Internet Voting System

Dissertation

Submitted for the module **CO3201**

Author: **Charlie Hall**

University of Leicester

School of Computing and Mathematical Sciences

May 2025

Declaration

All sentences or passages quoted in this report or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people's work have been specifically acknowledged by clear citation of the source, specifying author, work, date and page(s). Any part of my own written work or software code, which is substantially based upon other people's work, is duly accompanied by clear citation of the source, specifying author, work, date and page(s). I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

[illegible]

Name : Charlie Hall

Date: 01/05/25

Abstract

This project implements an internet voting system focused on transparency and immutability of election results. A Proof-of-Authority blockchain stores votes on an immutable public distributed ledger, where all vote-containing blocks are cryptographically linked. Multiple validators authenticate blocks before they are added to the ledger. All data for determining transaction validity is stored on the blockchain, allowing anyone with access to the ledger's history to verify the election. The blockchain relies on a set of validator nodes, which act as the source of truth and cross-check all transactions for authenticity. The system is deployed over a peer-to-peer network, supporting node discovery, secure communication, and resilience against DDoS attacks. Votes are authenticated using public-key cryptography, with voter credentials stored on NFC tags for convenient voter access. A desktop client enables interaction with the blockchain for relevant operations, while an NFC-enabled companion app facilitates the scanning of voter credentials. This approach demonstrates the feasibility of secure, scalable, and verifiable online voting through a blockchain-based system.

Contents

Abbreviations

Abbreviation	Meaning
API	Application Programming Interface
EVM	Ethereum Virtual Machine
GUI	Graphical User Interface
PoA	Proof of Authority
PoS	Proof of Stake
PoW	Proof of Work
UI	User Interface

Introduction

1.1 Problem Statement

The traditional server-client model is problematic when implementing a secure Internet voting system. Voting requires strict guarantees of anonymity, integrity, and transparency. Fundamentally, a server-client is a poor choice for achieving this because it has a single point of failure. In a system architecture that uses a single server to store and count votes, a hacker only needs to gain access to one machine, and they have unlimited access to undetectably alter the vote data. Additionally, participants in an election in a centralised architecture must blindly trust that their vote has been counted; no receipt mechanism can be validated independently, as voters don't have access to the source of truth, the centralised database.

1.2 Aim

This project aims to develop a software-based internet voting system emphasizing integrity and transparency. It will use a permissioned centralised blockchain network managed through a Proof-of-Authority (PoA) consensus mechanism to provide a tamper-proof platform where users can vote and verify their votes independently.

1.3 Objectives

The objectives of this project are:

- To design and implement a Proof-of-Authority (PoA) blockchain allowing tamper-proof storage of votes with transparent counting and authentication of votes.
- To develop a desktop client with a graphical user interface that enables users to vote and allows elections to be hosted.

- To conduct testing of the entire system using unit and integration testing, ensuring that all functionalities work as intended and that the software requirements are implemented correctly.
- To implement a companion app that allows user-friendly storage of voter credentials used by the blockchain to authenticate voters.

Background

2.1 Cryptography

This section outlines the key cryptographic methods relevant to the project, as outlined in [1].

2.1.1 Public–Private Key Cryptography

Public-private key cryptography uses a pair of mathematically related keys: a public key and a private key. Data encrypted with one key can only be decrypted with the other, enabling secure communication.

2.1.2 Digital Signatures

Digital signatures are a cryptographic technique based on public-key cryptography for verifying message authenticity and integrity. A signature can be generated using a private key and verified using the corresponding public key.

2.1.3 Hashing Functions

Hash functions are algorithms that convert input data into a fixed-size string of characters, known as a hash or digest. Secure hash functions ensure that even a small change in input produces a significantly different output and that the input cannot be determined from a digest

2.2 Blockchain

This section provides an overview of blockchain fundamentals, as described in [2] .

2.2.1 Blockchain

A blockchain is a decentralised ledger composed of sequential blocks. Each block contains a list of transactions and a reference (via a hashing function) to the previous block, forming an immutable chain of blocks.

2.2.2 Block Structure

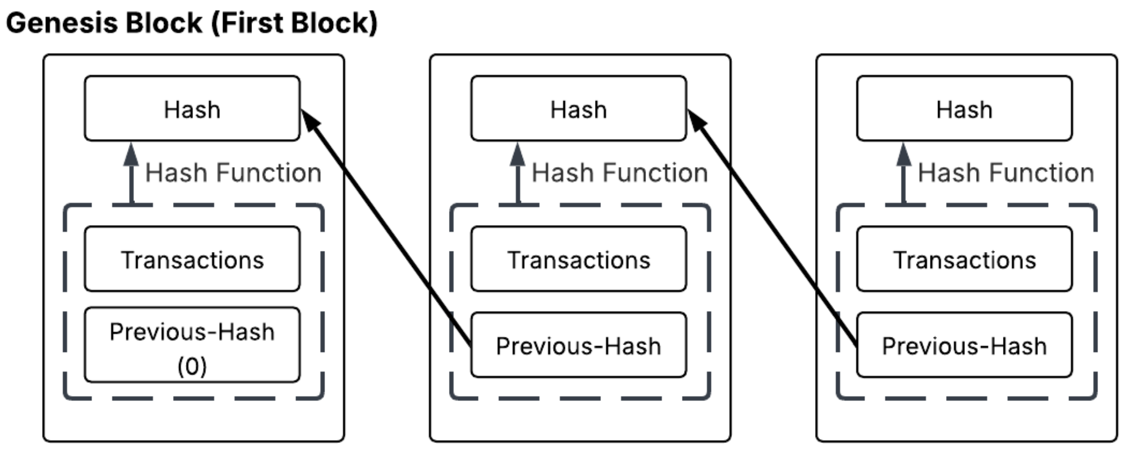


Figure 2.1: Block structure.

Blocks in a blockchain store transaction data and a cryptographic hash of the previous block. Including the previous block's hash ensures that any attempt to alter a past block invalidates the hash chain, making tampering easily detectable.

2.2.3 Decentralisation

Blockchain networks operate across a distributed set of nodes. Each node may maintain a full or partial copy of the ledger. This decentralisation eliminates single points of failure and enhances fault tolerance.

2.2.4 Consensus Mechanisms

Consensus mechanisms are protocols that blockchain participants use to agree on the blockchain's state. Prominent mechanisms include Proof of Work (Pow), Proof of Stake (PoS), and Proof of Authority (Poa).

2.2.5 Summary

Blockchain technology combines cryptographic techniques with distributed systems to provide secure, transparent, and tamper-resistant data storage. Its foundational elements include digital signatures, hashing, decentralised architecture, and consensus, enabling applications requiring trust without centralised control.

Literature Review

3.1 Voting System Principles

Voting systems must adhere to unique principles to ensure trust, transparency, and legitimacy. Watt [3] identifies six key principles that define the core requirements of a secure and trustworthy voting system that can be used for system comparison.

Table 3.1: Voting System Principles

	Principle Name	**Description**
P1	Voter Eligibility	”Those wishing to cast a vote are positively identified as eligible voters, and that no voter can cast more than a single ballot in a given election.”
P2	Impersonation	”Safeguards against personation are maintained.”
P3	Coercion	”The free exercise of the vote is safeguarded, both in terms of the opportunity to cast a ballot and voters being free from duress and unlawful undue influence.”
P4	Secrecy	”Secrecy regarding how an individual elector has, or has not, voted is preserved save in the face of a proper order of a competent Court.”
P5	Integrity	”The voting process is protected against tampering after any vote or votes have been cast.”
P6	Transparency and Auditability	”The counting procedure is verifiable, transparent, open to scrutiny, and accurate.”

3.2 Approaches and Techniques in Internet Voting

3.2.1 Democracy Guard

Democracy Guard [4] is a fully decentralised voting system built on Ethereum’s existing blockchain that uses Proof-of-Stake (PoS) consensus control. Democracy Guard uses smart contracts for all voting logic and operations, meaning the computation and reasoning for the voting system are done on the blockchain.

This means all nodes participate in determining the results of a voting operation, and no one node has the power to compromise the system. Using smart contracts allows the blockchain to be more transparent than an off-chain computation blockchain, because the inputs and outputs of the entire voting process are stored on-chain and available to all participants. Overall, this methodology closely aligns with the P6 transparency principle of voting systems, meaning the system is highly auditable.

However, this approach's trade-off is that it requires more computation than off-chain computation and thus increases the cost of deployment. This method also does not address privacy concerns arising from the votes' total traceability. There is a possibility that wallet addresses could be exposed, or voters could prove their vote by deliberately exposing their wallet addresses. This compromises the P4 voter secrecy principle, and in this particular implementation, there is no protection from coercion, violating the P3 coercion principle.

3.2.2 Votechain

VoteChain [5] is another example of a blockchain voting system. This system differs from Democracy Guard [4] because it uses off-chain computation rather than smart contracts, meaning VoteChain's blockchain stores an immutable record of finalised votes rather than the logic of voting operations. Votechain uses a variation of Proof-of-Work for consensus management; it differs from most proof-of-work blockchains in that mining nodes gain no financial reward. In their deployment context, they expect mining nodes to be either election authority machines or people volunteering their computation. Again, this highly decentralised approach helps ensure the P5 integrity principle by making it impossible to change the vote history without network consensus. Also, the ledger of previous votes is publicly accessible, making the system more trustworthy to voters, which is aligned with the P6 Transparency principle.

