Ethereum Blockchain-based Decentralized Preferential E-Voting Smart Contract using Solidity

[1]Saurav Gupta,[2]Manjunath CR

[1]PG Student,[2]Associate Professor,
[1]Departement of CSE, FET,
[1]Jain University, Bengaluru, India.

Abstract : Various e-voting systems have been provided with the goal of increasing security and reducing costs for a long time. Ethereum Blockchain is a major breakthrough in the tech industry that provides a highly secure platform. As Ethereum, a decentralized platform which runs decentralized applications (DApps), was introduced in the market, the idea of secured voting system finally seemed possible. Organizations have shifted their focus on applications that run on blockchain platforms. It is highly likely that a simple voting method will not lead to a clear majority. There can be a number of ways to deal with this issue, including other potentially voting process, thus costing time and resources which is exactly opposite of the main concern of e-voting systems. Thus, in our paper, we introduce the vote-trading concept where in the absence of a clear majority, votes can be redistributed to other candidates and also this 'majority' factor can be determined by the organization as per their requirement as different organizations can have different winning percentage. We discuss the design and implementation for the Ethereum blockchain based decentralized preferential e-voting system using Solidity programming language where we present the idea of giving preference to candidates, rather than one vote per candidate.

IndexTerms **- Blockchain, E-Voting, Ethereum, Decentralized Applications (DApps), Solidity, Smart Contract.**

I. INTRODUCTION

Basically speaking, smart contracts are programs that are executed on Ethereum platform. Ethereum Blockchain is a decentralized computing platform that not only provides a platform for crypto-currency transactions but also allows execution of certain programs called smart contract executes such programs [1]. Execution of such decentralized applications on Ethereum platform is possible using a virtual machine called Ethereum Virtual Machine shown in **Fig. 1**, EVM. We can think of EVM (Ethereum Virtual Machine) as a general-purpose machine that executes applications unlike Bitcoin [2] where only the bitcoins spending conditions are considered. The state of a smart contract can be changed in Ethereum blockchain, but to perform some changes, some fees or fuel is required. This fees or fuel or gas in Ethereum is called Ether. Thus, we can say that Ether is the fuel for operating this decentralized Ethereum platform [3].

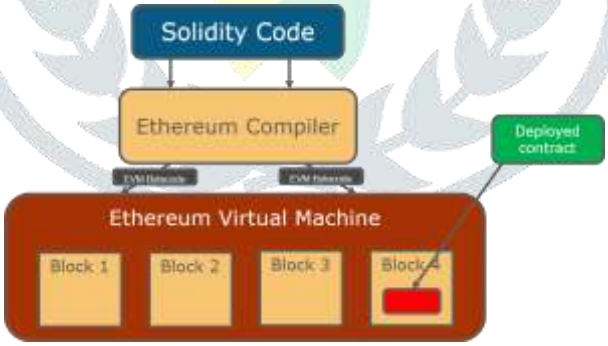

Fig. 1 Ethereum Architecture

A DApp is an decentralized application that is deployed on a decentralized network. A decentralized application (DApp) is basically application written as smart contract and can have a frontend user interface for user easiness.[4]. And as we have discussed, a "smart contract" is simply a program that runs on the Ethereum platform. Smart contracts are collection of code (its functions) and data (its state) that resides at a particular address on the Ethereum blockchain [5].

Blockchain can be thought of as a chain of blocks and these blocks contain transactions' information. If we break the word Blockchain, then the word "block" in this context means the digital information or transactions stored in the form of linked list, the "chain". Basically, the blocks store information about transactions like the amount, timestamp, and other information if we talk about the case of crypto- currency. Blocks also store information that helps in distinguishing one block from another. With other information in a block, a block also has a hash. All the contents of the block as well as the block itself can be uniquely identified by this hash. The hash works in such a way that once a block is created by adding transactions to it, any change, in fact, even a single character change in the data of a block causes the hash value of that block to change. Once a block's hash value changes, it does not remain the same block again. We can say that each block basically has 3 parts or we can say that each block stores the following information. These are Data, Hash of previous block and Hash.

