The Hong Kong Polytechnic University

Department of Computing

COMP4913 Capstone Project

Report (Final)

Blockchain Unleashed:

A Secure E-Voting Application

Student Name:	Yiu Kam Wing
Student ID NO:	20028987D
Programme-Stream Code:	61431-SYC
Supervisor:	Dr Yan Wang Liu Dennis
Co-Examiner:	Dr Li Chen Richard
2nd Assessor:	Dr Xiapu Luo Daniel
Submission Date:	13 April 2022

Abstract

Designing a secure e-voting system that protects the pricy of the voter and guarantees the correctness of the voting result meanwhile providing the transparency and immutability of the e-voting system is always a challenge. This report proposed a blockchain-based e-voting system to overcome the limitation of a centralized e-voting system. The trusted computing environment and public bulletin board are provided to guarantee that the voting result cannot tamper with the smart contract on a permissionless blockchain. A commitment scheme is used to computationally hide the sensitive information of the participant for registration in the blockchain. To only allow eligible participants to vote without revealing the voter's identity and prevent multiple voting, a linkable ring signature is used that each voter signs his vote on behalf of all eligible voters, and his signature is linked to the signer anonymously. A threshold cryptosystem is used to protect the voting result only can be viewed after the election ends. Because the public key encrypts each vote that anyone knows, the private key for decrypting the vote is separated from all voters, and reconstructing the private key requires the cooperation of some of the voters (exceeding the threshold). This report also compares our protocol with other blockchain-based e-voting protocols. Moreover, the security, time, and cost analysis are provided.

***Keywords*—** E-voting, Ethereum, Smart Contract, Commitment Scheme, Linkable Ring Signature, Threshold Cryptosystem

Contents

List of Figures

List of Tables

Chapter 1

Background and Problem Statement

1.1 Traditional Voting

Humans live in groups. Also, voting is widely used in our society to make collective decisions for groups. For example, the selection of a president in democratic societies is achieved by voting. However, the traditional voting system, such as the long time required, substantial costs, and cheating in paper balloting. For example, the United State election 2020 spending to almost $14 billion [1].

1.2 Centralized E-voting

Therefore, owing to the advances in technology, many electronic voting applications were developed such that the time and cost for an election were deceased. For example, Estonia is the first country in the world to hold nationwide elections using internet voting, which allows voters to cast their vote from anywhere in the world by using any internet-connected computer[2]. Traditional e-voting architecture is shown in figure 1.1. The voter sends the request through the font-end for voting and views the voting result from the back-end, then the back-end fetches or adds the data from the database and tally the ballots and then returns the result to the voter through the font-end.

Nonetheless, these e-voting systems cannot resolve the problem of election organization manipulation. There are two disadvantages of centralized systems:

Opacity: These organizations or authorities have complete control over the database and system such that they can manipulate the election result without anyone knowing. Therefore, the participant needs to rely on trust in the election organizer.

Single point of failure: The database and the server can be compromised by the hacker without anyone knowing.

A centralized system cannot guarantee that the ballot result did not modify by the election organization or

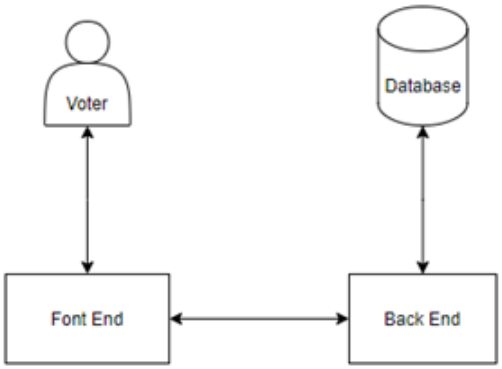


Figure 1.1: Traditional E-Voting Architecture

malicious hacker, and the tally ballot process cannot be shown transparently. Therefore, the centralized system cannot use in voting because voter privacy and the credibility of the voting result are the most critical criteria in voting.

1.3 Decentralized E-voting

In 2009, blockchain technology was born in response to eliminating the drawbacks of a centralized system [3]. A blockchain is a distributed public ledger in which all data in the blockchain are stored in a peer-to-peer (P2P) network and can be verified by the network without a trusted party.

1.3.1 P2P network

A P2P network is a decentralized network that communication within this network can transmit without a central server. Each node of the network acts as a server and a client simultaneously that owns a copy of data in the network and can update the data in the network. However, any update in the blockchain must be agreed upon by the consensus algorithm.

1.3.2 Cryptographic hash

Before we illustrate the consensus algorithm, we first need to understand the cryptographic hash and Merkle tree. A cryptographic hash is a function that maps an arbitrary length input to a fixed-length output and is mainly used for data integrity while fulfilling the properties below:

Deterministic: the hash function must always generate the same output given the same input.

Preimage- resistance: it is computationally infeasible to find the input from the hash function's output.

Collision- resistance: it is computationally infeasible to find the two different inputs have the same output of the hash function.

Avalanche effect: the output of hash function changes significantly if the input is changed slightly.

Computationally Efficient: it is computationally efficient to compute the output of the hash function.

1.3.3 Merkle tree

Merkle tree is a data structure tree in that each leaf node (transaction in the blockchain) is labelled with the cryptographic hash of a data block, and each non-leaf node is labelled with the cryptographic hash of the labels of its child nodes [4, Figure 1.2]. Therefore, the data changes at the leaf nodes lead to the root node hash change. Each block uses the Merkle tree to summarize all the transactions in the blockchain. There are two advantages of using the Merkle tree in the blockchain.

- To check whether two blocks have the same data, we only need to compare their Merkle root hash to whether they are the same instead of comparing all the leaf nodes.
- To prove the validity of data being part of a Merkle tree, we only need to provide and compute the data we want to prove and a few hashes at the tree. For instance, according to figure 2, we only need transaction C, Hash D, and Hash AB to verify that transaction C is part of the tree.

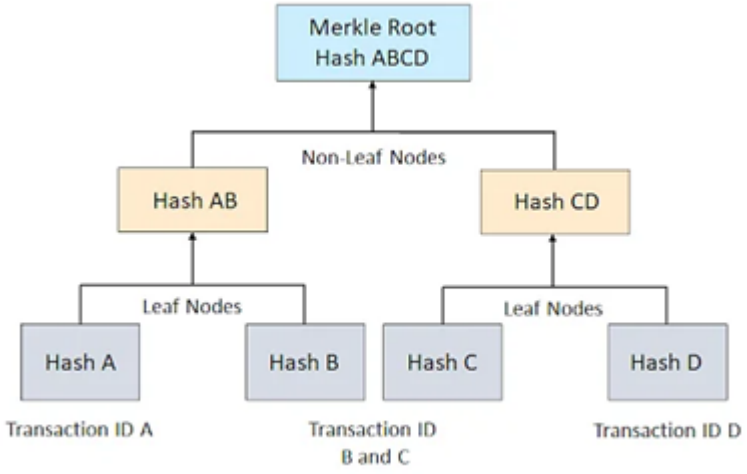


Figure 1.2: Merkle binary tree with 4 transactions

1.3.4 Consensus algorithm

Proof of work (PoW) and Proof of Stake (PoS) are two major consensus algorithms.

PoW: PoW was implemented in Bitcoin. Each block in the chain has a unique hash value (Merkle root) to represent the data of that block (including the hash value of the previous block) that the hash value of each block has a specific requirement to make miners use a large amount of computational power to satisfy the requirement. For example, the majority of the computing power on the network (miners)

need to find a binary string x of b bits such that the hash value H(data of that block$||x$), where the first w bits are all 0. Therefore, miners need to use brute force to try each x. However, we can easily verify the hash value by using the x given by the miners. If most miners are honest in the chain, then the honest chain will grow much faster than the forged chain. Because the malicious attacker needs to re-compute the block's hash value if he alters the data of the block.

PoS: There are no miners in the PoS system. PoS system order transactions and create new blocks by validators who have staked some coins in the system. A validator will be randomly chosen for validating and creating a new next block during each time slot, such as 10 seconds. A validator who stakes more coins in the system will have a higher probability to be chosen. Validators will lose a part of their stake if they approve a fraudulent transaction. Therefore, we trust the validators because of the financial motivators [5].

In other words, data in the blockchain cannot be changed without the majority agreement of the network. Therefore, blockchain technology can guarantee transparency, immutability, and verifiability. A simplified blockchain is shown in figure 1.3. Each node stores a copy of the blockchain that a blockchain is a chain of the block. Each block contains a hash of the previous block header for ensuring integrity.

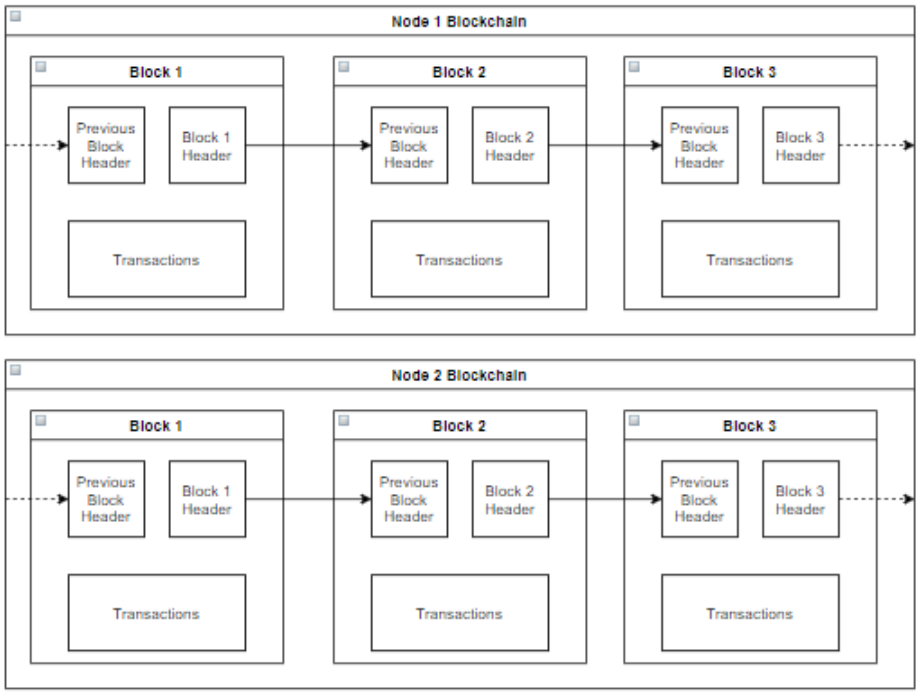


Figure 1.3: Simplified Blockchain

1.3.5 Smart contract

A smart contract is a computer program stored on a blockchain that executes in a decentralized way when pre-defined conditions are met. In other words, the code, the data, and the execution of the smart contract

are transparent and immutable. Noted that modification of the data of the smart contract is changing the final state of the data. Therefore, we can build a different decentralized application by using a smart contract.

1.3.6 Real life use case

Because of these blockchain technology properties, South Korea planned to test a blockchain-based voting system through the National Election Commission that can be used by more than 10 million people [5].

1.4 Related work in blockchain-based voting system

The secure blockchain-based voting system also is a trending research topic that integrates different cryptographic technologies with blockchain. Below are two related works about blockchain-based voting systems:

- Lyu at al.[6] published a decentralized e-voting based on a smart contract. They made linkable ring signatures and threshold cryptography to ensure the anonymity of voters and privacy, respectively.
- Uddin at al.[7] developed a blockchain-based e-voting that applies blind signatures and time-lock encryption to provide authentication of the voter and confidentiality of the ballot, respectively.

1.5 Project Goal

This project implements and analysis a secure blockchain-based e-voting application using the modified version of the voting protocol proposed by Lyu et al.[6] that guarantees the e-voting activity runs as usual even the malicious participants exist. Meanwhile, our protocol enhances the flexibility that allows participant registration anytime and anywhere. Moreover, our application supports creating multiple election events.

Chapter 2

Objectives and Outcome

1. To implement an e-voting application that the user can
    - Establish multiple elections that the election organization sets the list of eligible voters, candidates, and timers.
    - Vote the eligible candidate once for each election.
    - View and verify all elections results.
2. To implement a secure e-voting system based on a smart contract. The requirements of the secure e-voting system:

    Anonymity: No one can trace the voter from the ballot.

    Eligibility: Only the honest participant can cast a vote.

    Double-voting Avoided Each voter only can cast a vote once.

    Verifiability: Anyone can verify whether the election result is correct. Also, All voters can verify whether their vote exists.

    Fairness: Before the voting ends, no one can know the intermediate result.

    Correctness: The ballots can be tallied correctly.

    Robustness: Even if the malicious participants exist, the election will not break.

    Impersonation Attack Avoided: The attacker can not impersonate the participant to perform harmful actions to the participant and the election.
3. To compare our protocol with the e-voting protocol proposed by Lyu et al. [6].
4. To analyze the cost and the time of this e-voting system.

Chapter 3

Preliminary

This chapter is about a brief review of Ethereum, mathematical concepts and cryptography tools used in our project.

3.1 Ethereum

3.1.1 Ethereum

Ethereum is a programmable permissionless blockchain. Programmers can build different decentralized applications that run on blockchain technology by using a smart contract. Compared with permissioned blockchains such as Hyperledger and Corda, there are no restrictions that anyone can read or write on the Ethereum network. Therefore, our project uses Ethereum as a trusted computing environment because it is transparent and verifiable for the public. In this section, the main concepts about Ethereum, including transaction-based state machine, value, gas, world state, account, account storage, transaction, message call, block, and Ethereum virtual machine are explained based on the yellow paper of Ethereum [8]. Therefore, we can understand how decentralized applications can run on Ethereum guarantees immutable transactions and execution. Furthermore, the figure below shows the relationships between block, transaction, account state objects, and Ethereum tries [9, Figure 3.1].

3.1.2 Transaction-based state machine

Ethereum can view as a transaction-based state (world state) machine. Transactions thus represent a valid arc between two states. The machine begins with a genesis state, and the state will keep increment by transaction execution. The latest state is called the current state. During the state transition, Ethereum can perform arbitrary computation. Also, Ethereum can view as a chain of blocks. The transactions are batched into blocks. Each block is linked to its previous block by the hash of the previous block to prevent any changes in the history transactions by using a consensus algorithm.

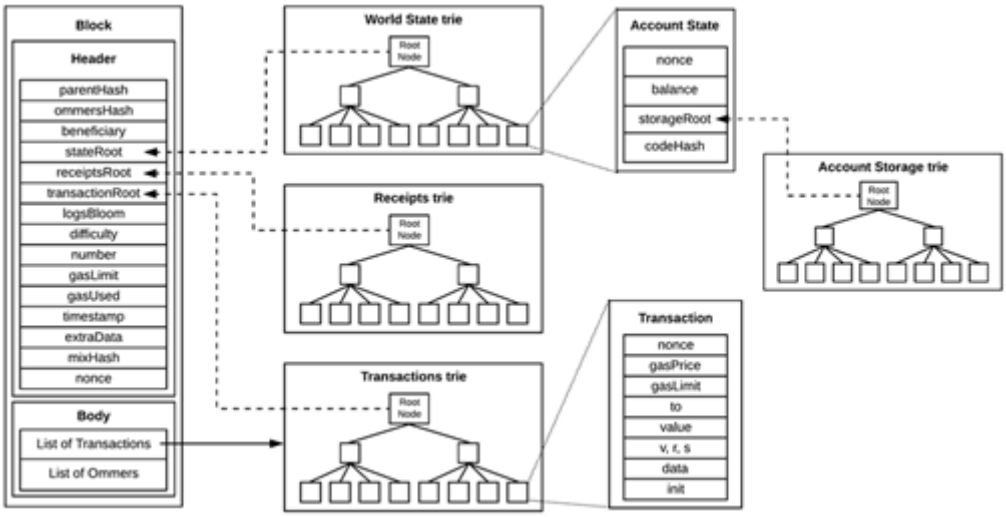


Figure 3.1: block, transaction, account state objects and Ethereum trees

3.1.3 Value

Like Bitcoin, Ethereum also has its currency called **Ether** for incentivizing computation within the network. **Wei** is the smallest sub denomination of Ether that one Ether equals to 10^{18} Wei.

3.1.4 Gas

All programmable computations such as creating contracts and making message calls in Ethereum require a fee called gas to perform to avoid network abuse. In every transaction, the sender needs to specify the amount of gas that he implicitly purchased from his account balance called **gasLimit**, and the amount of Wei he pays for each gas unit called **gasPrice**.

3.1.5 World State

A World state is a mapping of addresses and account states, and its data are stored in a modified Merkle Patricia tree. This tree is an improved Merkle tree that provides efficient authenticity and integrity verification, and it can be viewed as a global state since transaction executions constantly update its state. The address is the 20 bytes of the account's public key used as an identifier of the account.

3.1.6 Account State

There are two types of accounts in Ethereum, including externally owned accounts and contract accounts:

Externally Owned Account(EOA): Anyone with private keys controls this account. It can transfer ETH/-token between EOA and deploy and call smart contracts.

Contract Account(CA): This is a smart contract that is controlled by its contract code. The code in the contract account only executes when a transaction is sent to this account from EOA.

There are four fields in account state includes nonce, balance, storageRoot, and codeHash. The following table explains the details of four fields in account state.

Fields	EOA	CA
Nonce	number of transactions sent from this address	number of contract-creations made by this account.
Balance	the number of Wei owned by this address.	
StorageRoot	hash of an empty string.	hash of the root node of the account storage tree.
CodeHash	hash of an empty string.	hash of the EVM code of this account.

Table 3.1: Four fields in account state

3.1.7 Account Storage Tree

The account storage tree also is a Merkle Patricia tree. This tree is where the data associated with a contract account is stored, and all contract data is stored in the tree as a mapping between 32-bytes integers.

3.1.8 Transaction

Again, Transactions are stored in the Transaction tree which is a Merkle Patricia tree. There are two types of transactions:

Message Call: A transaction from one account to another. The from and to account both can be EOA or CA.

Contract creation: A transaction in that EOA create a smart contract. This transaction is without an address, and its data field contains a contract code.

The following are the fields of a transaction:

nonce: number of transactions sent by the sender.

gasPrice: number of Wei to be paid per gas unit for computation.

gasLimit: maximum amount of gas that should be used.

to: address of the recipient or empty in contract deployment transaction.

value: number of Wei to be transferred to the message call's recipient. The Wei will be added to the new contract in the contract deployment transaction.

r,s: values for verifying the sender of the transaction by using a digital signature.

init: an unlimited size byte array specifying the EVM-code for contract creation.