However, there are significant problems with this system. Proof-of-work is computationally expensive, and without a financial incentive, it's not proven that people will be willing to volunteer their computation reliably and at a large enough scale to guarantee decentralisation. In addition, if the computation pool were small and a hostile actor could provide a majority of computation to the pool, they would have total control to censor and change the blockchain. Additionally, there is no verification of vote validity by mining nodes; only the block structure and proof-of-work are verified. Any vote reaching the blockchain is assumed valid (Validated by a central web server), introducing a single point of trust and failure.

3.3 National implementations of Internet Voting

3.3.1 Estonia

As described in [6], Estonia uses a modular, centralised server model. Clients send votes to the vote-forwarding server, which verifies them and then sends them to a vote storage server. In this system, voters verify their vote using two RSA key pairs stored on a national ID card. Voters can access the keys using a key card scanner with their computer and enter a password to decrypt the keys. The first set of keys is used for identity authentication, and the second key pair is used to sign votes. At the end of the election, signatures are checked for validity and then stripped from votes; only the vote data is transferred to the counting server. The counting server tallies the votes and calculates the result. A unique feature of this system is that voters may cast multiple votes, and only the last is counted. This is effective in mitigating challenges to the P3 coercion principle.

While this system is secure, it is fully centralised and offers many bottlenecks for hackers to attack and derail it. There is also very little transparency; voters can use a smartphone app to get and decrypt their vote from the vote storage server. However, there is no proof that the vote-counting server tallied the vote correctly, violating the P6 principle to some extent.

3.3.2 Norway

As described in [7], Norway implemented a centralised voting system that relies on the traditional server-client model. The system implements vote check codes to enhance trust, which are delivered to voters through an independent channel before an election. Each candidate is assigned a unique code. After casting a vote, the voter receives a code from the system and compares it to their list to confirm that their intended vote was recorded correctly. This mechanism addresses P6 (transparency and verifiability) by giving voters an individual means of verifying their vote was recorded without modification. However, the system has significant limitations due to its centralised architecture. It relies on trust in the servers and administrators to handle vote storage and counting without independent verifiability, violating the P5 principle. Furthermore, because the check code is a receipt, it could allow voters to prove how they voted, violating the P4 secrecy principle.

3.4 Discussion

3.4.1 System Architecture

The architecture of an internet voting system fundamentally shapes its security, trust assumptions, and ability to meet the core principles of voting systems. Centralised models, such as those used in Estonia and Norway, benefit from simplicity. However, they introduce single points of failure, where a compromise of any key servers compromises the entire system undetectably. Decentralised architectures, such as public blockchain systems, reduce trust in any single actor by distributing vote data or processing across many nodes. This enhances integrity (P5) and transparency (P6), but can violate the secrecy (P4) if voter-identifying data is widely distributed across many nodes and publicly accessible.

Furthermore, decentralised blockchain systems may introduce technical barriers to participation or require voters to interact with blockchain tools, reducing the voting systems' usability and contradicting the P3 Coercion principle by creating unfair conditions.

3.4.2 Blockchain

Blockchain partnered with decentralisation offers an effective way to enhance integrity (P5) and transparency (P6), allowing voting users to have their machines become involved in the vote-counting process. In a blockchain system, the immutable record of votes and voting operations can form the basis for an audit and will also provide evidence of any attempted tampering. However, blockchains are weak because the public ledger of operations may be used to prove votes, violating the P4 secrecy principle.

A mitigation for this is to obfuscate voter identity using hashes, or in advanced implementations, zero-knowledge proofs could be used to prove vote eligibility without revealing identity.

3.4.3 Blockchain Computation Models

The two blockchain implementations differ in how the voting logic is computed. Democracy Guard uses an on-chain smart contracts system, making vote processing transparent and available to all network participants. Votechain relies on a central web server for vote processing outside the blockchain, making full auditability impossible. While centralising vote logic introduces a single point of trust and failure, executing logic on-chain is computationally expensive, can incur transaction costs, and is constrained by a limited instruction set and execution model. While these problems can be mitigated by implementing a custom blockchain rather than using Ethereum (such as Democracy Guard), it is still inefficient.

Between these two approaches, there is a middle ground: decentralized processing performed off-chain, directly on the participating nodes. This method retains the decentralization benefits and avoids a centralized point of failure. By distributing the vote logic across nodes rather than relying on a centralized server or the blockchain itself, systems can achieve greater efficiency without introducing a single point of failure.

3.4.4 System Comparison

System	Strengths	Weaknesses
DemocracyGuard	Highly transparent; network participants can trace voting operations. No single point of failure, the system is distributed, and there is a high level of redundancy.	On-chain computation is complex to implement and expensive to run. Developed on an already existing blockchain, limiting systems' control over blockchain mechanics.

System	Strengths	Weaknesses
		There are privacy concerns that blockchain voting operations could be linked to people proving how they voted.
VoteChain	Votes are transparent; voters can see their votes on the public ledger after voting. Public visibility of the vote ledger may support personal vote verification.	Vote processing is carried out by a single web server, which introduces a single point of failure for the system. Public visibility of votes violates the P4 secrecy principle if they can be linked to voting users.
Estonia	A centralised server is easy to implement and maintain. Multiple single points of failure are introduced by using central servers. National Voting ID cards with PIN protection provide strong authentication for voters.	Multiple single points of failure are introduced by using central servers. Vote processing and counting cannot be independently audited, so voters cannot verify that their vote was counted as tallied.
Norway	Strong alignment with the P3 Coercion principle by allowing multiple votes. The system is not auditable and has single points of failure.	The system is not auditable and has single points of failure.

Table 3.2: System Comparison

Software Requirements

ID	Requirement
4.1 Functional Requirements	
4.1.1 Networking	
FR-01	The system must operate over a semi-centralized peer-to-peer network.
FR-02	Nodes must be able to discover and connect to other nodes via a gossip-based discovery protocol.
FR-03	A join protocol must allow new nodes to securely authenticate and enter the network.
FR-04	Messages between nodes must include digital signatures and nonces to prevent tampering and replay attacks.
FR-05	Nodes must reject malformed, unsigned, or invalid messages.
FR-06	Validator nodes must maintain a shared directory of known nodes and their public keys.
4.1.2 Blockchain	
FR-07	A designated lead validator must propose blocks periodically (e.g. every 5 minutes).
FR-08	New blocks must only be accepted when validated by at least 70% of validators.
FR-09	All nodes must maintain a local copy of the blockchain or a valid snapshot.
FR-10	All transactions must include enough metadata to allow later validation (e.g. digital signatures, origin).
FR-11	The system must support reallocation of the lead validator in case of failure.
4.1.3 Voting Logic	

ID	Requirement
FR-12	Electoral roll must be maintained on-chain, with voters identified by public key.
FR-13	Only electors on the electoral roll may submit a ballot, and each elector may vote only once.
FR-14	Votes must be verifiable cryptographically, signed using the voter's private key.
FR-15	All vote metadata (e.g. signature, candidate ID, public key) must be stored on-chain for auditability.
FR-16	The system must prevent double voting using public key tracking.
FR-17	Votes must be verifiable by third parties but without linking identity to choice (see limitations under P4).

4.1.4 User Interface & Interaction

FR-18	A graphical interface must allow users to: Connect to the network. Submit votes. View blockchain status and election results.
FR-19	A companion mobile app must allow scanning of NFC-stored voter credentials and submission to the desktop client.

4.1.5 System Audit & Testing

FR-20	All system events (e.g. block additions, vote submissions, errors) must be logged.
FR-21	The system must implement automated unit and integration testing workflows.
FR-22	The system should generate logs for debugging and intergration with testing workflows..

4.2 Non-Functional Requirements

4.2.1 Security & Integrity

NFR-01	All communication between nodes must be encrypted and signed.
NFR-02	Blockchain data must be immutable and cryptographically verifiable.
NFR-03	The system must tolerate up to 30% validator node failure.

ID	Requirement
NFR-04	Private keys must be handled securely and not exposed in plaintext during runtime.
4.2.2 Privacy & Anonymity	
NFR-05	Public keys should be unlinkable to real-world identities where possible.
NFR-06	The system must allow auditability without exposing vote origin or contents together.
NFR-07	The design must consider, but is not yet resilient against, coercion and impersonation attacks.
4.2.3 Usability	
NFR-08	Voting interaction should take less than 2 seconds under normal network conditions.
NFR-09	The UI must be responsive and support keyboard/mouse interaction.
NFR-10	The interface must adapt to different screen sizes and follow basic accessibility guidelines.
NFR-11	Voter onboarding (e.g. scanning credentials, submitting vote) should require minimal technical knowledge.

Table 4.1: Software Requirements of the Voting Blockchain System

Implementation

5.1 Technologies Used

5.1.1 Python

Python was the language used to implement the desktop app. It was chosen for its well-developed cybersecurity libraries and because it is the language I feel most comfortable using. Thus, it would give the project the fastest and smoothest development. Several external Python libraries, beyond the standard library, were used in the project's development. External libraries were used where implementing a custom solution would be time-consuming and where existing libraries met security and functional requirements.

5.1.2 PyQt5

PyQt5 [8] was the library used to create all the user interfaces in the project. It was also used for thread management. In addition, the project mainly uses PyQt5's signals for communication between threads, particularly between the backend blockchain server thread and the UI.

5.1.3 Cryptography

The cryptography module [9] provided abstracted functions for handling key generation, encryption, decryption, signature generation, and verification.

5.1.4 Qrcode

The Qrcode module [10] generated the QR code for the mobile companion app to scan.

5.1.5 FastAPI and Uvicorn

Fastapi and Uvicorn [11, 12] hosted the HTTP server that receives the credentials post-request from the mobile app.

