Electronic voting using Ethereum Smart Contracts (PD2)

Lucas Bordignon, Matheus Bittencourt, Vinicius Macelai

November 23, 2018

1 Introduction and Functional Requirements

As shown on the previous report, we have a large amount of disadvantages when talking about physical elections, as the approach used in Brazilian's voting system. The two stronger worries when talking about such kind of system are the warranty of elections result, which is done currently just by the Brazilian Government, and the overall cost of the entire process, which is massive and hard to track as the country in discussion is very large and heterogeneous.

The proposed solution to such problem consists in host the electoral process inside a distributed blockchain, granting and solving some of the problems shown through the years of the current system.

When defining the project, some functional requirements were defined, given as following:

- **Record a vote**: People can vote as many time as they wish, the last vote should be the only one computed.
- **Authentication to vote**: Only authorized people can vote, the address should be on a white list.
- **Possibility to audit the votes**: The system should be verify every vote and that they are supposed to be there.
- **Vote credibility**:Voter can be ensured that her/his vote has been counted.
- **Integrity of the poll**: this network ensures the integrity of the Blockchain by securing it so that no adversary can alter entries.
- **Web interface to see and vote**: along with the smart contract, that should be a web application to see and vote.

2 Class Diagram

The class diagram shown at Figure 1 exposes the internals and definitions of the smart contract under development. It will be explored in more details on the following sections.

2.1 Variables

- **candidateList**: A map that links the name of each candidate to the number of the party;
- **votesReceived**: A map that links the number of each candidate to the amount of votes received;
- **whiteList**: A map that links the hash of a secret from an individual voter to a candidate number, representing that the voter voted for this candidate. The secret could anything that is hashed by the contract;
- **initTime**: Datetime to start the election. Initially not defined, can represent a Unix Timestamp or the reference for a specific block;

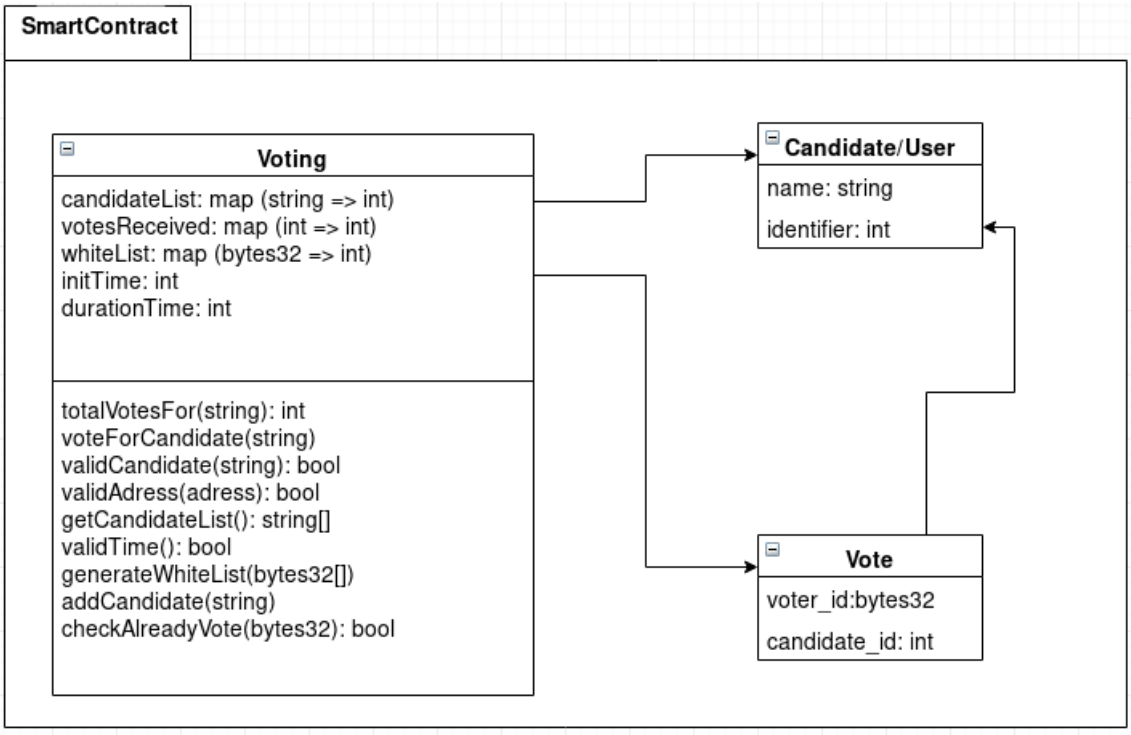

Figure 1: Class Diagram that exposes the public API of the proposed contract

- **durationTime**: Duration time of the election. Can be defined as a Unix Timestamp or the reference number of some block;