The type of information or data to be saved in the block may depend on the nature of blockchain application, but the given three are the general information that a blockchain stores. The first block in blockchain technology is known as the Genesis block. In

blockchain technology, every single block is chained to the previous block excluding the first block. Except the first block a.k.a. the Genesis block, each block holds the hash of previous block. This is one of the main reasons that have made blockchain technology so secure. To understand how this provides such security, we first explain the 'Previous Hash' concept with the following example.

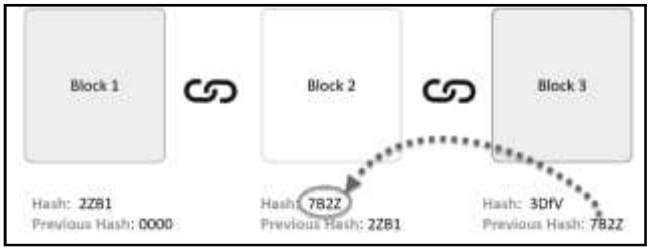

Fig. 2 'Previous Hash' value of Block 3 is the Hash value of Block 2

Figure 2 [6] represents a simple example of blockchain with just three blocks though a blockchain would generally contain thousands of blocks. The Genesis block or the first block has no predecessor in the blockchain and as a result, there is no reference to previous block. As can be seen in **Fig. 2**, the hash for the first block is '2ZB1' and the second block holds this hash value i.e. the hash value of previous block, i.e. the first block, as we see that the 'Previous Hash' value of Block 2 is same as the first block's hash value i.e. '2ZB1'.

We now see how this technique makes blockchain so secure. Assume an intruder or an attacker somehow changes the existing data in the second block i.e. Block 2. As the data with this block is tampered, the associated hash value of this block also changes. But Block 3 i.e. the third block still holds the old hash value of Block 2. As a consequence, this makes Block 3 and all the subsequent blocks in the blockchain invalid since they now don't contain the valid hash value of their respective previous block. Thus, tampering even a single block can cause all the succeeding blocks in the blockchain to become invalid. Say for example, an intruder changes some data in the Block 2 in **Fig. 2**. So, the hash value of Block 2 changes to some other value say, 'BB32'. But now the 'Previous Hash' value of Block 3 which is '7B2Z', doesn't match with new hash value of Block 2 which is 'BB32'. Thus, this makes the blockchain invalid, and we can detect these changes when an intruder changes some data in any of the blocks.

1.1 Problem Definition

Unlike nationwide voting where there are millions of voters, in an organization, there are very few voters, and very few nominees. There's a very high chance that a normal voting method won't lead to a clear majority [7]. Thus, another voting process has to take place which can be quite expensive in terms of time and resources or the other option is to have some kind of vote-trading where the votes can be redistributed to other candidates. As many organizations are now shifting their focus to blockchain technology because of its immense security and other reasons as mentioned in [8], voting through blockchain has been one of the main applications in the blockchain world. So how do we ensure that even if there isn't a clear majority in the voting process, we get a clear winner by not repeating the voting process and thus saving time and resources?

II. Existing System

The authors in [9] discuss a voting system which is based on Ethereum. The authors have stated that e-voting system must be secure in the sense that the system should be fully transparent should not duplicated votes. The authors have suggested deploying the E-Voting application as a smart contract. Only the users with valid EOAs are allowed to vote on that contract. However, if the voting doesn't result in clear majority, what can be done isn't discussed. The main advantages the paper offers are transparency and restriction to single vote per EOA.

The authors in [10] have discussed E-Voting and how it can be used or integrated with Blockchain [10].the authors have not only focused on some inherent features of Blockchain DApps which includes privacy, transparency, but their paper has proposed solution to verify the users' identities which is done in the registration phase. The first step of the protocol is registration and this is required for identity verification. However, a server which is centralized (Centralized Authority) is involved for the verification process and the same is also used to add the users' information to the database.

The author in [11], have introduced an e-voting system based on blockchain technology. The system utilizes smart contracts that make cost-efficient and secure election possible. Not only this, the voters' privacy are also guaranteed by this system. The authors have discussed how the blockchain technology can overcome limitations of e-voting systems. The authors have discussed how blockchain based e-voting system have laid the ground for transparency and how these systems ensures the election integrity and security.

