

北京理工大学

本科生毕业设计(论文)

基于以太坊私有链的投票系统设计与实现

Design and Implementation of Voting System Based on
Ethereum Private Blockchain

学 院 : 计算机学院

专 业 : 软件工程

学 生 姓 名 : 马睿

学 号 : 1820181084

指 导 教 师 : 孙建伟

校外指导教师: 无

2022 年 5 月 29 日

原创性声明

本人郑重声明：所呈交的毕业设计（论文），是本人在指导老师的指导下独立进行研究所取得的成果。除文中已经注明引用的内容外，本文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。

特此申明。

本人签名: _____ 日期: _____ 年 _____ 月 _____ 日

关于使用授权的声明

本人完全了解北京理工大学有关保管、使用毕业设计（论文）的规定，其中包括：①学校有权保管、并向有关部门送交本毕业设计（论文）的原件与复印件；②学校可以采用影印、缩印或其它复制手段复制并保存本毕业设计（论文）；③学校可允许本毕业设计（论文）被查阅或借阅；④学校可以学术交流为目的，复制赠送和交换本毕业设计（论文）；⑤学校可以公布本毕业设计（论文）的全部或部分内容。

本人签名: _____ 日期: _____ 年 _____ 月 _____ 日

指导老师签名: _____ 日期: _____ 年 _____ 月 _____ 日

基于以太坊私有链投票系统设计与实现

摘 要

随着物联网 (IoT) 的发展,投票变得越来越数字化。电子投票系统的好处是巨大的,从比传统纸质投票更方便,到更具成本效益和环保。然而,目前大多数电子投票系统都是基于中心服务器架构的,存在服务器管理人员篡改投票过程和投票结果的隐患。本文提出了一种基于以太坊私有区块链的新投票系统,通过分布式账本记录投票过程和投票结果,实现投票过程和结果无法篡改、真实可信。

该系统使用以太坊协议在私有区块链上创建和部署智能合约,以确保流程的去中心化和透明性。然而,由于以太坊的数据流方式使用每个人都可以见证的交易,这样的系统用户数据和投票选择的隐私性为零。针对这个缺陷,本文使用盲签名结合以太坊协议,以实现隐私保护,使其适用于董事会投票、学校选举,甚至社区公投。

关键词: 电子投票系统; 区块链; 以太坊; 智能合约; 权力下放; 私有区块链; 盲签名

Design and Implementation of Voting System Based on Ethereum Private Blockchain

Abstract

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

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

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

目 录

摘 要	I
Abstract	II
第 1 章 绪论	1
1.1 研究背景及意义	1
1.2 研究概述	1
1.3 论文结构	2
第 2 章 以太坊创建与部署	3
2.1 权威证明	3
2.2 GETH	3
2.3 创建私有 PoA 网络	4
2.4 网络运行	5
第 3 章 电子投票系统需求分析	8
3.1 系统总体要求	8
3.2 盲签名	8
3.3 投票程序	8
3.3.1 预注册期	9
3.3.2 注册期	9
3.3.3 投票期	9
3.3.4 理货结果期	10
第 4 章 系统功能设计	11
4.1 WEB2 和 WEB3 对比	11
4.2 智能合约设计	12
4.3 系统流程	13
4.3.1 操作员预注册期流程图	14
4.3.2 操作员注册期流程图	15
4.3.3 选民注册期流程图	16
4.3.4 选民投票期流程图	17

4.3.5 操作员理货结果期流程图	18
4.3.6 操作员理货结果期流程图	19
第 5 章 系统实现	20
5.1 VOTINGSYSTEM 智能合约文档	20
5.1.1 结构	20
5.1.2 事件	22
5.1.3 构造函数	23
5.1.4 添加选民	24
5.1.5 获得选票	24
5.1.6 获得选择	25
5.1.7 获取注册时间	25
5.1.8 获取投票时间	25
5.1.9 注册	25
5.1.10 设置选民签名的盲 UUID	27
5.1.11 计票	27
5.1.12 投票	28
5.1.13 映射	28
5.2 VOTINGSYSTEMFACTORY 智能合约文档	29
5.2.1 创建	29
5.3 盲签名 ^[18] 实现	30
5.3.1 新密钥对	30
5.3.2 新的请求参数	31
5.3.3 盲化消息	31
5.3.4 签署盲目的消息	32
5.3.5 Unblind signed blinded message	32
5.3.6 验证	33
5.4 NEXT.JS 网络应用程序	33
5.4.1 创建投票	33
5.4.2 添加选民	34

5.4.3 注册	36
5.4.4 签名	36
5.4.5 更新密钥	37
5.4.6 投票	37
5.4.7 计票	37
5.4.8 验证选票	38
5.5 签名者命令行界面应用程序	38
第 6 章 部署与测试	40
6.1 系统部署	40
6.1.1 开发环境	40
6.1.2 系统部署架构	40
6.2 系统模块功能测试	41
6.2.1 连接 MetaMask 钱包	41
6.2.2 创建新投票	42
6.2.3 添加选民	44
6.2.4 运行 signer-cli	45
6.2.5 注册	46
6.2.6 更新密钥	49
6.2.7 投票	49
6.2.8 计票	50
6.2.9 验证投票	52
结 论	54
参考文献	55
致 谢	57

第 1 章 绪论

本章比较了纸质投票系统和电子投票系统，并展示了它们的优缺点。本章还介绍了以前对电子投票系统和基于区块链的投票系统类型的研究。最后，阐述了论文的结构和系统的构建方式。

1.1 研究背景及意义

无论何时何地，人们的意见总是很重要。然而，随着技术的发展，获得这种意见的过程发生了变化。

几千年来，典型的投票过程包括将选票投到一个盒子里。随着时间的推移，这个过程并没有太大的变化。当前的离线纸质投票方法与两千年前罗马人和希腊人使用的^[1]没有太大区别。然而，这样的投票是低效的、时间、空间和资源消耗的^[2]。此外，选民无法验证他的投票是否被统计，因为没有办法在不取消匿名的情况下将选票追溯到选民。

纸质投票解决方案是电子投票（e-voting）^[3]。电子投票解决了许多问题。然而，基于 Web2 的投票系统并不透明和集中。

Web2 的解决方案是区块链^[4]技术和新的 Web3。随着以太坊的创建^[5]，一种新的系统方式成为可能。使用智能合约^[6]在以太坊虚拟机（EVM）上执行代码的去中心化、透明系统。

区块链投票系统有几个缺陷^[9]，具体取决于它们的构建和操作方式。但是，如果构建得当，这样的系统可以比 Web2 系统更加透明和公平。

1.2 研究概述

自信息时代开始以来，人们一直在努力改进和开发人们投票的新方式。

David Chaum 在这项研究中发挥了重要作用。1979 年，他是第一个提出^[7]比特币中发现的区块链协议的所有元素（工作证明除外）的人。后来，Chaum 于 1981 年开发了第一个电子投票系统^[8]，使用无法追踪的电子邮件、回信地址和数字化名。他也是 1983 年引入盲签名^[9]的人，后来在他的职业生涯中开发了几种不同类型的电子投票系统^{[10][11]}。

自比特币发布以来，区块链研究蓬勃发展，催生了以太坊协议及其 EVM。借助 EVM，可以使用智能合约在区块链上执行代码，从而使投票等活动更加透明和去中心化。这导致开发人员研究不同类型的基于区块链的电子投票系统。

有许多不同的提议。简单的是投票智能合约的直接实施，没有任何选民隐私或匿名性。然后，尽管以太坊协议具有数据可见性的性质，一些系统仍提供隐私和匿名性。此类系统使用零知识证明^{[12][17]}、智能代理^[15]、盲签名^{[13][14][16]}或其他类型的密码学来保护选民的身份和选择。

本文提出了一种基于私有以太坊区块链的投票系统实现，该系统使用智能合约结合椭圆曲线盲签名方案^[18]。

1.3 论文结构

本论文旨在利用以太坊协议和盲签名，创建一个可验证的投票系统。

本文分为六个部分，组织如下：

第 1 章：绪论。本章解释了投票系统的类型。建立对基于区块链的电子投票系统的研究并介绍论文结构。

第 2 章：私有区块链网络。本章确定系统将使用的专用网络类型。此外，本章还解释了如何设置网络及其参数并运行它。

第 3 章：系统理论分析。建立投票系统应满足的要求。解释盲签名及其使用原因。明确投票程序的时间段。

第 4 章：系统整体设计。这里描述了系统中使用的智能合约设计。接下来解释了 Web2 和 Web3 系统的区别，最后解释了系统的流程。

第 5 章：系统实现。本章介绍了智能合约的大量文档。解释签名者 CLI 应用程序的用途和用途。它还显示了前端用于与合约交互的代码。

第 6 章：部署与测试。解释系统的架构并显示在系统上执行的测试结果。

第 2 章 以太坊创建与部署

电子投票系统必须是安全的，并且不惜一切代价保护过程的完整性。在基于私有区块链的系统中，最好的决策是使用权威证明（PoA）区块链，其中网络中的所有节点都是可信的，并且受到攻击的危险最小化。

本章解释了为什么系统使用 PoA 私有以太坊网络。这里还展示了如何设置这样的网络、它的选项以及最后如何运行它。

2.1 权威证明

在 2017 年 2 月 24 日对 Ropsten 测试网发起拒绝服务（DoS）攻击后，目前以太坊的联合创始人兼首席技术官 Gavin Wood 提出了权威证明。

这次攻击证明了工作量证明共识算法不适合小型网络，因为小型网络不会以任何经济回报激励他们的验证者。投票系统将使用相同类型的网络。如果没有激励措施，将没有足够多的诚实矿工提供哈希率，从而导致 DoS 攻击和 51%攻击，其中不良行为者用自己的链分叉。

PoA 验证器节点操作员可以是已知且受信任的人，这意味着每次挖掘新块时，操作员都会将其声誉置于风险之中。考虑到这一点，PoA 中的验证者数量非常少，放弃了去中心化，但通过消除对高端采矿硬件的需求和通过消除未知验证者的安全性来获得可持续性。

2.2 Geth

Go Ethereum(Geth)是用 Go 的编程语言编写的 Ethereum 协议的开源实现。Geth 为用户提供对以太坊协议的贡献，并使用以太坊协议连接到以太坊的主网或任何其他网络。它还允许开发人员创建专用网络并开发在以太坊虚拟机（EVM）上运行的去中心化应用程序（dApp）。

Geth 提供了两种共识算法：Ethash（PoW）和 Clique（PoA）。Ethash 与以太坊主网本身使用的算法相同。然而，PoA Clique 算法更适合电子投票系统将在其上运行的区块链。

每个区块链网络都必须从某个地方开始。一个新的区块链需要一个创世区块。创世区块中包含网络规范：链 ID、共识算法、区块周期和气体限制。

[illegible]

```
"c5e6c9f8893cb397c3a49ad2257cf6543e55afbf": {  
  "balance": "0x2000000000000000000000000000000000000000000000000000000000000000"  
},  
},  
"number": "0x0",  
"gasUsed": "0x0",  
"parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",  
"baseFeePerGas": null  
}
```

