TITLE OF PROJECT

CENG 3550, DECENTRALIZED SYSTEMS AND APPLICATIONS

Mandana Zooyousefin
180709711@posta.mu.edu.tr

Wednesday 15th January, 2025

Abstract

This study explores how decentralized voting systems provide a reliable and transparent election environment. Using blockchain technology, the system aims to eliminate the possibility of tampering with election results, thus increasing trust in the voting process. The study discusses the fundamental principles of the system, comparisons with existing works in the literature, and the prototype implementation of the proposed system.

1 Introduction

Modern election systems are often criticized for security vulnerabilities and risks of manipulation. Decentralized voting systems have been developed to address these challenges. Blockchain technology provides an ideal infrastructure for these systems with its immutability and transparency features. This study examines how a decentralized voting system can be designed and highlights its advantages over existing methods.

2 Fundamentals

Blockchain Technology:

Blockchain is a distributed ledger technology where data is cryptographically chained and does not rely on a central authority.

Smart Contracts:

Smart contracts are programs that run on a blockchain and execute automatically when predefined conditions are met.

Decentralized Systems:

Unlike traditional centralized systems, decentralized systems distribute data across multiple nodes and update them simultaneously.

3 RELATED WORKS

Previous studies in this area include Nakamoto's proposal on blockchain infrastructure [1] and the evaluation of the Quorum platform by Baliga et al. [2]. However, existing works mainly focus on general blockchain applications rather than election security. The proposed system is specifically optimized for election scenarios.

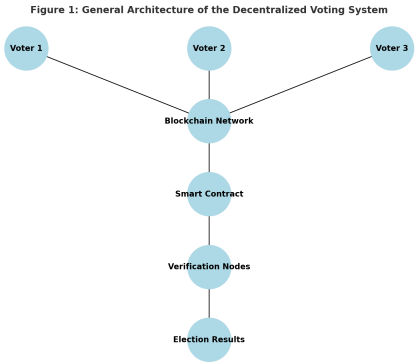

Figure 1: General Architecture of the Decentralized Voting System

4 SYSTEM PROPOSAL

The proposed system consists of three main modules:

Registration Module: A decentralized authentication system for voter identity verification. Voting Module: A smart contract that ensures votes are securely stored on the blockchain. Verification Module: A mechanism to neutrally verify election results. Figure 1 presents the overall architecture of the system.

5 IMPLEMENTATION

5.1 Overview

The decentralized voting system is implemented using Python to simulate the core functionalities of a blockchain-based voting process. The system allows voters to securely cast their votes, ensures that each voter can vote only once, and records each vote as a cryptographic hash to simulate a blockchain. The implementation is modular, enabling easy addition of candidates, voters, and future system enhancements.

5.2 Scenario Description

This implementation demonstrates a simple election process with three candidates: Alice, Bob, and Charlie, and three voters: Voter1, Voter2, and Voter3. The voting system operates through the following steps:

Implementation Steps

Candidate Registration

The administrator registers the candidates for the election. In this case, *Alice*, *Bob*, and *Charlie* are added as candidates.

Voter Registration

The administrator registers eligible voters in the system. *Voter1*, *Voter2*, and *Voter3* are registered for this scenario.

Start of Voting

The administrator starts the voting session. During this phase, voters can cast their votes.

Vote Casting

Each voter casts their vote for their preferred candidate. The system ensures that:

- Each voter can vote only once.
- Votes are recorded securely with a cryptographic hash.

End of Voting

Once the voting session ends, the administrator closes the voting process. Results are then calculated based on the votes recorded during the session.

Result Visualization

The final results are presented in a bar chart to show the distribution of votes among the candidates.

Detailed Steps

Step 1: Adding Candidates

The system uses the `add_candidate()` method to add *Alice*, *Bob*, and *Charlie* as candidates. Each candidate is stored in a dictionary with their names as keys and their vote counts initialized to zero.

```
voting_system.add_candidate("Alice")
voting_system.add_candidate("Bob")
voting_system.add_candidate("Charlie")
```

Step 2: Registering Voters

The system registers *Voter1*, *Voter2*, and *Voter3* using the `add_voter()` method. Voters are stored in a dictionary to track whether they have voted.

```
voting_system.add_voter("Voter1")
voting_system.add_voter("Voter2")
voting_system.add_voter("Voter3")
```

Step 3: Starting the Voting Session

The administrator initiates the voting process using the `start_voting()` method. This opens the voting session, allowing voters to cast their votes.

```
voting_system.start_voting()
```

Step 4: Casting Votes

Each voter casts their vote for their preferred candidate using the `vote()` method. The system records the votes securely as cryptographic hashes, which simulate a blockchain.

```
voting_system.vote("Voter1", "Alice")
voting_system.vote("Voter2", "Bob")
voting_system.vote("Voter3", "Alice")
```

If a voter attempts to vote more than once, the system blocks the action and issues a warning.