The authors in [12] have proposed a decentralized voting platform which is based on Ethereum Technology. Restriction of multiple votes per mobile (MSISDN) is the main contribution of this platform by the authors. The future work for this system could be to develop the system further to make it more suitable for nationwide elections using some of the technologies located in the voting centres such as fingerprint.

Authors in [13] argued that before Blockchain came into the picture, e- voting arrangements were unsatisfactory in the sense that they were not effortlessly auditable and adequately straightforward neither for the organizers nor for voters. They have provided with a decentralized application that address various security factors like verification, transparency, integrity and non-repudiation.

In [14], authors have proposed Votereum, an Ethereum-based E-voting system that utilizes blockchain technology and smart contract. The authors in their system have introduced the concept of verifiable E-voting platforms and have discussed the evaluation and security concerns to the system. Not only this, the authors have discussed the requirements, architecture and design of the system.

We know that Ethereum platform provides immense security to the decentralized applications, thus our main focus have been on how can we deal with the situation when the voting process leads to no clear majority defined by the organization.

III. System Design

Figure 3 describes the system architecture which gives us an idea of the entire system. We have user interface which is a web application developed in ReactJS. The user interface is for easiness with which voters can cast their vote in the voting system.

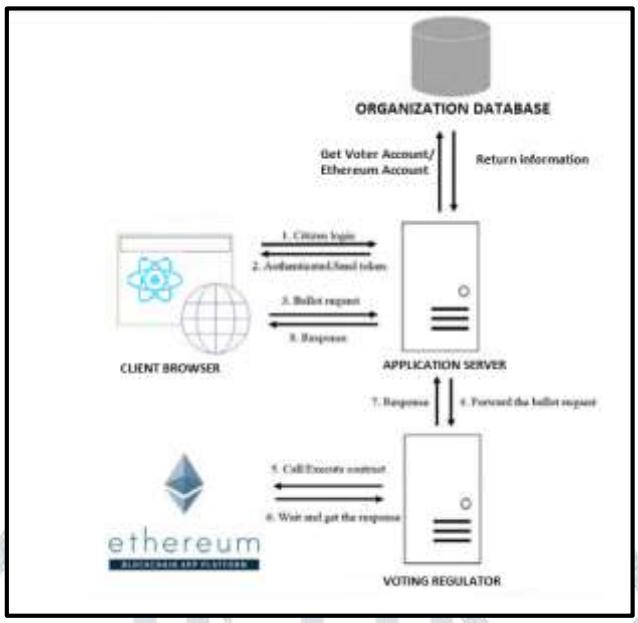

Fig. 3 System Architecture

We have application server which basically handles the request/response from the voters or organization database. For request/response which are related to blockchain/Ethereum, the application server will communicate with the voting regulator. Voting regulator acts as a mediator which handles request to Ethereum from the application server. The voting regulator executes the functions and methods in the smart contract. After receiving the response from the Ethereum, the voting regulator will send back it to the application server. We also have the organization database which stores various information about the members of the organization.

IV. Implementation

To validate our proposed methodology and system, we implemented our idea using several technologies which include Remix IDE [15], Ganache [16] and Solidity programming language. Remix IDE helps us to run the Solidity programming language and Ganache provides us with a local Ethereum network, thus we can perform all operations on it without any cost.

```
// imformation about the candidates standing in the election
struct Candidate
{
    string candidate_name;  // name of the candidate or the party
    uint vote_count;        // vote counts for this candidate
    address[] address_of_voters_for_this_candidate;
}
```

Fig. 4 Code block to define struct Candidate

In the code block in **Fig. 4**, we define the struct 'Candidate' which contains information such as the candidate name, vote count and the addresses of the voters who have voted for this candidate. We need the address of the voters who have voted for the particular candidate so that if the candidate receives the least vote, then the votes of this candidate will be redistributed to other candidates.

```
struct Voter
{
    bool voted;                    // ha
    bool has_voting_rights;        //doe
    uint [] voting_preferencing;


    uint redistributing_index;
}
```