从创世区块中提取的重要信息是：

A. 链 ID 是 1234。Truffle 需要此值才能将智能合约部署到正确的网络。此外，此值是在任何 Web 浏览器钱包（例如 MetaMask）中网络的指标；

B. Clique 算法挖出的区块周期为 0。新区块将不是在预定的时间段被挖掘，而是在提交新交易时被挖掘。这种方式比没有任何理由地无缘无故地挖掘空块要好；

C. extraData 包含签名节点的地址：

a) node1: 0xc5e6c9f8893cb397c3a49ad2257cf6543e55afbf

b) node2: 0xb022517f75b90a22e6e7da738bb9a98f1342c24b

D. 网络中每笔交易使用的 gas 为 0。因此，在 EOA 内无需任何 ETH 即可发送交易。无需预先筹资。

创世块设置完毕后，创建新网络的唯一方法是使用 Geth 对其进行初始化：

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

2.4 网络运行

运行区块链网络意味着运行验证交易的节点。在这个网络的情况下，只有两个节点用于演示目的，但即使在有更多节点的情况下，过程也是一样的。用于启动 node1 的 BASH 脚本如下：

```
#!/bin/bash  
geth --networkid 1234 --nousb --datadir=$pwd --syncmode full --port 30310 --miner.gasprice 0  
--miner.gastarget 470000000000 --http --http.addr '127.0.0.1' --http.port 8545 --http.corsdomain "*" --http.vhosts "*" --http.api admin,eth,miner,net,txpool,personal,web3 --ws --ws.port 8547 --ws.api admin,eth,miner,net,txpool,personal,web3 --mine --allow-insecure-unlock --unlock "0xC5e6c9f8893Cb397C3A49Ad2257cF6543E55aFbF" --password password.txt
```

上述 Geth 命令的选项如表 2-1 所示。

表 2- 1 Geth 命令

选项	选项 说明
--networkid 1234	此处指定的网络 ID 是为了确保 Truffle 等框架编译部署工件时内部具有正确的网络 ID。
--nousb	禁用对 USB 硬件钱包的监控和管理。
--datadir=\$pwd	数据库和密钥库的目录。它设置为 \$pwd，这意味着启动节点的脚本必须从节点的目录中执行。
--syncmode 'full'	将节点与其余节点完全同步。
--port 30310	为 Geth 进程分配一个端口。
--miner.gasprice 0	Gas 价格。在私有网络中，无需支付 gas 费。
--miner.gastarget limit	Gas 目标/限制。确切的值是来自创世块的气体限制，但实际上，气体限制是动态的并且基于前一个块。
--http	启用 HTTP-RPC 服务器。
--http.addr '127.0.0.1'	HTTP-RPC 服务器监听接口。
--http.port 8545	HTTP-RPC 服务器监听端口。
--http.corsdomain "*"	允许从网页访问 API。
--http.vhosts "*"	可以接受来自任何主机名的请求。
--http.api	启用通过 RPC 调用使用的有用 API。
--ws	启用 WS-RPC 服务器。
--ws.port 8547	WS-RPC 服务器监听端口。
--ws.api	启用通过 RPC 调用使用的有用 API。
--mine	启用挖矿。
--allow-insecure-unlock	当与账户相关的 RPC 被 HTTP 暴露时，允许不安全的账户解锁。
--unlock address	解锁 node1 Keystore 文件。
--password file	包含密钥库文件密码的文件。

北京理工大学本科生毕业设计（论文）

在所有节点启动后，私有以太坊网络启动并运行，准备好在其上部署智能合约（图 2-1）。

```

[05-08:15:40:12.383] Loaded most recent local full block
[05-08:15:40:12.383] Loaded most recent local fast block
[05-08:15:40:12.383] Loaded local transaction journal
[05-08:15:40:12.383] Regenerated local Transaction Journal
[05-08:15:40:12.383] Gasprice oracle is ignoring threshold set threshold=2
[05-08:15:40:12.383] Unclean shutdown detected
[05-08:15:40:12.383] Unclean shutdown detected
[05-08:15:40:12.383] Unclean shutdown detected
[05-08:15:40:12.383] Unclean shutdown detected
[05-08:15:40:12.383] Starting peer-to-peer node
[05-08:15:40:12.383] New local node record
[05-08:15:40:12.383] Started P2P networking
[05-08:15:40:12.383] IPC endpoint opened
[05-08:15:40:12.383] HTTP server started
[05-08:15:40:12.383] Websocket enabled
[05-08:15:40:12.383] New local node record
[05-08:15:40:12.383] Unlocked account
[05-08:15:40:12.383] Transaction pool price threshold updated
[05-08:15:40:12.383] Updated mining threads
[05-08:15:40:12.383] Transaction pool price threshold updated
[05-08:15:40:12.383] Ethereum automatically configured
[05-08:15:40:12.383] Commit new sealing work
[05-08:15:40:12.383] Block sealing failed
[05-08:15:40:12.383] Commit new sealing work
[05-08:15:40:12.383] Block sealing failed
[05-08:15:40:12.383] Looking for peers
[05-08:15:40:12.383] Snapshot extension registration failed
[05-08:15:40:12.383] Looking for peers

[05-08:15:40:22.130] Opened ancient database
[05-08:15:40:22.132] Initialized chain configuration
[05-08:15:40:22.132] Initialized chain configuration
[05-08:15:40:22.132] Initializing Ethereum protocol
[05-08:15:40:22.132] Loaded most recent local full block
[05-08:15:40:22.132] Loaded most recent local fast block
[05-08:15:40:22.132] Loaded local transaction journal
[05-08:15:40:22.132] Regenerated local Transaction Journal
[05-08:15:40:22.132] Gasprice oracle is ignoring threshold set threshold=2
[05-08:15:40:22.132] Unclean shutdown detected
[05-08:15:40:22.132] Unclean shutdown detected
[05-08:15:40:22.132] Unclean shutdown detected
[05-08:15:40:22.132] Starting peer-to-peer node
[05-08:15:40:22.132] New local node record
[05-08:15:40:22.132] Started P2P networking
[05-08:15:40:22.132] IPC endpoint opened
[05-08:15:40:22.132] HTTP server started
[05-08:15:40:22.132] Websocket enabled
[05-08:15:40:22.132] New local node record
[05-08:15:40:22.132] Unlocked account
[05-08:15:40:22.132] Transaction pool price threshold updated
[05-08:15:40:22.132] Updated mining threads
[05-08:15:40:22.132] Transaction pool price threshold updated
[05-08:15:40:22.132] Ethereum automatically configured
[05-08:15:40:22.132] Commit new sealing work
[05-08:15:40:22.132] Block sealing failed
[05-08:15:40:22.132] Commit new sealing work
[05-08:15:40:22.132] Block sealing failed
[05-08:15:40:22.132] Looking for peers
[05-08:15:40:22.132] Snapshot extension registration failed
[05-08:15:40:22.132] Looking for peers
```

图 2-1 运行私有以太坊网络

第 3 章 电子投票系统需求分析

尽管有不同的投票系统，但它们都必须满足确切的要求。然而，在基于区块链的电子投票系统中满足其中一些要求被证明比传统的纸质投票方法更复杂。因此，通过智能联系人和密码学，我打算使系统公平且端到端可验证。

本章解释了投票系统需要满足的要求。此外，这里还介绍了盲签名是什么以及如何使用它们进行盲投票。

3.1 系统总体要求

基于区块链的投票系统应该继承传统投票系统的所有属性，并且只建立在它们之上。以下列表是一个公平的、端到端的可验证投票系统必须具备的要求：

- A) 资格，只有有投票权的人才能参与该过程；
- B) 完整性，系统不得允许不良行为者篡改选民数据或投票本身；
- C) 隐私，没有选民可以追溯到他所做的选择；
- D) 可验证，每个人都可以验证他们的投票，从而验证最终结果的完整性；
- E) 去中心化，多个以太坊节点在任何给定时间运行，以在任何节点发生故障时保持私有区块链同步和活跃。

3.2 盲签名

盲签名确保选民选择的隐私。与数字签名一样，盲签名用于验证消息的真实性。不同之处在于消息是盲目和加密的，这意味着签名数据与原始消息不同。此外，该方法确保签名消息的接收者可以对其进行揭开盲区们和解密，从而获得原始消息。

以上所有内容都使消息的真实性可验证，并且不回顾人的身份。因此，使该方法适用于需要隐私的系统。

3.3 投票程序

由于以太坊协议的性质，每笔交易都是公开的。因此，任何与区块链有联系的人都可以看到它们。所有这些都对透明度有利，但对隐私不利。没有隐私是投票系统的一个重大缺陷。出于这个原因，系统使用盲签名来创建盲投票（图 3-1）。因此，为了应用盲选，系统分为四个阶段。

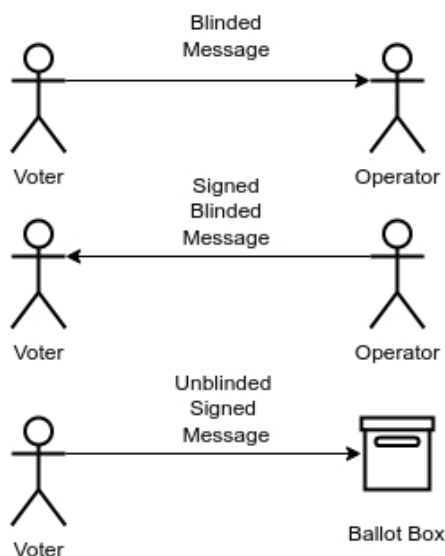


图 3-1 盲投票

3.3.1 预注册期

组织者在此期间通过部署新的智能合约来创建新的投票。合约的构造函数需要注册和投票的时间、投票的名称、选择和运营商的公钥用于盲签名过程。合约创建后，所有这些变量都是不可变的，以防止不良行为者篡改它们。

部署后，运营商插入所有可以参与投票的合格选民的外部所有者帐户（EOA）。为此，组织者必须事先知道选民的 EOA，以防止选民访问其他选民的个人信息，从而保持系统的隐私和完整性。

3.3.2 注册期

所有符合条件的选民都可以通过在本地生成一个通用唯一标识符（UUID）和一组公钥和私钥来注册投票。由于 UUID 的性质，碰撞概率接近于零。所以 UUID 将是选民在注册时盲注的消息，并发送给合约进行签名。

注册行为通知运营商有要签名的盲消息。盲消息签名后，运营商将其上传到合同中。然后选民可以在投票期间使用它。

3.3.3 投票期

所有想要投票的登记选民必须解盲的签盲 UUID。然后，组成一个由非盲签

UUID、UUID 本身、公钥和选择组成的选票。之后，如果选民希望匿名，则选民会使用与用于登记的不同的 EOA 发送选票。UUID 和非盲签 UUID 值无法追溯到原始 EOA。此外，用于对 UUID 进行盲化和解盲的公钥从未共享过，因此无法追踪。通过这种方式投票，选民的身份即使对运营商也是保密的。因此，该系统为投票提供了隐私。

3.3.4 理货结果期

投票期结束后，主办方必须对投票结果进行统计。链下，使用盲签名模式^[18]，操作员使用非盲签 UUID 和来自投选票的公钥来验证其完整性。然后，如果选票有效，则将选票添加到合同中。验证程序完成后，将公布结果。然后选民可以通过提供他们的 UUID 来验证他们的选择。