3.1.9 Block

There are three components in a block: block header, transactions, and ommer block headers. The Block header contains the hash of its previous block for building an immutable chain. The fields in the block header are:

parentHash: hash of the block header from the previous block.

ommersHash: hash of ommer block headers of this block.

beneficiary: account address of who gets the fees for successfully mining this block.

stateRoot: hash of the root node of the state tree, after all transactions are executed, and finalizations applied.

transactionsRoot: hash of the root node of the transaction tree that the tree contains all the transactions of this block.

receiptsRoot: hash of the root node of the transaction receipt tree. Whenever a transaction is executed, a receipt of that transaction will be generated that contains the information about that transaction and the receipt is stored in that tree.

logsBloom: Bloom filter is used to find out the indexable information such as logger address and log topics generated on this block's transactions. The purpose of it is to prevent the block saves many duplicate data.

difficulty: the difficult level of mining this block.

gasLimit: maximum amount of gas that transactions in this block can be used.

gasUsed: total gas used in transactions in this block.

timestamp: Unix timestamp when this block was created.

extraData: an arbitrary byte array less than or equal to 32 bytes.

mixHash: hash combined with the nonce for proving this block is mined.

nonce: a 64 value used in mixHash.

In short, an ommer block is a block that is not chosen to be the next confirmed block when more than one block is mined and broadcasted to the network roughly simultaneously. In Ethereum, ommer block miners also can get smaller rewards for appreciating their effort. The structure of ommer block header is the same as the block header.

3.1.10 Ethereum Virtual Machine

Ethereum Virtual Machine (EVM) is a machine maintained by all Ethereum nodes. It is a runtime environment for all accounts, including smart contracts, to compute a new valid state from block to block.

3.2 Mathematical Concepts

All cryptography tools used in our project are over Elliptic Curve Cryptography (ECC). ECC is over a finite field in a prime field because a finite field contains challenging mathematical problems and provides practical implementations of theoretical ideas. Therefore, the section is about the finite field and its related mathematical foundations based on chapter 5 of the textbook "CRYPTOGRAPHY AND NETWORK SECURITY PRINCIPLES AND PRACTICE" for understanding the cryptography tools [10].

3.2.1 Modular Arithmetic

Modular arithmetic is a system of arithmetic for integers, where values wrap around when they reach a fixed value called **modulus** to leave a **remainder**, and the modular arithmetic is often tied to a prime number in cryptography [11]. The properties of normal arithmetic such as commutativity, associativity, and distributivity are also exiting in modular arithmetic[12].

Definition 1 (Remainder)**.** A remainder is a mod n where $a \in \mathbb{Z}, n \in \mathbb{Z}^+$ and remainder $\in \mathbb{Z}_n$.

Congruence Modulo

Let $a, b, n \in \mathbb{Z}$, we say a and b are congruent modulo n if

$$a \bmod n = b \bmod n$$

and we can also express the congruence by

$$a \equiv b \ (\bmod n)$$

An example of congruence modulo 4:

$$7 \equiv 3 \ (\bmod 4)$$

$$-1 \equiv 3 \ (\bmod 4)$$

$$-5 \equiv 3 \ (\bmod 4)$$

Properties of modular arithmetic

The following are the equalities about the addition, subtraction, and multiplication properties of modular arithmetic that are used in our project for avoiding bit overflow and efficient computation:

$$(a + b) \bmod n = [(a \bmod n) + (b \bmod n)] \bmod n$$

$$(a - b) \bmod n = [(a \bmod n) - (b \bmod n)] \bmod n$$

$$(a \times b) \bmod n = [(a \bmod n) \times (b \bmod n)] \bmod n$$

Modular Inversion

There is no division operation in modular arithmetic. However, if the number co-primes to modulus, the modular inverse is still exited.

Definition 2 (Modular Inversion)**.** Let x, $y \in Z_n$, y is the inverse of x which is denoted $x^{-}1$ if $(x \times y) \equiv 1(\bmod n)$.

Definition 3 (Division Algorithm Equation)**.** Let a and $b \in \mathbb{Z}$, then there exist unique integers q and r such that $a = q \times b + r$, where $0 \leq r \leq b$.

Euclidean Algorithm

Euclidean algorithm is a efficient method to find the modular inverse by computing the greatest common divisor (GCD) of two integers.To find the GCD(a,b) by using euclidean algorithm, we repeatedly compute

$$a_i = b_i \times q_i + r_i$$

that $a_{i+1} = b_i$ and $b_{i+1} = r_i$ in next round until the r_i equals to 0, where q_i is a quotient, r_i is a remainder, and $a > b$. Then the GCD(a, b) is last non-zero remainder. An example of find the GCD(25,13)=1 by using euclidean algorithm:

$$25 = 13(1) + 12$$

$$13 = 12(1) + 1$$

$$12 = 1(12) + 0$$

Extended Euclidean Algorithm

Extended euclidean algorithm is extension of euclidean algorithm to find integer x and y such that $ax + by$ = GCD(a, b). An example of find the modular inverse of 13 where modulus is 25 by using extended euclidean algorithm:

$$25(1) - 13(1) = 12$$

$$13 - 12(1) = 13 - (25(1) - 13(1)) = 13(2) + (-25(1)) = 1$$

The inverse of 13 is 2 because 13(2) + (-25(1)) = 1.

3.2.2 Group

A group G denoted by $\{G, \bullet\}$ is a set of elements with a binary operation[1] $\bullet$ that associate with two elements a, b in G and satisfy the following properties:

Closure: if a and b belong to G, then $a \bullet b$ also belongs to G

Associative: $a \bullet (b \bullet c) = (a \bullet b) \bullet c$ for all a, b, c in G

Identity element: there is an element e in G such that $a \bullet e = e \bullet a = a$ for all a in G.

Inverse element: for every element a in G, there is an element a' such that $a \bullet a' = e$ where e is the identity element.

Finite Group

A finite group is a group that has finite elements. The **order** of the group = number of elements in the group.

Abelian Group

If a group also satisfy the following properties, then it is a abelian(commutative) group:

Commutative: $a \bullet b = b \bullet a$ for all a, b in G.

Cyclic Group

A group G is cyclic if every element of G is a power a^k of a fixed element $a \in G$ (a **generator** of G) where k is an integer.

An example of $\{\mathbb{Z}_n, +\}$

A set $\mathbb{Z}_n$ is a set of remainders modulus a positive integer n = $\{0, 1, 2, ..., n\}$. $\{\mathbb{Z}_n, +\}$ is a abelian and finite group that the order of $\{\mathbb{Z}_n, +\}$ is n and satisfy the following properties:

Closure: $a + b \equiv c$ (mod n) , $0 \leq c \leq n$.

Associative: $a + (b + c) \equiv (a + b) + c$ (mod n) .

Identity element: $a + 0 \equiv a$ (mod n) , 0 is the identity element.

Inverse element: $a + (n - a) \equiv 0$ (mod n)

Commutative: $a + b \equiv b + a$ (mod n)

[1] Addition, subtraction, multiplication, and division are the binary operations.

3.2.3 Ring

A ring R is denoted by $\{ R, +, X \}$ is a set of elements with two binary operations such that all $a, b, c \in R$ satisfy abenlian group properties and the following addition properties:

Closure under multiplication: If $a, b \in R$, then $ab \in R$ as well.

Associativity of multiplication: $a(bc) = (ab)c$ for all $a, b, c \in R$.

Distributive laws: $a(b + c) = ab + ac$ for all $a, b, c \in R$.

Commutative Ring

A ring is commutative if it satisfy the following addition condition:

Commutativity of multiplication: $ab = ba$ for all $a, b \in R$.

Integral Domain

Integral domain is a non-zero commutative ring that obey the following axioms:

Multiplicative identity: the is an element $1 \in R$ such that $a1 = 1a = a$ for all $a \in R$.

No zero divisors: if $a, b \in R$ and $ab = 0$, then $a = 0$ or $b = 0$.

An example of {$\mathbb{Z}, +, X$}

A set $\mathbb{Z}$ includes positive, negative and zero is an integral domain. We can easily verify it by just looking at the above properties of the integral domain.

3.2.4 Field

A field F is denoted by $\{ F, +, X \}$, is a set of elements with two binary operations that all $a, b, c \in R$ satisfy the properties of integral domain and multiplicative inverse axiom.

Multiplicative inverse: For each $a \in F$, except 0, there is an element $a^{-1} \in F$ such that $aa^{-1} = a^{-1}a = 1$.

Subtraction And Division

A field also contains subtraction and division operations because we can achieve subtraction by adding an additive inverse and achieve division by multiplying a multiplicative inverse.

Finite Field

A finite F is a set of finite elements. Then it is a finite field. In elliptic curve cryptography, the finite fields F_p and F_{2^m} where p is a prime number, and m is an integer are used according to different considerations such as efficient computation and the level of security. In our project, the F_p is used. The F_p is a prime field of order p that contains {0,1,2,...,p-1} with integer modular arithmetic. The prime number p is used in a finite field because it makes every element of the field have a multiplicative inverse.

3.3 Cryptography Tools

In this section, the cryptography tools including elliptic curve cryptography, ElGamal encryption, elliptic curve digital signature scheme, commitment schemes, zero-knowledge proof, schnorr's protocol, chaum-pedersen protocol, linkable ring signature, Shamir secret sharing scheme, verifiable secret sharing (VSS), Feldman VSS, threshold cryptosystem, and threshold cryptosystem based on an elliptic curve are explained.

3.3.1 Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) is a modern public-key cryptosystem that is based on the algebraic structures of the elliptic curves over finite fields, and on the difficulty of the Elliptic Curve Discrete Logarithm Problem (ECDLP)[13].

Elliptic Curve

An elloptic curve is a plane algebraic curve that contains points $\{x, y\}$, which its simplified equation is:

$$y^2 = x^3 + ax + b$$

, where $4a^3 + 27b^2 \neq 0$ and a extra point O at "infinity". The following figure is the example of an elliptic curve used in Bitcoin (secp256k1).

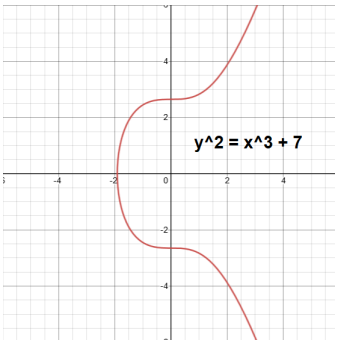


Figure 3.2: elliptic curves $E : y^2 = x^3 + 7$

Elliptic Curve over Finite Field

The elliptic curve over finite field F_p where $p > 3$ is used in ECC. All points $\{x, y\}$ of the curve are in F_p. Then the elliptic curve equation over F_p is:

$$y^2 = x^3 + ax + b \ (\text{mod } p)$$

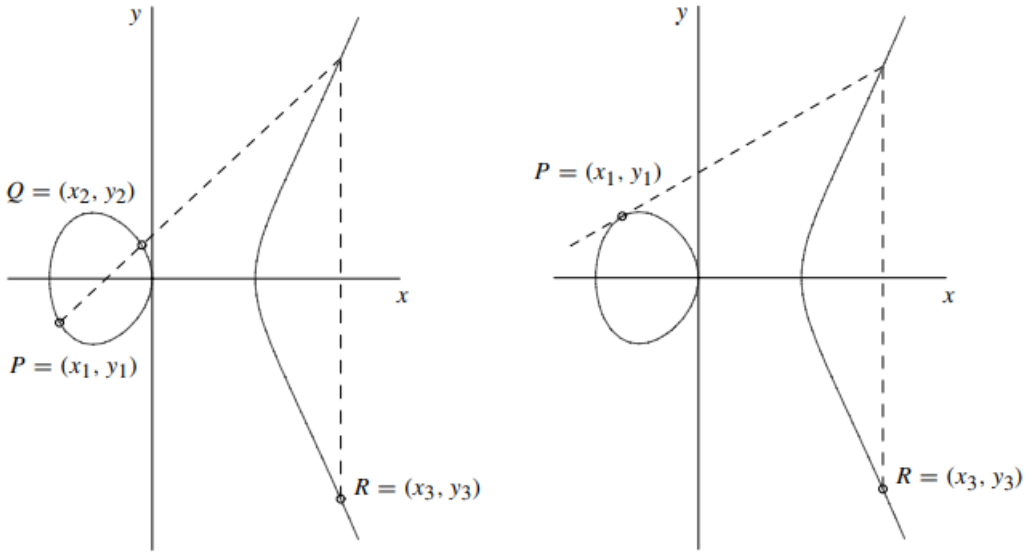
,where $4a^3 + 27b^2 \neq 0$, $a, b, x, y \in F_p$ and an extra point O at "infinity".

Point Operations

In this subsection, the explanation of point addition, point doubling, and point multiplication of an elliptic curve is provided.

Point Addition: Let P, Q, R are the points on an elliptic curve E. The $P + Q = R$ is defined as follows. First draw a line L through P and Q. The line L intersects the curve E at the third point. Then R is the reflection of the third point about the x-axis [14, Figure 3.3].

Point Doubling: Let P, R are the points on an elliptic curve E. The $P + P = R$ is defined as follows. First, draw the tangent line to the elliptic curve at P. This line intersects the elliptic curve at a second point. Then R is the reflection of the third point about the x-axis [14, Figure 3.3].



(a) Addition: $P + Q = R$. (b) Doubling: $P + P = R$.

Figure 3.3: Geometric addition and doubling of elliptic curve points

Point at Infinity: Let P, Q are two points on an elliptic curve E that are respective negatives. If we try to perform $P + Q$, we first draw a line L through P and Q. However, we cannot find L intersects the curve E at any point [15, Figure 3.4]. To solve this problem, the point at infinity O is created to be

the identity element of elliptic curve arithmetic that its rule is:

$$\forall P \in E : O + P = P$$

such that all operations on points within the finite field.

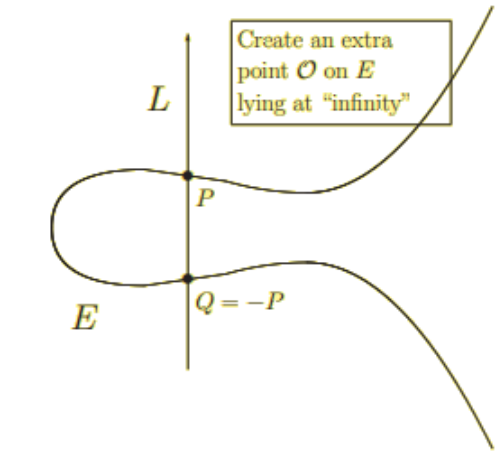


Figure 3.4: Point at Infinity

Point Negation: Let $P = \{P_x, P_y\}$ is point on an elliptic curve E. Point negation is finding a point $-P$ that $P + (-P) = O$. From Figure 3.4, we can know that $P+$ the reflected point of $P = O$. Therefore,

$$-P = \text{ the reflected point of } P = \{P_x, -P_y\}$$

Point Multiplication: Basically, point multiplication is repeated point addition. Therefore, we can compute point multiplication by point addition and point doubling. Also, there exist some efficient algorithms to perform point multiplication, such as the Montgomery ladder approach [16].

Order, Subgroup and Generator

For the elliptic curves E over finite fields F_p, the **order** n of the curve is the total number of points on $E(F_p)$. Since the finite fields are also a finite cyclic group, the points on the curve are divided into h number of subgroups called **cofactor** where the order of each subgroups is r. Then the order of the curve is $n = hr$ [13, Figure 3.5]. The **generator** also called **base point** is a point on $E(F_p)$ which is publicly known parameter for generating other points on its subgroup by rG.

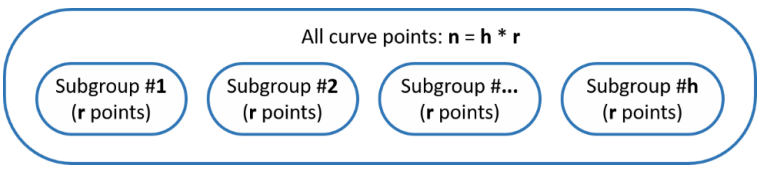


Figure 3.5: Relationship of order and subgroup.

Elliptic-Curve Discrete Logarithm Problem (ECDLP)

The informal definition is ECDLP: given the elliptic curves E over finite fields F_p, generator G, and point P, it is computational infeasible to find k such that $P = kG$ where $k \in \mathbb{Z}_n$ and n is the order of the curve.

Private Key and Public Key

Because of the ECDLP and fast point multiplication, in ECC, the private key is an integer k, and the public key is $P = kG$ where G is the generator point.

secp256r1

secp256r1 also known as prime256r1 is an elliptic curve E over F_p used in our project [17]. The parameters of this curve are shown in below:

p: FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF

a: FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFC

b: 5AC635D8 AA3A93E7 B3EBBD55 769886BC 651D06B0 CC53B0F6 3BCE3C3E 27D2604B

G: 04 6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0 F4A13945 D898C296
4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE CBB64068 37BF51F5

n: FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84 F3B9CAC2 FC632551

h: 01

Elliptic Curve Diffie–Hellman Key Exchange (ECDH)

The private and public key pair cannot be directly used to perform encryption and signing in ECC. To perform encryption and signing between two parties, they need to agree on a shared secret first by **ECDH**. ECDH is a key agreement scheme based on Diffie–Hellman key exchange to allow two parties to establish a shared secret anonymous [18]. The protocol works as follows. Let Alice and Bob have elliptic curve key pairs $\{k_A, P_A = k_A G\}$ and $\{k_B, P_B = k_B G\}$ respectively. First, they exchange their public keys $\{P_A, P_B\}$. Then their shared secret P_{AB} is established by computing $P_{AB} = k_A P_B = k_B P_A$.

Elliptic Curve Point Encoding

There are two types of elliptic curve point encoding ways which are **irreversible** and **reversible**. Let message $M \in \mathbb{Z}_n$ is encoded to the elliptic curve point P_M.

Irreversible: Irreversible elliptic curve point encoding represents P_M cannot reverse to M. The way used in the project to encode the message to the elliptic curve point irreversibly is to multiply M with the base point G.