Fig. 5 Code block to define struct Voter

In the code block in **Fig. 5**, we define the struct 'Voter' which contains information such as whether the voter has voted or not and whether the voter has voting rights or not. The voter gives preferences to the candidates and the preferences are stored in an array named 'voting_preferencing', with candidate at 0th index has highest preference while the candidate at the last index has the lowest preference. We also have the redistributing index for each voter which helps us in knowing that which index or in another sense which preference we are redistributing at the moment or while the redistributing happens.

```
function set_majority(uint majority) public
{
    require(msg.sender == organizer, "Only organizer is allowed to set the winning majority");
    majority_percentage = majority;
}
```

Fig. 6 Code block for setting the majority percentage

Figure 6 shows the code block for setting the majority percentage. Only the owner or organizing committee has the privilege to set the majority percentage.

```
function give_voting_rights(address voter) public
{
    require(msg.sender == organizer, "Only organizer is allowed to give voting rights");

    voters[voter].has_voting_rights = true;

}
```

Fig. 7 Code block for giving voting rights

Figure 7 shows the code block for giving voting rights to the voters. Only the owner or organizing committee has the privilege to give voting rights to the voters. The address of the voter is passed as argument to this function.

```
function vote(uint[] memory preference_array) public
{

    require(voters[msg.sender].has_voting_rights == true, "You are not authorized to vote");
    require(voters[msg.sender].voted == false, "You have already casted your vote");
    require(preference_array.length == get_number_of_candidates(), "Please enter the exact numbers of that candidates");

    for(uint i=0; i<preference_array.length; i++)
    {
        preference_array[i]--;
    }

    voters[msg.sender].voting_preferencing = preference_array;
    voters[msg.sender].voted = true;

    uint candidate_index = voters[msg.sender].voting_preferencing[0];
    candidates[candidate_index].address_of_voters_for_this_candidate.push(msg.sender);

    address_of_voters.push(msg.sender);

}
```

Fig. 8 Code block for vote

Figure 8 shows the voting function of our system. The voter needs to fulfil three requirements before it can cast its vote. First, the voter should be authorized to vote. Second, it should not have already voted meaning a voter can vote only once. And lastly, the voter should give preference to all the candidates. Thus, if the last criteria are met, then voter's vote is casted. Each voter gives an array of preferences as input. The 1st input has highest preference while the last input has least preference.

Figure 9 shows the code block for calculating votes. The first preference of each voter is counted as single vote, and thus the vote count for that candidate is increased.

```
function calculate_votes() public
{
    uint number_of_voters_who_voted = get_number_of_voters_who_voted();

    for(uint i=0; i<number_of_voters_who_voted; i++)
    {
        address address_of_voter = address_of_voters[i];
        uint candidate_index = voters[address_of_voter].voting_preferencing[0];

        candidates[candidate_index].vote_count++;
    }
}
```

Fig. 9 Code block for calculating votes

Figure 10 shows the code block for checking the majority and finding the winner of the election. We find the candidate with the maximum votes and the candidate with the least votes and their respective vote counts. If the vote percentage of the candidate with maximum votes is greater than or equal to the majority percentage, then that candidate is declared winner. Or else, we redistribute votes calling the function 'redistribute_votes' passing the candidate with the minimum votes as argument.

```
function check_majority() public
{
    uint number_of_voters_who_voted = get_number_of_voters_who_voted();
    uint max_vote_count = 0;
    uint min_vote_count = number_of_voters_who_voted;
    uint max_voted_candidate;
    uint min_voted_candidate;

    uint number_of_candidates = get_number_of_candidates();

    for(uint i=0; i<number_of_candidates; i++)
    {
        if(candidates[i].vote_count > max_vote_count)
        {
            max_vote_count = candidates[i].vote_count;
            max_voted_candidate = i;
        }

        if(candidates[i].vote_count < min_vote_count)
        {
            min_vote_count = candidates[i].vote_count;
            min_voted_candidate = i;
        }
    }

    uint max_candidate_vote_percentage = max_vote_count*100/number_of_voters_who_voted;

    if(max_candidate_vote_percentage >= majority_percentage)
    {
        winner = candidates[max_voted_candidate].candidate_name;
    }
    else
    {
        redistribute_votes(min_voted_candidate);
    }


}
```