第 4 章 系统功能设计

Web2 和 Web3 应用程序彼此不同。因此，必须采取不同的设计方法。实施基于时段并划分为选民操作员角色的系统是此类任务的唯一解决方案。

在本章中，首先对这两种类型的系统进行了比较。接下来，解释用于智能合约的设计模式。最后，系统中的每个流程都有一个表示流程图。

4.1 Web2 和 Web3 对比

在 Web2 中，应用程序分为客户端和服务端，但在 Web3 中，情况不再如此。相反，服务器和数据库应用程序在区块链上运行（图 4-1 与图 4-2）。

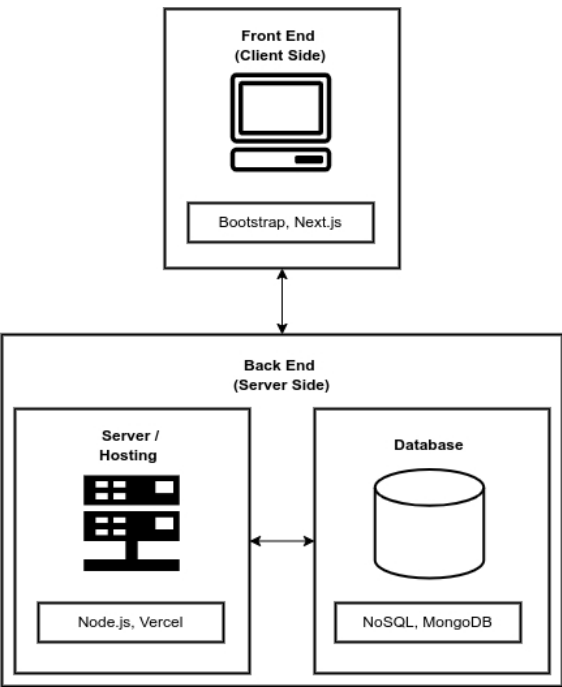


图 4-1 常规系统

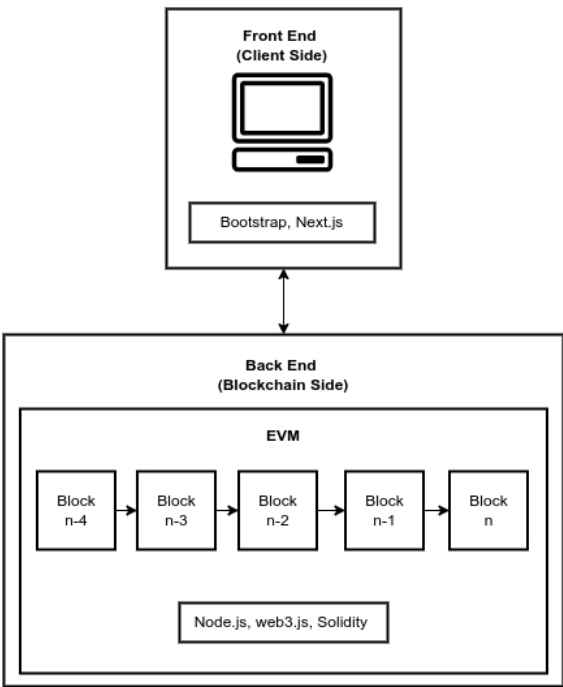


图 4-2 dApp 系统

去中心化应用程序（dApp）使用区块链作为服务器来运行代码和作为数据库来存储数据。以这种方式构建应用程序放弃了速度和成本效率，但获得了区块链保证的流程透明度、用户信任和数据安全性。在投票系统的情况下，与传统的 Web2 应用程序相比，dApp 提供的好处远远超过了缺点。

4.2 智能合约设计

该论文提议的投票系统预计将用于董事会投票、近距离社区公投和其他类似的小规模投票。由此得出，一旦设置了系统，它将多次用于不同的民意调查。它使工厂设计模式最适合进行投票的智能合约。对于每个新的投票实例，运营商将部署一个新的智能合约 `VotingSystem`，使每个投票独立于其他投票，如 UML 所示（图 4-3）。

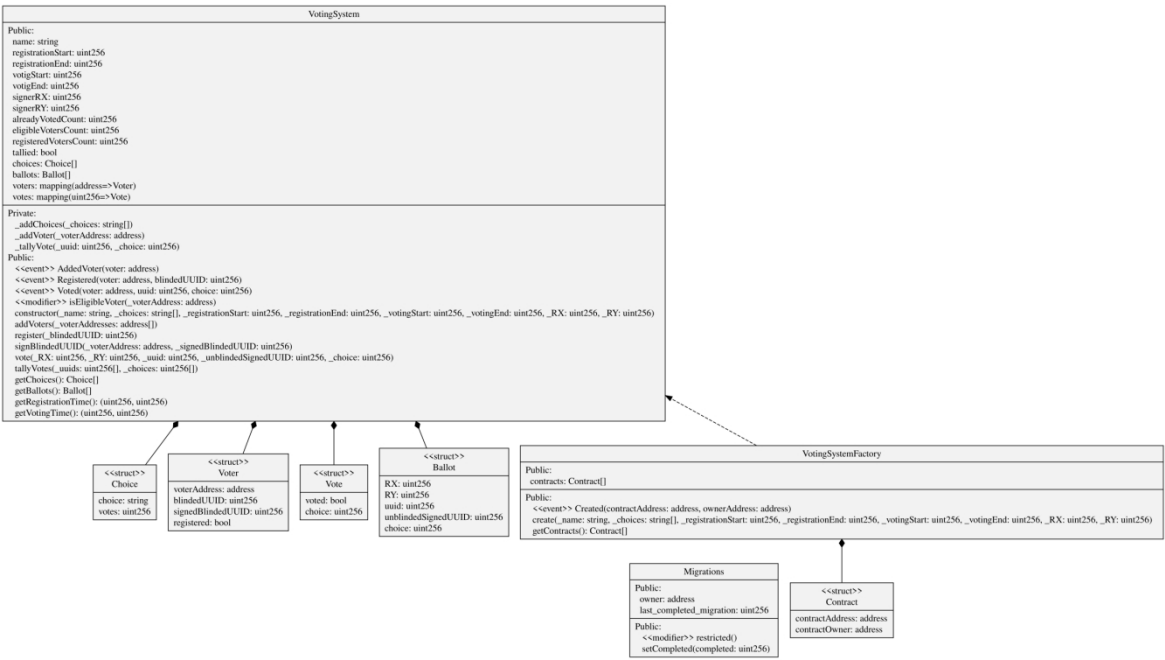


图 4-3 智能合约 UML

在使用这种模式的系统开发的早期阶段出现的一个问题是系统是 100% 去中心化的，具有 PoW 类型的共识算法，并且开放给所有人加入网络并部署新的投票实例。不幸的是，像这样的开放系统允许垃圾邮件。不良行为者可以通过使用无效投票或创建名称与有效投票实例相似的投票实例来使其无法使用，从而使系统容易受到网络钓鱼。

对整个系统进行了重新设计，以解决像这样的一个问题，并在不补偿可用性的情况下修复整体安全性。区块链验证共识算法更改为 PoA，只有受信任的人才能运行区块签名节点。结合使用已经测试过的 OpenZeppelin 的 Ownable 智能合约可以部署合约的代码限制。通过这些改进 VotingSystem 智能合约只能由签名节点部署，

使用本地创建的帐户，放弃自由和去中心化，但大大提高了安全性。

4.3 系统流程

投票是一种时间锁定的过程。因此，系统中的投票流程也是时间锁定的。由于系统中投票者和操作员角色的时间锁定和划分，没有一个单一的从开始到结束的流
程，而是一个跨职能流程（图 4-4）。

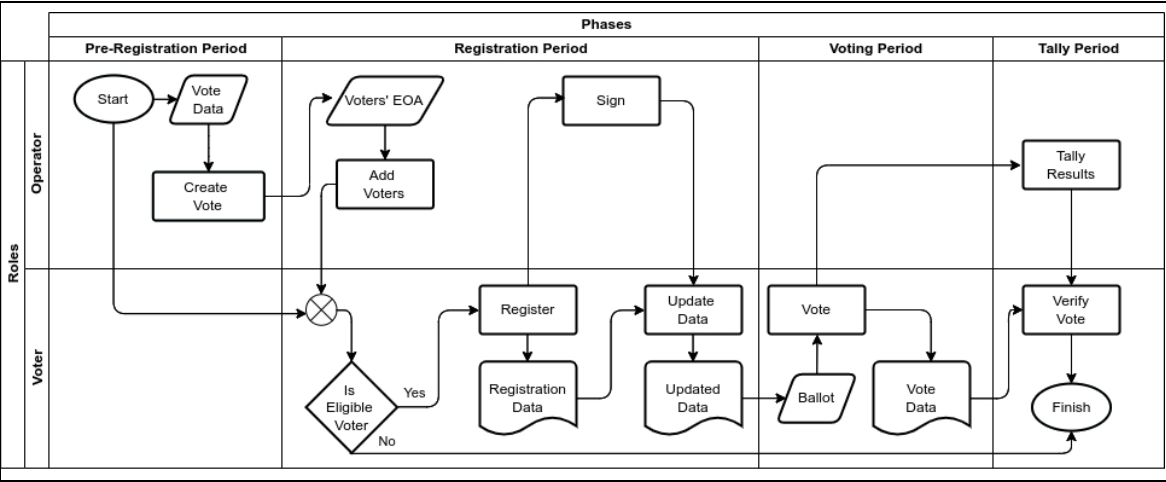


图 4-4 系统跨职能流程图

4.3.1 操作员预注册期流程图

运营商必须通过部署新的 VotingSystem 智能合约来创建投票。

投票所需的数据在创建选民期间进行验证，然后部署新的投票实例。在（图 4-5）中显示了新投票创建的流程。

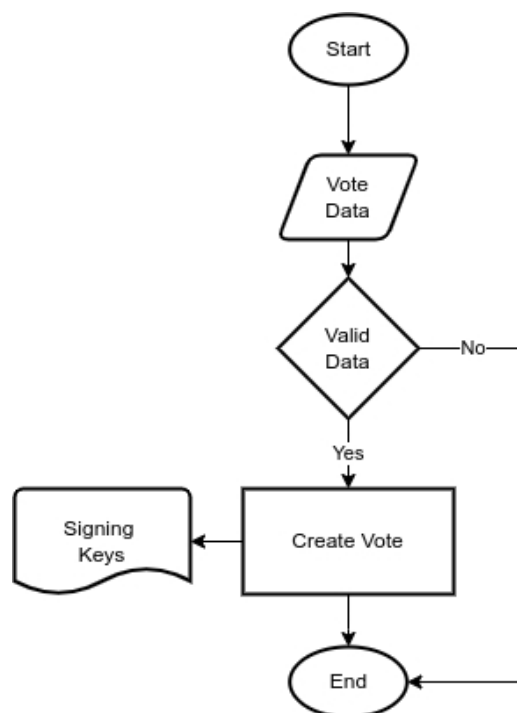


图 4-5 操作员预注册期流程图

4.3.2 操作员注册期流程图

在此期间（图 4-6），操作员起着至关重要的作用。首先，他将符合条件的 EOA 添加到智能合约中。然后，操作员必须确保他对每个请求签名的盲 UUID 进行签名。

整个投票的完整性依赖于运营商签署传入请求的能力。如果在此过程中的某个地方出现问题，则必须尽可能立即修复。如果无法修复签名过程中的严重错误，则运营商应创建一个新的投票实例并重新开始投票。