Reversible: Reversible elliptic curve point encoding represents P_x can reverse to x. For each given value of x, there exists two values of y from the cubic equation of elliptic curve [19]. Therefore, the reversible approach to inject x as a point P_x by computing this according to the elliptic curve equation:

$$P_M = \{M, \sqrt{x^3 + ax + b \ (\text{mod } p)}\}$$

Then the x can reverse from the P_x by getting the x of coordinate P_x and ignore the y of coordinate P_x.

Since all the cryptography tools used in this project are over an elliptic curve. Therefore, the following reports will use the parameters of secp256r1 as the domain parameters in other cryptography tools and the key pair is referred to as the elliptic curve key pair without explicit notations.

3.3.2 ELGAMAL ENCRYPTION

ElGamal encryption is asymmetric encryption[2] scheme that encrypts a message by a one-time-key that the key is generated by using a variant of ECDH. The ElGamal encryption is used to encrypt the votes and the shares of the secret-sharing scheme in our project. The encryption is defined as follows:

Public parameters: recipient(Bob)'s public key $P_b = bG$, where $b \in \mathbb{Z}_n$.

Encryption:

1. choose a random $k \in \mathbb{Z}_n$.
2. compute $C = kG$ as public key.
3. compute $C' = kP_b$
4. map message M as point P_M on E inversely.
5. the ciphertext of $M = (C, D = C' + P_M)$.

Decryption (by Bob):

1. compute $C' = bC$.
2. retrieve $P_M = D - C' = C' - C' + P_M$.
3. obtain the M from P_M that $M = x$ coordinate of P_M.

[2]In asymmetric encryption, there has a key pair called the public key and private key. The public key is publicly known for encrypting the message by anyone. The encrypted message only can be decrypted by the corresponding private key so that only the key pair owner knows the private key.

3.3.3 Elliptic Curve Digital Signature Algorithm (ECDSA)

The ECDSA is a secure digital signature scheme[3] which uses keys derived from ECC [20]. The ECDSA sign and verify algorithms are defined as follows:

Public parameters: signer(Alice)'s public key $P_a = aG$, where $a \in \mathbb{Z}_n$.

Sign (by Alice):

Input: the message M and the signer's private key a.

1. compute the message hash h by using cryptographic hash function $H() : h = H(M)$, where $h \in \mathbb{Z}^+$.
2. generate a random integer $k \in \mathbb{Z}_n$.
3. compute random point $R = kG$ and $r = R_x = x$ coordinate of R.
4. compute the signature proof $s = k^{-1} \times (h + ra)$ (mod n).
5. return signature $\{s, r\}$.

Verify:

Input: the message M, signature $\{s, r\}$, and the Alice's public key.

1. compute the message hash h' by using cryptographic hash function $H() : h' = H(M)$, where $h' \in \mathbb{Z}^+$.
2. compute $s_{inv} =$ the modular inverse of $s = s^{-1}$ (mod n).
3. recover random point $R' = (h's_{inv})G + (rs_{inv})P_a$.
4. $r' = R'_x = x$ coordinate of R'.
5. return $r \stackrel{?}{=} r'$.

Proof Verification Works: If the message M was not modified and the public key P_a is corresponding to the signer's private key, then $h' = h$ and the math behind the $s_{inv} =$ the modular inverse of $s = s^{-1}$ (mod n):

$$s_{inv} = s^{-1} \text{ (mod}n \text{)} = (k^{-1} \times (h + ra))^{-1} \text{ (mod}n \text{)} = k \times (h + ra))^{-1} \text{ (mod}n \text{)}$$

Then we replace the $h' = h$,$P_a = aG$, and $s_{inv} = k \times (h + ra))^{-1}$ (mod$n$) in R':

$$R' = (h's_{inv})G + (rs_{inv})P_a$$

[3]Digital signature offers the functionalities of a handwritten signature and data integrity. In other words, only the signer can use his identity to sign the message, and the verifier can determine whether the message was modified and the signer of the signed message. The digital signature scheme uses the public key cryptography and hash function to achieve the above functionalities.

$$R' = (hs_{inv})G + (rs_{inv})aG$$

$$R' = (h + ra)s_{inv}G$$

$$R' = (h + ra)(k \times (h + ra))^{-1} \text{ (mod} n \text{)}G$$

$$R' = kG = R$$

Realization

In the implementation of ECDSA and other cryptographic algorithms, the cryptographic hash function used in this project is **soliditysha3** which is provided by the web3.js package and Ethereum. In the frontend, the output of soliditysha3 function is a hex string whose output size is 256 bits and the hex string will be converted to be an integer. In Ethereum, the output of soliditysha3 function is an integer.

3.3.4 Commitment Schemes

Cryptographic commitment Schemes are digitally equivalent to a physical sealed envelope that allows people to commit a value while hiding the value, with the ability to reveal the committed value [21]. For example, Alice commits her message m by putting it into the envelope and then sending the sealed envelope (commitment) to Bob. Since the envelope is sealed and only Alice can open it, Bob cannot reveal the m from the envelope. Later on, Alice opens the envelop and recovers them. Then Alice can convince the commitment is indeed the commitment of the m to Bob. From this example, we can know commitment schemes have two phases including the **commit phase** and **reveal phase** [22]:

Commit Phase: The sender commits a message m by computing $C = \text{commit}(m, r)$ where r is a random number. Then sender sends C to the receiver.

Reveal Phase: The senders open the commitment C by sending m and r to the receiver. Then the receiver computes $C' = \text{commit}(m, r)$ and verify $C \stackrel{?}{=} C'$.

The security properties of commitment schemes are **binding** and **hiding**:

Binding: The probability of generating r and r' such that commit(m, r) and commit(m', r') is negligible where $m \neq m'$. Moreover, assume the adversary have unlimited computational power. If the adversary still find commit(m, r) and commit(m', r') with a negligible probability, then this is called **unconditional binding**. Otherwise, this is called **computational binding**.

Hiding: The commitment reveals nothing to the committed message. In other words, commit(m, r) and commit(m', r) is indistinguishable where $m \neq m'$. Also, hiding property comes in two flavors. If commit(m, r) and commit(m', r) are computationally indistinguishable that it is called **computational hiding**. If commit(m, r) and commit(m', r) are statistically indistinguishable, then it is called **unconditional hiding**.

Commitment Realization

In the implementation, the commitment of the message m is mG. This commitment is not unconditional hiding because there is no blinding factor r. The commitment is used to hide and blind the secret while allowing the provider to prove he knows the value of the commitment without revealing the commitment and verifying the shares whether correctly distributed in a secret sharing scheme. In my research, it is hard to use a commitment scheme with unconditional hiding that can achieve the above functionalities non-interactively. Therefore, this commitment method is used because of the time limitation.

3.3.5 Zero Knowledge Proof

A zero-knowledge proof allows the prover convinces a statement is true to the verifier by revealing no information except the statement. There are three properties that the proof needs to be satisfied [23]:

Correctness: If the prover and verifier are honest and the statement is true, then the verifier accepts the proof with overwhelming probability.

Soundness: If the statement is false, the verifier accepts this proof with negligible probability.

Zero Knowledge: Nothing can be learned from the proof except the validity of the statement.

Proof of Knowledge

Proof of knowledge is a zero-knowledge proof that the prover not only proves the statement is true but also proves he knows the **witness** of the statement to the versifier. Specifically, the soundness and zero-knowledge properties in proof of knowledge are different to zero acknowledge proof [24]:

Special soundness: There exits an extractor E that can obtain the witness for the proven statement with high probability for all provers. The verifier only accepts the proof if the prover knows the witness.

Honest-verifier Zero knowledge: zero-knowledge against an honest verifier only.

In our project, we treat the smart contract as the honest-verifier.

Sigma Protocol

The structures of many interactive zero knowledge proofs are the same which are three steps including **commitment**, **challenge**, and **response** and they are called **sigma protocols** [25]. The sigma protocol works as follows:

Commitment: Prover sends a message m called commitment to the verifier.

Challenge: Verifier sends a random number r as a challenge to the prover.

Response: Prover computes the proof based on m and r and sends the proof to the verifier.

Fiat-Shamir transformation

Fiat-Shamir transformation is a technique to transform an interactive zero-knowledge proof into a non-interactive one. The meaning of non-interactive is that the prover can directly send his final proof to the verifier, and the verifier can directly determine whether the proof can be accepted. The key idea is that the prover computes the random challenge by himself using a cryptographic hash function rather than the verifier sending the random value to him [26].

3.3.6 Schnorr's Protocol

Schnorr's protocol is a proof of knowledge that the prover proves the knowledge of a where $A = aG$ is public without revealing a. In our project, the user uses schnorr's protocol to verify he is a legitimate register without revealing his personal information [27]. The elliptic curve version of non-interactive Schnorr's Protocol works as follows:

Prover:

Input: secret a.

1. generate random $r \in \mathbb{Z}_n$ and compute point $R = rG$.
2. compute random $c = H(G, R, A)$ where $H()$ is a cryptographic hash function.
3. compute $m = r + ac$ (mod n) and send $\{R, c, m\}$ to verifier.

Verifier:

Input: A and the proof $\{R, c, m\}$.

1. check $R \stackrel{?}{=} mG - cA = (r + ac)G - cA = rG + acG - cA = rG + acG - acG = rG$.

3.3.7 Chaum-Pedersen Protocol

Chaum-Pedersen Protocol is a proof of knowledge for the equality of discrete logarithms. In other words, the prover can claim he knows x of xG and xH where G, H are two different generators on curve E [28]. Chaum-Pedersen protocol is used to verify whether the user provided ElGamal decryption key is correct such that the correctness of the ElGamal decryption can be ensured. The non-interactive Chaum-Pedersen protocol is defined as follows:

Public Parameters: base points G, H, points $A = xG, B = xH$, and $n \in \mathbb{Z}_n$ where only prover knows x.

Prover:

Input: base points G, H and secret x.

1. generate random $k \in \mathbb{Z}_n$ and compute points $K = kG, L = kH$.

2. compute $c = H(K, L)$.
3. compute $r = k - xc$ (mod n)
4. send the proof $\{r, c\}$ to verifier.

Verifier:

Input: base points G, H, points A, B, and the proof $\{r, c\}$.

1. compute $K' = rG + cA = rG + cxG = (k - xc)G + cxG = kG$.
2. compute $L' = rH + cH = rH + cxH = (k - xc)H + cxH = kH$
3. check $c \stackrel{?}{=} c' = H(K', L')$.

3.3.8 Linkable Ring Signature

Ring Signature

A ring signature is a group signature without a group manager and cooperation between group members that allows a signer to sign a message on behalf of the group without revealing which group member signed this message. In addition, anyone can verify the validity of the signature. [29]. Moreover, in a ring signature, each group member i has a key pair $\{sk_i, sk_i\}$, to produce a signature for entity i, the message m, secret key of $i = sk_i$, and public keys of all members $\{pk_i, ..., pk_n\}$ are needed.

Linkable Ring Signature

A linkable Ring Signature is a modification of a ring signature that detects whether the same signer generates two signatures. There are three properties that this signature needs to be satisfied with [30]:

Anonymity: No one can know the signer of the signature.

Linkability: Signatures signed by the same signer are linked.

Spontaneity: No group manager and secret-sharing setup stage.

The following are the steps of how to generate and verify a linkable ring signature with a group size of z.

Public Parameters: a list of public keys of the group members $L = \{pk_1, pk_2, ..., pk_z\}$ where $pk_i = sk_iG$.

Signature Generation:

Input: the message $m \in \mathbb{Z}_n$, the signer's secret key $sk_i \in \mathbb{Z}_n$, L.

1. compute $H = H_2(L)$ and $K = sk_iH$ where $H_2()$ maps an integer to an elliptic curve point.

2. generate random $c \in \mathbb{Z}_n$ and compute $u_{i+1 \text{ (mod } z)} = H_1(L, K, m, cG, cH)$ where $H_1()$ is an cryptographic hash function.
3. For $j \in [1, z)$,
   (a) compute $k = i + j$ (mod z).
   (b) generate random $v_k \in \mathbb{Z}_p$.
   (c) compute $u_k = H_1(L, K, m, v_kG + u_kpk_k, v_kH + u_kK)$.
4. compute $v_i = c - sk_iu_i$ (mod p).
5. return signature $\{u_1, v_1, v_2, ..., v_z, K\}$.

Signature Verification:

Input: signature $\{u_1, v_1, v_2, ..., v_z, K\}$, message m, and public keys L.

1. compute $H = H_2(L)$.
2. For $j \in [1, z)$,
   (a) compute $u_{j+1} = H_1(L, K, m, v_jG + u_jpk_j, v_jH + u_jK)$.
3. check $u_1 \stackrel{?}{=} u_z$.

Linkage between two signatures

Since $H = H_2(L)$, $K = sk_iH$, signer i will produce same K for all signatures with the same public key list L and the signer identity is protected by the ECDLP that no one can know sk_i from K and public cannot check the equality of $pk_i = sk_iG$ and $K = sk_iH$ because they do not know sk_i. Therefore, any one can verify whether signature $\{u_1, v_1, v_2, ..., v_z, K\}$ and signature $\{u'_1, v'_1, v'_2, ..., v'_z, K'\}$ with the same public key list are generated by the same signer by checking $K \stackrel{?}{=} K'$.

Realization

The key pairs of all participants are elliptic curve key pairs but are not the Ethereum account key pair. Because each transaction on Ethereum contains the account address(compressed public key) of the transaction executor. If we use the Ethereum account key pair, the transaction will expose the signer's public key, and the linkable ring signature will become useless.

For $L = \{pk_1, pk_2, ..., pk_z\}$ where $pk_i = sk_iG$, the implementation of computing $H_1(L, K, m, v_jG + u_jpk_j, v_jH + u_jK)$ is

$$t_1 = pk_1.x + pk_1.y + pk_2.x + pk_2.y + ... + pk_z.x + pk_z.y$$

$$t_2 = K.x + K.y$$

$$t_3 = m \text{ (mod } n \text{)}$$

$$t_4 = (v_jG + u_jpk_j).x + (v_jG + u_jpk_j).y$$

$$t_5 = (v_jH + u_jK).x + (v_jH + u_jK).y$$

$$t_6 = (t_1 + t_2 + t_3 + t_4 + t_5) \text{ (mod } n \text{)}$$

$$t_7 = H(t_6) \text{ (mod } n \text{)}$$

t_7 is the result of $H_1(L, K, m, v_jG + u_jpk_j, v_jH + u_jK)$.
Furthermore, the implementation of computing $H = H_2(L) = H_1(L)G$.

3.3.9 Secret Sharing Scheme

The idea of a secret sharing scheme is splitting the secret into many shares that each share is distributed to each group member. If the group members publish a sufficient number of shares, the secret can be reconstructed. Otherwise, no one can reconstruct the secret. In general, a secret sharing scheme for a dealer D and participants $P_1, P_2, ..., P_m$ has two phrases:

Distribution: A dealer D splits the secret s into shares $s_1, s_2, ..., s_m$ and distributes s_i to participant P_i for $i \in [1, m]$.

Reconstruction: A set of participants $Q \subseteq \{P_1, P_2, ..., P_m\}$ pool their shares s_i together such that the secret s can be recovered.

3.3.10 Shamir Secret Sharing Scheme

Shamir secret sharing scheme is one of the secret sharing scheme based on polynomial interpolation such that the secret s is split into n shares and any combination of k shares can reconstruct s while any combination of $k - 1$ shares provide no information of s [31]. Therefore, before we discuss the details of this scheme, we explain the polynomial and its properties first.

Polynomial

A polynomial $f(x)$ is a mathematical expression in the form expression in the form:

$$a_nx^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + ... + a0$$

, where $a_n, a_{n-1}, a_{n-2}, ..., a0$ are called **coefficients**, $a0$ is **constant**, and the highest exponent of x is called **degree**. A **irreducible polynomial** is a polynomial that only divisible by itself and constant. Also, a **polynomial over a field** F represents the coefficients of this polynomial are taken from F. For example, $f(x) = 3x^2 + 6x + 4$ is a polynomial that its coefficients are $3, 6, 4$, degree is 2 and constant is 4.

Theorem 1 (Polynomial interpolation)**.** *Given $d+1$ points $(x_1, y_1), (x_2, y_2), ..., (x_{d+1}, y_{d+1})$ where $x_1, x_2, ..., x_{d+1}$ are distinct numbers, there is only one polynomial $f(x)$ of degree $\leq d$ that $f(x_i) = y_i$ for $i \in [1, d+1]$.*

Theorem 2 (Lagrange interpolation)**.** *Given $d+1$ points $(x_1, y_1), (x_2, y_2), ..., (x_{d+1}, y_{d+1})$ where $x_1, x_2, ..., x_{d+1}$ are distinct numbers, the unique polynomial $f(x)$ of degree $\leq d = \sum_{i=1}^{d+1} y_i \lambda_i(x)$ where $\lambda_i(x) = \prod_{j=1, i \neq j}^{d+1} \frac{x - x_j}{x_i - x_j}$.*

Assume we have three points $(3, 2), (4, 1), (2, 5)$, we can can figure out the degree-2 polynomial $f(x)$ that interpolates these three points by using lagrange interpolation as follows:

i	1	2	3
x_i	3	4	2
y_i	2	1	5

First, we compute $\lambda_i(x)$ for $i \in [1, 3]$.

$$\lambda_1(x) = \frac{(x-4)(x-2)}{(3-4)(3-2)}$$
$$\lambda_2(x) = \frac{(x-3)(x-2)}{(4-3)(4-2)}$$
$$\lambda_3(x) = \frac{(x-3)(x-4)}{(2-3)(2-4)}$$

Then we compute $f(x)$,

$$f(x) = y_1\lambda_1(x) + y_2\lambda_2(x) + y_3\lambda_3(x)$$
$$= 2\frac{(x-4)(x-2)}{(3-4)(3-2)} + \frac{(x-3)(x-2)}{(4-3)(4-2)} + 5\frac{(x-3)(x-4)}{(2-3)(2-4)}$$
$$= x^2 - 8x + 17$$

Then, we can see that the degree-2 polynomial $f(x) = x^2 - 8x + 17$ that interpolates three points $(3, 2), (4, 1), (2, 5)$ from Figure 3.6.