Step 5: Ending the Voting Session

Once all voters have cast their votes, the administrator ends the voting session using the `end_voting()` method. After this, no further votes can be cast.

```
voting_system.end_voting()
```

Step 6: Viewing Results

The `get_results()` method is used to display the vote counts for each candidate. In this scenario:

- Alice receives 2 votes.
- Bob receives 1 vote.
- Charlie receives 0 votes.

```
voting_system.get_results()
```

The results are also visualized using a bar chart generated by the `plot_results()` method:

```
voting_system.plot_results()
```

6 RESULTS and DISCUSSION

Voting Results

The voting results for this scenario are as follows:

- **Alice:** 2 votes
- **Bob:** 1 vote
- **Charlie:** 0 votes

Alice is declared the winner. The vote distribution is shown in the bar chart below:

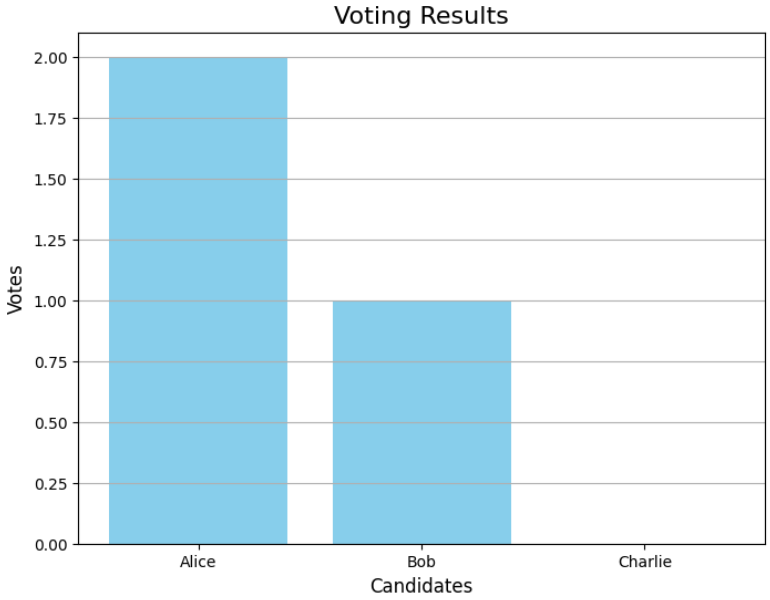

Figure 2: Vote Distribution Bar Chart

Blockchain Simulation

Each vote is recorded as a unique cryptographic hash to simulate the behavior of a blockchain. For example:

- Voter1's vote for *Alice* generates a hash: `2c26b46b68ffc68ff99b453c1d304134134e0f4c3e7fdf19fb37b0f99d04`

These hashes ensure data integrity and prevent tampering.

Key Features

- **Security:** Each vote is securely recorded with a cryptographic hash.
- **Anonymity:** The system does not store voter preferences directly, ensuring voter anonymity.
- **Visualization:** Results are presented graphically for better understanding.
- **Integrity:** The system prevents duplicate voting and ensures that votes cannot be altered.

7 CONCLUSION

This Python-based decentralized voting system provides a simple yet effective simulation of a secure and transparent voting process. While this implementation is not a full blockchain solution, it effectively demonstrates the key principles of decentralized voting systems. Future enhancements could include integration with actual blockchain frameworks like Ethereum or Hyperledger.

ACKNOWLEDGEMENT

References

[1] Nakamoto, Satoshi. Bitcoin: A peer-to-peer electronic cash system. Manubot, 2019.

[2] Baliga, A., Subhod, I., Kamat, P., amp; Chatterjee, S. (2018). Performance evaluation of the quorum blockchain platform. arXiv preprint arXiv:1809.03421.

[3] Fan, Chao, et al. "Disaster City Digital Twin: A vision for integrating artificial and human intelligence for disaster management." International Journal of Information Management (2019): 102049.

[4] Al-Saqaf, Walid, and Nicolas Seidler. "Blockchain technology for social impact: opportunities and challenges ahead." Journal of Cyber Policy 2.3 (2017): 338-354.