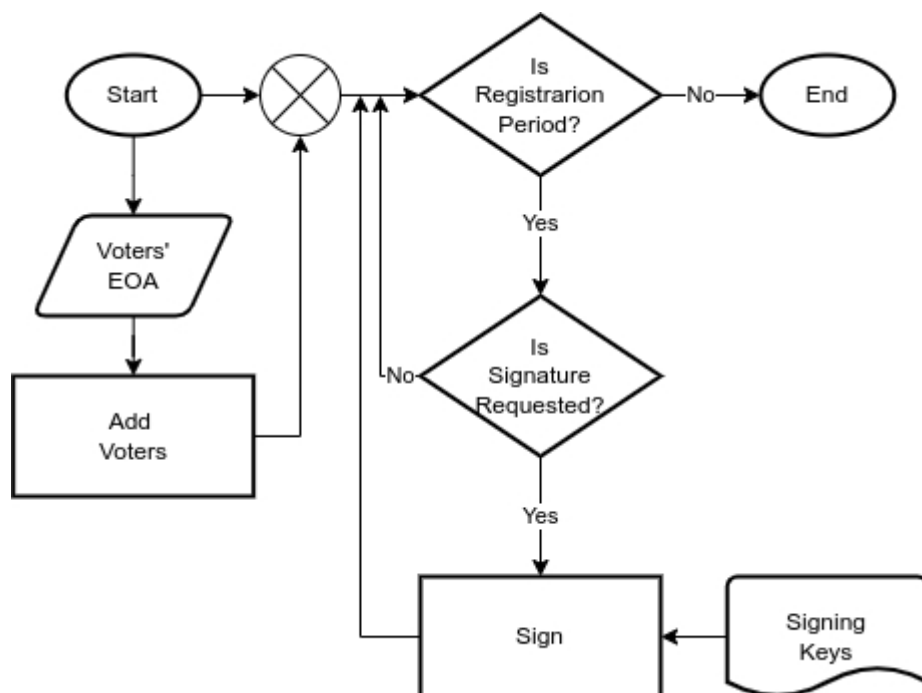


图 4-6 操作员注册期流程图

4.3.3 选民注册期流程图

选民必须在登记期间登记投票（图 4-7）。注册时，投票者在本地生成盲签名模式所需的 UUID 和密钥。然后他将 UUID 盲化 并将其上传到合约中，要求操作员签名。提出请求后，选民必须下载他的密钥。需要下载密钥，因为它们没有存储在其它任何地方以防止其他人看到它们。当操作员签署盲 UUID 时，选民必须上传他的注册密钥并将其更新为投票密钥。为防止去匿名化，选民绝不能与任何人共享任何密钥。

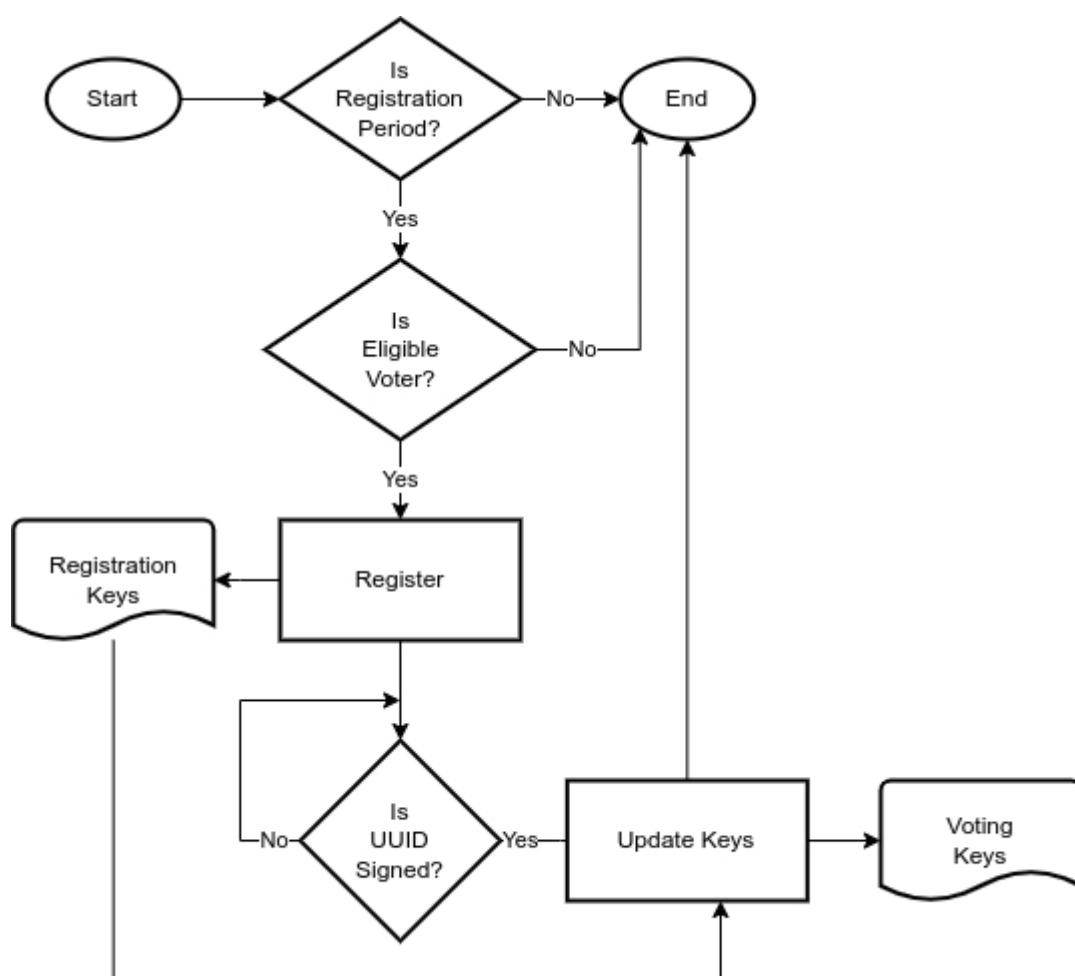


图 4-7 选民注册期流程图

4.3.4 选民投票期流程图

在投票期间，所有登记的选民都有资格投票（图 4-8）。首先，注册选民必须上传他的投票密钥。然后，拥有密钥数据的投票者创建一个发送到智能合约的选票。如果选民想要在统计结果后验证他的投票，他必须下载他的验证密钥。为了防止去匿名化，选民决不能与任何人共享这些密钥。

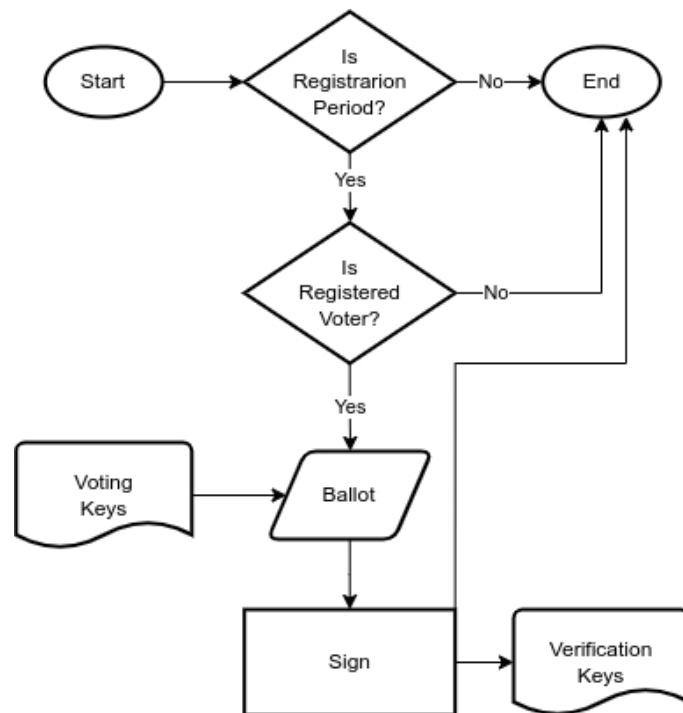


图 4-8 选民投票期流程图

4.3.5 操作员理货结果期流程图

投票期结束后，操作员必须对结果进行统计（图 4-9）。他从合同中获取选票并使用盲签名模式和签名密钥对其进行验证。计数程序完成后，操作员更新最终结果。

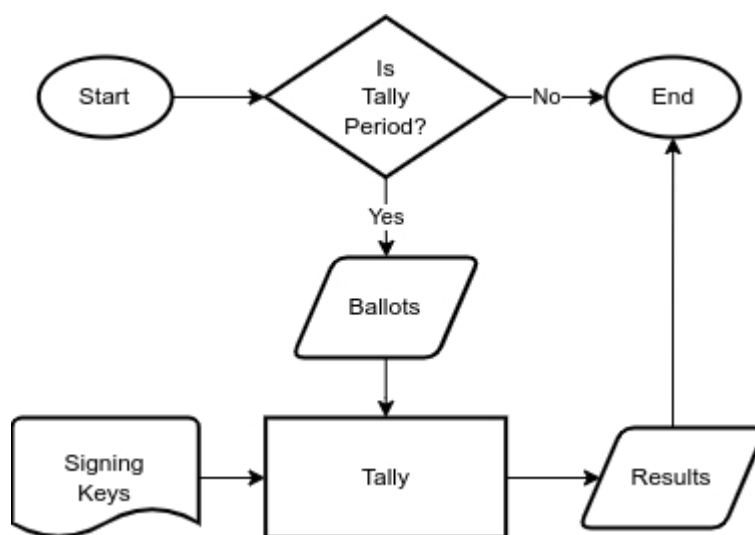


图 4-9 操作员理货结果期流程图

4.3.6 操作员理货结果期流程图

整个投票过程的最后阶段是选民验证他的投票（图 4-10）。选民必须等待运营商统计结果以验证他的投票。然后使用验证密钥，他可以看到他的选择。为了防止去匿名化，如果选民不再使用密钥，就必须删除它们。

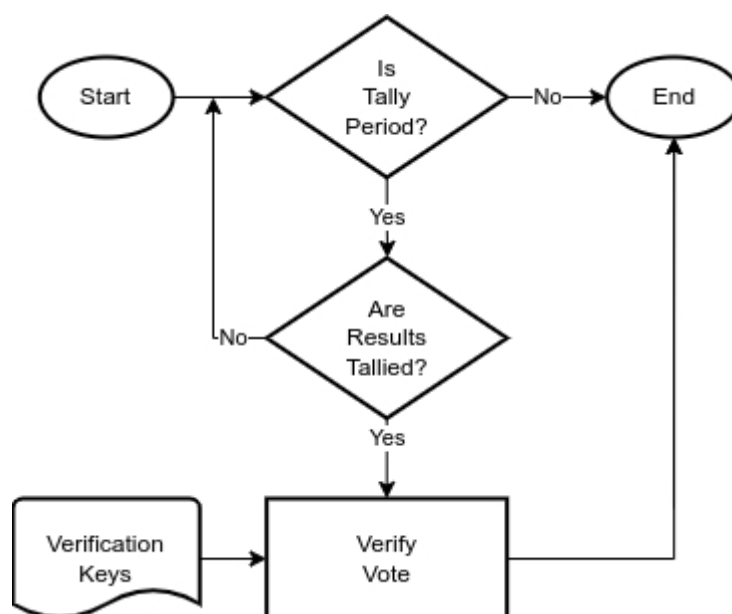


图 4-10 操作员理货结果期流程图

第 5 章 系统实现

该系统的软件分为三个部分。以 Truffle 作为开发框架的 Solidity 智能合约。前端 Next.js Web 应用程序。还有一个 Node.js 命令行界面（CLI）应用程序，它监听和记录事件，并签署选民的盲 UUID。

5.1 VotingSystem 智能合约文档

当工厂 VotingSystemFactory 合约将智能合约 VotingSystem 部署到区块链时，会创建一个新的投票实例。

5.1.1 结构

几个结构数据类型被初始化并在整个合同中使用。这些结构背后的主要原因是为了便于开发人员和代码可读性。

5.1.1.1 选择

名称：Choice

描述：Choice 结构包含单个选择和给定选择在投票过程中获得的票数。

代码：

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

5.1.1.2 选民

名称：Voter

描述：Voter 结构体用于存储选民的数据。由于该系统旨在保护选民隐私，因此与选民相关的唯一数据是他的地址、盲 UUID、签名的盲 UUID 以及他是否已注册投票。

代码：

```
struct Voter {  
    address voterAddress;  
    uint256 blindedUUID;  
    uint256 signedBlindedUUID;  
    bool registered;  
}
```

5.1.1.3 投票

名称：Vote

描述：每个投票的选民都有一个与他的 UUID 相对应的结构投票，该 UUID 存储在智能合约的映射中。此结构的目的是检查给定的 UUID 在计票时是否已投票，并存储选民所做的选择。