Fig. 10 Code block for checking majority and getting the winner

Figure 11 shows the code block for redistributing votes. The last candidate is removed from the competition and the vote of the voters who voted for this candidate is redistributed to their next preferred candidate. The size of number of candidates is decreased by one.

```
function redistribute_votes(uint min_voted_candidate) public
{
    uint voters_for_min_voted_candidates = candidates[min_voted_candidate].address_of_voters_for_this_candidate.length;

    for(uint i=0; i<voters_for_min_voted_candidates; i++)
    {
        address voter_address = candidates[min_voted_candidate].address_of_voters_for_this_candidate[i];
        voters[voter_address].redistributing_index++;
        uint candidate_index_to_vote = voters[voter_address].voting_preferencing[voters[voter_address].redistributing_index];

        candidates[candidate_index_to_vote].vote_count++;
    }


    candidates[min_voted_candidate] = candidates[candidates.length-1];
    candidates.length--;

}
```

Fig. 11 Code block for redistributing votes

V. TESTING AND RESULTS

We tested our code by deploying our smart contract on our local Ethereum network provided by Ganache. **Figure 12** shows the deploy button in Remix IDE which requires two arguments, one is the name of the election and the other one is majority function. Election name because our voting system can be used in many organizations for various election purpose.

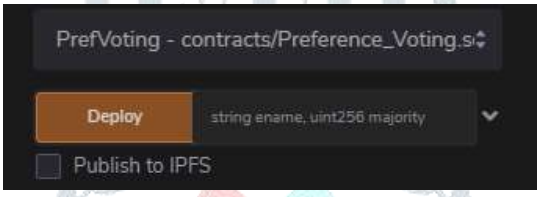

Fig. 12 Deploying code in Remix IDE

Figure 13 shows the output once the contract is deployed. We have information like transaction hash, the block in which the transaction is recorded, the fuel used and other such information. We see that the transaction is stored in Block 1. We can cross-check the details with the details in our local Ethereum network in Ganache.