Shamir Threshold Scheme

After you understand the properties of polynomial, you might already know how Shamir's secret sharing scheme works. The Shamir (t, m)-threshold for sharing a secret $s \in \mathbb{Z}_p$ where p is a prime and $0 \leq t < m < p$ is defined as follows:

Distribution: A dealer picks a random polynomial

$$f(x) = a_t x^t + a_{t-1}x^{t-1} + a_{t-2}x^{t-2} + ... + a0 \text{ (mod } p \text{)}$$

, where the coefficients $a_t, a_{t-1}, ..., a0$ and $f(x) \in \mathbb{Z}_p$ and the secret $s = f(0) = a0$. Then the dealer sends $s_i = f(i)$ to participants P_i for $i \in [1, m]$.

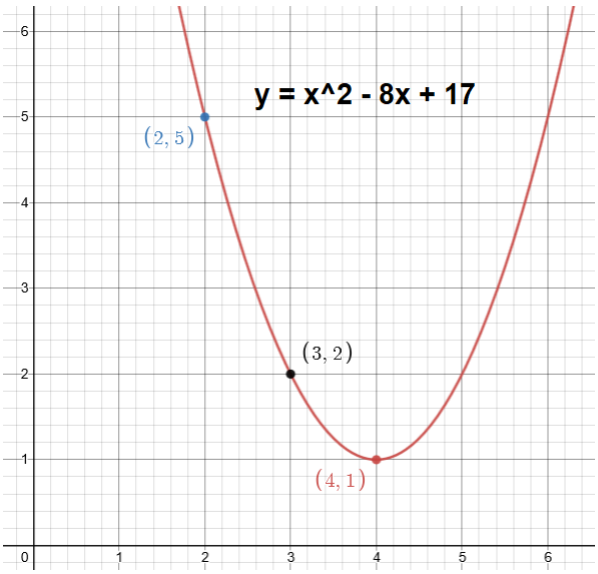


Figure 3.6: the degree-2 polynomial $f(x) = x^2 - 8x + 17$

Reconstruction: Any set of $t + 1$ participants can use their shares s_i reconstruct secret s by lagrange interpolation:

$$\sum_{i \in Q} s_i \lambda_i(0), \text{ where } \lambda_i(0) = \prod_{j \in Q, i \neq j} \frac{0 - j}{i - j} = \prod_{j \in Q, i \neq j} \frac{j}{j - i} \pmod{p}$$

Assume a set Q of t participants P_i pool their shares s_i together. Then they can know a unique polynomial $f'(x)$ of degree $t - 1$ such that $f'(0) = s'$ and $f'(i) = s_i$. However, $f'(0) \neq s$ and $f'(x)$ reveal nothing information about s because there is p possible polynomials of degree t equally likely that can interpolates t points [32, Figure 3.7].

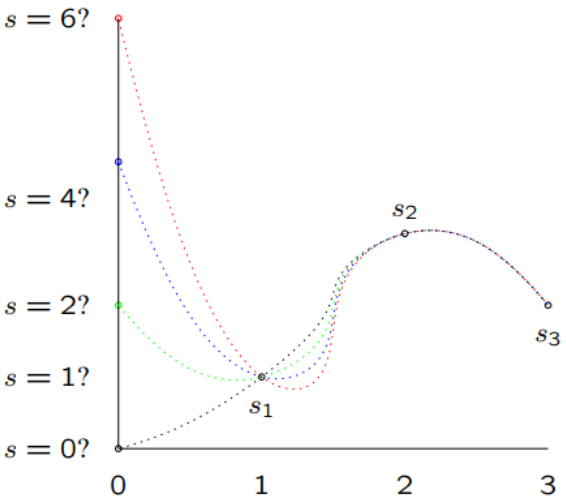


Figure 3.7: Security of Shamir's scheme illustrated: 5 degree-3 polynomials that can interpolates s_1, s_2, s_3.

3.3.11 Verifiable Secret Sharing (VSS)

The basic secret sharing scheme requires the dealer and all participants to be honest. Therefore, the VSS comes in for resisting the following malicious participant's active attacks:

Distribution: A dealer sends incorrect shares to some or all of the participants.

Reconstruction: The participants submit incorrect shares.

3.3.12 Feldman VSS

Feldman VSS is an extension of Shamir's secret sharing scheme. The idea behind it is that the dealer not only sends the share s_i to participant P_i but also broadcasts a verification value to all participants such that the participants can use them to validate their shares [33].
Let an elliptic curve E over finite field F_p where p is prime and the order of E is n. Then, the elliptic curve based Feldman VSS works as follows:

Distribution: A dealer picks a random polynomial

$$f(x) = a_t x^t + a_{t-1}x^{t-1} + a_{t-2}x^{t-2} + ... + a_0 \text{ (mod } n \text{)}$$

, where the coefficients $a_t, a_{t-1}, ..., a0$ and $f(x) \in \mathbb{Z}_p$ and the secret $s = f(0) = a0$. Then the dealer sends $s_i = f(i)$ to participants P_i for $i \in [1, m]$. Furthermore, the dealer broadcasts commitments $A_j = a_j G$ for $j \in [0, t]$ to all participants. Upon receipt of share s_i, the participant P_i can verify the correctness of the share by checking the following equation:

$$s_i G = a_t i^t G + a_{t-1} i^{t-1} G + a_{t-2} i^{t-2} G + ... + a_0 G$$

$$= A_t \cdot i^t + A_{t-1} \cdot i^{t-1} + A_t \cdot i^{t-2} + ... + A_0$$

$$= \sum_{j=0}^{t} A_j \cdot i^j$$

Reconstruction: Each share submitted by participant P_i is verified by using the above equation. Then, the secret s can be recovered by using Lagrange interpolation if $t + 1$ valid shares are contributed.

3.3.13 Threshold Cryptosystem

In (t, m)-threshold cryptosystem, a private key S is distributed to a group with participants $P_1, P_2, ..., P_m$ such that at least $t + 1$ participants cooperation is required for decryption [34].
To achieve the above goal, we might first consider a unsound approach based on a Feldman VSS as follows:

1. A dealer first generate a private public key pair $\{S, H\}$.

2. A dealer distributes part of private key s_i to participant P_i by using the distribution protocol of Feldman VSS.

3. Participants use public key H for encryption.

4. Participants use private key S for decryption. Therefore, the reconstruction protocol of Feldman VSS needs to be run first for recovering private key S such that participants can use S for decryption.

The main problem of the above approach is there exits a dealer that knows the private key S such that he can use H without anyone noticing.
Therefore, (t, m)-threshold cryptosystem is defined as follows to tackle this problem where $0 \leq t < m$:

Distributed key generation: A protocol for generating public key H by participant $P_1, P_2, ..., P_m$ (without trusted dealer) such that each participant P_i has a secret shares s_i and public verification key h_i for $i \in [1, m]$.

Encryption: An algorithm that produce a ciptertext C of message M under public key H.

Threshold decryption: A protocol that any set of $t + 1$ participants $P_1, P_2, ..., P_{t+1}$ obtain M from C by using secret shares $s_1, s_2, ..., s_{t+1}$ and public verification keys $h_1, h_2, ..., h_{t+1}$.

3.3.14 Threshold Cryptosystem Based On Elliptic Curve

The main idea of the implementation of threshold cryptosystem is each participant P_i generates a random polynomial $f_i(x)$ of degree t for $i \in [1, m]$. Then the random group polynomial $f(x) = \sum_{i=1}^{m} f_i(x)$ [Figure 3.8]. Then, the private key and public key between $P_1, P_2, ..., Pm$ are $\sum_{i=1}^{m} f_i(0)$ and $\sum_{i=1}^{m} f_i(0)G$ respectively.

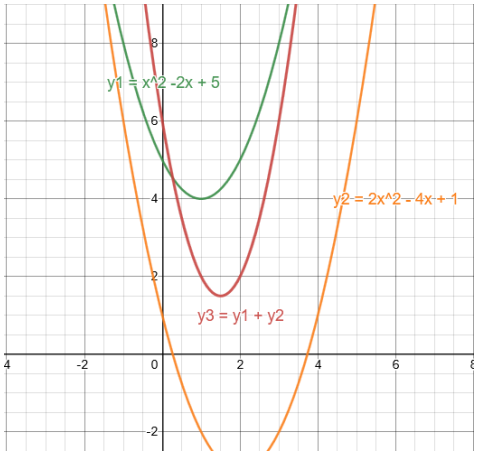


Figure 3.8: A visual representation of summing two polynomials

Distributed Key Generation Protocol

The distributed key generation protocol is defined as follows:

1. Each participant P_i generates a random polynomial $f_i(x) = a_{i,t}x^t + a_{i,t-1}x^{t-1} + a_{i,t-2}x^{t-2} + ... + a_{i,0}$ (mod n) of degree t where all coefficients $a_{i,j} \in \mathbb{Z}_p$ and $f_i(x) \in \mathbb{Z}_n$, and broadcasts a commitment $A_{i,j} = a_{i,j}G$ for $j \in [0, t]$.

2. Each participant P_i computes the public key $H = \sum_{j=1}^{m} A_{j,0}$.

3. Each participant P_i executes Feldman's VSS scheme once that lets $a_{j,0}$ as the secret value. P_i plays the role of the dealer and P_j plays the role of the participant for $i \neq j$ and $j \in [1, m]$.

4. Each participant P_i receives $f_j(i)$ for $j \in [1, m]$ [Table 3.2]. Then, P_i computes share $f(i) = \sum_{j=1}^{m} f_j(i)$. Participant P_i verifies $f_j(i)$ by checking $f_j(i)G = \sum_{k=0}^{t} A_{j,k} \cdot i^k$. The public verification key $h_i = f(i)G$.

		P_j			
		P_1	P_2	...	P_m
P_i	P_1	$f_1(1)$	$f_2(1)$	...	$f_m(1)$
	P_2	$f_1(2)$	$f_2(2)$	...	$f_m(2)$
	$\vdots$		$\vdots$		
	P_m	$f_1(m)$	$f_2(m)$	...	$f_m(m)$

Table 3.2: Participant P_i receives $f_j(i)$ for $j \in [1, m]$.

Threshold Decryption Protocol

The threshold decryption protocol works as follows:

1. Each participant publishes share $f(i)$ with a proof that shows $f(i)G = h_i$.

2. A Q is a set of $t+1$ participants publishes valid shares $f(i)$. Then the private key S can be recovered by using lagrange interpolation:

$$S = \sum_{i \in Q} f(i)\lambda_i(0), \text{ where } \lambda_i(0) = \prod_{j \in Q, i \neq j} \frac{j}{j-i} \text{ (mod } n \text{)}$$

3. The ciphertext can be decrypted by using S.

There is one problem with distributed key generation protocol. If the participant P_i publishes a wrong $h_i = f(i)'G \neq f(i)G$ in step 4, then the incorrect share $f(i)'$ will be accepted in threshold decryption protocol. Therefore, although the idea of this threshold cryptosystem is used in our application, the approach for verifying shares is different.

Chapter 4

E-Voting Application

In this chapter, the overview of our e-voting application is explained. Then, the notations of the parameters and functions are summarised. Next, the details of our e-voting protocol are described in stages.

4.1 Overview

In this section, the overview of our application is described in four aspects: users, stages, punishment, stages transition, and procedure description.

4.1.1 Users

There are four types of users in our application:

Election Organization: A person/organization who controls the election preparation stage, creation stage, and set up stage.

Register: A person who participates in the registration stage with the ability to be a participant.

Participants: A person who participates in each stage of the e-voting application except preparation, creation, and set up stages with approval of the election organization.

General Public: A person who participates in the verification stage and result stage that can report non-contributed participants and view and verify the election result.

Noted that one person can be multiple types of users.

4.1.2 Stages

There are ten stages in an e-voting activity [Figure 4.1]. The first two stages (preparation and creation) are apart from the e-voting protocol stages. The following is a brief description of each stage:

Preparation: A stage in which the election organization collects and verifies participants' information including a unique elliptic curve public key. Our application does not provide the functionality of this stage. Therefore, this stage is a prerequisite to election organization for using our application.

Creation: A stage where the election organization creates a new election smart contract instance in Ethereum for conducting the election activity.

Set Up: A stage that the election organization fills in the election information such as public key list, candidates list, registration time, and vote time. After the election organization submits the election information, the election information is not allowed to change anymore.

Registration: An optional stage that allows a legal register to become a participant in the election by proving his identity. Suppose the election organization owns the participants' personal information but not the public key. In that case, it can turn on this stage for people to submit their public keys with their identity proof at any time and anywhere through the election smart contract. If this stage does not turn on, then the stage will directly go to the distribution stage after the setup stage.

Distribution: A stage that runs the distributed key generation protocol of threshold cryptosystem based on an elliptic curve. However, the public key is not set yet, and there is no public verification key. The communication between participants is conducted in the election smart contract by storing values with the help of a digital signature and public-key encryption.

Verification: A stage that identifies and filters out the malicious participants by honest participants such that the identified malicious participants do not allow to participate in the vote stage and reconstruction stage. The malicious participant is the participant who did not distribute values or distributed incorrect values to other participants in the distribution stage.

Vote: A stage that allows honest participants to vote once without revealing their identity by using a linkable ring signature. Furthermore, the vote public key is generated, and the vote is encrypted by the vote public key.

Reconstruction: A stage for reconstructing the private key such that the votes can be recovered by the participants computing and uploading their shares. The participants who do not submit their shares also can count as malicious participants.

Result: A stage that allows all people to view and verify the election result. To achieve the above purpose, the vote private key will be recovered first. Then, all ballots will be decrypted and tallied.

Failed: A stage that represents the election is corrupt and no longer available because the number of honest participants is less than the threshold of reconstructing the key such that the vote private key cannot be reconstructed.

4.1.3 Punishment

In this application, the punishment for the malicious participants is not allowing them to vote only. However, the election organization can know the malicious participants' actual identities from their public keys such that the election organization also can decide their punishment.

4.1.4 Stages Transition

According to the timer settled by the election organization in the setup stage, the stage will transition into another stage. For example, the current time is larger than the distribution end time and smaller than the verification end time. Then, the current stage is the vote stage. Moreover, if the number of honest participants is less than the threshold of reconstructing the private key, the current stage will transit into the failed stage.

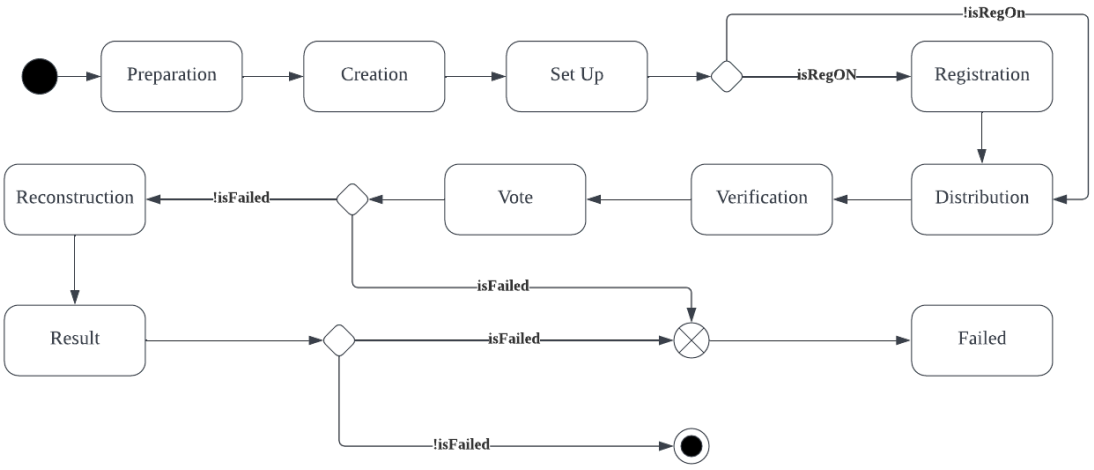


Figure 4.1: An Overview of E-Voting Stages

4.1.5 Procedure Description

After the notations section, the procedures of each stage are explained in detail. If one step of the procedure is "throw an error", then the following steps of that procedure will not be executed.

For a simple illustration, the data stored in the election contract can be directly used in the front end without retrieval.

For some function calls, when the function is successfully executed, it will emit the event to the client to update the contract's current status. However, in the procedure description, the event is skipped and assumes the client always knows the latest status of the contract.

4.2 Notations

The following are the notations of the parameters and functions used in our application:

p**:** a large prime number.

F_p**:** a finite field over p.

$E(F_p)$**:** an elliptic curve on finite field F_p.

G**:** the generator point of the curve $E(F_p)$.

n**:** the order of the curve $E(F_p)$.

m**:** the number of participants / public keys.

q**:** the number of malicious or disqualified participants.

d**:** the number of candidates.

h**:** the number of registers' information.

b**:** the number of ballots.

t**:** the number of minimum shares for threshold key reconstruction.

B_i**:** the encrypted ballot B_i.

C_i**:** the candidate C_i.

$C_i.voteCount$**:** the number of votes of candidate C_i. Initially, $C_i.voteCount = 0$ for $i \in [1, d]$.

P_i**:** the participant P_i.

R_i**:** the register's information R_i (points of $E(F_p)$).

sk_i**:** the private key of P_i.

pk_i**:** the public key of P_i.

B**:** A set of encrypted ballots $\{B_1, B_2, ..., B_b\}$.

C**:** A set of candidates $\{C_1, C_2, ..., C_d\}$.

P**:** A set of participants $\{P_1, P_2, ..., P_m\}$.

pk**:** A set of public keys $\{pk_1, pk_2, ..., pk_m\}$.

R**:** A set of registers' information $\{R_1, R_2, ..., R_h\}$.

P'**:** A set of honest participants $P' \subseteq P$. Initially, $P = \emptyset$.

Q**:** A set of malicious or disqualified participants where $Q \subseteq P$. Initially, $Q = \emptyset$.

$f_i(x)$**:** random polynomial of $P_i = a_{i,t-1}x^{t-1} + a_{i,t-2}x^{t-2} + a_{i,t-3}x^{t-3} + ... + a_{i,0}$ (mod n) of degree $t-1$ where all coefficients $\{a_{i,0}, a_{i,1}, ..., a_{i,t-1}\}$ and $f_i(x) \in \mathbb{Z}_n$.

$A_{i,j}$**:** a commitment of $a_{i,j} = a_{i,j}G$ for $j \in [0, t-1]$.

H**:** the vote public key $H = \sum_{j=1}^{m} A_{j,0}$.

$f(i)$**:** the secret share of P_i.

S**:** the threshold vote private key $S = \sum_{i=1}^{m} f(i)$ (mod n).