2.2 Functions

- **totalVotesFor**: Only to see the number of votes to a specific candidate.
- **voteForCandidate**: Vote for a specific candidate, in this function it will be required to be a valid candidate, a valid adress, valid time as well.
- **validCandidate**: Check if a specific candidate is in the candidate list.
- **validAdress**: Check if an address is on the whitelist.
- **getCandidateList**: Return the list of all candidates.
- **validTime**: Check if the current time is between initTime and the duration time.
- **genererateWhiteList**: Populate the list of the hashes of the secrets of each voter.
- **addCandidate**: Add a candidate to the candidate list.
- **checkAlreadyVote**: Check if an address already voted in this election, if so, only the last vote should be counted

3 Example code

The following block shows a partial implementation of the smart contract, showing through the functions *voteForCandidate()*, for example, how the voting interface work on the current project.

We are using the language solidity [1], an object-oriented language for implementing smart contracts with focus on the Ethereum Virtual Machine (EVM).

```
pragma solidity ^0.4.6;

contract Voting {

  mapping (bytes32 => uint8) public votesReceived;
  bytes32[] public candidateList;

  function Voting(bytes32[] candidateNames) {
    candidateList = candidateNames;
  }

  function totalVotesFor(bytes32 candidate) returns (uint8) {
    require(validCandidate(candidate));
    return votesReceived[candidate];
  }

  function voteForCandidate(bytes32 candidate) {
    require(validCandidate(candidate));
    votesReceived[candidate] += 1;
  }

  function validCandidate(bytes32 candidate) returns (bool) {
    for(uint i = 0; i < candidateList.length; i++) {
      if (candidateList[i] == candidate) {
        return true;
      }
    }
    return false;
  }

  function getCandidateList() constant returns (bytes32[]) {
    return candidateList;
  }
}
```

4 Questions

1. Should the Brazilian Election run on this platform?

   For the reason of sovereignty, a full election should not be run fully on Ethereum Blockchain. The main reason would be the need to anonymity, which means that the contract should ensure that the vote is secret, which is not the main focus of our proposal.

   An appropriate way to use a blockchain in the Brazilian Election case would be related to use along with the current system, where you could verify the result on the blockchain. It's a way to grant auditability of the election result, being open to the community to exert it.

2. What about to use a private blockchain?

   We don't think that this would a good idea, because when you use a private blockchain you lose most part of the desirable proprieties of a open blockchain, as being decentralized and collaborative, allowing it to be easily auditable.

   There will be a central (single or a small group of) point of failure and this centralized system doesn't ensure that the data is immutable. Moreover, this system could censor some voters, where in a open blockchain you can't control all nodes to achieve this kind of power. It grants a higher reliability on the system.

5 Potential problems

The main problem with this scheme, is the need to trust who generated the contract and populate the whitelist. Since the whitelist is a list of hashes of secrets and they are hashed before put in the contract, the person/organization who populated know everyone's secret and therefore

can vote being this person. But everyone can verify whether your vote is right, so, it's possible to fraud voting for someone else, but the person will know.

For an election with multiples roles involved, like the Brazilian one, where you need to vote for President, Governor and so on... You can instantiate several contracts, one for which role. The main problem to unify this on a single contract is the candidate list, since every state has his own.

The Ethereum Blockchain is the most appropriated one, given the reasons we discussed already, therefore will be a cost for each transaction/vote. The organization can't send the ether needed to every voter, due to the fact that they will be able to link every address to a vote, so the workaround for this is that they send you fiat money and you buy your own ether.

References

[1] **Solidity language -** https://solidity.readthedocs.io