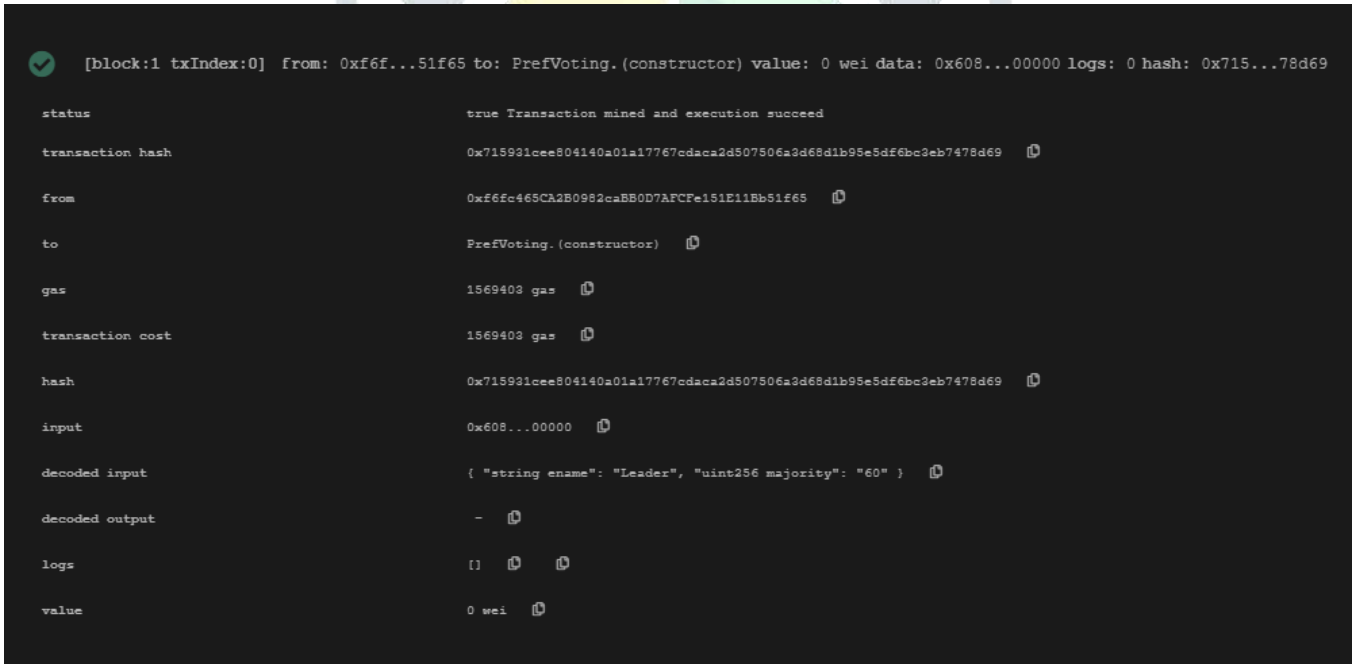

Fig. 13 Output after deploying the smart contract in Remix IDE

In **Fig. 14**, we see the information stored in Block 1. We can see in the right side of the image, a red button is there with 'Contract Creation' written on it. Also we can check the transaction hash in **Fig. 16** with the transaction hash in **Fig. 15** to confirm that they are the same hash.

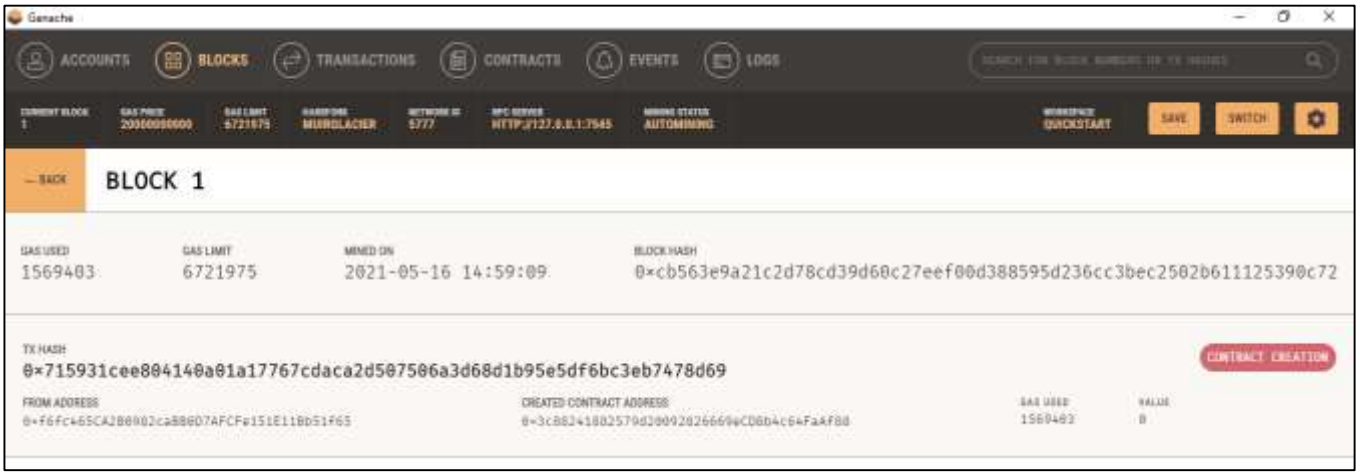

Fig. 14 Local Ethereum network in Ganache

After deploying the contract and giving the required inputs, we can see all blocks and all the transactions in our local blockchain as shown in **Fig. 15** and **Fig. 16** respectively.