代码：

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

5.1.1.4 选票

名称：Ballot

描述：Ballot 对象具有验证投票是否有效所需的所有数据。它包含选民的公钥 X 和 Y 值、UUID、非盲签名 UUID 和选择。在整个注册和盲签名过程中，这些数据从未与任何人共享。因此，使用这样的结构投票可以使投票匿名并确保投票者的隐私。

代码：

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

5.1.2 事件

事件用于记录数据并标记盲 UUID 签名的需要。

5.1.2.1 添加选民

名称：AddedVoter

描述：添加投票者时，会发出事件 AddedVoter。它的目的是让 signer-cli 记录合格的选民。

表 5- 1 AddedVoter 事件的参数

名称	类型	描述
Voter	address	选民地址。

代码：

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

5.1.2.2 注册

名称：Registered

描述：其目的是通知 signer-cli 应用程序对已注册选民的蒙蔽 UUID 进行签名。

表 5- 2 Registered 事件的参数

名称	类型	描述
Voter	address	选民的地址。
blindedUUID	uint256	选民的失明 UUID。

代码：

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

5.1.2.3 投票

名称：Voted

描述：此事件在选民投票时发出。该活动的目的是记录选票。

表 5- 3 Voted 事件的参数

名称	类型	描述
voter	address	选民的地址。
uuid	uint256	选民的 UUID。
choice	uint256	选民的选择。

代码：

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

5.1.3 构造函数

名称：constructor;

描述：初始化投票。一旦设置，所有参数都是不可变的。投票名称不能为空字符串。选择数组不能为空或包含空字符串。注册开始时间戳的值必须小于注册结束时间戳。投票开始时间戳的值必须小于投票结束时间戳但大于或等于注册结束时间。最后，运营商的公钥值不能为零；

表 5- 4 constructor 函数的参数

名称	类型	描述
_name	string	投票的名称。
_choices	string[]	一组选择。
_registrationStart	uint256	注册期的开始。
_registrationEnd	uint256	注册期结束。
_votingStart	uint256	投票期的开始。
_votingEnd	uint256	投票期结束。
_RX	uint256	运营商签名公钥的 X 值。
_RY	uint256	运营商签名公钥的 Y 值。

5.1.3.1 添加选择

名称：_addChoices;

描述：在构造函数中执行的私有函数以验证选择；

表 5- 5 _addChoices 函数的参数

名称	类型	描述
_choices	string[]	一组选择。

5.1.4 添加选民

名称: addVoters;

描述: 运营商将符合条件的选民地址添加到合约中;

表 5- 6 addVoters 函数的参数

名称	类型	描述
_voterAddresses	address[]	一组选民地址。

5.1.4.1 添加投票人

名称: _addVoter;

描述: 该函数针对数组_voterAddresses 中的每个地址执行。该函数确保给定地址有效。它还确保地址之前没有添加, 因此防止了双重投票的机会。成功添加地址后, 该函数会发出 AddedVoter 事件。

表 5- 7 _addVoter 函数的参数

名称	类型	描述
_voterAddress	address	选民的地址。

5.1.5 获得选票

名称: getBallots;

表 5- 8 getBallots 函数的返回值

名称	类型	描述
	tuple[]	包含所有选票的 Ballot 类型数组。

5.1.6 获得选择

名称: getChoices;

表 5- 9 getChoices 函数的返回值

名称	类型	描述
	tuple[]	包含所有选择的 Choice 类型的数组。

5.1.7 获取注册时间

名称: getRegistrationTime;

表 5- 10 getRegistrationTime 函数的返回值

名称	类型	描述
	uint256	注册期的开始时间。
	uint256	注册期结束时间。

5.1.8 获取投票时间

名称: getVotingTime;

表 5- 11 getVotingTime 函数的返回值

名称	类型	描述
	uint256	投票期的开始时间。
	uint256	投票期结束时间。

5.1.9 注册

名称: register;

描述: 该函数由投票者自己调用, 以便将盲化的 UUID 上传到要签署的合同中。将选民标记为登记册, 使他能够在投票期间投票。该功能只能在注册期间由符合条件的选民执行;

表 5- 12 register 函数的参数

名称	类型	描述
_blindedUUID	uint256	投票者 UUID 的盲值。

5.1.10 设置选民签名的盲 UUID

名称：signBlindedUUID;

描述：运营商在链下签署选民的盲 UUID 并使用此功能将其上传到合约；

表 5- 13 signBlindedUUID 函数的参数

名称	类型	描述
_voterAddress	address	给定选民的地址。
_signedBlindedUUID	uint256	给定选民的签名盲 UUID。

5.1.11 计票

名称：tallyVotes;

描述：该函数在结果被操作员验证为有效后对结果进行统计。要求执行时间在投票时间结束之后。只能调用一次。只有操作员可以调用它；

表 5- 14 tallyVotes 函数的参数

名称	类型	描述
_uuids	uint256[]	选民 UUID 的数组
_choices	uint256[]	选民选择的数组

5.1.11.1 计单票

名称：_tallyVote;

描述：在每个 UUID 和选择对的 tallyVotes 中执行的私有函数。它验证 UUID 和选择。如果两者都有效，则给定选择已收到的票数加一。

表 5- 15 _tallyVote 函数的参数

名称	类型	描述
_uuid	uint256	选民的 UUID。
_choice	uint256	选民的相应选择。

5.1.12 投票

名称：vote;

描述：选民使用此功能投票。它需要组成选票并进行投票所需的所有值。投票者的公钥值不能为零。UUID 和非盲签名 UUID 也是如此。该功能只能在投票期间执行；

表 5- 16 vote 函数的参数

名称	类型	描述
_RX	uint256	投票者公钥的 X 值。
_RY	uint256	投票者公钥的 Y 值。
_uuid	uint256	选民的 UUID。
_unblindedSignedUUID	uint256	选民的非盲签名 UUID。
_choice	uint256	选民的选择。

5.1.13 映射

5.1.13.1 选民

名称：voters;

描述：以选民地址为键、结构 Voter 为值的映射；

表 5- 17 voters 映射键

名称	类型	描述
	address	选民的地址。

表 5- 18 voters 映射值

名称	类型	描述
voterAddress	address	选民的地址。
blindedUUID	uint256	选民的盲 UUID。
signedBlindedUUID	uint256	操作员签名的盲 UUID。
registered	bool	标记选民是否已登记的标志。

5.1.13.2 投票

名称: votes

描述: 以投票者的 UUID 作为键, 以结构体投票作为值的映射。

表 5- 19 votes 映射键

名称	类型	描述
	uint256	选民的 UUID。

表 5- 20 votes 映射值

名称	类型	描述
voted	bool	表示选民是否投票的标志。
choice	uint256	选民的选择。

5.2 VotingSystemFactory 智能合约文档

该系统的开发理念是仅使用一种设置即可实现多种用途。在这样的系统中, 每次发生新投票时都必须部署 VotingSystem 智能合约。VotingSystemFactory 智能合约执行此任务。

5.2.1 创建

名称: create;

描述: 部署 VotingSystem 智能合约。一旦设置, 所有参数都是不可变的。投票名称不能为空字符串。选择数组不能为空或包含空字符串。注册开始时间戳的值必须小于注册结束时间戳。投票开始时间戳的值必须小于投票结束时间戳但大于或等

于注册结束时间。最后，运营商的公钥值不能为零；

表 5- 21 create 函数的参数

名称	类型	描述
_name	string	投票的名称。
_choices	string[]	一组选择。
_registrationStart	uint256	注册期的开始。
_registrationEnd	uint256	注册期结束。
_votingStart	uint256	投票期的开始。
_votingEnd	uint256	投票期结束。
_RX	uint256	运营商签名公钥的 X 值。
_RY	uint256	运营商签名公钥的 Y 值。

5.3 盲签名^[18]实现

该系统使用椭圆曲线盲签名方案的 JavaScript 实现。该方案使用 secp256k1 曲线。以太坊使用这条曲线来签署交易。

要工作，签名者（投票操作员）必须生成签名密钥对和验证密钥对。请求签名的人（选民）生成两个秘密值和一个公钥。

该方案在文件 blind-secp256k1.js 中实现。以下文档描述了文件中的功能。

5.3.1 新密钥对

名称：newKeyPair;

描述：生成新的椭圆曲线密钥对进行验证。该密钥由操作员在本地保密。

表 5- 22 newKeyPair 函数的参数

名称	类型	描述
d	string	私钥。
QX	string	签名者的公钥 X 值。
QY	string	签名者的公钥 Y 值。

5.3.2 新的请求参数

名称: newRequestPair;

描述: 生成用于签名的新椭圆曲线密钥对。此函数生成的公钥是上传到智能合约的公钥。选民使用该公钥将 UUID 盲化。

表 5- 23 newRequestPair 函数的参数

名称	类型	描述
k	string	私钥。
signerRX	string	签名者的公钥 X 值。
signerRY	string	签名者的公钥 Y 值。

5.3.3 盲化消息

名称: blindMessage;

描述: 盲化消息。在系统的情况下, 选民使用此功能将他的 UUID 盲化。

表 5- 24 blindMessage 函数的参数

名称	类型	描述
message	string	消息来盲化。
signerRX	BN	签名者的公钥 X 值。
signerRY	BN	签名者的公钥 Y 值。

表 5- 25 blindMessage 函数的返回值

名称	类型	描述
a	string	第一个致盲秘密值。
b	string	第二个致盲秘密值。
RX	string	用户的公钥 X 值。
RY	string	用户的公钥 Y 值。
hm	string	消息哈希。
blindedMessage	BN	盲目的消息。

5.3.4 签署盲目的消息

名称：signBlindedMessage;

描述：将消息盲化。操作员从合约中获取选民的盲 UUID 并使用此功能对其进行签名。

表 5- 26 signBlindedMessage 函数的参数

名称	类型	描述
blindedMessage	BN	盲目的消息。
d	string	签署私钥。
k	string	验证私钥。

表 5- 27 signBlindedMessage 函数的返回值

名称	类型	描述
signedBlindedMessage	BN	签名的盲消息。

5.3.5 Unblind signed blinded message

名称：unblindSignedBlindedMessage;

描述：解盲已签名的盲消息。投票者使用此函数来获取投票所需的非盲签名消息。

表 5- 28 unblindSignedBlindedMessage 函数的参数

名称	类型	描述
signedBlindedMessage	BN	签名的盲消息。
a	string	第一个致盲秘密值。
b	string	第二个致盲秘密值。

表 5- 29 unblindSignedBlindedMessage 函数的返回值

名称	类型	描述
unblindedSignedMessage	BN	解盲的签名的消息。

5.3.6 验证

名称：verify;

描述：此函数验证提供的消息哈希是否是在获取非盲签名消息的过程中使用的哈希。

表 5- 30 verify 函数的参数

名称	类型	描述
hm	BN	消息哈希。
unblindedSignedMessage	BN	一个非盲签名的消息。
RX	BN	用户的公钥 X 值。
RY	BN	用户的公钥 Y 值。
QX	string	签名者的验证公钥 X 值。
QY	string	签名者的验证公钥 Y 值。

表 5- 31 verify 函数的返回值

名称	类型	描述
	bool	如果哈希有效则为真，否则为假。

5.4 Next.js 网络应用程序

应用程序的前端是使用 Next.js 框架构建的。Next.js 是一种基于 React.js 的框架类型。UI 是使用 JavaScript 文件中的 JSX 而不是纯 HTML 和 CSS 构建的。

前端应用程序的目的是与后端交互。这里的后端是区块链和智能合约。

为了与智能合约交互，我为 VotingSystem 和 VotingSystemFactory 创建了包装类。使用类是因为使用 web3.js 库时的代码比较乱，合约函数返回的数据比较乱，需要改造。

