

Oliver Aschwanden

Prof. Thomas H. Austin

CS 168 - Blockchain and Cryptocurrencies

5/16/2022

E-Voting application on the Ethereum blockchain Project Report

Introduction

Voting and Election are the backbone of many important decision a group or society needs to make. Even though most of our activities have been brought into the digital world voting is not one of them and many votes are still conducted on a paper basis. This project explores the possibility to move these voting mechanisms into the digital realm with the help of blockchain technology. After some literature review a prototype for this idea was created with Solidity and web3.js.

Background

As the consequences of certain votes can be very impactful it is important certain requirements are fulfilled when creating a voting system. The Equal Justice Foundation has created 6 basic principles which were used as a guideline when creating this project. They are the following: secret ballot, one man, one vote, voter eligibility, transparency, accuracy, and reliability. (Charles E. Corry) The topic of digital voting is not new and has been around since the beginning of the 2000s. But the rise of blockchains made the idea of digital voting popular and feasible again. There are different rules for different votes, some of them are based on the one man one vote principle others give the voter with a bigger stake more power. Further the

way the basic principle of voting are different. Some rely on blind signatures while other use homomorphic encryption to anonymize the votes.

Smart Contract

The smart contract was written in Solidity as this was the language already somewhat familiar from class. The central piece of the smart contract is a voter directory which contains all the public keys of the eligible voters and keeps track if they already voted on this proposal or not. The mapping was chosen because of the performance benefits compared to an array.

```
struct voter {  
    address voterAddress;  
    bool voted;  
}  
  
mapping(address => voter) public voterDirectory;
```

As mappings in Solidity are not iterable four counter variables were used to tally up the votes during the election. All the variables except the countResult are private. This was done so that the voters can not see the current result of the vote and be potentially be influenced by it.

```
uint private countResult = 0;  
uint public finalResult = 0;  
uint public totalVoter = 0;  
uint public totalVote = 0;
```

The contract has two states (InProgress, Concluded) this was done so that voting is only possible during certain times and prevent the voters being influenced by the current result of a vote.

Modifiers were another feature that was used in this smart contract. In this project two modifiers were created they simplified the check for the owner of the smart contract and verified that the function was only able to be called when it was in the correct state.

Frontend with web3.js

To interface with the smart contract the web3.js library was used. There are 2 separate views for managing and voting on proposals. The main view is to setup a vote, add eligible voters to the smart contract, change the state of the voting process and show the results.

eVote Admin Panel

New Proposal

or

Voting Address

Editing Proposal: Approval of Budget 2022
Voting Address: 0xe443675437936A4f7ee05b2Ab8607500F81713FF

Addresses(comma-separated)

0x2aF1408aACf96fD117eeBc148B11767B354cb37;0xAe750a1154F072
A130bb90609c05Ff1f6bDFA7e6;0xfd03B687D6741aE0FeAE836c294526
49a3de0478;0xa8c02Ae7f035650b93a2F6364A82CF036111fDC5;

Total Voters: 4
Number of Votes: 2
Final Result: 2 Yes, 0 No

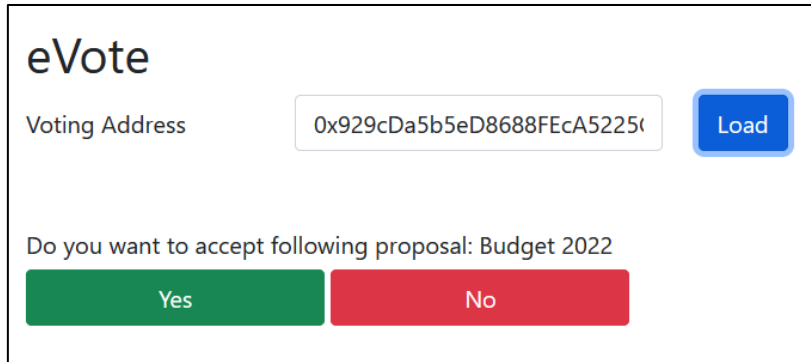
After the main page is loaded web3.js is initialized and connected to a MetaMask account. By default it will always take the first account in MetaMask, but the accounts can be changed in MetaMask itself.

```

window.web3 = new Web3(window.ethereum);
let address = await window.ethereum.request({method: 'eth_requestAccounts'});
from = address[0]

```

All the calls to the smart contract are executed in an asynchronous way and are handled with promises.



The screenshot shows a web interface titled 'eVote'. It features a 'Voting Address' field containing the hexadecimal string '0x929cDa5b5eD8688FEcA5225c'. To the right of the field is a blue 'Load' button. Below this, a message asks 'Do you want to accept following proposal: Budget 2022'. At the bottom of the interface are two large buttons: a green 'Yes' button and a red 'No' button.

On the voting page a voter can look up a current vote with the contract address he got from the voting admin and decide if he wants to approve

or decline the proposal. He will get an alert when he is trying to vote a second time .

Conclusion and Discussion

To conclude, the project was successful and it could be proven that electronic voting on the Ethereum blockchain are possible. Unfortunately two of the established voting principles are not fulfilled completely. Especially the secret ballot principle needs some more work. As the owner of the smart contract can still view what each voter choose. One possibility would be to use homomorphic encryption to encrypt the vote value before sending it to the blockchain, and then do the calculation on the encrypted value which is still possible because of its homomorphic property. Secondly to make sure that the correct person is voting, and not a malicious actor with the voters private key, an additional verification step off the blockchain is required. This could be a verification of a biometric property, which is verified before the vote is cast.

Another point of improvement is the scalability of this solution. The smart contract is being run a single line after another, which will most likely lead to bottlenecks during a big vote.

This project was, albeit time consuming, very interesting and improved my knowledge about blockchains and how to build decentralized applications with it.

Works Cited

Charles E. Corry, Ph.D. *ejfi.org*. 14 June 2009. <<http://www.ejfi.org/Voting/Voting-12.htm>>.