5.2 System Architecture Summary

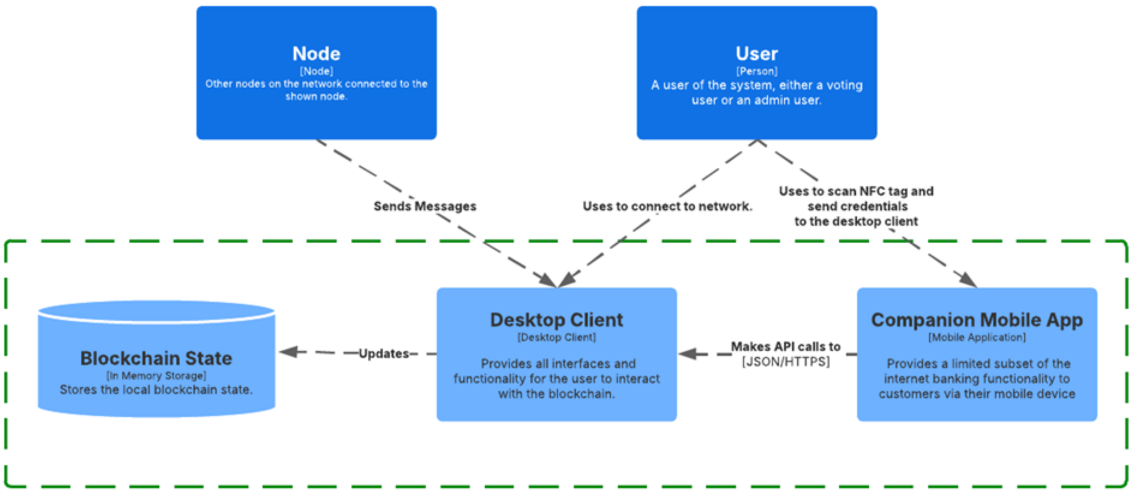

Node
[Node]
Other nodes on the network connected to the shown node.
User
[Person]
A user of the system, either a voting user or an admin user.
Sends Messages
Uses to connect to network.
Uses to scan NFC tag and send credentials to the desktop client
Blockchain State
[In Memory Storage]
Stores the local blockchain state.
Updates
Desktop Client
[Desktop Client]
Provides all interfaces and functionality for the user to interact with the blockchain.
Makes API calls to
[JSON/HTTPS]
Companion Mobile App
[Mobile Application]
Provides a limited subset of the internet banking functionality to customers via their mobile device

Figure 5.1: A C4 container diagram showing the system overview.

The project architecture, shown in 5.1, comprises two independent software components: a desktop blockchain client and a companion mobile app.

5.2.1 Companion Mobile App

The mobile app allows voters to scan NFC tags containing elector credentials. Once scanned, these credentials are transmitted via an HTTP request to the desktop application for loading to the desktop client.

5.2.2 Desktop Blockchain Client

The desktop app is a graphical interface that allows users to interact with the blockchain. It is structured into several components that modularise functionality. This has benefits such as enabling a more straightforward development process and easier maintenance.

The client has a small API component: an HTTP server implemented with FastAPI that runs with Uvicorn. The server receives the elector credential data from the companion mobile app via a single post endpoint /credentials. Upon a valid request, the API layer will pass the credential data to the blockchain component for vote proposals from the client.

The terminal component allows the user to interact with the blockchain server directly through the terminal rather than the UI. It interprets user input sent via the command line and queues the relevant actions to be executed on the server thread.

The UI component is implemented with the PyQt5 framework. It controls the main thread and is the application's entry point. The UI component is responsible for creating all pages for the user to interact with and, like the terminal layer, can queue functions to be executed by the server component. The UI can be optionally disabled with a command-line argument; all input is via the terminal layer in this mode.

Finally, the blockchain server is the core component that implements the peer-to-peer network with other desktop clients and handles all blockchain operations. TCP communication is implemented using asyncio, but the blockchain and peer-to-peer network were developed from scratch. This component is responsible for most system functionality. Some key responsibilities are managing connections between nodes, receiving from nodes, sending messages to nodes and handling blockchain operations, including proposing votes to the network.

5.3 System Usage

5.3.1 Nodes and Voters

In this system, a node is any machine participating in the network, whether through the desktop client or a terminal, that runs an instance of the blockchain server. In contrast, a voter holds valid credentials (as defined by the electoral roll) and can cast a vote.

Importantly, not all nodes are voters. Nodes serve as infrastructure for the network, enabling blockchain operations and acting as interfaces through which voters or administrations can interact with the system. Voting is only permitted for authenticated voters; nodes facilitate the process but do not inherently possess voting rights.

5.3.2 Election Authority

This system assumes that the elections take place with a trusted central authority. The networking and blockchain implementation relies on some degree of trust in the bootstrap and validator nodes.

5.3.3 User Permissions and Roles

System users can be categorised into two roles: voting users and election officials. Although both roles use the same software, their permission and control over the system will be determined by the permission level of the node to which their desktop client is connected. Election Officials interact with the system via validator nodes, which grant them elevated permissions to manage blockchain operations directly. Voting Users, on the other hand, connect through regular nodes, allowing them only to propose votes to validator nodes.

This means user permissions are determined indirectly by node connectivity instead of the system delegating roles to specific users, as is typical of a server-client architecture. Election officials will be able to;

- Load and add electors to the electoral register.
- Add candidates.
- Delegate other nodes as validators.

Voting users will be able to;

- See the blockchain state
- See the blockchain history.
- Propose a vote for a candidate.

5.3.4 Use Cases

The following section will list the system's key use cases.

Use Case #1: Create Network

- Description: This is the first step in setting up the peer-to-peer network that all other nodes will join.
- User: Election Official
- Steps:
  1. The user launches the program.
  2. The user selects the host option on the index page.
  3. The user inputs the host and port of the bootstrap node.z
  4. The node starts as a network bootstrap on the host and port given.

Use Case #2: Create a new node and join the network.

- Description: Creates a new node and joins the network as a regular node.
- User: Election Official and Voting User
- Steps:
  1. The user launches the program
  2. The user selects the connect option on the index page.
  3. The user inputs the host and port of the bootstrap node of the network they wish to connect to.
  4. The node starts and attempts to connect to the given host and port.

Use Case #3: Add a validator

- Description: An election official can delegate another node as a validator from an already existing validator node. In a real-world context, this would be another node that the election official started.

- User: Election Official using a validator node.
- Prerequisite cases: #1, #2,
- Steps:
  1. The user selects the add validator option from the advanced menu on an existing validator node.
  2. The user inputs the node ID of the new validator.
  3. The input node ID is designated as a validator and propagated through the network.
  4. Once the new validator detects that it has been designated as a validator, it will fully sync with the entire blockchain history and start operating as a new validator node on the network.

Use Case #4: Add a candidate

- Description: An election official can add a candidate, who will be an option on the user ballot.
- User: Election Official using a validator node.
- Prerequisite cases: #1, #2, #3
- Steps:
  1. The user selects the add candidate option from the advanced menu on an existing validator node.
  2. The user inputs the candidate's name that should be added.
  3. The new candidate's name is added as a candidate on the blockchain.

Use Case #5: Add a vote

- Description: A user can fill in a ballot and propose it to a validator to be submitted to the blockchain.
- User: A Voting User

- Prerequisite cases: #1, #2, #3
- Steps:
  1. The user selects the cast vote option from the connected menu.
  2. The user loads their credentials from a file or uses the mobile companion app to link to the desktop and scan an NFC containing their voter information.
  3. The user selects and confirms a candidate option to submit their ballot.
  4. The vote is proposed to the lead validator, who authenticates the ballot. If the vote is authenticated, it is added to the blockchain.
  5. Once the vote has been added, the user receives a pop-up confirming their vote has been successfully added to the network.

Use Case #6: View Election Results

- Description: Election officials and users can view current vote tallies and results.
- User: Election Official & Voting User
- Prerequisite cases: #1, #4, #5
- Steps:
  1. The user selects the ”View Results” option from the menu.
  2. The system retrieves blockchain data and tallies current votes.
  3. Current results are displayed clearly to the user.

5.4 Companion Android App and API

5.4.1 Summary

A companion mobile app was developed as a usability feature for voters. Elector credentials comprise public-private key pairs that are too long for a user to remember or too large to store in a QR code. Instead, the elector's credentials are stored on an NFC tag. In a deployment context, the voting user will receive an NFC storing credentials from the election authority before the election. The Companion app was developed using Android Studio and is only compatible with Android devices. It targets Android API level 24.