通过前端与区块链交互的函数代码如下：

5.4.1 创建投票

```
const create = async (  
  factoryContract,  
  name,
```



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

5.4.2 添加选民

```
const add = async (contract, voters) => {
```

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

5.4.3 注册

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

5.4.4 签名

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

5.4.5 更新密钥

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

5.4.6 投票

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

5.4.7 计票

```
const tally = async (contract, keys) => {  
  // Uses the “VotingSystem” contract wrapper class to get the ballots  
  const ballots = await contract.object.ballots();  
  const uuids = [];  
  const choices = [];  
  ballots.forEach((ballot) => {  
    // Verify the ballot integrity using the blind-secp256k1 implementation
```

```
const isValid = verify(
  ballot.uuid,
  ballot.unblindedSignedUUID,
  ballot.RX,
  ballot.RY,
  keys.QX,
  keys.QY
);
if (isValid) {
  uuids.push(ballot.uuid);
  choices.push(ballot.choice);
}
});
if (uuids.length === choices.length && uuids.length !== 0)
  // Uses the "VotingSystem" contract wrapper class to tally the votes
  await contract.object.tallyVotes(uuids, choices);
else alert("No ballots to tally");
};
```

5.4.8 验证选票

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

5.5 签名者命令行界面应用程序

signer-cli 是一个 Node.js 模块。其主要目的是自动签署选民的盲 UUID，而不是操作员手动检查选民是否请求签名，然后从用户界面（UI）签署他的盲 UUID。

当应用程序运行时，它使用 WebSocket 连接到区块链和 web3.js API 来监听事件并生成日志。

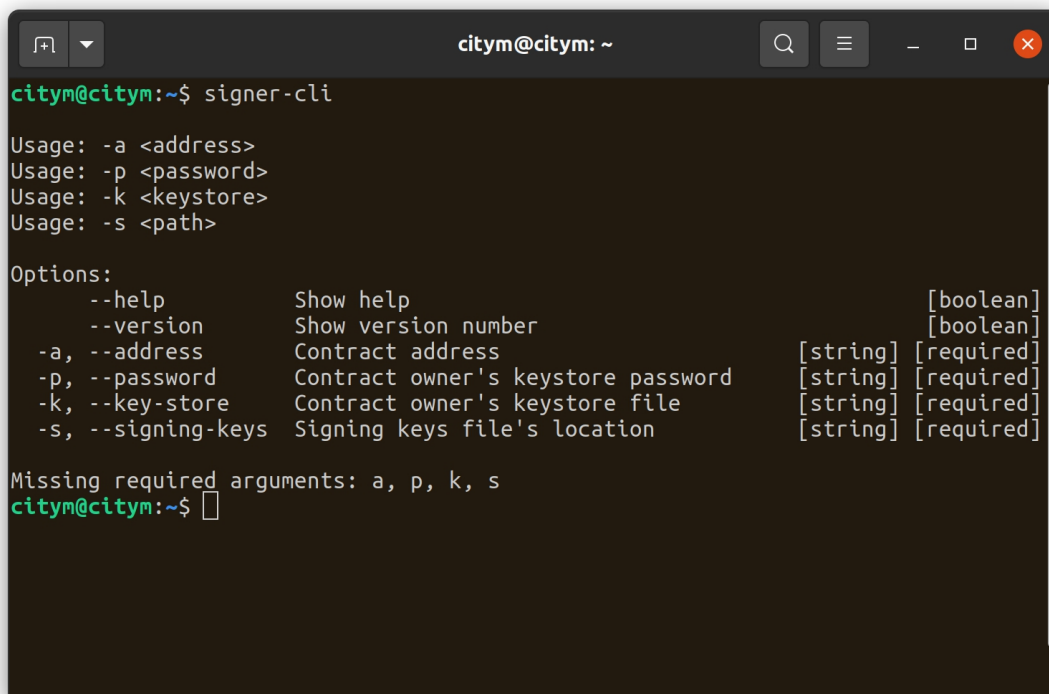
CLI 应用程序是一个 Node.js 模块，因此可以使用 npm 在本地安装。通过转到终端中的模块目录并运行以下命令来完成安装：

```
npm i -g .
```

一旦安装了模块，就可以从任何地方调用它。例如，以下命令用于启动 signer-cli 模块：

```
signer-cli
```

该命令本身，不带任何选项，将返回以下消息，显示所需选项及其描述（图 5-1）：



```
citym@citym: ~  
citym@citym:~$ signer-cli  
Usage: -a <address>  
Usage: -p <password>  
Usage: -k <keystore>  
Usage: -s <path>  
  
Options:  
  --help          Show help [boolean]  
  --version       Show version number [boolean]  
  -a, --address   Contract address [string] [required]  
  -p, --password  Contract owner's keystore password [string] [required]  
  -k, --key-store Contract owner's keystore file [string] [required]  
  -s, --signing-keys Signing keys file's location [string] [required]  
  
Missing required arguments: a, p, k, s  
citym@citym:~$
```

图 5-1 signer-cli 选项说明

第 6 章 部署与测试

在最后一章中，系统被部署和测试。

6.1 系统部署

6.1.1 开发环境

软硬件环境如表 6-1 所示：

表 6- 1 开发环境

类型	规格
硬件环境	RAM: 2x8 GB SODIMM DDR4 Synchronous 2667 MHz CPU: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
软件环境	操作系统：Ubuntu 20.04 LTS 开发环境：Go-Ethereum client, VS Code IDE JavaScript 框架：Node.js, Next.js 部署框架：Truffle

6.1.2 系统部署架构

所有 Web2 应用程序都有一个后端，它为前端提供数据。同样，Web3 dApp 也有同样的要求。所以唯一的区别在于后端，Web2 应用程序将连接到数据库，而 Web3 dApps 将连接到区块链。

该系统使用 Geth 提供的 RPC 连接连接到区块链。Truffle 框架在系统运行之前通过此连接部署 VotingSystemFactory 合约。稍后，当系统通过该 RPC 连接运行时，操作员将从前端部署投票实例。

使用 RPC，参与投票的用户将他们的钱包连接到私有区块链连接，并通过前端与智能合约进行交互。

部署系统的架构如图 6-1 所示。

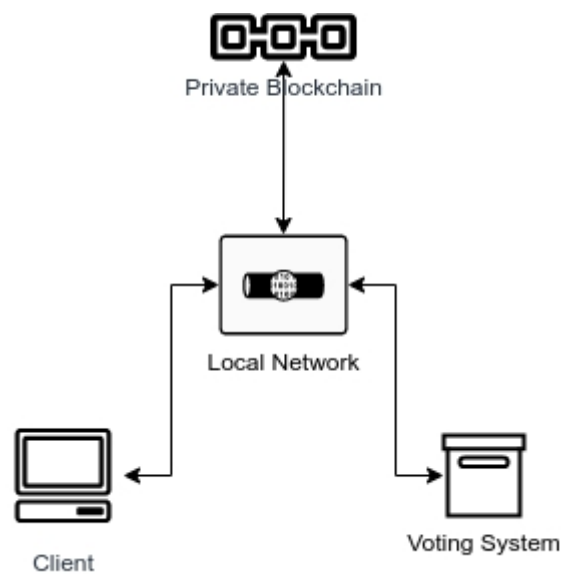


图 6-1 部署系统

6.2 系统模块功能测试

此部分显示黑盒、功能、集成和系统测试。

6.2.1 连接 MetaMask 钱包

运营商和选民必须使用 MetaMask 钱包连接到私有区块链才能与系统交互。

表 6- 2 连接 MetaMask 钱包测试用例及结果

描述项	具体说明
用例 id	Case 1
用例名称	连接 MetaMask 钱包
条件	已经安装了 MetaMask。
测试目的	测试选民和操作员能否连接到区块链。
操作步骤	(1) 在浏览器中打开应用程序； (2) 点击 “Connect” 按钮。
第 1 组预期结果	操作员可以使用节点地址进行连接。
第 1 组实际结果	操作员可以使用节点地址进行连接。
第 2 组预期结果	选民可以使用自己的地址进行连接。

- 第 2 组实际结果 选民可以使用自己的地址进行连接。
- 第 3 组预期结果 尝试连接到错误的网络时会弹出警报窗口。
- 第 3 组实际结果 尝试连接到错误的网络时会弹出警报窗口。

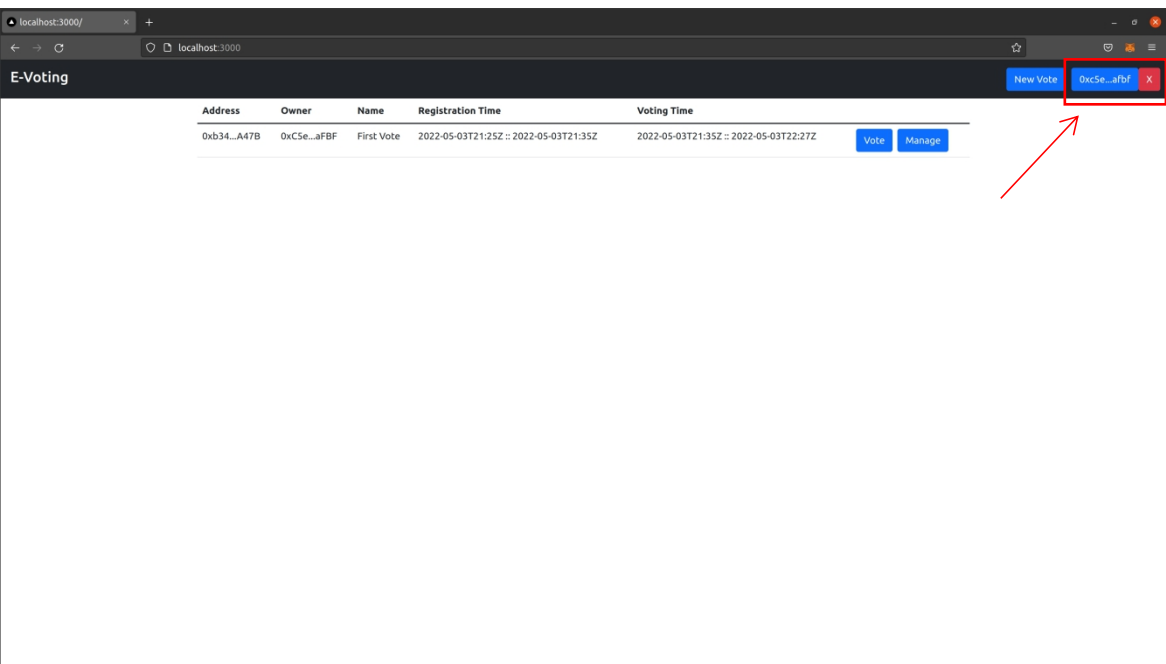


图 6-2 操作员可以使用节点地址进行连接。

6.2.2 创建新投票

操作员必须创建一个新的投票实例。创建新投票会在区块链上部署新的 VotingSystem 智能合约。这是通过 UI 完成的。以下测试（表 6-3）显示此功能是否正常工作。

表 6- 3 创建新投票测试用例及结果

描述项	具体说明
用例 id	Case 2
用例名称	创建新投票
条件	操作员与他的节点地址相连。
测试目的	测试操作员是否可以创建新投票。

北京理工大学本科生毕业设计（论文）

操作步骤	<ul style="list-style-type: none">(1) 在浏览器中打开应用程序；(2) 点击 “New Vote” 按钮；(3) 输入投票数据；(4) 点击 “Submit” 按钮。
第 1 组测试数据	<p>Vote Name: Fist Vote</p> <p>Choice 0: A</p> <p>Choice 1: B</p> <p>Choice 2: C</p> <p>Registration Start Date: 05/18/2022 , 07:20 PM</p> <p>Registration End Date: 05/18/2022 , 08:20 PM</p> <p>Voting Start Date: 05/18/2022 , 08:20 PM</p> <p>Voting End Date: 05/18/2022 , 09:20 PM</p>
第 1 组预期结果	当输入正确的数据时，就会创建一个新的投票，并且操作员可以下载特定投票的签名密钥。
第 1 组实际结果	当输入正确的数据时，就会创建一个新的投票，并且操作员可以下载特定投票的签名密钥。