$isReg_i$**:** A boolean value indicates whether R_i is already used for registration.

$isSetup$**:** A boolean value for indicating whether the registration stage is on.

$isFailed$**:** A boolean value for indicating whether the current stage is failed stage.

$isA_{i,j}$**:** A boolean value for indicating whether $A_{i,j}$ and its related data are stored in the contract.

$isF_i(k)$**:** A boolean value for indicating whether $F_i(k)$ and its related data are stored in the contract.

now**:** the current time.

$isf(i)$**:** A boolean value for indicating whether $f(i)$ is uploaded to the contract.

now**:** the current time.

$regEndTime$**:** the registration stage end time.

$disEndTime$**:** the distribution stage end time.

$verEndTime$**:** the verification stage end time.

$voteEndTime$**:** the vote stage end time.

$reconEndTime$**:** the reconstruction stage end time.

K_b**:** the tag value of linkable ring signature signing algorithm output.

K**:** A set of $K_1, K_2, ..., K_b$.

M**:** the arbitrary message.

$hash(M)$**:** the cryptographic hash function where the input is M.

$schnorrProve(a)$**:** the schnorr's prove protocol where the input is secret $a \in \mathbb{Z}_n$.

$schnorrVer(A, p)$**:** the schnorr's verify protocol where the inputs are $A = aG$ and the schnorr proof p.

$ecdsaSign(M, sk_i)$**:** the ecdsa's signing algorithm where the inputs are M and private key sk_i.

$ecdsaVer(M, sig, pk_i)$**:** the ecdsa's verify algorithm where the inputs are M, ecdsa signature sig, and signner's public key pk_i.

$elgamalEnc(M, pk_i)$**:** the elgamal encryption algorithm where the inputs are M and public key pk_i. The output of this function is $(C, D = C' + P_M)$.

$elgamalDec(C, sk_i)$**:** the elgamal decryption algorithm where the inputs are elgamal ciphertext C and receiver's private key sk_i. The output of this function is plaintext M.

$cpProve(G, H, x)$**:** the Chaum-Pedersen prove protocol where inputs are two base point G, H and a secret value $x \in Z_n$. the output is a proof $p = (r, c)$.

$cpVer(G, A, H, B, p)$**:** the Chaum-Pedersen verify protocol where inputs are two base point G, H, a point A that its the base point is G, a point B that its the base point is H, and a Chaum-Pedersen proof.

$lrsSig(M, L, sk_i)$**:** the linkable ring signature signing algorithm where inputs are M, list of public keys $L = \{pk_1, ..., pk_m\}$, and signer's private key sk_i. The output of this function is $sig = (u_1, V = \{v_1, ..., v_m\}, K)$.

$lrsVer(M, sig)$: the linkable ring signature signing algorithm where inputs are M, and the proof of $lrsSig(M, L, sk_i)$.

4.3 Creation Stage

There are two smart contracts except for the libraries in our application:

VotingApp: A simple contract for creating a smart Election contract. It stores the total number of Election smart contracts and the addresses of all Election smart contracts.

Election: Anyone can get all constructed election contract addresses from the Votingapp contract. Then the user can access election contracts and do operations such as vote and view the result. Different operations in the election have different requirements. For example, vote operation requires a valid linkable ring signature.

The following are the steps of creating a new election [Figure 4.2]:

1. The election organization establishes an EOA account.
2. The election organization send a message call: `addElection()` to the VotingApp.

3. Upon receipt of `addElection()`, the votingApp construct a new instance of Election smart contract with the current total number of Election smart contracts id and the address of sender $msg.sender$.

4. Then the new election is created that its id and owner is id and $msg.sender$ respectively such that only the owner can set up this election.

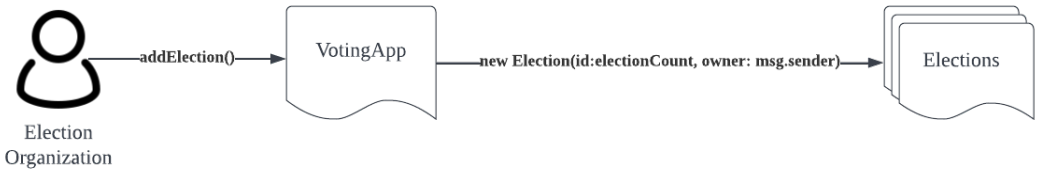


Figure 4.2: Election Creation Process

4.4 Set Up Stage

4.4.1 Front End Side

The election organization set the following election information by sending a transaction to Election Contract function `setElectionInfo(data)` :

title: the election title (integer).

description: the election description (integer).

candidates: list of candidates' names (integer).

publickeys: list of eligible participants' public keys.

minShares: number of minimum shares for threshold key reconstruction.

regTime: the time limit of registration time.

disTime: the time limit of distribution time.

verTime: the time limit of verification time.

voteTime: the time limit of vote time.

reconTime: the time limit of reconstruction time.

regTimeUnit: minute / hour / day

disTimeUnit: minute / hour / day

verTimeUnit: minute / hour / day

voteTimeUnit: minute / hour / day

reconTimeUnit: minute / hour / day

isRegOn: a boolean value for indicating whether turn on the registration stage.

regInfo: list of eligible register' information.

4.4.2 Election Contract Side

The function `setElectionInfo(data)` is defined as follows:

1. if the sender is not the owner $msg.sender \neq owner$, throw an error.
2. if this election has already setup $isSetup = true$, throw an error.
3. if the number of $data.candidates$ or $data.minShares < 2$, throw an error.
4. set $title = data.title, description = data.description, C = data.candidates,$ $pk = data.publickeys, t = data.minShares, isRegOn = data.isRegOn, postTime = now.$
5. if the registration stage is open $isRegOn = true$, then set $R = data.regInfo$ and compute the registration end time and distribution end time

$$regEndTime = postTime + data.regTime \cdot data.regTimeUnit$$

$$disEndTime = regEndTime + data.disTime \cdot data.disTimeUnit$$

Otherwise, compute the distribution end time

$$disEndTime = postTime + data.disTime \cdot data.disTimeUnit$$

6. compute the remaining timers including verification end time, vote end time, and reconstruction end time

$$verEndTime = disEndTime + data.verTime \cdot data.verTimeUnit$$

$$voteEndTime = verEndTime + data.voteTime \cdot data.voteTimeUnit$$

$$reconEndTime = voteEndTime + data.reconTime \cdot data.reconTimeUnit$$

7. set $isSetup = true$.

4.4.3 Implementation Consideration

Since the contract code size cannot be more than 24576 bytes, the $title$, $description$, and C are integer for saving the contract code size. Therefore, they are encoded into integers from the string for storing them in the election contract. Later On, they need to be encoded back into string such that the user can understand their meaning.

4.5 Registration Stage

4.5.1 Front End Side

For registration to become a new participant P_{m+1}, the following steps are needed:

1. generate a key pair $\{sk_{m+1}, pk_{m+1}\}$.
2. input his personal information $R_?$ and compute $r = hash(R_?)$ (mod n).
3. check the validity of identity by checking $rG \stackrel{?}{=} R_i$ for $i \in [1, h]$.
4. if $rG = R_i$, then generate a schnorr proof $p = schnorrProve(r)$ and send $\{pk_{m+1}, i, p\}$ to the election contract. Otherwise, the current user is not eligible and do nothing.

4.5.2 Election Contract Side

The procedures of after receiving $\{pk_{m+1}, i, p\}$ is defined as follows:

1. if the registration stage is not open $isRegOn = false$, throw an error.
2. if current stage is not registration stage $now > regEndTime$, throw an error.
3. if R_i has registered $isReg_i = true$, throw an error.
4. if the proof p is incorrect $schnorrVer(R_i, p)$, throw an error.
5. add participant $P = P \cup \{P_{m+1}\}$, public keys $pk = pk \cup \{pk_{m+1}\}$, and update $m = m + 1$.

4.6 Distribution Stage

4.6.1 Front End Side

The procedures of front end side work as follows:

1. Each participant P_i generates a random polynomial $f_i(x) = a_{i,t-1}x^{t-1} + a_{i,t-2}x^{t-2} + a_{i,t-3}x^{t-3} + ... + a_{i,0}$ (mod n) of degree $t-1$ where all coefficients $a_{i,j} \in \mathbb{Z}_p$ and $f_i(x) \in \mathbb{Z}_n$, and computes a commitment $A_{i,j} = a_{i,j}G$ for $j \in [0, t-1]$.

2. Each participant P_i computes hash $hA_{i,j} = hash(A_{i,j}, i, j)$ and signature $sigA_{i,j} = ecdsaSign(hA_{i,j}, sk_i)$ for $j \in [0, t-1]$.

3. Each participant P_i computes ciphertext $F_i(j) = elgamalEnc(f_i(j), pk_j)$, hash $hF_i(j) = hash(F_i(j), i, j)$, and signature $sigF_i(j) = ecdsaSign(hF_i(j), sk_i)$ for $j \in [1, m]$.

4. Each participant P_i sends $\{ A_{i,j}, i, j, hA_{i,j}, sigA_{i,j} \}$ and $\{ F_i(k), i, k, hF_i(k), sigF_i(k) \}$ to the election contract for $j \in [0, t-1]$ and $k \in [1, m]$.

4.6.2 Election Contract Side

The procedures of election contract side works as follows:

1. if current stage is not distribution stage $now > disEndTime \vee now < regEndTime$, throw an error.

2. For $j \in [0, t-1]$,

   (a) if $A_{i,j}$ and its related data are stored in the contract $isA_{i,j} = true$, throw an error.

   (b) compute $hA'_{i,j} = hash(A_{i,j}, i, j)$ and if $hA'_{i,j} \neq hA_{i,j}$, throw an error.

   (c) if the signature $sigA_{i,j}$ is incorrect $ecdsaVer(hA_{i,j}, sigA_{i,j}, pk_i) = false$, throw an error.

   (d) store $\{ A_{i,j}, i, j, hA_{i,j}, sigA_{i,j} \}$ in this contract.

   (e) set $isA_{i,j} = true$.

3. For $k \in [1, m]$,

   (a) if $F_i(k)$ and its related data are stored in the contract $isF_i(k) = true$, throw error.

   (b) compute $hF_i(k)' = hash(F_i(j), i, j)$ and if $hF_i(k)' \neq hF_i(k)$, throw an error.

   (c) if the signature $sigF_i(k)$ is incorrect $ecdsaVer(hF_i(k), sigF_i(k), pk_i) = false$, throw an error.

   (d) store $\{ F_i(k), i, k, hF_i(k), sigF_i(k) \}$ in this contract.

   (e) set $isF_i(k) = true$.

4.6.3 Adversary Actions In Front End

In the front end, the adversaries may attempt the following three actions for making this election fail:

- The malicious participant P_i does not send $\{ A_{i,j}, i, j, hA_{i,j}, sigA_{i,j} \}$ and $\{ F_i(k), i, k, hF_i(k), sigF_i(k) \}$ to the election contract for $j \in [0, t-1]$ and $k \in [1, m]$ for constructing the threshold key pair.
- The malicious participant P_i distribute incorrect $F_i(k) = q$ where $q \in \mathbb{Z}_n$ and $q \neq F_i(k)$ to some or all honest participants P_k.
- The malicious participant P_i distribute incorrect $F_i(k) = q$ where $q \in \mathbb{Z}_n$ and $q \neq F_i(k)$ to the malicious participants P_k.

In the verification stage, there are some methods for handling these actions.

4.7 Verification Stage

The main goal of the verification stage is to identify and filters out malicious participant to ensure the election can run well.

4.7.1 Report Non-contributed Participant

First, we need to report non-contributed participant by simply send a transaction with no data to the election contract function. The function works as follows:

1. if the current stage is not verfication stage $verEndTime < now \vee now < disEndTime$, throw an error.
2. if this function has already executed $isNoConParCheck = true$, throw an error.
3. For $i \in [1, m]$,
   (a) check whether participant P_i sends his commitment A_{i_j} by checking $isA_{i_j} \stackrel{?}{=} true$ for $j \in [0, t-1]$. If any $isA_{i_j} = false$, add participant P_i in disqualified participants set $\{P_i\} \cup Q$.
   (b) verify whether participant P_i sends $F_i(k)$ to participant P_k by checking $isF_i(k) \stackrel{?}{=} true$ for $k \in [1, m]$. If any $isF_i(k) = false$, add participant P_i in disqualified participants set $\{P_i\} \cup Q$.
4. set this function has already executed $isNoConParCheck = true$.

Noted that this function can trigger once by anyone.

4.7.2 Report Malicious Participant

The following are the steps of honest participants verify the validity of the $f_j(i)$ and generate proof for reporting malicious participants:

1. Each participant P_i decrypt $F_j(i) = (C, D = C' + f_j(i))$ to get $f_j(i) = elgamalDec(F_j(i), sk_i)$ for $j \in [1, m]$ and $P_j \notin Q$.
2. Each participant P_i verifies $f_j(i)$ by checking $f_j(i)G \stackrel{?}{=} \sum_{k=0}^{t-1} A_{j,k} \cdot i^k$ for $j \in [1, m]$ and $P_j \notin Q$. If $f_j(i)G \neq \sum_{k=0}^{t-1} A_{j,k} \cdot i^k$, compute proof of correct decryption key $p = cpProve(G, F_j(i).C, sk_i)$ and the decryption key $C' = sk_i \cdot F_j(i).C$.
3. Each participant P_i sends $\{C', j, i, p\}$ for P_j sent incorrect $f_j(i)G$ to election contract.

Upon the election receipt of $\{C', j, i, p\}$ for P_j sent incorrect $f_j(i)G$, the following procedures is executed:

1. if the current stage is not verfication stage $verEndTime < now \vee now < disEndTime$, throw an error.
2. For each $\{C', j, i, p\}$,
   (a) verify C' whether is correct decryption key by check $cpVer(G, P_i, F_j(i).C, C') \stackrel{?}{=} true$. if $cpVer(G, P_i, F_j(i).C, C', p) = false$, throw an error.
   (b) decrypt $F_j(i)$ by using C' such that $f_j(i)$ is obtained.
   (c) if $f_j(i)G \neq \sum_{k=0}^{t-1} A_{j,k} \cdot i^k$, add participant P_j in disqualified participants set $\{P_j\} \cup Q$.

The reason why $cpVer(G, P_i, F_j(i).C, C')$ can prove the whether the key is correct is this function is checking $log_G(P_i)$=$log_{F_j(i).C}(C')$ where $P_i = sk_iG, F_j(i).C = rG$, and $C' = sk_irG$. If $C' \neq sk_irG$, then $log_G(sk_iG) \neq log_{rG}(C')$.

4.7.3 Non-detected Adversary

If the malicious participant P_i distribute incorrect $F_i(k) = q$ where $q \in \mathbb{Z}_n$ and $q \neq F_i(k)$ to the malicious participants P_k, then the honest participants cannot detect and report them because each participant only can verify his values. However, if $t \leq (m-a)$ where a is number of malicious participant including non-detected adversaries, then the election still works. Because t number of honest participants can compute their shares and the threshold private key can be reconstruct if have t number of shares.

4.8 Vote Stage

Vote Stage is a stage that allows participants to vote for their target candidate once without revealing their identity, and the ballots are encrypted.

4.8.1 Set Vote Public Key

The vote public key H need to be computed first such that the ballots can be encrypted by H. The election contract function for setting vote public key is defined as follows:

1. if current stage is not vote stage $now > voteEndTime \vee now < verEndTime$, throw an error.
2. if this function already executed $isVotePubKeySet = true$, throw error.
3. set honest participants $P' = P - Q$.
4. remove malicious participants' public key $pk = pk - \{pk_j\}$ for $j \in [1, m], P_j \in Q$.
5. compute $H =$ sum of pk_i for $pk_i \in pk$.
6. if $|P'| < t$, set current stage is failed stage $isFailed = true$.
7. set this function has already executed $isVotePubKeySet = true$.

Noted that this function can trigger once by anyone.

4.8.2 Vote

The following are the steps of how a participant sends his ballot to the election contract:

1. Participant P_i select his target candidate C_j and encrypt his ballot $B_b = elgamalEnc(C_j, H)$ where $j \in [1, d]$ and b is number of ballots.
2. Participant P_i hash his encrypted ballot $hB_b = hash(B_b)$ and sign the encrypted ballot using linkable ring signature $sig_{hB_b} = lrsSign(hB_b, pk, sk_i) = \{u_b 1, V_b, K_b\}$.
3. Participant P_i sends $\{B_b, hB_b, sig_{hB_b}\}$ to election contract.

The election contract function handles $\{B_b, hB_b, sig_{hB_b}\}$ by following steps:

1. if current stage is not vote stage $now > voteEndTime \vee$ $now < verEndTime \vee isFailed = true$, throw error.
2. if the public key H is not computed, throw an error.
3. if the participant has already voted $sig_{hB_b}.K_b \in K$, throw an error.
4. if $hash(B_b) \neq hB_b$, throw an error.
5. if the signature is invalid $lrsVer(hB_b, sig_{hB_b}) = false$, throw an error.
6. stores $\{B_b, hB_b, sig_{hB_b}\}$ in the contract.
7. update $\{K_b\} \cup K$ and $b = b + 1$.

4.9 Reconstruction Stage

4.9.1 Front End Side

The procedures of front end side works as follows:

1. Each participant P_i decrypt $F_j(i) = (C, D = C' + f_j(i))$ to get $f_j(i) = elgamalDec(F_j(i), sk_i)$ for $j \in [1, m]$ and $P_i, P_j \in P'$.
2. Each participant P_i sends $\{f_j(i), i, j\}$ to election contract for $j \in [1, m]$ and $P_i, P_j \in P'$.

4.9.2 Election Contract Side

Upon the election receipt of $\{f_j(i), i, j\}$ for $j \in [1, m]$ and $P_i, P_j \in P'$, the following procedures is executed:

1. if current stage is not reconstruction stage $now > reconEndTime \vee now < voteEndTime \vee isFailed = true$, throw an error.
2. if the secret share $f(i)$ has uploaded $isf(i) = true$, throw an error.
3. set $f(i) = 0$.
4. For each $\{f_j(i), i, j\}$,
   (a) if $f_j(i)G \neq \sum_{k=0}^{t-1} A_{j,k} \cdot i^k$, throw an error.
   (b) $f(i) = f(i) + f_j(i)$.
5. store $\{f(i), i\}$.
6. set the secret share $f(i)$ has uploaded $isf(i) = true$.