5.4.2 Usage

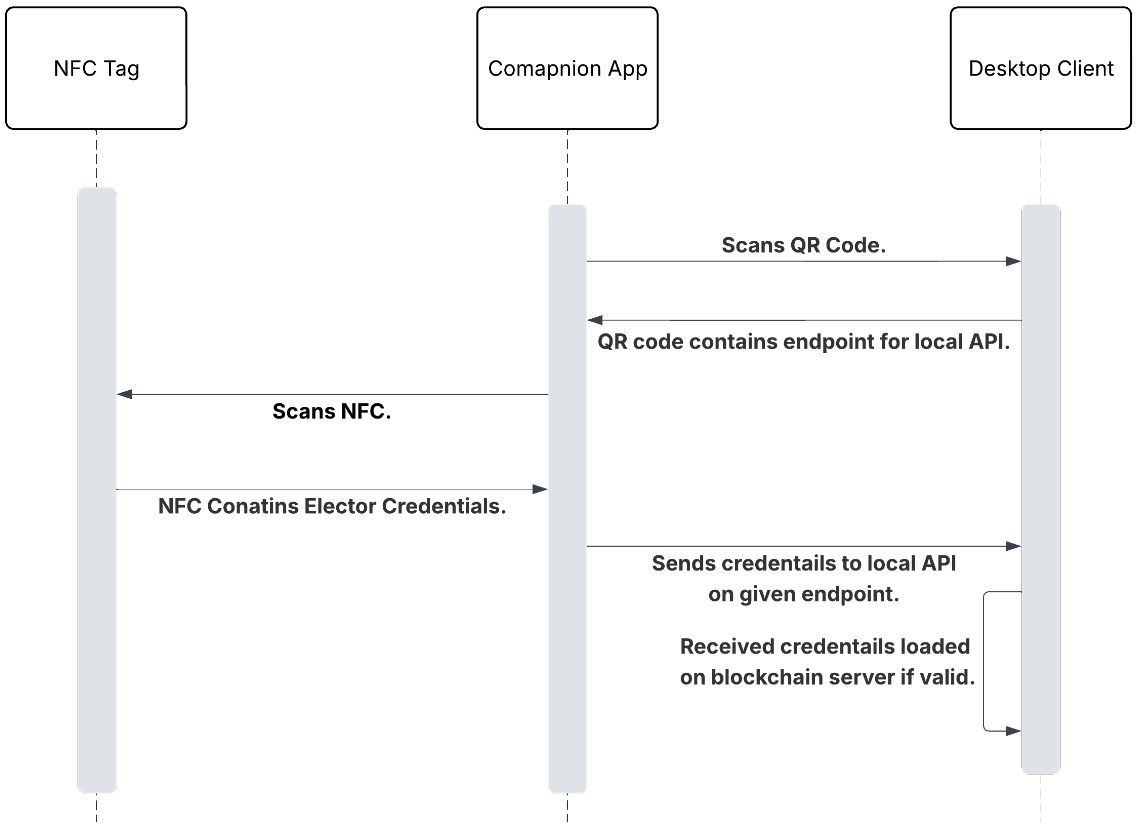

NFC Tag
Comapnion App
Desktop Client
Scans QR Code.
QR code contains endpoint for local API.
Scans NFC.
NFC Conatins Elector Credentials.
Sends credentails to local API on given endpoint.
Received credentails loaded on blockchain server if valid.

Figure 5.2: Android app sequence usage diagram.

The user can load their voter credentials using the phone app when voting. A QR code is generated on the desktop app that the companion app can scan. The endpoint for the desktop credential loading API is embedded in the QR code. The companion app can parse the QR code to see if it is formatted as a valid endpoint. If so, it saves the endpoint and allows users to scan an NFC tag. The companion app will parse the NFC to determine if it stores a valid credential. If the credentials are valid, the user can then press send, which sends a post request to the endpoint with the loaded credentials from the NFC as the request body. The API thread on the client receives this and passes the voter credentials to be loaded on the blockchain server for future vote proposals.

5.5 User-Interface

The user interface was implemented using PyQt5; each page was a separate Python class. When data from the blockchain server must be rendered, the user interface can get this from the server interface class rather than crossing a thread boundary to get it from the server class. The main.py is the main PyQt5 app class; It has a method switch_page that takes a page as input and makes it the visible page.

5.6 Networking

This section focuses mainly on the peer-to-peer network implementation, including node connections, message propagation, and the node discovery protocol. Blockchain-specific designs are covered in a separate section, with references to integration points where relevant.

5.6.1 Summary

A traditional server-client voting system introduces a single point of control and failure, which is incompatible with the project's aim of an immutable vote history. The system adopts a permissioned peer-to-peer (P2P) network of interconnected nodes to address this, which is a prerequisite for building the blockchain system. A permissioned peer-to-peer system was used because it integrates well with the consensus mechanism chosen for the blockchain, which uses a set of trusted validator nodes. In this implementation, the same blockchain validator nodes also have an increased network responsibility because they have already trusted actors through the blockchain. Because of this tight integration between the two parts of the software, several blockchain features rely directly on the network implementation.

5.6.2 Network Topology

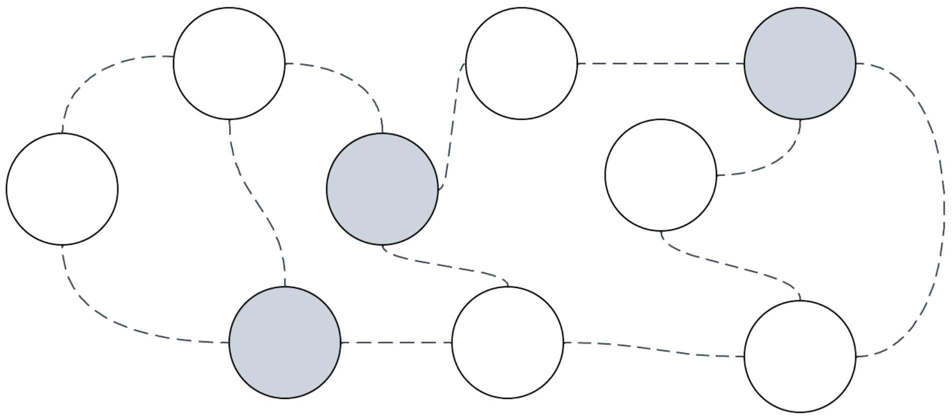


Figure 5.3: An example network topology, Validator nodes are shown in grey.

This project's implementation of a peer-to-peer network uses a gossip-based topology. This topology does not require all nodes to be directly connected as shown in the example topology, Figure 5.3. This design improves scalability and eliminates the need for nodes to know the addresses of every other node in the network. As a result, it reduces the surface for DDoS attacks and obscures the overall network topology, making Sybil attacks more difficult to carry out.

5.6.3 Validator and Bootstrap Nodes

In this system, validator nodes serve dual roles in the blockchain and network layers. The bootstrap node is the first validator node in the network and acts as the initial point of contact for other nodes joining the system. From a networking perspective, validator nodes manage a public directory that maps Node IDS to Public Keys, enabling all nodes to authenticate messages and ensure communication integrity. Additionally, validators maintain a private ledger containing host and port details for known nodes. This allows them to recommend peers to new nodes that will balance the network and avoid partitions. Importantly, a new node can initiate a connection with any validator because validators synchronise and maintain the node directory collaboratively.

5.6.4 Join Protocol

A node joins the network by opening a TCP socket and sending a request connection message to a validator. The host and port of the validator/bootstrap would be required to start this connection. This means the host and port of a validator or a bootstrap node must be pre-distributed by the election authority before an election.

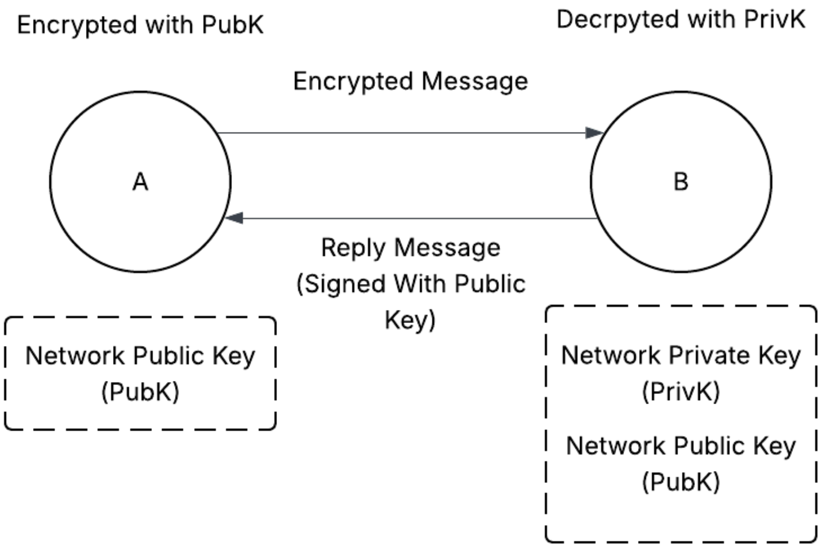

Encrypted with PubK
Decrpyted with PrivK
Encrypted Message
A
B
Reply Message (Signed With Public Key)
Network Public Key (PubK)
Network Private Key (PrivK)
Network Public Key (PubK)

Figure 5.4: Initial join message procedure.

Because this is the initial connection, no information that can be used to verify the node is stored on the public node ledger, which is a problem because the initial join message could be tampered with (man-in-the-middle attack). To address this, a public key (known to all nodes) is used to encrypt their initial connection message; the private key is only known by the validator, who can decrypt it when it receives it, as seen in figure 5.4 Once a validator receives a node's join request, it registers the public key and node ID on the public ledger. The node is then free to connect to other nodes in the network and propose data.

5.6.5 Join Suggestions

If a new node requests to join a node with too many connections, the node will refuse the connection and reply with a set of suggested connections to attempt to join instead. This is possible only through validator nodes because only they have a global node directory, which stores node IDS with their last known address.

5.6.6 Node Authentication

When nodes join the network, they generate their own RSA public-private key pair. All nodes sign their messages with their private key. Node public keys are stored on a public ledger and are available to all nodes. They can be used to verify messages from nodes. Node authentication is especially vital when verifying messages from validators that control the blockchain state. Still, authentication of all node communication adds extra resilience to the network.

