STAVELEY E. Greek and Roman voting and elections[M]. London: Thames and Hudson, 1972.

[2] KRIMMER R., DUENAS-CID D., KRIVONOSOVA I. et al. How Much Does an e-Vote Cost? Cost Comparison per Vote in Multichannel Elections in Estonia[A]. In: KRIMMER R. et al. Electronic Voting[M]. Cham: Springer International Publishing, 2018: 117-131.

[3] HEIBERG S, WILLEMSON J. Verifiable internet voting in Estonia[J]. 2014 6th International Conference on Electronic Voting: Verifying the Vote (EVOTE), 2014: 1-8.

[4] NAKAMOTO S. Bitcoin: A Peer-to-Peer Electronic Cash System[EB/OL]. Bitcoin.org, 2008. https://bitcoin.org/bitcoin.pdf.

[5] WOOD G. Ethereum: A Secure Decentralized Generalized Transaction Ledger[EB/OL]. Ethereum.github.io, 2014. https://ethereum.github.io/yellowpaper/paper.pdf.

[6] CLACK C, BAKSHI V, BRAINE L. Smart Contract Templates: foundations, design landscape and research directions[J]. CoRR, 2016, abs/1608.00771.

[7] CHAUM D. Computer Systems Established, Maintained and Trusted by Mutually Suspicious Groups[D]. UNIVERSITY OF CALIFORNIA, BERKELEY, 1979.

[8] CHAUM D. Untraceable electronic mail, return addresses, and digital pseudonyms[J]. Commun. ACM, 1981, 24: 84-88.

[9] CHAUM D. Blind Signatures for Untraceable Payments[A]. In: CHAUM D. et al. Advances in Cryptology[M]. Boston, MA: Springer US, 1983: 199-203.

[10]] CHAUM D. Elections with Unconditionally-Secret Ballots and Disruption Equivalent to Breaking RSA[A]. In: BARSTOW D., GRIES D. et al. Advances in Cryptology --- EUROCRYPT '88[M]. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988: 177-182.

[11] CHAUM D. Secret-ballot receipts: True voter-verifiable elections[J]. IEEE Security Privacy, 2004, 2(1): 38-47.

[12] MCCORRY P., SHAHANDASHTI S., HAO F. A Smart Contract for Boardroom Voting with Maximum Voter Privacy[A]. In: KIAYIAS A. Financial Cryptography and Data Security[M]. Cham: Springer International Publishing, 2017: 357-375.

[13] QIXUAN Z., BOWEN X., HAOTIAN J. et al. Ques-Chain: An Ethereum Based E-Voting System[J]. CoRR, 2019, abs/1905.05041.

[14] LIU Y., WANG Q. An E-voting Protocol Based on Blockchain[J]. IACR Cryptol. ePrint Arch., 2017.

[15] PAWLAK M., PONISZEWSKA-MARAŃDA A., KRYVINSKA N. Towards the intelligent agents for blockchain e-voting system[J]. Procedia Computer Science, 2018, 141(1877-0509): 239-246.

[16] CANARD S.，GAUD M.，TRAORéJ. Defeating malicious servers in a blind signatures based voting system[A]. In: DI CRESCENZO G., RUBIN A. Financial Cryptography and Data Security[M]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006: 148-153.

[17] PANJA S., ROY B. A Secure End-to-End Verifiable E-Voting System Using Zero-Knowledge Proof and Blockchain[A]. In: CHAUDHURI A., GUPTA S., ROYCHOUDHURY R. A Tribute to the Legend of Professor C. R. Rao: The Centenary Volume[M]. Singapore: Springer Singapore, 2021: 45-48.

[18] MALA H., NEZHADANSARI N. New blind signature schemes based on the (elliptic curve) discrete logarithm problem[A]. In: et al. ICCKE 2013[M]. 2013: 196-201.

[19] SPECTER M., KOPPEL J., WEITZNER D. The Ballot is Busted Before the Blockchain: A Security Analysis of Voatz, the First Internet Voting Application Used in {U.S}. Federal Elections[A] In: 29th USENIX Security Symposium (USENIX Security 20). USENIX Associatio, 2020: 1535--1553.

Acknowledgment

I want to sensiarly thank all my mentor Sun Jianwei for the help I have received from him. Providing me guidance and keeping me in track to finish the thesis.

I show my gratitude towards the teachers from the foreign students office, especially (Dasy), and the China Scholarship Council (CSC) for providing me with a scholarship and making the dream to study in China come true.

During the wring of the thesis, I received tremendous support from my family, both morally and financially. Therefore, I also want to express my gratitude to them.

Thanks to all my friends who helped me throughout the writing process. Of all these people, I would like to especially thank my friend Stefan Shutev for letting me spend time in his home while writing my dissertation.

Finally, I would like to bid farewell to China and all the great Chinese people and wish them great success and prosperity.