Noted that this function can be trigger by anyone. Because if the malicious party impersonate honest participant P_i and sends correct $\{f_j(i), i, j\}$ for $j \in [1, m]$ and $P_i, P_j \in P'$ to the election contract luckily. Then, the opportunity of successful recover threshold private key S is increased actually and the honest participant P_i does not need to pay the cost of this function execution.

4.10 Result Stage

A result stage is a stage that allows all people to view and verify the election result. To achieve the above purpose, the vote private key S needs to be recovered first. Then, all ballots will be decrypted and tallied.

4.10.1 Front End Side

The procedures of front end side is defined as follows:

1. let P'' is a set of t number of participants who submitted their shares, compute the private key

$$S = \sum_{i \in P''} f(i)\lambda_i(0), \text{ where } \lambda_i(0) = \prod_{j \in P'', i \neq j} \frac{j}{j-i} \ (\text{mod } n)$$

2. send S to election contract.

Noted that this function can trigger once by anyone. The private key is computed in the front end because it can reduce the transaction cost.

4.10.2 Election Contract Side

The following are the steps of tallying the ballots:

1. if current stage is not result stage $isFailed = true \vee now < voteEndTime$, throw an error.
2. if the private key cannot be recovered $|\{f(i), i\}| < t$, change the stage to be failed stage $isFailed = true$ and then throw an error.
3. if the ballots are tallied $isVoteTallied = true$, throw an error.
4. if the threshold vote private key S is incorrect $S \cdot G \neq H$, throw an error.
5. For each encrypted ballot $B_i \in B$,
   (a) compute $C_j = elgamalDec(B_i, S)$.
   (b) if $C_j \in [1, d]$, increase vote count of C_j such that $C_j.voteCount = C_j.voteCount + 1$ and update $\{C_j\} \cup \{B_i, hB_i, sig_{hB_i}\}$ = $\{B_i, hB_i, sig_{hB_i}, C_j\}$.
6. set the ballots are tallied $isVoteTallied = true$.

Noted that this function can trigger once by anyone. In the vote stage, malicious participants P_i can sends encrypted ballot $B_i = (C, D = C' + C'_j)$ where $C'_j \notin [1, d]$ to the election contract that the election contract would accept this ballot. To make $C'_j \notin [1, d]$, malicious participants can use another encryption key S' to encrypt C_j where $S' \neq S$ or simply randomly pick a point on curve $E(F_p)$. However, the elgmal decryption algorithm still works $C'_j = elgamalDec(B_i, S)$ because the algorithm must can output the x coordinate of $elgamalDec(B_i, S) = D - S \cdot C$. Therefore, the simple solution for this attack is ensure the $C_j \in [1, d]$ before update the vote count.

4.10.3 Verifiability

Since all ballots $\{B_i, hB_i, sig_{hB_i}, C_j\}$ are stored in the contract including its signature and ciphertext, the general public can easily verify the validity and linkability of the signature through our front end function.

Chapter 5

Security Analysis

In this chapter, the security aspect of our application will be discussed:

Anonymity: No one can trace the voter from the ballot.

The ballot is signed by the linkable ring signature such that the probability of knowing the signer is $\frac{1}{n}$ where n is the total number of honest participants. In addition, the key pair used to sign the ballot is no relationship with the EOA account's key pair. Therefore, if n is large enough, anyone including participants and the election organization cannot know the voter from the ballot.

Eligibility: Only the honest participant can cast a vote.

The linkable ring signature ensures the people have the private key corresponding to one of the public keys in the participants' public key list. To add the corresponding public key to the public key list, the participant needs to register with the election organization first. Therefore, like many real scenarios, only the election organization can control the public key list that can control who can vote. If the participant's public key should be in the public key list but does not, the participant needs to contact the election organization to handle this situation. Furthermore, the malicious participants who want to corrupt the election by making the threshold private key cannot be recovered. These participants can be detected and not allowed to vote as a punishment.

Double-voting Avoided Each voter only can cast a vote once.

The linkability of the linkable ring signature allows the election smart contract to identify whether the same voter signs the two ballots by storing the tags of each ballot. If the requested ballot has the same tag as one of the tags stored in the contract, then deny this ballot to avoid double voting.

Verifiability: Anyone can verify whether the election result is correct. Also, All voters can verify whether their vote exists.

All the valid ballots are stored in the election smart contract. Anyone can check the validity of the ballots' signatures and the uniqueness of the ballots to ensure that all ballots are correct. Then, anyone can verify the election result by counting the ballots by them. In addition, the voter can use the linkable

ring signature's tag to check whether his ballot exists in the election contract to verify whether his vote has been counted.

Fairness: Before the voting ends, no one can know the intermediate result.

All the ballots are encrypted by the vote public key H, and the ballots only can be decrypted after the vote stage. Therefore, no one can know the intermediate election result in the vote stage such that the election result does not affect by the intermediate result.

Correctness: The ballots can be tallied correctly.

The ballots and the election result are stored in the election smart contract. In other words, the ballots and the election result are stored a decentralized computing environment that ensures others cannot modify them. Therefore, the election result is tallied by the correct ballots such that the election result is also correct.

Robustness: Even the malicious participants exist, the election will not break.

Let the number of participants be m and the minimum shares for recovering the vote private key by t. Then, the election still can work well if the number of malicious participants is a where $a < (m - t)$. For example, in the distribution stage, if some participants do not share their values or distribute incorrect values to other participants, the election can run as usual. Moreover, in the reconstruction stage, if some participants do not submit their shares, the election result can still be tallied.

Impersonation Attack Avoided: The attacker can not impersonate the participant to perform harmful actions to the participant and the election.

In the distribution stage, the malicious participants cannot impersonate other honest participants to send the incorrect values to others, which leads the honest participants to incorrectly be malicious because the participant uses his private key signs the values. Therefore, other people can verify the sender of these values by his public key.

Chapter 6

Application Architecture

In chapter 1, the transitional application architecture is introduced. In this chapter, our decentralized web application architecture is briefly explained, which differs from the transitional architecture.

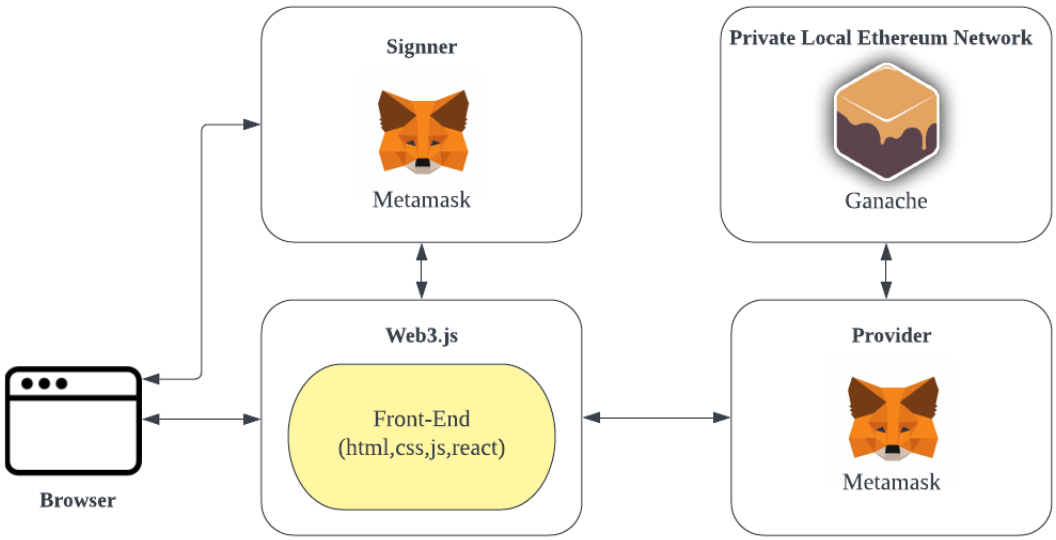


Figure 6.1: Decentralized Web Application Architecture

The following are the explanations of Figure 6.1:

Web3.js: Web3.js is a collection of libraries that allow the browser can communicate with a local or remote Ethereum node through a provider such that the front end can interact with smart contract [35]. Web3 provider/provider is used to determine which Ethereum client our app interacts with.

MetaMask: MetaMask is a browser extension and mobile application for handling account management and connecting the user to the Ethereum blockchain. [36]. The interaction between the user and Ethereum client needs an Ethereum account to sign each transaction or function call. In our application, users can select different EOA accounts for signing transactions by MetaMask. In addition, the

functionality of the Web3 provider is achieved by MetaMask that MetaMask communicates with our local Ethereum's private network.

Ganache: Ganache is a personal blockchain for rapid Ethereum and Corda distributed application development [37]. It can be used as our local Ethereum private network.

Chapter 7

Protocol Comparison

In this chapter, we will compare our e-voting protocol with the e-voting protocol proposed by Lyu et al.[6]. First, we introduce the e-voting protocol proposed by Lyu et al. Then, we discuss the problems with their protocol. Next, we explain how we modify the protocol for solving the problems.

7.1 The E-voting Protocol Proposed by Lyu et al

Their protocol also based on elliptic curve $E(F_q)$ whose order is a prime q, the generator point of $E(F_q)$ is G and the order of G is l. In addition, all the data are sent to the smart contract with the signatures to ensure the data are from legitimate voters. Their protocol is defined as follows:

Register Stage: Voter P_i generates a key pair $(pk_i = sk_iG, sk_i)$ and sends pk_i to the election administrator EA secretly for $i \in [1, n]$.

Set Up Stage: EA sends to following data to the smart contract:

- n: number of voters.
- all voter public keys $pk_1, pk_2, ..., pk_n$.
- a list of timers for controlling election progresses.
- t: number of minimum shares to reconstruct the threshold private key.

After that, EA publishes the address of the election smart contract.

Key generation Stage: Each P_i choose a random polynomial $f_i(z) \in l$ of degree $t-1$ that $f_{i,j} \in Z_q$ for $j \in [1, t-1]$ and $f_i(0) = x_i$.

$$f_i(z) = (f_{i,0} + f_{i,0}z + ... + f_{i,t-1}z^{t-1}) \bmod l$$

The each P_i do the following things:

1. computes $F_{i,j} = f_{i,j}G$ for $j \in [0, t-1]$ and sends $(F_{i,j}, j, j)$ with P_i signature to the smart contract.
2. computes $s'_{i,j} = f_i(j)$ for $j \in [1, n]$ and gets $s_{i,j} =$ encrypted form of $s'_{i,j}$ by pk_j.
3. sends $(s_{i,j}, i, j)$ with P_i signature to the smart contract for $j \in [1, n]$.

The smart contract verifies the signatures of every $F_{i,j}$ and $s_{i,j}$ and broadcasts that an error if the signature is invalid. After all P_i finished their procedures, the public key g is computed as $g = \sum_{i=1}^{n} F_{i,0}$.

Vote Stage: P_i computes his vote V'_i according to his choice and the coding rules and gets $V_i =$ the encrypted form of V'_i by g. Then, P_i sends V_i with a linkable ring signature by using voter public key list.

The smart contract verifies the validity and the linkability of the signature to ensure the vote is from an eligible voter and avoid double voting.

Sub secret generation Stage: The each P_i do the following things:

1. obtains $s'_{j,i}$ by decrypting $s_{j,i}$ with sk_i for $j \in [1, n]$.
2. verifies $s'_{j,i}$ for $j \in [1, n]$ by the equation:

$$s'_{j,i} \cdot G = \sum_{l=0}^{t-1} F_{j,l} \cdot i^l$$

3. computes $s_i = \sum_{j=1}^{n} s'_{j,i}$.

Secret Upload Stage: P_i sends (s_i, i) with his signature to the smart contract. The smart contract use t of all s_i to recover the secret key x by the equation:

$$x = \sum s_j \cdot \prod_{h=1, h\neq j}^{t} \frac{h}{h-j} \text{ (mod } q \text{)}$$

Tally: The smart contract uses x to decrypt all the votes V_i, then tallies the votes, and then publishes the result.

7.2 Problems of the protocol

In the key generation stage and secret upload stage, the malicious voters can destroy the election such that the election needs to restart.

Key Generation Stage: if one malicious voter P_i sends $(F_{i,j}, j, j)$ with P_i signature to the smart contract but P_i sends incorrect $s'_{i,j} =\neq f_i(j)$ for $j \in [1, n]$ or does not send to the smart contract, then the secret key x cannot be reconstructed. The reason behind it is:

- each P_j computes incorrect $s_j = \sum_{k=1}^{n} s'_{k,j}$ since $s'_{k,j}$ is incorrect/nothing if $k = i$.
- then the secret key x cannot be recover since all s_j are wrong.

Secret Upload Stage: the malicious voter P_i can sends incorrect s_i where $s_i \neq \sum_{j=1}^{n} s'_{j,i}$ to the smart contract and the smart contract does not know it is incorrect. Therefore, the smart contract may recover the wrong secret key x if s_i is used in the recover secret key equation.

In addition, the protocol does not mention how to handle the incorrect $s'_{j,i}$ is found in the sub secret generation stage. Furthermore, the equation for recovering the secret key x only works if all P_i submitted their s_i to the smart contract for $i \in [1, t]$. In other words, we cannot arbitrarily pick t of s_i to apply this equation.

7.3 Our Improvement

Let the number of voters be n and the minimum shares for recovering the vote private key is t. To ensure the election still can work well if the number of malicious participants is a where $a < (n-t)$, we first filter out the malicious voters by smart contract and honest voters before construct our public private key pair (g, x). Instead of P_i sends s_i to the smart contract for recovering x, P_i sends $s'_{j,i}$ to the smart contract for $j \in [1, m]$ where m is number of honest voters such that the smart contract can verify each $s'_{j,i}$ to make sure s_i is correct by the equation:

$$s'_{j,i} \cdot G = \sum_{l=0}^{t-1} F_{j,l} \cdot i^l$$

Let $P = 1, 2, ..., n$ is a set of honest voters and Q is any set of t number of voters in P. To reconstruct the private key, compute

$$x = \sum_{j \in Q} s_j \cdot \prod_{h \in Q, j \neq h} \frac{h}{h - j} \ (\text{mod } q\ )$$

Chapter 8

Time and Cost Analysis on Ethereum'S Private Network

In this chapter, we discuss the computation time of executing the front-end function, and the gas cost of the election smart contract function call is also discussed. Before we analyze the time and the cost of our application on Ethereum's private network, the configuration of the test case and the testing methodology is explained first.

8.1 Testing Configuration

the testing configuration is defined as follows:

- number of participants is n.
- number of honest participant is $m = \lfloor \frac{n}{2} \rfloor$.
- number of non-contributed participants is $q = \lceil \frac{n}{2} \rceil$.
- number of minimum shares to reconstruct the threshold private key is $t = m = \lfloor \frac{n}{2} \rfloor$.
- number of participants cast their vote is $s = m = \lfloor \frac{n}{2} \rfloor$.
- number of candidates is 2.
- $isRegOn = true$. The registration stage is on each test case.
- The election organization sends three numbers of registers' information to the smart contract. Only 1 of the register will become the participant such that $n = n + 1$ but m, q, t, and s will not change.
- the timers is off.

8.2 Testing Methodology

the following are our testing methodology:

Front-End Function: The `console.time()` and `console.timeEnd()` methods is used for tracking the execution time of our front-end functions.

Smart Contract Function: After the function call is completed, the ganache will output the gas usage of this function call in the command-line interface [Figure 8.1]. Therefore, we use this method to test the gas usage of the election contract function calls.

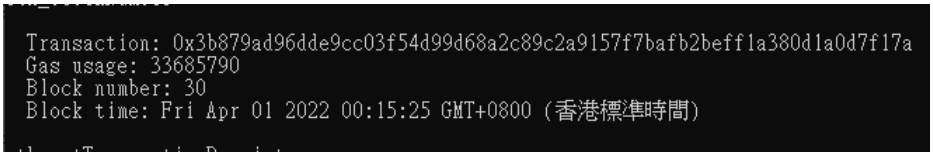


Figure 8.1: Gas usage of a transaction.

8.3 Time Analysis

Table 8.1: The computation times of front end operations for participant

Operations	**Time(ms)**		
-	n = 5	n=10	n=20
Registration Stage: Verify Identiy	1175.6896		
Distribution Stage: generate random polynomials, values, and commitments	48.1970	107.625	220.6791
Distrubition Stage: encrypt the values	329.9948	437.8510	845.3510
Distribution Stage: generate signatures of values and commitments	177.1350	308.5910	585.3620
Vote Stage: encrypt vote	219.4848	207.9550	204.7231
Vote Stage: generate LRS	479.5280	923.1291	1752.7351
Reconstruction Stage: generate sub secret	275.9550	612.1489	1125.2580
Total	2,705.9842	3,772.9896	5,909.7979

8.4 Cost Analysis

Table 8.2: The gas costs of computing the transactions.

Operations	**Gas**		
-	n = 5	n=10	n=20
Create stage: create a election contract	4180519		
Set Up stage: set up a election	728673	959689	1237850
Registration stage: add and verify new participant	1409589		
Distribution stage: store and verify values and commitment	11039918	23809808	47183928
Verification stage: verify and filter out non-contributed participant	243056	887264	2793665
Vote stage: compute public key and LRS parameters	1082344	1498844	1981617
Vote stage: add and verify vote	8857119	7585490	33685790
Reconstruction: add and verify shares	2460207	7681470	22928825
Total	30,001,425	48,012,673	113,992,194

Chapter 9

Conclusion

9.1 Summary

This report first discussed the drawbacks of traditional e-voting and the advantages of blockchain-based e-voting. Then, the project objectives and outcomes are given. Next, the theory behind our application is explained. Then, we described our application in detail by stages. After that, we analyzed the security aspects of our application. Then, we introduced our decentralized web application architecture. Then, we compare our e-voting protocol with the e-voting protocol proposed by Lyu et al.[6]. Finally, we provided the time and cost analysis.

9.2 Future Work

The time and gas cost of our application is very high such that it is impractical in large-scale elections. Therefore, we can investigate or find some efficient cryptography algorithms that can achieve the security requirements for an election in the future.

We can extend our verifiable threshold cryptosystem to a publicly verifiable threshold cryptosystem such that anyone can verify the correctness of the shares [38]. We can let the smart contract by the verifier in the distribution stage such that we can reduce the stage in our application.