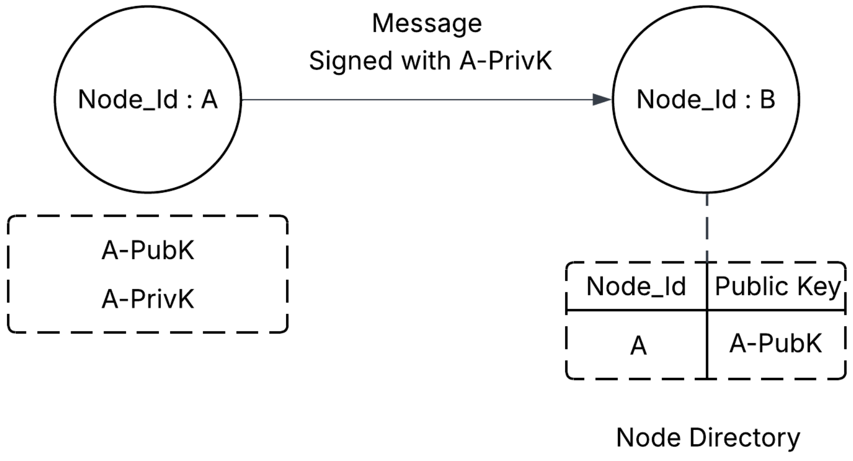


Figure 5.5: Authentication procedure for messages.

Figure 5.5 shows a message from sent from node A to node B; node A uses its private key (A-PrivK) to generate a signature for the message. Node A includes the signature in the message and sends it through the network to B. When B receives the message, it removes the signature from the message and then compares the message, the known public key of A on the ledger, and the signature to verify the message hasn't been changed and that it originates from node A.

5.6.7 Discovery Protocol

Nodes are discovered using a gossip protocol where each node gossips to its neighbouring nodes about its current connections. A target connectivity is set (currently a fixed variable in the constants) such that if a node receives information about a node that it is not currently connected to and is below its target connectivity, it will establish a new connection with the node. This approach ensures enough connectivity between nodes to allow nodes to go offline without partitioning the network.

5.6.8 Message Protocol

As previously described, nodes in the network may not have direct connections to each other, so they will have to communicate through other nodes. Nodes maintaining a direct connection may send messages directly between themselves using port and IP to indicate destination, but this is only used for connection management (discovery, gossip, or connection requests).

All non-connection management messages are broadcast to the network, using a TTL and a message ID system. Nodes propagate messages to peers until they reach the intended target Node Id or the TTL value reaches 0. The initial TTL value is calculated with the formula $\lceil \log_k(n) \rceil + 1$, where n is the number of nodes known globally and k is the minimum connections in the network each node maintains. The TLL formula will overestimate the number of jumps needed to reach a node, due to the +1; this is to ensure the message reaches even in instability

This approach has the advantage that connected nodes don't need to share their Node ID with their neighbours. That means validator nodes' hosts and ports are more challenging to obtain, decreasing their vulnerability to DDoS attacks.

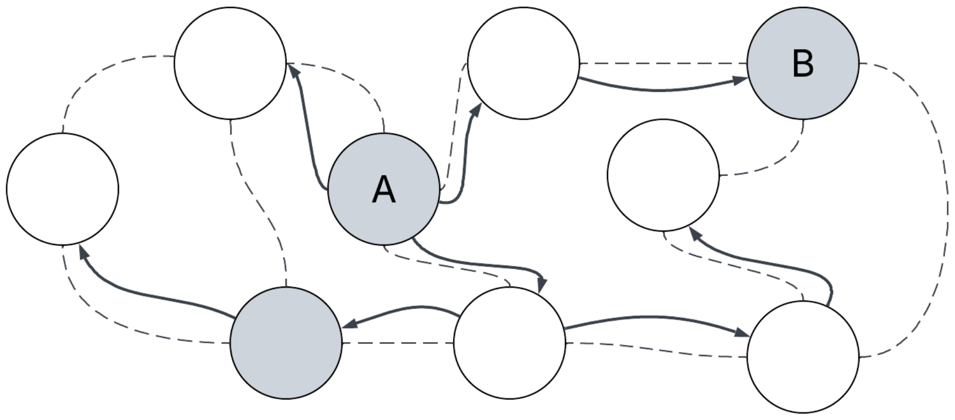


Figure 5.6: Example message flow, sent from A targeting B.

Each message also has a unique ID generated by hashing its contents to prevent it from being repeatedly processed by nodes. Once a node has seen a message, it is stored in a set and will not be rebroadcast or processed again. This means that nodes cannot return to a node from which it has come (as seen in 5.6) , meaning the message always progresses to a target node.

5.6.9 Message Structure

As mentioned in this section, all node messages are validated before processing. Messages are JSONs of 7 fields: code, sender, nonce timestamp, message_id, signature and data. Code stores the message's intent, and data may store any information needed, depending on the message's code. The other 5 fields are present for every message as metadata fields that help with verification. A signature is generated from all 6 fields that help guarantee message integrity; the message cannot be changed, or its signature will no longer match the message. The nonce is a random number so that an attacker cannot replay messages; each message is unique.

5.7 Blockchain Implementation

5.7.1 Summary

As the literature review summary describes, this project will use a Proof of Authority blockchain with off-chain computation to store an immutable record of votes and their validity. Blockchain was chosen for this project mainly because its transparency and integrity align with the P4, P5, and P6 principles.

5.7.2 Blockchain Components

Transactions

Transactions are the smallest unit in the project's blockchain. Transactions have two fields: operation and data. The operation indicates how the transaction has changed the blockchain's state; for example, "add_validator" or "add_vote". The data field can store any additional metadata about a transaction necessary to give the transaction meaning, for example, the Node ID of the new validator for a "add_validator" transaction. Metadata should also be used so that someone can verify the transaction's validity after it has been added to the blockchain. For example, any operation that requires a validator's approval should have a signed note to prove that it originates from a validator node. This is important to allow for independent auditability of the system.

Blocks

Blocks are transaction groupings assigned a hash based on their contents. Each block contains a list of transactions and the hash of the previous block in the blockchain, forming a chain of linked blocks.

Serialisation

All these components were implemented as classes in the project, and all have serialisation methods to make them easy to send over the network as JSON.

5.7.3 Blockchain Audit

A blockchain can be determined to be valid if:

1. All blocks correctly point to the previous block's hash, all the way back to the first block.
2. All transactions within each block meet the predefined transaction rules.

As this system stores all metadata about the transactions on-chain, a blockchain history can be validated at any point following the above rules.

5.7.4 Local Blockchain

Blockchain partnered with decentralisation offers an effective way to enhance integrity (P5) and transparency (P6), allowing voting users to have their machines become involved in the vote-counting process. In a blockchain system, the immutable record of votes and voting operations can form the basis for an audit and will also provide evidence of any attempted tampering. However, blockchains are weak because the public ledger of operations may be used to prove votes, violating the P4 secrecy principle. A mitigation for this is to obfuscate voter identity using hashes, or in advanced implementations, zero-knowledge proofs could be used to prove vote eligibility without revealing identity.

5.7.5 State Snapshot

This system establishes the blockchain's state by parsing all transactions. For example, if a node wants to obtain the list of validator nodes, it can parse all transactions from the first block in the chain to the head of the blockchain. However, this is computationally slow, with complexity $O(N)$, where N is the number of transactions in the entire history.

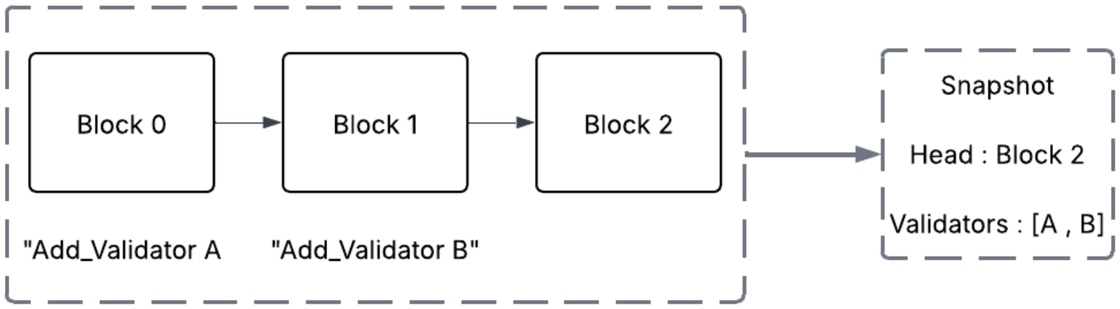


Figure 5.7: An example snapshot generated from a sequence of blocks.

To mitigate this problem, this project implements a state snapshot, a cached summary of all transactions in the network. For example, Figure 5.7 shows a sequence of blocks with some transactions shown below and their corresponding state snapshot.

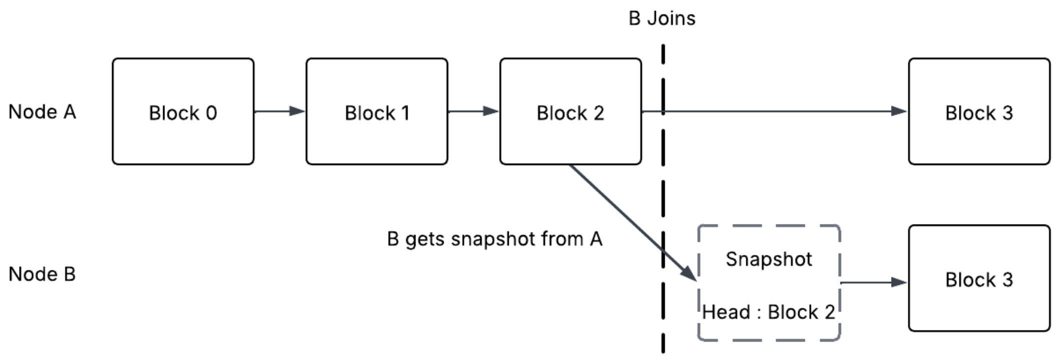


Figure 5.8: Node B joins a network syncing with Node A using a snapshot.