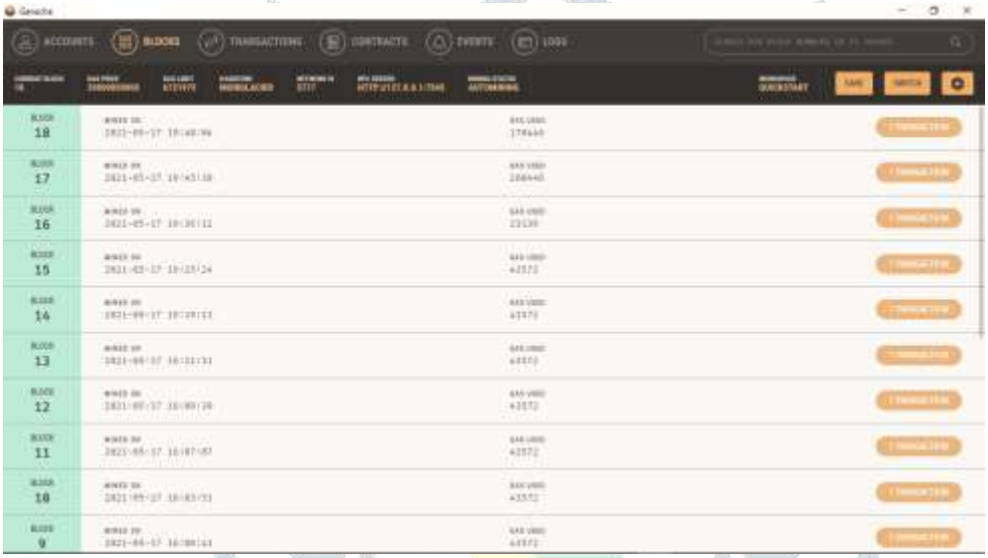

Fig. 15 Blocks in the local blockchain

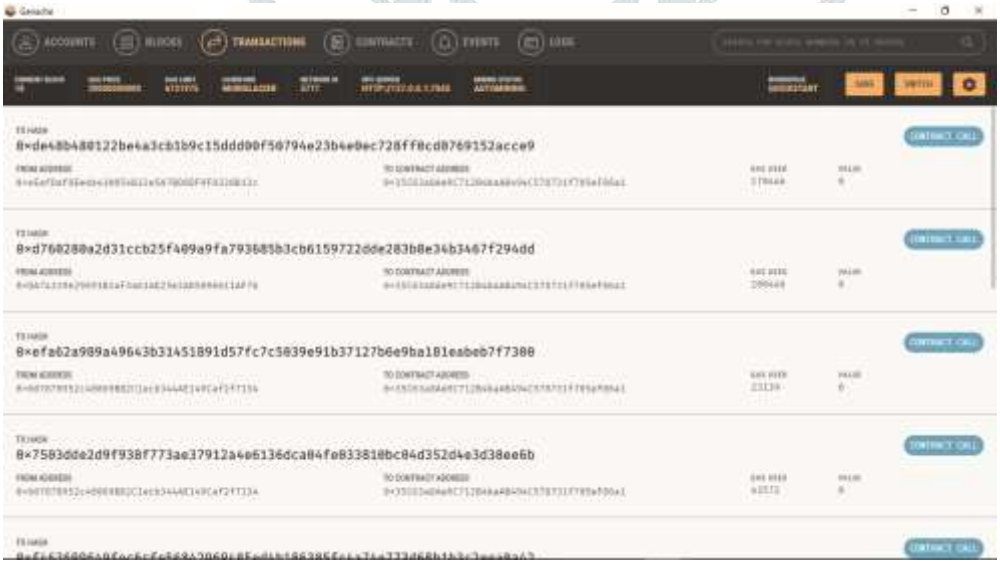

Fig. 16 Transactions taken place in the local blockchain

Figure 17 shows the button of winner. When the voting has taken place, and the vote count has been done, then we can click on the winner button in the Remix IDE to get the winner of the election as shown in **Fig. 17**.

Fig. 17 'winner' button to get the winning candidate name

One of the main reasons for decentralized voting system is because centralized systems are subjected to Denial-of-Service attacks. And since the votes are stored in a centralized server, the integrity can be compromised and thus users don't have that transparency to trust online voting systems. However, with the Ethereum blockchain technology used in our voting system, the transparency in the election is obvious on the Ethereum network because all the information regarding the voting is open to public and thus the voters can be assured that their votes are being counted. Not only this, the information regarding the voters is also hidden or encrypted as the users are known by the Ethereum address. As we saw in the results section that when voting takes place, each vote is stored as transaction in the blocks in the blockchain. These transactions or votes take place from Ethereum accounts which are just huge numbers, and thus only the voter knows whom they vote and no one else can know who the voter is and to whom a particular voter has voted. Thus, Ethereum based voting system helps us in providing privacy of voters, integrity of votes, and transparency in the results.