图 6-3 显示了第一次测试成功创建的投票。时间被转换为 UTC，以便来自不同时区的人。他们可以通过 VPN 连接到专用网络并进行投票。UTC 在全球范围内流行和使用，尤其是在区块链社区中。

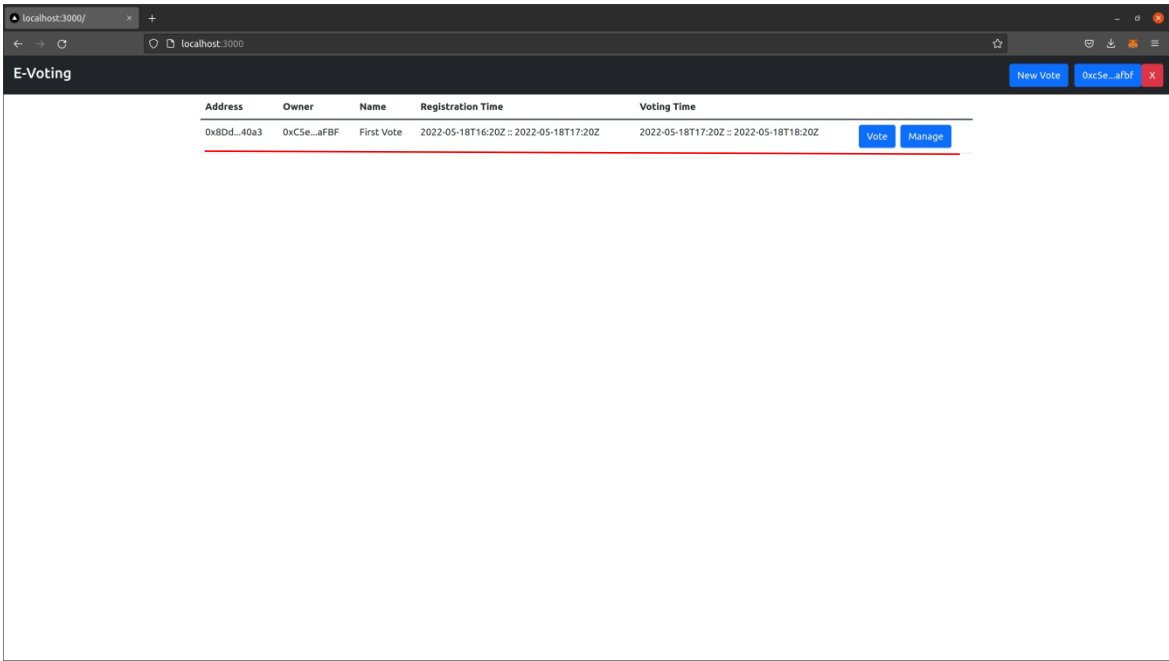


图 6-3 创建投票成功

6.2.3 添加选民

创建投票后，操作员必须将投票者的地址添加到智能合约中。表 6-4 是对此功能执行的测试。

表 6- 4 添加选民测试用例和结果

描述项	具体说明
用例 id	Case 3
用例名称	添加选民
条件	操作员与他的节点地址相连。
测试目的	测试操作员是否可以添加选民。
操作步骤	(1) 在浏览器中打开应用程序； (2) 在本投票上点击“Manage”按钮； (3) 选“Add Voters”选项卡； (4) 添加选民； (5) 点击“Add”按钮。

第 1 组测试数据	Voter 0: 0x871ac0c271Bf817517b05F52A540Bf317525Ba2e Voter 1: 0xe4BACd4F44Ddc422c7CefB7c90C8a2f02f346078 Voter 2: 0xDc47CDdB2f71564e0b7cA1d40a730E6352125B04
第 1 组预期结果	成功添加选民
第 1 组实际结果	成功添加选民
第 2 组测试数据	Voter 0:
第 2 组预期结果	显示带有错误消息的警报。
第 2 组实际结果	显示带有错误消息的警报。

6.2.4 运行 signer-cli

signer-cli 应用程序用于签署选民的 UUID 并制作日志。操作员将其用于特定投票。该应用程序必须在节点机器上运行。对 signer-cli 执行的测试如表 6-5 所示。

表 6- 5 signer-cli 黑盒测试

描述项	具体说明
用例 id	Case 4
用例名称	运行 signer-cli
条件	signer-cli 必须在节点的机器上启动。
测试目的	测试操作员是否可以为给定投票启动 signer-cli。
操作步骤	(1) 打开一个终端窗口 (2) 在终端中输入带有所需选项的 signer-cli 命令。
第 1 组测试数据	-a: 0x8Dd2ce04169a7936d2686c22D2fC6d576B1940a3 -p: 123456 -k: ~/e-voting-system/blockchain/node1/ keystore/UTC--2022-04-20T09-10-06. 743241232Z--c5e6c9f8893cb397c3a49ad2257cf6543e55afbf -s: ~/Downloads/signing-keys.json
第 1 组预期结果	成功运行 signer-cli。
第 1 组实际结果	成功运行 signer-cli。

第 2 组测试数据	-a: <无效地址> 其余与测试 1 相同。
第 2 组预期结果	signer-cli 不会启动并抛出错误。
第 2 组实际结果	signer-cli 不会启动并抛出错误。
第 3 组测试数据	-p: <无效地址> 其余与测试 1 相同。
第 3 组预期结果	signer-cli 不会启动并抛出错误。
第 3 组实际结果	signer-cli 不会启动并抛出错误。
第 4 组测试数据	-k: <无效地址> 其余与测试 1 相同。
第 4 组预期结果	signer-cli 不会启动并抛出错误。
第 4 组实际结果	signer-cli 不会启动并抛出错误。
第 5 组测试数据	-s: <无效地址> 其余与测试 1 相同。
第 5 组预期结果	signer-cli 不会启动并抛出错误。
第 5 组实际结果	signer-cli 不会启动并抛出错误。



图 6-4 singer-cli 运行

6.2.5 注册

操作员添加地址后，选民必须注册才能投票。他们必须通过按下按钮进入投票页面并从注册选项卡注册。测试结果见表 6-6。

表 6- 6 选民注册测试

描述项	具体说明
用例 id	Case 5
用例名称	选民注册
条件	连接到区块链并且是合格的选民。
测试目的	测试选民是否可以注册投票。
操作步骤	(1) 在浏览器中打开应用程序； (2) 在本投票上点击“Vote”按钮； (3) 选“Register”选项卡； (4) 点击“Register”按钮。
第 1 组预期结果	符合条件的选民成功注册并下载了注册密钥。
第 1 组实际结果	符合条件的选民成功注册并下载了注册密钥。
第 1 组预期结果	MetaMask 在注册期间外尝试注册时显示错误。
第 1 组实际结果	MetaMask 在注册期间外尝试注册时显示错误。
第 2 组预期结果	不符合条件的选民会看到“Not Eligible For Registration”消息。
第 2 组实际结果	不符合条件的选民会看到“Not Eligible For Registration”消息。

当投票者注册时,signer-cli 应用程序正在运行并对传入的请求进行签名(图 6-5)。



图 6-5 signer-cli 签名盲 UUID

6.2.6 更新密钥

选民被签署后的时候，他必须更新他的密钥。密钥更新测试见表 6-7。

表 6- 7 更新密钥测试

描述项	具体说明
用例 id	Case 6
用例名称	更新密钥
条件	连接到区块链并注册过的选民。
测试目的	测试操作员是否可以添加选民。
操作步骤	(1) 在浏览器中打开应用程序; (2) 在本投票上点击“Vote”按钮; (3) 选“Register”选项卡; (4) 上传注册密钥; (5) 点击“Update”按钮。
第 1 组预期结果	已成功更新注册密钥并下载投票密钥。
第 1 组实际结果	已成功更新注册密钥并下载投票密钥。

6.2.7 投票

投票只能在投票期间进行。为了投票，选民必须上传他的投票密钥。如果选民想保持匿名，他必须使用与注册时使用的地址不同的地址进行投票。

表 6- 8 投票测试

描述项	具体说明
用例 id	Case 7
用例名称	投票
条件	连接到区块链并且是合格的选民。
测试目的	测试选民是否可以投票。
操作步骤	(1) 在浏览器中打开应用程序;

	(2) 在本投票上点击“Vote”按钮；
	(3) 选“Vote”选项卡；
	(4) 上传投票密钥；
	(5) 选择一个选项；
	(6) 点击“Vote”按钮。
第 1 组预期结果	在投票期间成功投票。
第 1 组实际结果	在投票期间成功投票。
第 2 组预期结果	在投票期间外失败投票。
第 2 组实际结果	在投票期间外失败投票。

在图 6-6 中可以看到选民投票的日志。所有投票均来自同一地址，以表明没有指向起始注册地址的链接。因此，使投票程序私密且匿名。

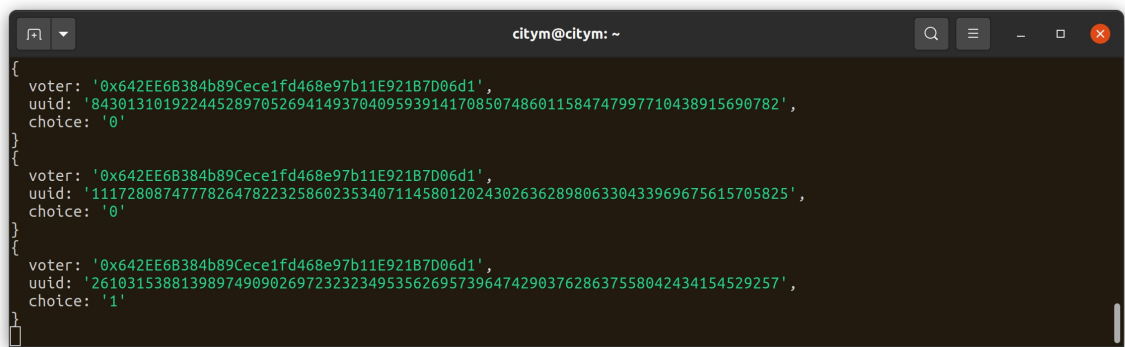


图 6-6 投票记录

6.2.8 计票

投票期结束后，操作员必须对选票进行统计。他必须进入 Tally Votes 选项卡，上传签名密钥并计算结果。该测试的结果如表 6-9 所示。

表 6- 9 计票测试

描述项	具体说明
用例 id	Case 8

用例名称	计票
条件	时间是在投票期之后。操作员连接到区块链。
测试目的	测试计算结果的过程。
操作步骤	<p>(1) 在浏览器中打开应用程序；</p> <p>(2) 在本投票上点击“Manage”按钮；</p> <p>(3) 选“Tally Votes”选项卡；</p> <p>(4) 上传签名密钥；</p> <p>(5) 点击“Tally”按钮。</p>
第 1 组预期结果	投票期结束前无法统计结果。
第 1 组实际结果	投票期结束前无法统计结果。
第 2 组预期结果	成功计算结果。
第 2 组实际结果	成功计算结果。
第 3 组预期结果	如果已经统计过，则无法统计结果。
第 3 组实际结果	如果已经统计过，则无法统计结果。