When nodes initially join, they receive a snapshot state from validators that they will update as they receive new blocks. This means that the new node will be aware of the blockchain's state without needing to parse the entire blockchain history. New blocks the node receives past the sync point can be added to the current state as seen in Figure 5.8.

5.7.6 Consensus Mechanism

Summary

The blockchain state is determined by the consensus and order of all its operations. The network must reach a consensus regarding the order of operations to ensure consistency. Without a consensus over the order of operations, there could be potential double-spend anomalies, where voters could use their votes twice. This project implements a proof-of-authority system that uses a set of validators to maintain consensus. A quorum of validators must agree to append a block to the blockchain. Proof of Authority was chosen for this project because it is easy to implement, doesn't need vast amounts of computational power like proof of state, and the context of the project has an authority that can already be pre-trusted to some degree.

Validator Node

The role of the validator node is to control consensus. Any node can become a validator via a transaction on the blockchain. The system gives a validator node an elevated level of trust, and it is used to help control the network (as described in the networking section) and perform consensus control.

Lead Validator

A single validator is chosen as the lead validator. The lead validator proposes blocks to the network periodically. The period during which lead validators propose blocks can be changed in the constants. The period value does not significantly affect the mechanism itself; it only influences the delay between transactions being proposed and added to the blockchain.

Proposal of Blocks

The lead validator starts the protocol proposal by broadcasting the "add_block" message to the network. When validator nodes receive the "add_block" message, they ensure;

1. The previous block matches the head of the blockchain.
2. All the transactions within the block are valid.
3. The lead validator started the block proposal, which is verified using a "submit_block" transaction with a signed nonce as its metadata.

Once a validator has verified a block, the new block is propagated to the network. When a regular node receives a new block broadcast (the original from the lead validator or the propagation from validators), it places the message into memory. It does not add the new block to its local blockchain until 70% of the validators have propagated the new block transaction, which is when the network has reached consensus. The 70% validator threshold allows for some network instability or loss of connections to validators while maintaining a high degree of confidence in the blockchain's validity. In a real-world context, the lead validator and other validator nodes will run on machines physically controlled by the authority running the election and ideally geographically separated for redundancy and security. This allows up to 30% of the validator machines to be compromised before that affects the network state.

The ”submit_block” message includes a signed nonce by the lead validator's private key to prove that it was correctly proposed by the lead validator.

Dynamic Lead-Validator

As the lead validator is a single point of failure, the blockchain allows for a dynamic reallocation of the lead validator. Other validators track the time since the last block; if a block is not added for a set period (set by a variable in the system constants), the validator will propose a block with one transaction, ”set_lead_validator” to a new node Id. As per the consensus mechanism, this new block will have to be proposed by at least 70% of validator nodes to be generally accepted by all nodes and become the blockchain state. The following procedure determines the new lead validator node ID:

- All validator IDs are ordered alphabetically in a list.
- A random number is generated between 0 and the length of the validator ID list, using the blockchain head as a seed.

- The random number indicates the position of the new lead validator ID within the order list.

This procedure is pseudorandom, meaning it is random enough not to be predictable by attackers, but it is also deterministic so that all validator nodes using this procedure will come to the same conclusion on the node_id of the new lead validator.

5.7.7 Election Actions

Elector Credentials

```
{
  "elector_id": "usqq",
  "public_key": "usQqJVyMzTzUgvYHIqSNf9dLrkh9hmR8f2R0tu5SCW8=",
  "private_key": "FgVurjx/5o63Z9YYE8YZ2800Nqi3q9L0NKHtwXUEwJQ="
}
```

Figure 5.9: Example elector credentials entry.

Each elector will have a public and private key generated with the ECDSA algorithm as seen in figure 5.9. The algorithm was chosen because it produces small key sizes, allowing the public and private keys to be loaded on the NFCS, most of which have a storage capacity of 800 bytes. Voters will only be identified by an elector's IDS derived from their public keys, not names, ensuring anonymity and protecting the vote-proof principle. The voter credentials will be loaded off-chain into memory on the current client through the API or loaded from a file in preparation for a vote proposal request. Clients cannot send vote proposals without any loaded credentials.

Electoral Roll

```
{
"data": {
  "elector_public_key":"qCFit8eqr6aFDiiNXjWPrpFLkqD07+gMyiz162sLP+U=
     "
  },
"operation":"ADD_ELECTOR"
}
```

Figure 5.10: Example add "elector_transaction".

The electoral roll is a component of the blockchain. Validators use it to determine voter eligibility to cast a ballot. First, the lead validator must designate eligible electors by adding an "add_elector" transaction on the blockchain, storing the voter's elector public key as seen in figure 5.10. Whether an elector has cast a ballot can also be derived from the blockchain by checking all votes and whether any votes have used the elector's public key for authentication. The blockchain snapshot will track used public keys, meaning validators don't need to parse the entire blockchain each time they check a vote.

Candidates

Candidates can be added to the election similarly to electors, designated with an "add_candidate" transaction. Candidates are primarily identified with a generated integer ID, which maps to a display name that can be given to the user.

Voting Procedure

When the user fills out and submits a ballot, it is sent as a proposal message to the lead validator. A proposed vote consists of the vote package and a signature. The package contains the elector's public key, the chosen candidate ID, and a nonce value used to stop replay attacks. The signature is from the vote package using the voter's private key, which is obtained from the voter's credential NFC or file. The vote is broadcast to the entire network, handled by the lead validator node, and cross-checked by other validators.

As shown in Appendix 1, a vote is valid if:

1. The public key in the vote package is registered in the electoral roll.
2. The public key has not yet been used to authenticate a vote.
3. The signature can be authenticated with the voter's public key and vote package.

Once the vote is valid, it is added to the next block in the blockchain. When other validators cross-validate new blocks, they can follow the same procedure to validate the vote.

All the data that the validator nodes used to verify the vote's validity is stored publicly on the blockchain. This allows vote transparency as any individual with the technical knowledge can audit any vote.

5.8 Key Classes and Modules

File	Description
server.py	The main class holds all the state data for the blockchain server and depends on several component classes.

File	Description
elector_actions.py	This module can change the blockchain server state or send messages to other nodes. It contains all the electoral functionality for a voting user, like proposing a vote or loading electoral credentials.
validator_actions.py	This module can change the blockchain server state or send messages to other nodes. It contains all the functions specific to validator nodes.
new_block_proccessor.py	Handles and processes incoming new block messages, adding them to the local state or propagating them if needed.
lead_validator_actions.py	This module can change the blockchain server state or send messages to other nodes. It contains all the functions that are specific to lead validator nodes.
command_runner.py	Parses the input from the user from the terminal and will carry out subsequent actions.
message_processor.py	Connection handler deals with setting up and receiving connections from other nodes.
connection_handler.py	Connection handler deals with setting up and receiving connections from other nodes.
server_events.py	Server events hold the PyQt signals used to communicate with the user interface. The signals are emitted when certain predefined events happen.
server_interface.py	The server interface stores a copy on the user interface thread for other classes to read it conveniently.
ui_event_handler.py	This class connects to user interface events and makes thread-safe calls to the blockchain server.
api_thread.py	This class contains the FastAPI HTTP server.

File	Description
main.py	This is the entry point for the desktop client and contains the MainApp class, which renders the user interface. The main.py also creates other threads to run the API server, the terminal, and the blockchain server.

Table 5.1: Key class descriptions

5.9 Communication Between Threads

This project required multiple threads because the asyncio sockets (run on the blockchain server) and the PyQt5 user interface each require separate event loops.

All threads are initialised on the main.py class. PyQt5 signals communicate from the server to the UI thread, enabling thread-safe data transfer and function calls. These signals are organised into dedicated classes to support maintainability and documentation. Each signal can carry typed parameters and is fully documented with appropriate docstrings.

Server-side events are directly connected to functions in the relevant UI classes on the main thread or the ServerInterface class. The server interface class stores a snapshot of the server's state, allowing UI components to access this data without needing to access any data stored on the server thread.

5.10 Maintainability

During Development, the focus was on developing readable and maintainable code. PEP8 style guidelines were adhered to, and docstrings were used for function and class clarity. Python's Type-hinting library was also extensively used to provide better documentation for functions and classes.

Testing

Since most of the development time was dedicated to the desktop client and blockchain implementation, the mobile app was tested manually during development. This section will outline the testing methods used for the desktop app.

6.1 Logging

A custom logging framework was developed for manual testing and later integration testing to generate a text-file log of all events on the blockchain server. There are three classes of event tags: information, warning and error. Logs are generated in a /log file and named based on the node's IP and port.

6.2 Unit Testing

Unit testing was implemented using Python's unit testing module. The tests are in the Node/tests/unit directory. Unit testing was used mainly for the pure blockchain classes and their methods. These classes didn't have direct network integration, meaning meaningful tests could be conducted on them as standalone components.

6.3 Integration Testing

A custom-built integration testing workflow was developed to allow the creation of test cases targeting individual system features. Implemented in Python using subprocesses, the framework was designed for modularity and customisation. It will enable each test case to configure the number of nodes to be launched, specify their address information, designate bootstrap nodes, and define a sequence of input commands for each node. To determine a test's completeness, the logs generated by a node are parsed and checked for required tags indicating test completion.

Evaluation and Future Work

7.1 Voting Principles

This section will evaluate the system against the six principles in table 3.1.

