Blockchain CS-2361
12th April 2020

Harpreet Virk
Varsheel Deliwala
Vir Jhangiani

HyperVoter : A Distributed E-Voting system implemented on Hyperledger Fabric

INTRODUCTION

Existing voting systems, especially for elections, are usually carried out in a way where the participating members - Election Commission, Candidates, and Voters don't trust each other completely. This leaves the system vulnerable to attacks not only from outside adversaries, but also by the system participants themselves.

Traditional paper-ballot based voting systems are vulnerable to malpractices such as election riggings and forced ballot. The existing voting system relies on third party tally agents to count votes resulting in inaccuracies in the tally and even misreporting. Moreover, the fact that the voter turnout has decreased in the recent years suggests that these may not be the most effective and convenient systems going forward.

Online voting systems have often been considered as potential alternatives to the traditional methods of voting. In this paper, we will propose to devise a Blockchain based E-Voting System that will try to address these issues of security, privacy, convenience and trust.

MOTIVATION FOR BLOCKCHAIN BASED VOTING SYSTEM

E-Voting systems have often been rejected in practical scenarios over concerns regarding lack of security, privacy, and trust due to their susceptibility to hacking. Malicious users can manipulate the election results by tampering with the way in which the identity of voters is verified, or the way in which the votes are submitted and counted.

In our E-Voting system, we aim to tackle these issues by ensuring the following:

- The identity of every voter is verified and remains anonymous throughout the process.
- Every voter should be eligible to vote, and pre-authorised to do so.
- No one should be able to cast their vote more than once in the same election (double spend problem).
- Eliminating a third-party responsible for counting votes.
- Votes should not be modifiable once cast.

- Election results should be publicly visible to the entire network and vote transactions should be openly available for verification to ensure transparency (Voter's identity will still be anonymous)

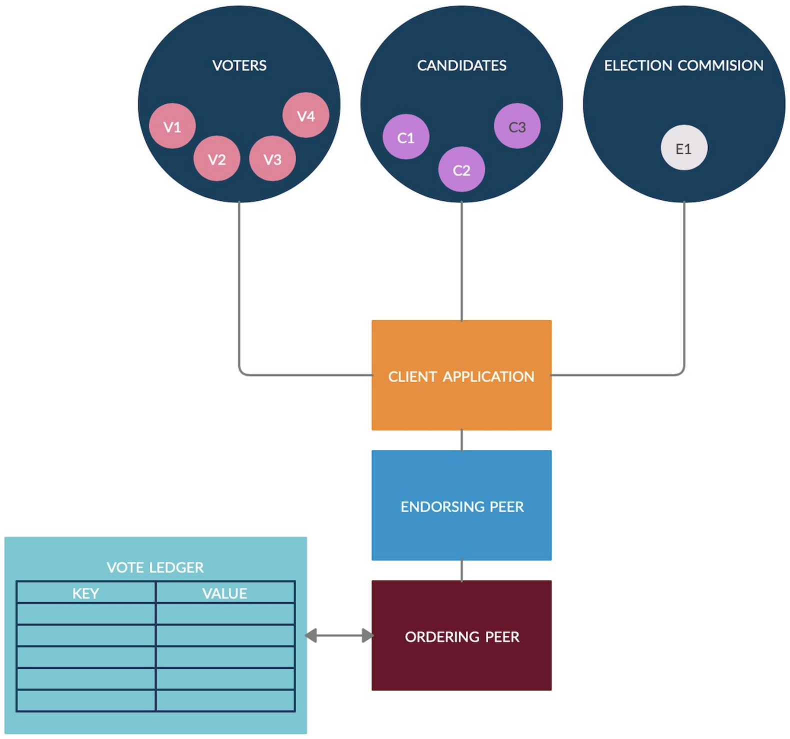
Naturally, features of blockchain technology can help us provide solutions to these problems. With its distributed and decentralised public ledger, peer-to-peer consensus, immutability, and verifiability, blockchain is ideal for a voting platform. It guarantees that malicious parties cannot influence the voting procedures, thereby ensuring a secure and transparent E-Voting system.

PROJECT ARCHITECTURE

In this project, we will be using Hyperledger as the incubator for blockchain technology, with Hyperledger Fabric providing the basic framework for the architecture layer of the application.

With a permissioned ledger like Hyperledger Fabric, we can ensure that only verified voters are able to participate to vote in the network, while ensuring that anonymity of the voter is maintained.

Our model will be built using a single client application with multiple interfaces/dashboards for different types of users. One endorsing peer will manage the execution of chaincode, and one ordering node will record the transactions on the Vote Ledger after they have been approved.