The election will start after the election organization sets up the information in the smart contract. In other words, our application does not allow the election organization to set up a future election. Therefore, we can add one more timer called start time to allow them to set up a future election.

Furthermore, a list of timers can compute in the front end to reduce the cost of the setup function call.

9.3 Acknowledgement

Lastly, I would like to thank my supervisor: Dr Yan Wang Liu Dennis, for providing guidance, suggestion, and feedback about the project direction and implementation throughout this capstone project.

Bibliography

[1] B. Schwartz, “Total 2020 election spending to hit nearly $14 billion, more than double 2016ś sum,” Available at https://www.cnbc.com/2020/10/28/2020-election-spending-to-hit-nearly-14-billion-a-record.html (2020/10/28).

[2] e Estonia, “e-governance,” Available at https://e-estonia.com/solutions/e-governance/e-democracy/.

[3] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” http://bitcoin.org/bitcoin.pdf.”

[4] Simplilearn, “Merkle tree in blockchain: What is it, how does it work and benefits,” Available at https://www.simplilearn.com/tutorials/blockchain-tutorial/merkle-tree-in-blockchain (2021/11/08).

[5] L. Insights, “Korea to trial blockchain in large scale online voting,” Available at https://www.ledgerinsights.com/korea-to-trial-blockchain-in-large-scale-online-voting/ (2021/6/23).

[6] J. Lyu, Z. L. Jiang, X. Wang, Z. Nong, M. H. Au, and J. Fang, “A secure decentralized trustless e-voting system based on smart contract,” in *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2019, pp. 570–577.

[7] M. N. Uddin, S. Ahmmed, I. A. Riton, and L. Islam, “An blockchain-based e-voting system applying time lock encryption,” in *2021 International Conference on Intelligent Technologies (CONIT)*, 2021, pp. 1–6.

[8] G. Wood *et al.*, “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[9] L. Saldanha, “Merkle tree and ethereum objects - ethereum yellow paper walkthrough (2/7),” Available at https://www.lucassaldanha.com/ethereum-yellow-paper-walkthrough-2/ (2018/12/11).

[10] W. Stallings, “Finite field,” in *CRYPTOGRAPHY AND NETWORK SECURITY PRINCIPLES AND PRACTICE*, W. Sutton, Ed. Malaysia: Pearson Education Limited, 2017, pp. 141–157.

[11] C. W. S. R. Mel LI, Sharky Kesa, “Modular arithmetic,” Available at https://brilliant.org/wiki/modular-arithmetic/.

[12] A. Kak, Ed., *PART 2: Modular Arithmetic Theoretical Underpinnings of Modern Cryptography*, ser. Lecture Notes on Computer and Network Security. Avinash Kak, Purdue University, 2022.

[13] S. Nakov, Ed., *Practical Cryptography For Developers*, ser. Lecture Notes on Computer and Network Security. MIT License, 2018.

[14] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Berlin, Heidelberg: Springer-Verlag, 2003.

[15] J. H. Silverman, Ed., *An Introduction to the Theory of Elliptic Curves*, ser. An Introduction to the Theory of Elliptic Curves, 2006.

[16] P. L. Montgomery, "Speeding the pollard and elliptic curve methods of factorization," *Mathematics of Computation*, vol. 48, pp. 243–264, 1987.

[17] C. Research, "Certicom researchsec 2: Recommended elliptic curve domain parameters," Available at http://www.secg.org/sec2-v2.pdf.

[18] R. C. Merkle, "Secure communications over insecure channels," *Commun. ACM*, vol. 21, no. 4, p. 294–299, apr 1978. [Online]. Available: https://doi.org/10.1145/359460.359473

[19] R. O.SRINIVASA and s. Setty, "Efficient mapping methods for elliptic curve cryptosystems," *International Journal of Engineering Science and Technology*, vol. 2, 08 2010.

[20] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *Int. J. Inf. Secur.*, vol. 1, no. 1, p. 36–63, aug 2001. [Online]. Available: https://doi.org/10.1007/s102070100002

[21] P. Shiu, "Foundations of cryptography: basic tools, by oded goldreich. pp. 372. £40. 2001. isbn 0 521 79172 3 (cambridge university press)." *The Mathematical Gazette*, vol. 87, no. 509, p. 381–382, 2003.

[22] B. Schoenmakers, "Commitment schemes," in *Lecture Notes Cryptographic Protocols*.

[23] S. Rubinstein-Salzedo, *Zero-Knowledge Proofs*. Cham: Springer International Publishing, 2018, pp. 173–189. [Online]. Available: https://doi.org/10.1007/978-3-319-94818-8_16

[24] B. Schoenmakers, *Proof of Knowledge Versus Proof of Membership*. Boston, MA: Springer US, 2011, pp. 983–984. [Online]. Available: https://doi.org/10.1007/978-1-4419-5906-5_10

[25] C. Hazay and Y. Lindell, *Sigma Protocols and Efficient Zero-Knowledge1*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 147–175. [Online]. Available: https://doi.org/10.1007/978-3-642-14303-8_6

[26] D. Bernhard, O. Pereira, and B. Warinschi, "How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios," in *Advances in Cryptology – ASIACRYPT 2012*, X. Wang and K. Sako, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 626–643.

[27] C. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, pp. 161–174, 01 1991.

[28] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '92. Berlin, Heidelberg: Springer-Verlag, 1992, p. 89–105.

[29] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *Advances in Cryptology — ASIACRYPT 2001*, C. Boyd, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 552–565.

[30] J. K. Liu and D. S. Wong, "Linkable ring signatures: Security models and new schemes," in *Computational Science and Its Applications – ICCSA 2005*, O. Gervasi, M. L. Gavrilova, V. Kumar, A. Laganà, H. P. Lee, Y. Mun, D. Taniar, and C. J. K. Tan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 614–623.

[31] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, p. 612–613, nov 1979. [Online]. Available: https://doi.org/10.1145/359168.359176

[32] H. Lipmaa, "Lecture 9: Secret Sharing, Threshold Cryptography, MPC," 2004, uRL: http://www.tcs.hut.fi/Studies/T-79.159/2004/slides/L9.pdf. Last visited on 2022/03/23.

[33] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, 1987, pp. 427–438.

[34] B. Schoenmakers, "Threshold cryptography," in *Lecture Notes Cryptographic Protocols*.

[35] web3.js, "web3.js - ethereum javascript api," 2022. [Online]. Available: https://web3js.readthedocs.io/en/v1.7.1/

[36] MetaMask, "Introduction," 2022. [Online]. Available: https://docs.metamask.io/guide/#why-metamask

[37] T. Suite, "Overview," 2022. [Online]. Available: https://trufflesuite.com/docs/ganache/index.html

[38] B. Schoenmakers, "A simple publicly verifiable secret sharing scheme and its application to electronic voting," in *Advances in Cryptology — CRYPTO' 99*, M. Wiener, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 148–164.

[39] mariocao, "elliptic-curve-solidity," 2020. [Online]. Available: https://github.com/witnet/elliptic-curve-solidity

[40] T. Wu, “Rsa and ecc in javascript,” 2009. [Online]. Available: http://www-cs-students.stanford.edu/~tjw/jsbn/

Appendices

Appendix A

Project Schedule

The figure below is a Gantt chart representing our project schedule.

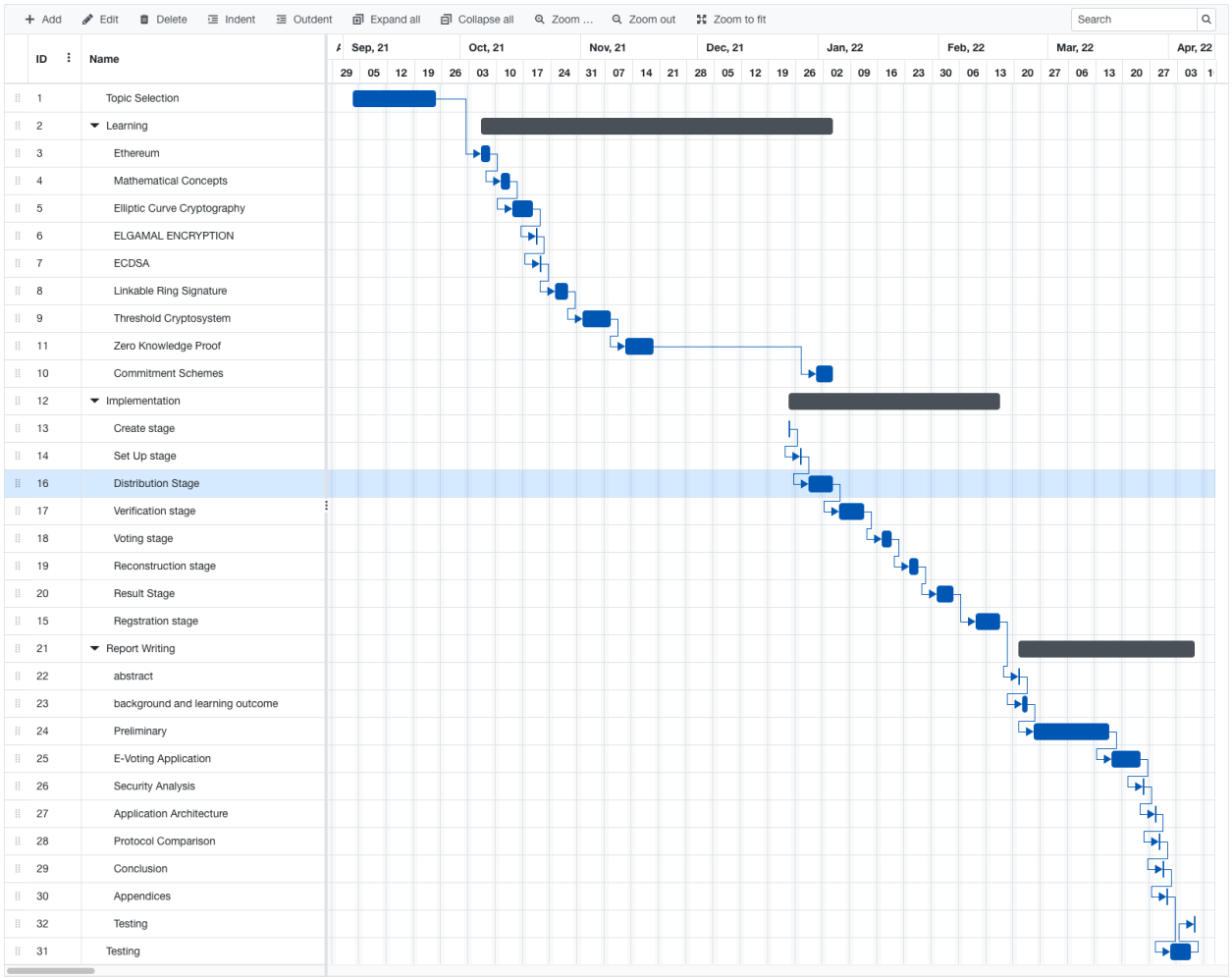


Figure A.1: Project Schedule

Appendix B

Set Up Guide

The following are the steps for running our application.

B.1 Download Source Code

First, download our source code from GitHub: `https://github.com/timyiu478/e-voting`.

B.2 Download and Install Node.js

Then, download node.js from this webpage: `https://nodejs.org/zh-tw/download/`.

B.3 Install Packages, truffle and ganache-cli

After you have installed the node.js, go to the source code directory, then install the packages, truffle, and ganache-cli by the following commands:

```
npm install
npm install -g truffle
npm install -g ganache-cli
```

B.4 Run the Ethereum Private network

Then, run the Ethereum Private network by this command:

```
ganache -i "1337" -l "3000000000000000" -p "8545"
```

B.5 Deploy Smart Contract

Then, deploy the smart contracts to the private network by this command:

```
truffle migrate --reset
```

B.6 Run the Web Application

Then, run the web application by this command where the URL of our application is `127.0.0.1:3000`:

```
npm start
```

B.7 Install MetaMask for browser

Then, add MetaMask into the browser from this webpage: `https://metamask.io/download/`.

B.8 Add a new network in MetaMask

Then, add a new network in MetaMask according to Figure B.1 and figure B.2.

B.9 Import Account in MetaMask

To import an account in MetaMask, we first copy a private key from ganache-cli [Figure B.3]. Then, open MetaMask and click the top right corner icon [Figure B.4]. Then click import account, paste the copied private key into the input, and click import [Figure B.5].

Figure B.1: Click add network.

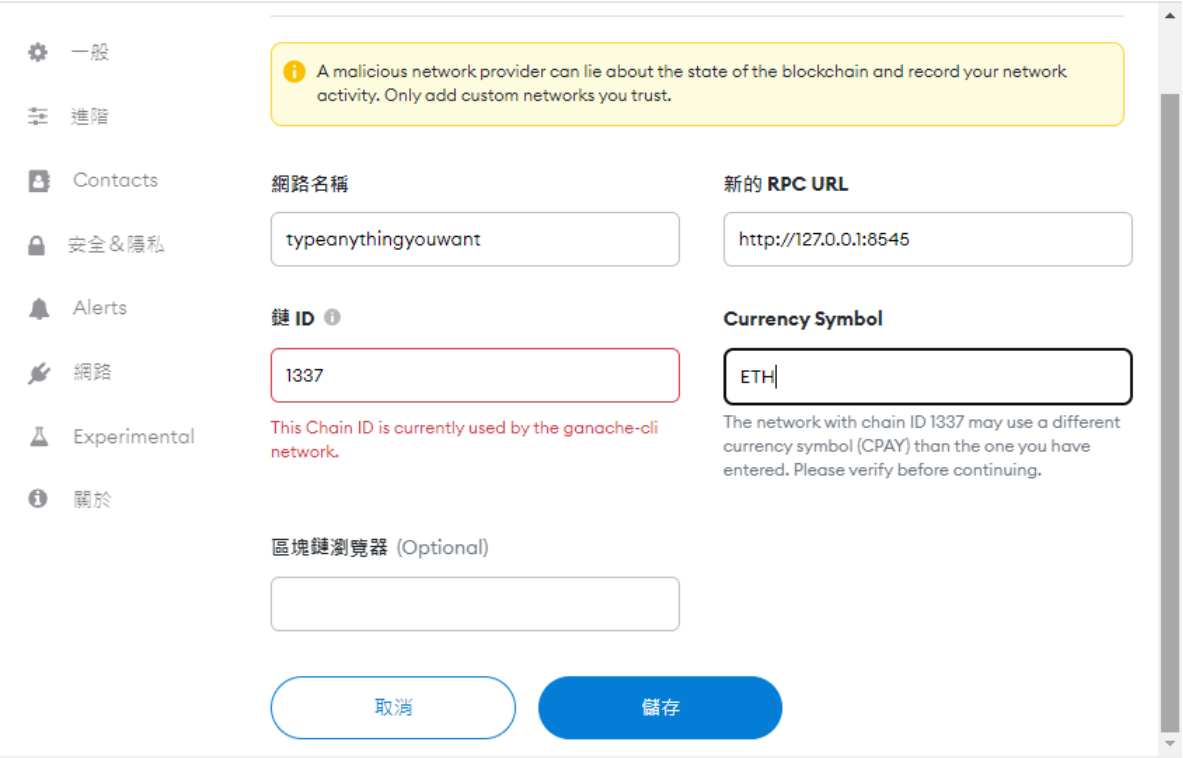


Figure B.2: URL: http://127.0.0.1:8545;chain ID: 1337;Currency Symbol:ETH. Click save after filling in the information

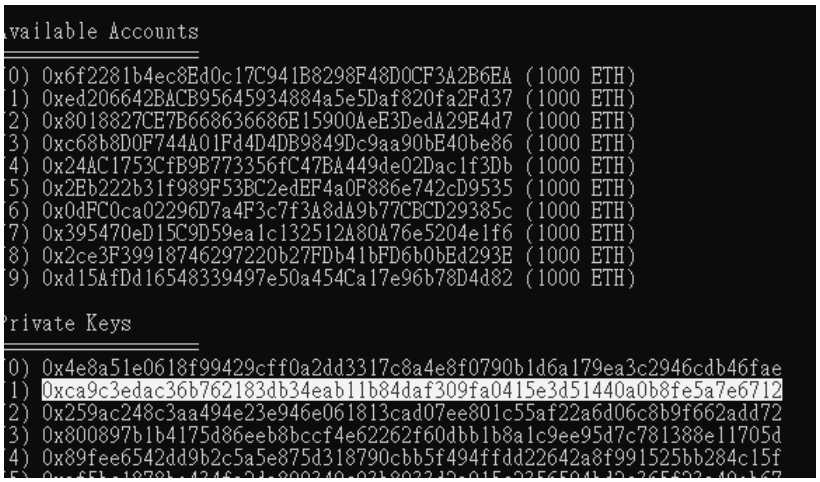


Figure B.3: copy an private key from ganache-cli.

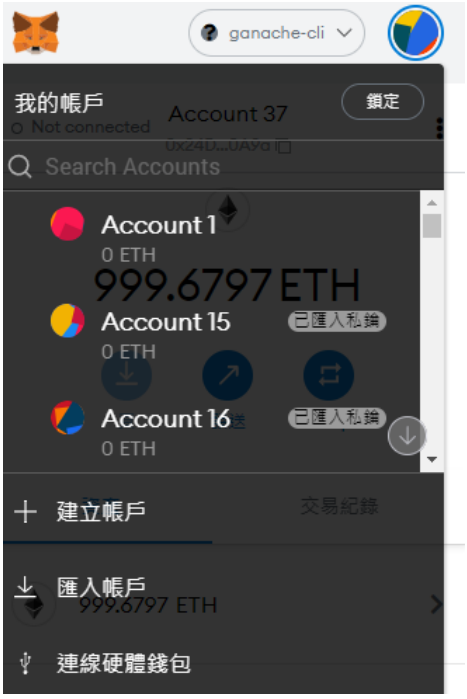


Figure B.4: MetaMask account panel

Figure B.5: MetaMask import account panel.