7.1.1 Eligibility

The system enforces eligibility via the electoral roll and will not allow double-counting votes from one public key. Impersonation In the current implementation, there is no further authentication for voter identity than private key ownership, making this system vulnerable to impersonation (P2). If voters lose their credentials storing NFC, an unauthorised person could use their vote undetectably without recourse to undo the vote. A solution to this is password-encrypting the private keys; this would mean a PIN would be needed to access the Voter's private key. This could not be implemented in the project's current iteration as encrypted EDSCA keys become too large to store on normal NFC tags. NFC tags that are large enough to store passwords require proprietary software to encode, which was not feasible to obtain for this project. Future work could implement either NFC tags with built-in cryptography or bigger NFC tags that allow for password-protected keys.

7.1.2 Coercion

This system does not protect against coercion in voting (P3) and does not allow double voting, as in the Estonian implementation. This may be especially problematic for this project's implementation, as vote secrecy is weak, and thus, voters could be targeted for their votes. In future work, there is no reason why voters could not vote multiple times to mitigate this. If modified slightly, the current system would allow for this; however it would make parsing the blockchain state harder.

7.1.3 Vote Secrecy

The fact that vote transactions are stored on the network publicly gives transparency to the system, meaning that everyone can see the voter's public key that accessed a ballot and its contents. This is a positive when building trust in the system, but it is problematic regarding the P4 secrecy principle. A person can see their transaction on the network, and if they can get their public key from the NFC, they can prove this is their vote. Although this is not simple, it still breaks the P4 secrecy principle that requires total vote secrecy. A Zero Knowledge Proof-based verification method for votes would be implemented to mitigate this in future iterations. Ideally, this would be implemented so that votes can be proven valid without giving any information on the voter's identity, while still maintaining independent verifiability.

7.1.4 Integrity and Transparency

The system developed is strongly aligned with the P6 transparency and auditability principle, as the blockchain is entirely public and auditors can trace the operations for correctness. The system is also well aligned with the P5 integrity principle, assuming an honest election authority and a deployment of enough validators, so it would be complex for an attacker to compromise the vote record.

7.2 Usability

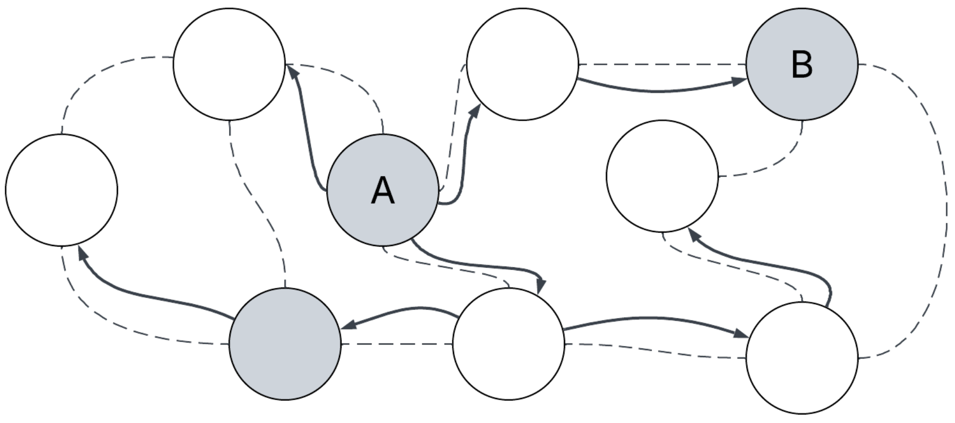
7.2.1 User-Experience

The current system is more designed as a prototype and is overwhelming and hard for the average user to understand. A more straightforward blockchain client or a wrapper of the current implementation could be developed. This would abstract the process of joining the network and purely contain a vote-casting function for the end user. Alongside a wrapper, more functionalities could be included, such as a web client for the blockchain that easily allows users to search the blockchain history and find their votes and other transaction information. A complete API could be added to the blockchain client for better connectivity with other apps. This would allow other apps to integrate with all the blockchain operations the client supports and facilitate the development of new companion apps.

7.2.2 Accessibility

The system has currently only been tested on Windows 11 (the development environment). To ensure usability and compatibility, testing it on other operating systems and hardware configurations and developing a custom ported version if needed, is essential.

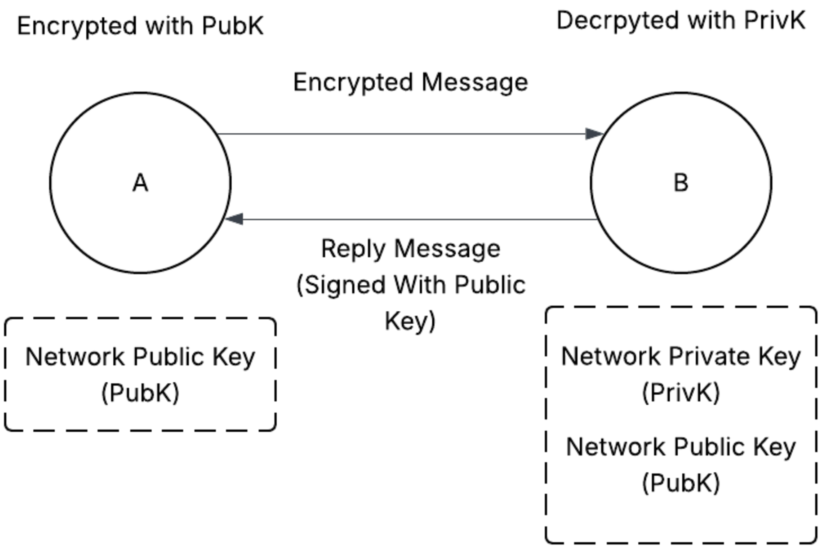
7.3 Message Protocol



Repeated Figure 5.6.

As seen in Figure 5.6, the message protocol is highly wasteful. Nodes indiscriminately forward messages without considering the direction of the intended recipient, as they do not have the required topology knowledge. As a result, many messages propagate through parts of the network that are irrelevant to the target node, consuming bandwidth resources. This isn't easy to mitigate in future work while maintaining decentralisation; a possible mitigation would be implementing routing tables maintained by the validators, allowing them to suggest the most efficient routes without exposing the topology.

7.4 Join Protocol


Encrypted with PubK
Decrpyted with PrivK
Encrypted Message
A
B
Reply Message (Signed With Public Key)
Network Public Key (PubK)
Network Private Key (PrivK)
Network Public Key (PubK)

Repeated Figure 5.4.

As shown in Figure 5.6, the join protocol is not fully secure. Although the initial message is encrypted, the reply message is only signed. Since Node A does not yet have access to Node B's public key at this stage, the reply message could potentially be tampered with. The network's private and public keys are also hardcoded into the implementation. In future development, a secure method should be implemented to allow joining nodes to safely obtain and verify reply messages. Furthermore, there should be a mechanism for validator nodes to access the network's private key without distributing it to all nodes.

7.5 Validator Removal

In the current implementation, validators are assumed to remain honest and continuously active. However, nodes may become compromised or unresponsive in practice. The system currently lacks a mechanism for dynamically detecting and removing such validators. Future work could incorporate a fault detection mechanism based on response rates and peer-reported misbehaviour. Similar to the logic used for the dynamic lead validator assignment, a removal protocol should remove nodes detected as offline or compromised.

7.6 Dynamic Lead-Validator

The lead validator is manually defined and then dynamically reassigned based on activity in proposing blocks. However, this mechanism does not account for misbehaviour, and the system maintains a single point of compromise. If a lead validator proposes malicious or invalid blocks, the lead validator reassignment protocol should be triggered, and the lead validator should lose validator status.

7.7 Testing

Generally, the system met most of the project's functional requirements, successfully implementing the blockchain client with the mobile app, and most of its functionality worked. However, some non-functional requirements have yet to be verified due to testing limitations that make it hard to quantify the system. Due to the project timescale limitations, a system could not be implemented so that all metrics could not be systematically evaluated. Future work should focus primarily on systematic load testing to uncover bugs that may only surface under high-stress conditions. Secondly, usability testing would be valuable for identifying accessibility flaws and informing the user experience improvements outlined earlier in this section. Implementing these testing methods will help guarantee the system meets non-functional requirements, which currently cannot be guaranteed.

7.8 Requirements Evaluation

This section will evaluate how well the system meets the requirements, categorizing each as Complete, Partially Complete, or Not Complete.

ID	Requirement	Completed
4.1 Functional Requirements		
4.1.1 Networking		
FR-01	The system must operate over a semi-centralized peer-to-peer network.	Complete.

ID	Requirement	Completed
FR-02	Nodes must be able to discover and connect to other nodes via a gossip-based discovery protocol.	Complete.
FR-03	A join protocol must allow new nodes to securely authenticate and enter the network.	Complete.
FR-04	Messages between nodes must include digital signatures and nonces to prevent tampering and replay attacks.	Complete.
FR-05	Nodes must reject malformed, unsigned, or invalid messages.	Complete.
FR-06	Validator nodes must maintain a shared directory of known nodes and their public keys.	Complete.
4.1.2 Blockchain		
FR-07	A designated lead validator must propose blocks periodically (e.g. every 5 minutes).	Complete.
FR-08	New blocks must only be accepted when validated by at least 70% of validators.	Complete.
FR-09	All nodes must maintain a local copy of the blockchain or a valid snapshot.	Complete.
FR-10	All transactions must include enough metadata to allow later validation (e.g. digital signatures, origin).	Complete.
FR-11	The system must support reallocation of the lead validator in case of failure.	Partially Complete.
4.1.3 Voting Logic		
FR-12	Electoral roll must be maintained on-chain, with voters identified by public key.	Complete.
FR-13	Only electors on the electoral roll may submit a ballot, and each elector may vote only once.	Complete.
FR-14	Votes must be verifiable cryptographically, signed using the voter’s private key.	Complete.