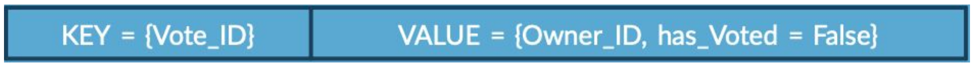


MAIN COMPONENTS:

- User - Election Commision
- User - Voters
- VOTE Object
- Vote Ledger - Public Ledger of all VOTE objects written to the fabric.

VOTING PROCESS

1. The Election Commision declares the elections, and requests eligible voters and candidates to register for voting on the registration dashboard.
2. Voters submit their voter applications on the registration dashboard to verify their identity on the system using their unique Aadhar Card ID, and an OTP based email address/phone number authentication.
3. The Election Commision verifies their details, and adds them as a registered Voter to the system network by assigning them a unique voterId and voterPin. These details are added onto the *voters.json* file by them, and also sent to the voters via e-mail.
4. The Election Commision also ensures that one new VOTE object is created for each voter participant. They create these by invoking the createVoteObject() chaincode to create new VoteObjects on the blockchain transaction ledger.
5. The Election Commision assigns candidateIds to every candidate, which are visible on the voting dashboard for voters to refer to.
6. On the day of elections, Voters open the Voting dashboard and enter their VoterId, PIN and select their preferred candidate from a dropdown list and submit their vote.
7. Once the vote is submitted, the transaction is sent to the endorsing nodes for executing the chaincodes for the transfer of ownership of the VOTE object from the voter to the candidate. Candidates are not allowed to further vote, i.e. transfer ownership of the received vote.
8. The VOTE objects comprises the chainstate of the ledger, with a unique **Vote_ID** as its key, and records the following parameters for its value:
   a. **Owner_ID:** Set to the Voter_Id when the object is initially created by the Election Commision. It is changed to Candidate_Id when the voter sends their vote to the candidate.
   b. **has_Voted:** Boolean Value set to False if the voter has not yet voted, set to True after voting.



KEY = {Vote_ID}	VALUE = {Owner_ID, has_Voted = False}

9. The chainstate will not contain any information on the original owner of the VOTE object once the vote has been casted.
10. Once the chaincode is executed and the transactions are approved by the endorsing nodes, they are added onto the Vote Ledger by the ordering nodes.
11. All approved transactions are publicly visible on the Vote Ledger. The voters can verify that their vote was sent to the desired candidate by the Vote_ID of the transaction.

12. Once the voting ends, the ledger can be searched for all votes, which can then be sorted by Candidate_IDs to count the number of votes received by each candidate. These results can be used to automatically collate election results, which can then be displayed publicly.

CLIENT:

The front end client will consist of four interfaces or dashboards to interact with the blockchain:

- **Registration Dashboard:** Eligible voters register themselves here by providing a proof of eligibility which is then sent for verification to the Election Commision.

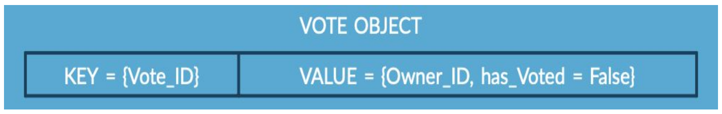
- **Election Commision Dashboard**: This has two views - a Candidate page and a Voter page:
    - The candidate page has options to view, modify or add candidates to a particular election before the elections are declared public.
    - The voter page is used to add new voters or to view the voters list. Once approved and registered, a Voter_ID (similar to a hash of public key) and a PIN is generated for these voters which is sent to their email address. Additionally, all registered voters get one VOTE object created for them, which they can use (or spend) to vote for a candidate.

- **Voting Dashboard:** This is the actual interface used during polling day to cast votes. It displays the pool of candidates. Voter participants cast their vote for their preferred candidates by transferring their VOTE object to the candidates' address. They do so by entering their unique Voter_ID, PIN and the preferred candidate from a drop-down menu.

- **Public Vote Ledger:** All VOTE transactions are registered on the ledger, which is published online and available for the network users to view on this dashboard. It shows the number of votes each candidate received. We also display the total number of voters that participated and other useful results such as number of votes received by a party or other demographic data. Since transactions are immutable and voter identity is anonymous, this provides accurate results while still allowing voters to ensure their vote was cast as intended.

VOTE OBJECT:

We will be using VOTE objects as our transaction assets on the ledger. The dictionary will be of the following format:



CHAINCODE:

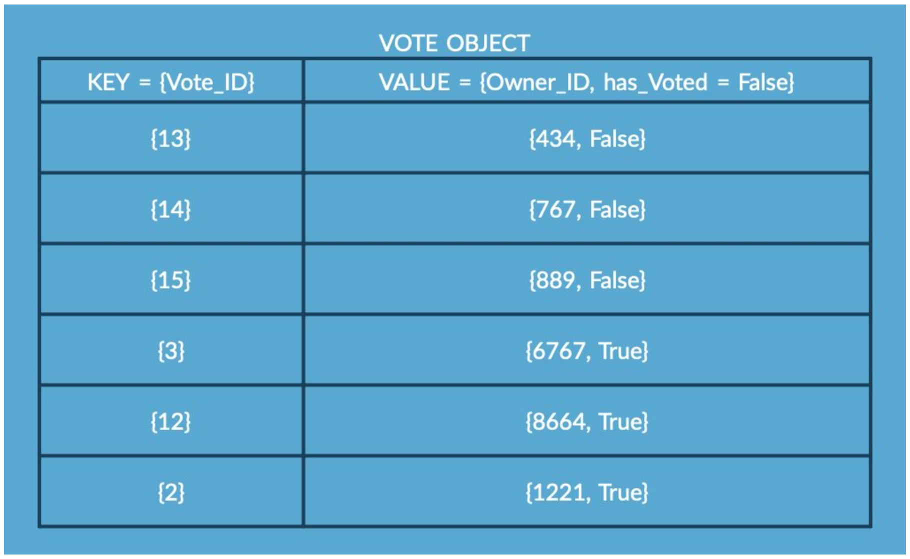
Our chaincode will consist of the following functions for VOTE object transactions:

A. **CreateVoteObject()** - This function can only be used by an Election Commision user account to generate a new VOTE object for every voter after they have been verified. The Vote_ID will be sequentially generated for all newly generated VOTE objects using this function. The Owner_ID will be set to the new voter's User_ID and the has_Voted boolean variable will be set to False.

B. **SendVoteObject()** - This function can only be used by a Voter user account to transfer the ownership of a VOTE object from self (Voter) to their preferred candidate. It will first check if the VOTE object has already been used for voting using the has_Voted variable. If it has been used, the transaction will be declined. Otherwise, the function will proceed. The Vote_ID will remain the same. The Owner_ID will be set to the Candidate_ID of the preferred candidate and the has_Voted boolean value will be set to True to signify that the VOTE object has been used and the vote has been cast.

These chain codes are executed by the endorsing nodes after the user nodes request to use them via the client. Once these chaincodes are executed (Vote transaction is made), the endorsed proposal is sent to the ordering service to be published on the public ledger.

SAMPLE CHAINSTATE:



VOTE OBJECT

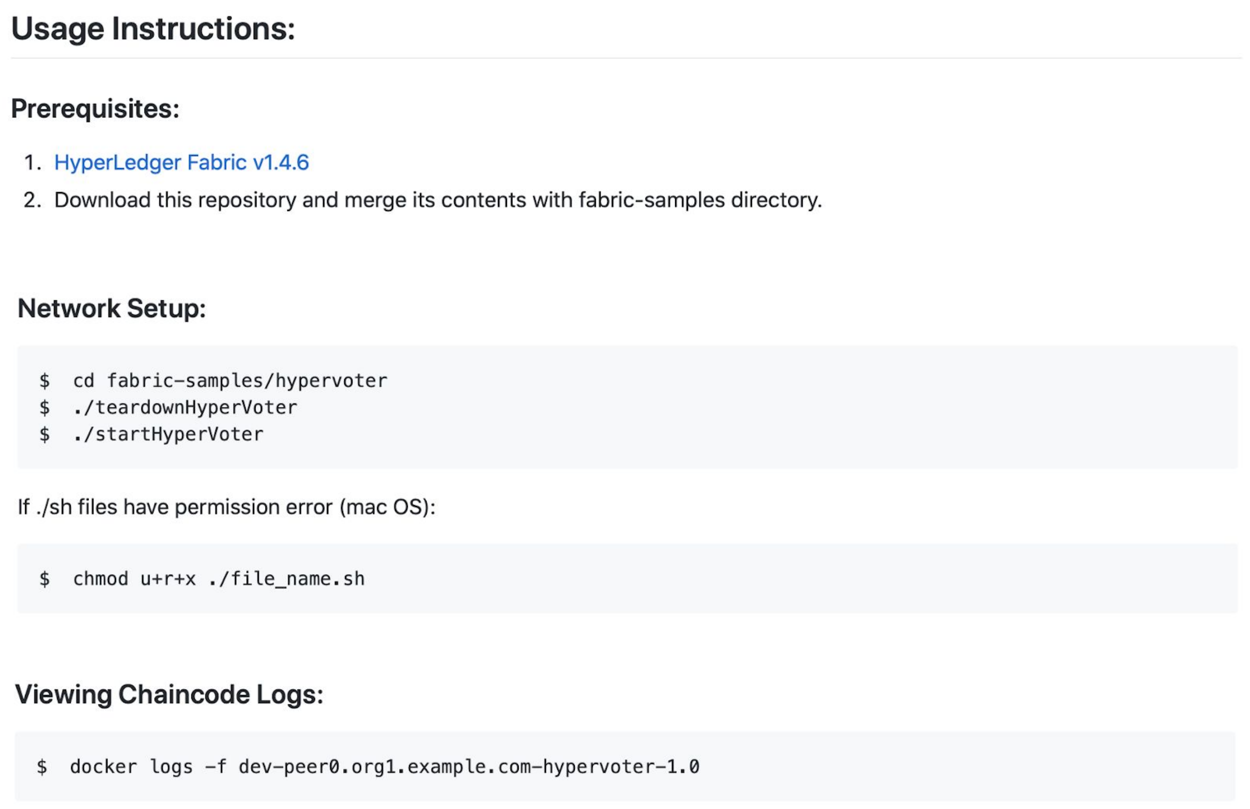
KEY = {Vote_ID}	VALUE = {Owner_ID, has_Voted = False}
{13}	{434, False}
{14}	{767, False}
{15}	{889, False}
{3}	{6767, True}
{12}	{8664, True}
{2}	{1221, True}