在图 6-6 中可以看到，两张票投 0 票，一张投 1 票，这意味着两名选民选择了 A，一名选民选择了 B。与图 6-7 中显示的结果相同。

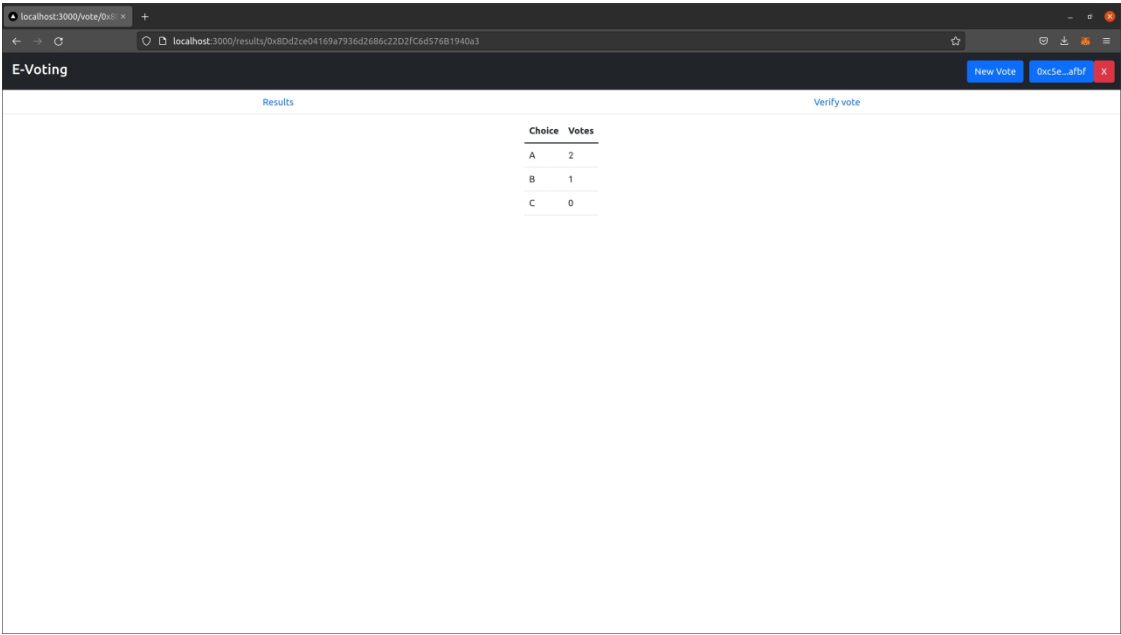


图 6-7 投票结果

6.2.9 验证投票

为了验证他的投票，选民必须上传他的验证密钥，然后将显示的结果与他在投票期间所做的选择进行比较。

表 6- 10 验证投票测试

描述项	具体说明
用例 id	Case 9
用例名称	验证投票
条件	时间是在投票期之后。 选民连接到区块链。
测试目的	测试能否验证投票。
操作步骤	(1) 在浏览器中打开应用程序； (2) 在本投票上点击“Results”按钮； (3) 选“Verify vote”选项卡； (4) 上传验证密钥； (5) 点击“Verify”按钮。
第 1 组预期结果	显示与投票期间做出的选择相同的选择。
第 1 组实际结果	显示与投票期间做出的选择相同的选择。。

在图 6-8 中，您可以看到身份验证如何显示给用户。在该具体的例子中，地址投票给了 A，验证显示为 A，所以功能起作用了。最好使用与注册验证投票时使用的地址不同的地址。另外，验证完成后，最好删除密钥。

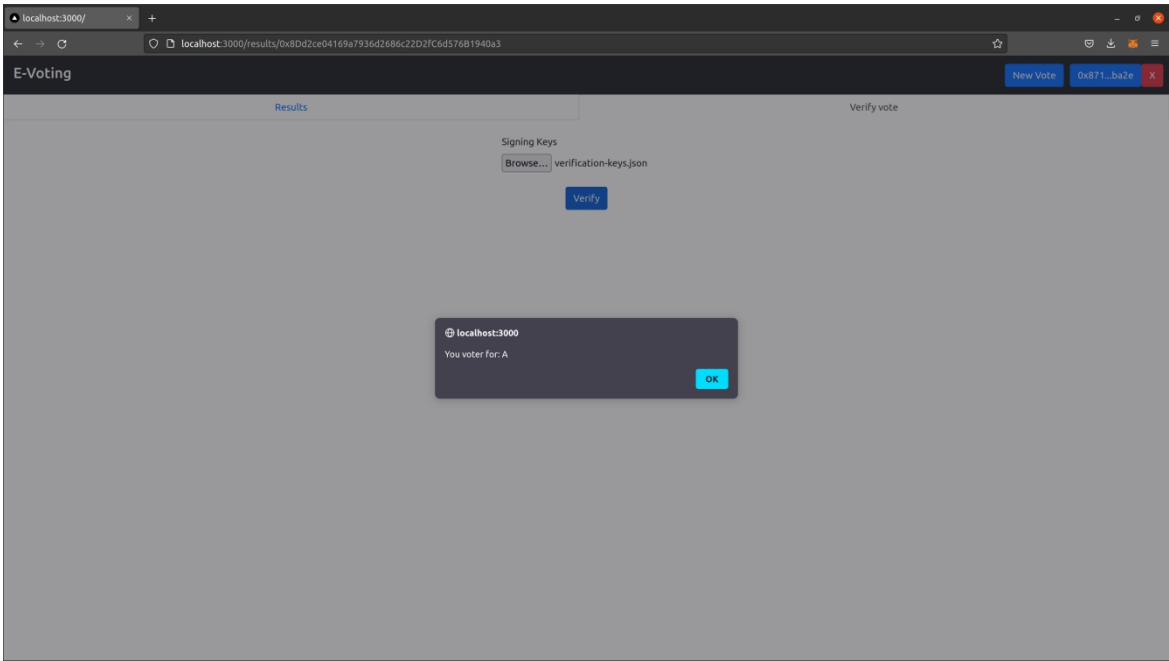


图 6-8 选民验证自己的投票

结 论

总结的整个项目是一个基于以太坊私有区块链的投票系统，它使用盲签名来强制隐私。提议的系统是信任的，这意味着选民必须相信运营商会诚实地签署和处理他们的选票。确保运营商诚实的唯一方法是让每个选民在结果出来后验证他的投票。这与智能合约相结合，可以证明投票是诚实的。

该系统由三部分组成，包括 Solidity 智能合约和以太坊私有链、Next.js 前端 Web 应用程序和 Node.js CLI 应用程序，用于签署选民和记录投票数据。

该系统采用私有以太坊区块链，采用 PoA 共识算法，确保网络安全稳定。

Web3.js 是通过为 JavaScript 代码提供与区块链交互的方式来连接各个部分的库。首先，Truffle 框架使用该库来部署 VotingSystemFactory 合约。然后在 web3.js 库和 Next.js 应用程序中的 MetaMask 的帮助下，用户可以轻松连接到区块链并与智能合约进行交互，而无需任何有关区块链的先验知识。最后，在 Node.js CLI 应用程序中使用 web3.js 库来监控事件和自动化签名过程。

最后，经过测试，系统实现了可信的端到端可验证的电子投票功能，并保证投票过程的隐私保护。

参考文献

- [1] STAVELEY E. Greek and Roman voting and elections[M]. London: Thames and Hudson, 1972.
- [2] KRIMMER R., DUENAS-CID D., KRIVONOSOVA I. et al. How Much Does an e-Vote Cost? Cost Comparison per Vote in Multichannel Elections in Estonia[A]. In: KRIMMER R. et al. Electronic Voting[M]. Cham: Springer International Publishing, 2018: 117-131.
- [3] HEIBERG S, WILLEMSON J. Verifiable internet voting in Estonia[J]. 2014 6th International Conference on Electronic Voting: Verifying the Vote (EVOTE), 2014: 1-8.
- [4] NAKAMOTO S. Bitcoin: A Peer-to-Peer Electronic Cash System[EB/OL]. Bitcoin.org, 2008. <https://bitcoin.org/bitcoin.pdf>.
- [5] WOOD G. Ethereum: A Secure Decentralized Generalized Transaction Ledger[EB/OL]. Ethereum.github.io, 2014. <https://ethereum.github.io/yellowpaper/paper.pdf>.
- [6] CLACK C, BAKSHI V, BRAINE L. Smart Contract Templates: foundations, design landscape and research directions[J]. CoRR, 2016, abs/1608.00771.
- [7] CHAUM D. Computer Systems Established, Maintained and Trusted by Mutually Suspicious Groups[D]. UNIVERSITY OF CALIFORNIA, BERKELEY, 1979.
- [8] CHAUM D. Untraceable electronic mail, return addresses, and digital pseudonyms[J]. Commun. ACM, 1981, 24: 84-88.
- [9] CHAUM D. Blind Signatures for Untraceable Payments[A]. In: CHAUM D. et al. Advances in Cryptology[M]. Boston, MA: Springer US, 1983: 199-203.
- [10] CHAUM D. Elections with Unconditionally-Secret Ballots and Disruption Equivalent to Breaking RSA[A]. In: BARSTOW D., GRIES D. et al. Advances in Cryptology --- EUROCRYPT '88[M]. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988: 177-182.
- [11] CHAUM D. Secret-ballot receipts: True voter-verifiable elections[J]. IEEE Security Privacy, 2004, 2(1): 38-47.
- [12] MCCORRY P., SHAHANDASHTI S., HAO F. A Smart Contract for Boardroom Voting with Maximum Voter Privacy[A]. In: KIAYIAS A. Financial Cryptography and Data Security[M]. Cham: Springer International Publishing, 2017: 357-375.
- [13] QIXUAN Z., BOWEN X., HAOTIAN J. et al. Ques-Chain: An Ethereum Based E-Voting System[J]. CoRR, 2019, abs/1905.05041.

- [14] LIU Y., WANG Q. An E-voting Protocol Based on Blockchain[J]. IACR Cryptol. ePrint Arch., 2017.
- [15] PAWLAK M., PONISZEWSKA-MARAÑDA A., KRYVINSKA N. Towards the intelligent agents for blockchain e-voting system[J]. Procedia Computer Science, 2018, 141(1877-0509): 239-246.
- [16] CANARD S., GAUD M., TRAORÉ J. Defeating malicious servers in a blind signatures based voting system[A]. In: DI CRESCENZO G., RUBIN A. Financial Cryptography and Data Security[M]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006: 148-153.
- [17] PANJA S., ROY B. A Secure End-to-End Verifiable E-Voting System Using Zero-Knowledge Proof and Blockchain[A]. In: CHAUDHURI A., GUPTA S., ROYCHOUDHURY R. A Tribute to the Legend of Professor C. R. Rao: The Centenary Volume[M]. Singapore: Springer Singapore, 2021: 45-48.
- [18] MALA H., NEZHADANSARI N. New blind signature schemes based on the (elliptic curve) discrete logarithm problem[A]. In: et al. ICCKE 2013[M]. 2013: 196-201.
- [19] SPECTER M., KOPPEL J., WEITZNER D. The Ballot is Busted Before the Blockchain: A Security Analysis of Voatz, the First Internet Voting Application Used in {U.S}. Federal Elections[A] In: 29th USENIX Security Symposium (USENIX Security 20). USENIX Association, 2020: 1535--1553.

致 谢

致谢

衷心感谢我的导师孙建伟给予我的帮助。他为我提供指导，让我按计划完成论文。

另外，我要感谢留学生办公室的老师，尤其是于春晓老师。我也感谢国家留学基金管理委员会（CSC）给了我奖学金，让我实现了留学中国的梦想。

在撰写论文期间，我得到了家人的大力支持，无论是精神上还是经济上。因此，我也想对他们表示感谢。

感谢在整个写作过程中帮助过我的所有朋友。在所有这些人中，我要特别感谢我的朋友 Stefan Shutev 让我在写论文的时候花时间在他家。

最后，我要告别中国和所有伟大的中国人民，祝愿他们繁荣昌盛。