ID	Requirement	Completed
FR-15	All vote metadata (e.g. signature, candidate ID, public key) must be stored on-chain for auditability.	Complete.
FR-16	The system must prevent double voting using public key tracking.	Complete.
FR-17	Votes must be verifiable by third parties but without linking identity to choice (see limitations under P4).	Complete.
4.1.4 User Interface & Interaction		
FR-18	A graphical interface must allow users to: Connect to the network. Submit votes. View blockchain status and election results.	Complete.
FR-19	A companion mobile app must allow scanning of NFC-stored voter credentials and submission to the desktop client.	Complete.
4.1.5 System Audit & Testing		
FR-20	All system events (e.g. block additions, vote submissions, errors) must be logged.	Complete.
FR-21	The system must implement automated unit and integration testing workflows.	Partially Complete.
FR-22	The system should generate logs for debugging and intergration with testing workflows.	Complete.
4.2 Non-Functional Requirements		
4.2.1 Security & Integrity		
NFR-01	All communication between nodes must be encrypted and signed.	Complete.
NFR-02	Blockchain data must be immutable and cryptographically verifiable.	Complete.

ID	Requirement	Completed
NFR-03	The system must tolerate up to 30% validator node failure.	Complete.
NFR-04	Private keys must be handled securely and not exposed in plaintext during runtime.	Partially Complete.
4.2.2 Privacy & Anonymity		
NFR-05	Public keys should be unlinkable to real-world identities where possible.	Not Complete.
NFR-06	The system must allow auditability without exposing vote origin or contents together.	Not Complete.
NFR-07	The design must consider, but is not yet resilient against, coercion and impersonation attacks.	Not Complete.
4.2.3 Usability		
NFR-08	Voting interaction should take less than 2 seconds under normal network conditions.	Not Complete.
NFR-09	The UI must be responsive and support keyboard-/mouse interaction.	Complete.
NFR-10	The interface must adapt to different screen sizes and follow basic accessibility guidelines.	Complete.
NFR-11	Voter onboarding (e.g. scanning credentials, submitting vote) should require minimal technical knowledge.	Not Complete.

Table 7.1: Requirements Evaluation for Voting Blockchain System.

Conclusion

8.1 Ethical Considerations

With these suggestions for future work, I believe this system could be realistically implemented in the real world. However, it's important to note that this software project was mainly designed as a proof of concept. A voting system is a high-consequence system; any failure would have disastrous consequences. Therefore, software of this nature must be peer-reviewed, certified, and rigorously tested. This voting system would be more appropriate for smaller company elections that require security, but would not have nation-level adverse effects if the system failed. The risks of using Internet voting for political elections currently outweigh the benefits. It is disturbing that crucial political polls in the UK have already been held online. In 2022, the Conservative Party indirectly elected a new prime minister in part using online voting [13]. The software was not independently audited, but cybersecurity experts doubted its security. I believe paper voting offers a more secure election without any real disadvantages. Overall, it remains essential to continue researching Internet voting to improve general security. Still, Internet voting has a place in political elections and should only be used where it is impossible to vote on paper, for example, overseas voters. Personal development

8.2 Personal Development

This project has been an extremely valuable learning experience for me. From a project management perspective, I've gained insight into planning and handling unforeseen challenges; I wish I had spent more time on the planning stages, precisely detailing what work would be done when, rather than a vague plan. It also taught me to manage the expectations of my system; I could work for hours tweaking a small part of a subsystem while many features weren't yet implemented. I learnt that an imperfect implementation is better than none, and to balance my expectations of project progress. From a technical standpoint, I've developed my knowledge of blockchain systems, networking, and cryptography. I've not taken any cryptography or networking modules, so I found these elements most interesting to implement. Studying networking for this project has deepened my understanding of all internet technologies. On a more personal level, I've learned the importance of resilience and sustainable pacing. I've realised that constantly pushing yourself to achieve short-term goals is unhealthy and that long-term productivity can be helped by taking breaks and focusing on other things that the project.This is something I struggled with, as seen on my git log and commits, there was a period of a few weeks where few commits were made. However, I got back into the project with the same enthusiasm I initially had and managed to keep progressing with development.

Bibliography

[1] J. Katz and Y. Lindell, *Introduction to Modern Cryptography, Second Edition*, 2nd ed. Chapman & Hall/CRC, 2014.

[2] H. Guo and X. Yu, “A survey on blockchain technology and its security,” *Blockchain: Research and Applications*, vol. 3, no. 2, p. 100067, 2022, iD: 778493. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2096720922000070

[3] B. Watt, “Implementing electronic voting,” Electoral Commission, Tech. Rep., 2002. [Online]. Available: https://www.electoralcommission.org.uk/sites/default/files/electoral_commission_pdf_file/Implementationofe-votinglegalannex_6722-6270__E__N__S__W__.pdf

[4] M. S. Peelam, G. Kumar, K. Shah, and V. Chamola, “Democracyguard: Blockchain-based secure voting framework for digital democracy,” *Expert Systems*, vol. 42, no. 2, p. e13694, 2025. [Online]. Available: https://doi.org/10.1111/exsy.13694

[5] A. Pandey, M. Bhasi, and K. Chandrasekaran, “Votechain: A blockchain based e-voting system,” in *2019 Global Conference for Advancement in Technology (GCAT)*, 2019, pp. 1–4, iD: 1.

[6] D. Springall, T. Finkenauer, Z. Durumeric, J. Kitcat, H. Hursti, M. MacAlpine, and J. A. Halderman, “Security analysis of the estonian internet voting system,” ser. ACM Conferences. New York, NY, USA: ACM, Nov 3, 2014, pp. 703–715. [Online]. Available: http://dl.acm.org/ft_gateway.cfm?id=2660315&type=pdf

[7] S. Kardaş, M. S. Kiraz, M. A. Bingöl, and F. Birinci, “Norwegian internet voting protocol revisited: ballot box and receipt generator are allowed to collude,” *Security and Communication Networks*, vol. 9, no. 18, pp. 5051–5063, 2016, 11. [Online]. Available: https://doi.org/10.1002/sec.1678

[8] R. Computing, “Pyqt5,” 2024, version 5.x. [Online]. Available: https://pypi.org/project/PyQt5/

[9] C. A. T. Python, “cryptography,” 2024, provides cryptographic recipes and primitives to Python developers. Version 42.x. [Online]. Available: https://pypi.org/project/cryptography/

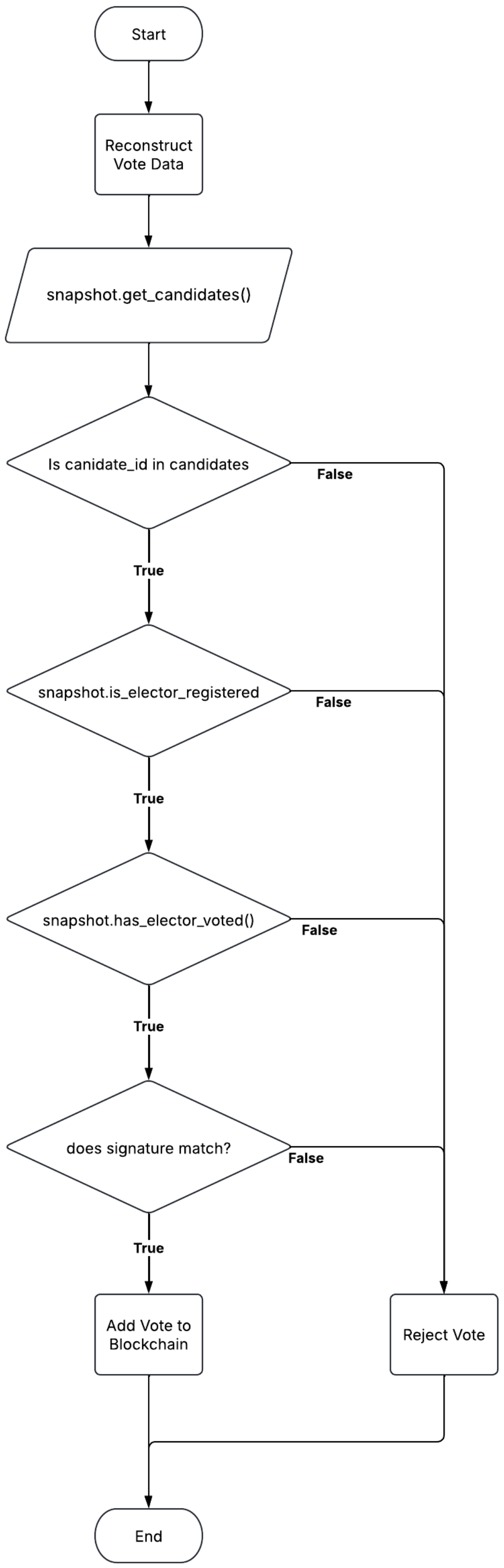
[10] L. Loop, “qrcode,” 2024, python library for generating QR codes. Version 7.x. [Online]. Available: https://pypi.org/project/qrcode/

[11] S. Ramírez, “Fastapi,” 2024, fast web framework for building APIs with Python 3.7+ based on standard Python type hints. Version 0.110.x. [Online]. Available: https://pypi.org/project/fastapi/

[12] Encode, “Uvicorn,” 2024, aSGI web server implementation for Python. Version 0.27.x. [Online]. Available: https://pypi.org/project/uvicorn/

[13] B. Quinn, “Experts question security of online vote to pick tory leader,” 2022. [Online]. Available: https://www.theguardian.com/politics/2022/aug/22/experts-question-security-online-vote-pick-tory-leader

Appendix



Appendix 1: A flow-chart diagram of the verify logic procedure.