NETWORK CONFIGURATION:

The application uses the sample network 'basic-network' which bootstraps the following instances:

---

1. 1 Orderer
2. 1 Certifying Authority
3. 1 org (org1) maintaining 1 peer (peer0)
4. 1 CouchDB
5. 1 CLI



Usage Instructions:

Prerequisites:

1. HyperLedger Fabric v1.4.6
2. Download this repository and merge its contents with fabric-samples directory.

Network Setup:

```
$  cd fabric-samples/hypervoter
$  ./teardownHyperVoter
$  ./startHyperVoter
```

If ./sh files have permission error (mac OS):

```
$  chmod u+r+x ./file_name.sh
```

Viewing Chaincode Logs:

```
$  docker logs -f dev-peer0.org1.example.com-hypervoter-1.0
```

Running the Application:

1. Change working directory:

```
$  cd fabric-samples/hypervoter/javascript
```

2. Install Application Dependancies:

```
$  npm install
```

3. Enroll admin (stores CA admin credentials in wallet dir)

```
$  node enrollAdmin.js
```

4. Register and enroll EC:

```
$  node registerUser.js EC <EC_PIN>
$  node registerUser.js EC 555000555
```

EC_PIN is preset in voters.json file, and can be set by the EC before the elections start. Only EC user can create VoteObjects.

5. Register and enroll three voters:

```
$  node registerUser.js <voterId> <voterPin>
$  node registerUser.js 101 001
$  node registerUser.js 202 002
$  node registerUser.js 303 003
```

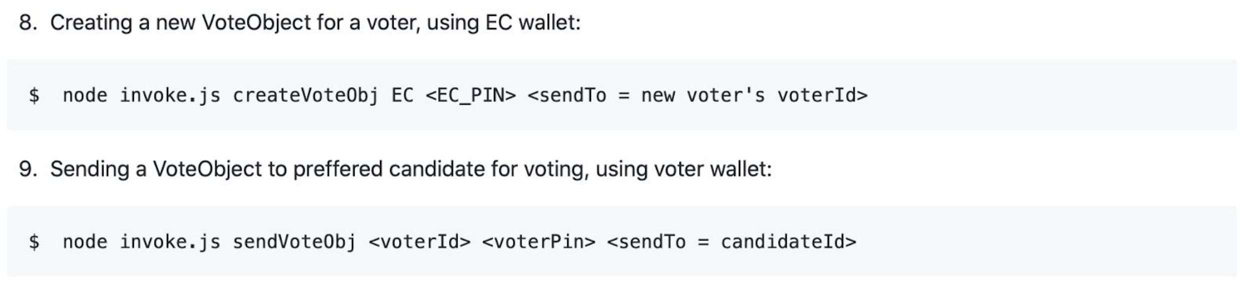
VoterId's and voterPin's are preset in voters.json file, and are set by the EC after they verify individual voters before the elections start. Voters can only send VoteObjects, not create them.

6. Query all votes using a voters wallet:

```
$  node query.js -1 <voterId>
```

7. Query a specific vote using a voters wallet:

```
$  node query.js <voteId> <voterId>
```



8. Creating a new VoteObject for a voter, using EC wallet:

```
$  node invoke.js createVoteObj EC <EC_PIN> <sendTo = new voter's voterId>
```

9. Sending a VoteObject to preffered candidate for voting, using voter wallet:

```
$  node invoke.js sendVoteObj <voterId> <voterPin> <sendTo = candidateId>
```

BENEFITS OF USING HYPER-VOTER!

- Increase in vote count, as the process is made more convenient.
- Reduction in the election costs.
- Transparency in vote counting to reduce malpractices.
- Anonymity of Voters is maintained, whilst allowing them to self-verify that their vote went to the desired candidate using Vote_ID at the same time.
- Does not allow double spending of votes.
- Eliminates the possibility of error in vote counting.
- Decentralised networks enhance trust in the system.