Appendix C

Capture Screens

C.1 Participants Public Keys

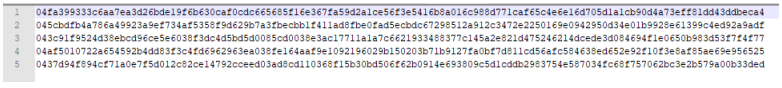


Figure C.1: Participants Public Keys:test.txt. Each line represents one public key

C.2 Participants KeyPairs

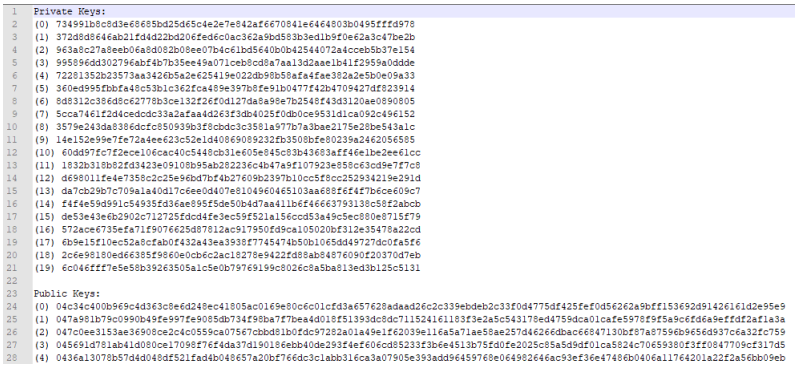


Figure C.2: Participants KeyPairs:keyPairs.txt. (1) private key is corresponding to (1) public key.

C.3 New Election

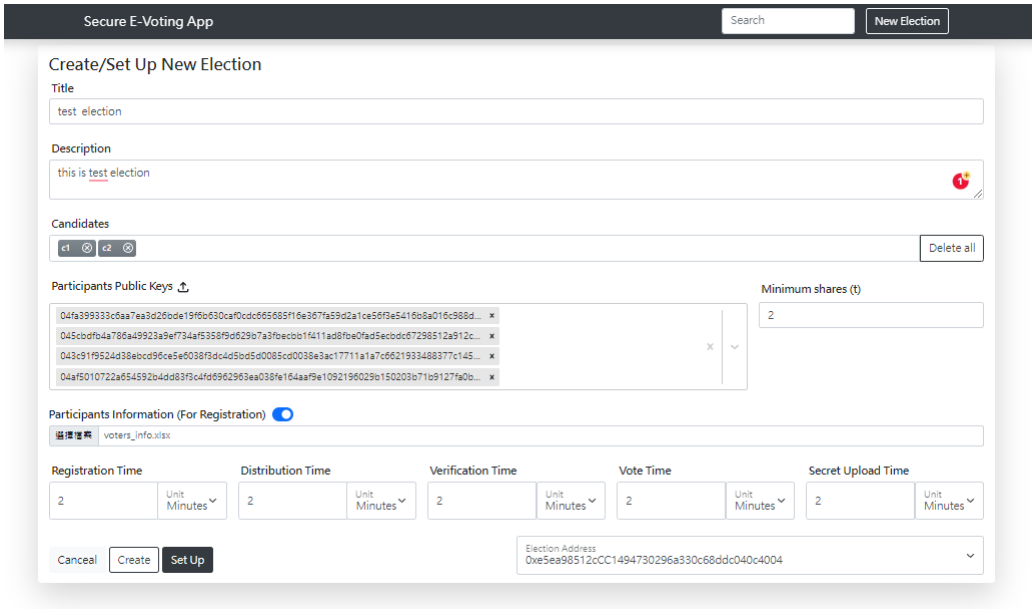


Figure C.3: New Election User Interface

C.4 Register Information

	A	B	C
1	**Name**	**Birth Date (yyyy-mm-dd)**	**Identity Number**
2	Tom Chan	1990-10-20	A123456
3	Ken Wong	1995-10-21	Z654321
4	Mark Lee	1999-01-22	F098765

Figure C.4: Register Information: voters_info.xlsx. Each row represents one register's information.

C.5 Application Home Page

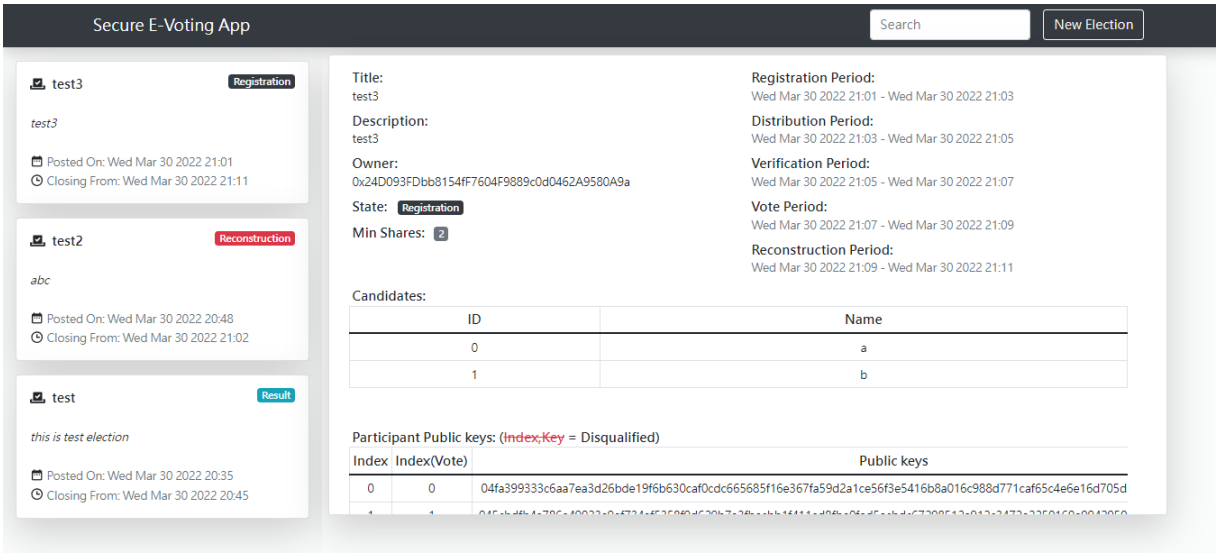


Figure C.5: Application Home Page. On the left-hand side is the election list. Users can one of them then the details of that election will show on right-hand side.

C.6 Distribution Stage

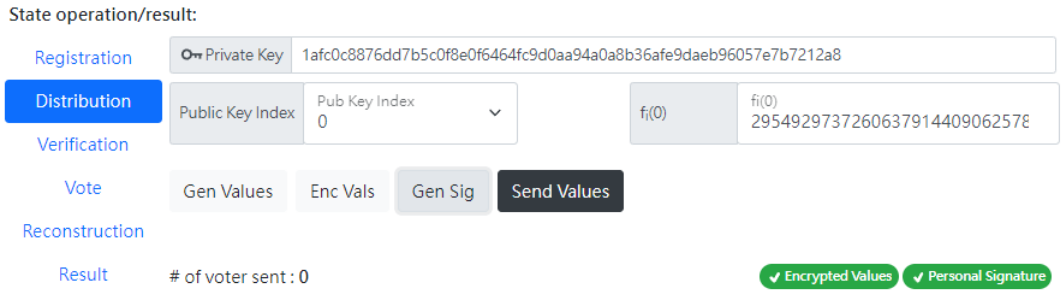


Figure C.6: Distribution Stage User Interface.

C.7 Verification Stage

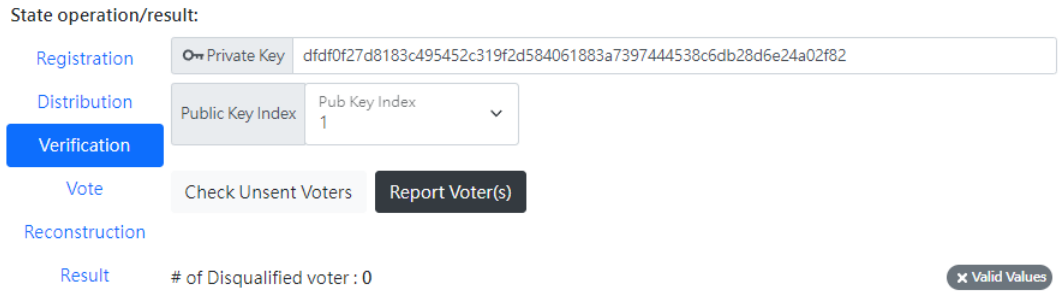


Figure C.7: Verification Stage User Interface.

C.8 Vote Stage

C.8.1 Compute Vote Public Key and Other Parameters

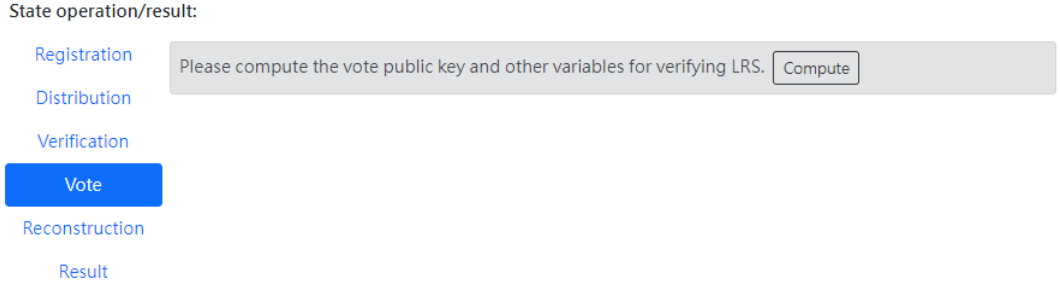


Figure C.8: Vote Stage. Action: compute vote public key and linkable ring signature parameters.

C.8.2 Vote

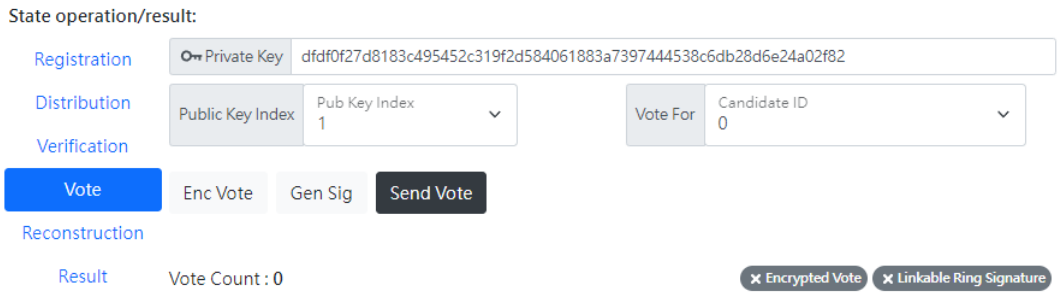


Figure C.9: Vote Stage. Action: vote candidate.

C.9 Reconstruction Stage

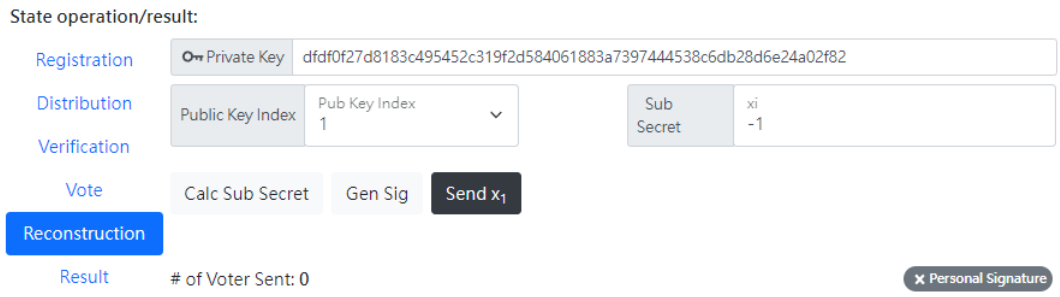


Figure C.10: Reconstruction Stage User Interface.

C.10 Result Stage

C.10.1 Decrypt and Tally the Ballots.

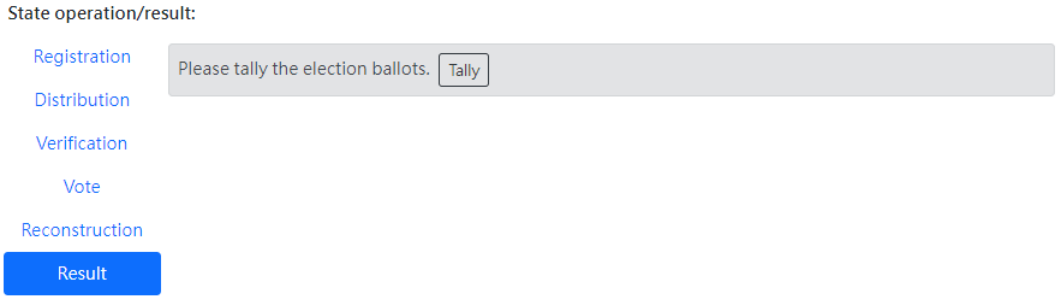


Figure C.11: Result Stage. Action: decrypt and tally the ballots.

C.10.2 Election Result

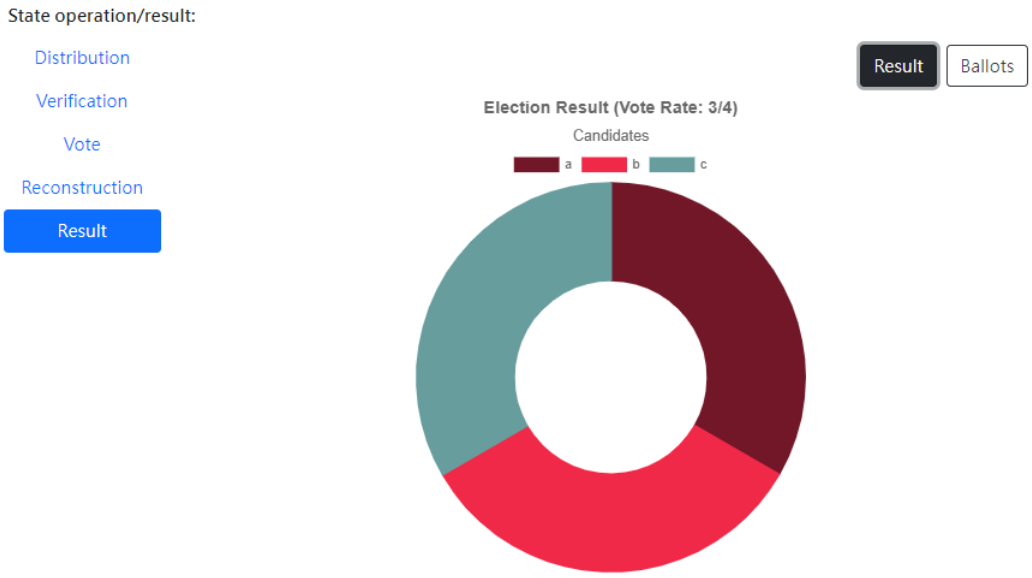


Figure C.12: Result Stage. Action: view election result.

C.10.3 Verify Ballots

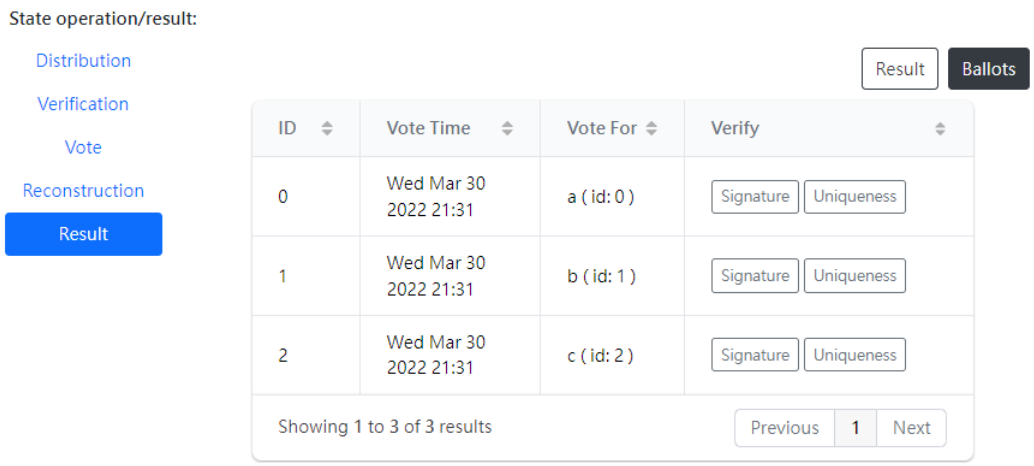


Figure C.13: Result Stage. Action: verify ballots.

C.11 An Example of Error

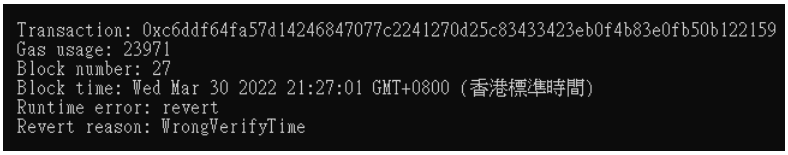


Figure C.14: Error: Wrong verification Time.

C.12 Disqualified Participants

Participant Public keys: (~~Index,Key~~ = Disqualified)

Index	Index(Vote)	Public keys
~~0~~		~~045cbdfb4a786a49923a9ef734af5358f9d629b7a3fbecbb1f411ad8fbe0fad5ecbdc67298512a912c3472e2250169e0942950~~
1	0	043c91f9524d38ebcd96ce5e6038f3dc4d5bd5d0085cd0038e3ac17711a1a7c66219334883 77c145a2e821d475246214dcede
2	1	04af5010722a654592b4dd83f3c4fd6962963ea038fe164aaf9e1092196029b150203b71b9127fa0bf7d811cd56afc584638ec

Figure C.15: Public key list with disqualified participants.

Appendix D

Computer Specification

Component	Specification
Operating System	Microsoft Windows 10 Home
CPU	Intel Core i5-4460 @ 3.20GHz
GPU	NVIDIA GeForce GT 720
RAM	DDR3 16GB
Motherboard	acer aspire xc-705

Table D.1: Specification of the computer used for testing the application

Appendix E

Packages and Libraries

- Most of the packages/dependencies used in our project is listed in `/package.json`.
- The `src/contracts/EllipticCurve.sol` is a library to perform elliptic curve operations written in Solidity [39].
- In addition, the codes in `src/components/linkable_ring_signature/lib` are the codes written in JavaScript by Tom Wu that the codes allow us the handle big integer, generate pseudo-random integers, and compute elliptic curve arithmetic operations [40].