VI. CONCLUSION

Different blockchain based e-voting systems have been provided but all of the voting systems fail when a voting results in no majority or no party/candidate winning the election. In this case, there is second round of voting and the process continues until we have a clear majority. This is quite time consuming process and can be quite expensive in terms of resources required for voting. Our voting system uses the concept of preference based voting, where instead of giving vote to particular candidate, the voters provide with preference for the candidates. The votes are counted, and if we have a clear majority as defined by the organization, then our system will declare the winner. But if we don't have clear winner, the votes of the last candidate are distributed according to the preferences given by the voters for that candidate. Thus, the last candidate is removed from the competition and its votes distributed to other candidates according to the preferences given in the votes of the eliminating candidates. This process continues as long as we don't get a clear majority winner. Ethereum platform has been used which runs smart contract on it and is known to be one of the immensely secured blockchain platform. The proposed system was implemented and tested successfully and we also analyzed how our system provides privacy of voters, integrity of votes, and transparency in the results.

REFERENCES

[1] V.Buterin et al., "A next-generation smart contract and decentralized application platform," white paper, 2014.
[2] Nakamoto, Satoshi. (2009). Bitcoin: A Peer-to-Peer Electronic Cash System.
[3] "INTRO TO ETHEREUM" https://ethereum.org/en/developers/docs/intro-to-ethereum/
[4] "INTRODUCTION TO DAPPS" https://ethereum.org/en/developers/docs/dapps/
[5] "Blockchain Tutorial for Beginners: Learn Blockchain Technology" https://www.guru99.com/blockchain-tutorial.html
[6] "INTRODUCTION TO SMART CONTRACTS" https://ethereum.org/en/developers/docs/smart-contracts/
[7] Tullock, Gordon. "Problems of Majority Voting." Journal of Political Economy, vol. 67, no. 6, 1959, pp. 571–579. JSTOR, www.jstor.org/stable/1827311.
[8] Why Your Business Wants To Adopt Blockchain Technology" https://senlainc.com/blog/why-your-business-wants-to-adopt-blockchain-technology/
[9] Yavuz, E.A., Koç, A., Çabuk, U.C., & Dalkiliç, G. (2018). Towards secure e-voting using ethereum blockchain. 2018 6th International Symposium on Digital Forensic and Security (ISDFS), 1-7
[10] P. Tarasov and H. Tewari, "The future of e-voting." IADIS International Journal on Computer Science & Information Systems, vol. 12, no. 2, 2017
[11] F. Þ. Hjálmarsson, G. K. Hreiðarsson, M. Hamdaqa and G. Hjálmtýsson, "Blockchain-Based E-Voting System," 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, 2018, pp. 983-986, doi: 10.1109/CLOUD.2018.00151.
[12] D. Khoury, E. F. Kfoury, A.Kassem and H. Harb, "Decentralized Voting Platform Based on Ethereum Blockchain," 2018 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET), Beirut, 2018, pp. 1-6, doi: 10.1109/IMCET.2018.8603050
[13] S. Shukla, A. N. Thasmiya, D. O. Shashank and H. R. Mamatha, "Online Voting Application Using Ethereum Blockchain," 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Bangalore, 2018, pp. 873-880, doi: 10.1109/ICACCI.2018.8554652.
[14] L. V. Thuy, K. Cao-Minh, C. Dang-Le-Bao and T. A. Nguyen, "Votereum: An Ethereum-Based E-Voting System," 2019 IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF), Danang, Vietnam, 2019, pp. 1-6, doi: 10.1109/RIVF.2019.8713661
[15] https://remix.ethereum.org/
[16] "Ganache" https://www.trufflesuite.com/docs/ganache/overview