TrustVote: On Elections We Trust with Distributed Ledgers and Smart Contracts

Majd Soud, Sigurður Helgason, Gísli Hjálmtýsson, Mohammad Hamdaqa
School of Computer Science
Reykjavík University, Iceland
{majd18, sigurdurhel15, gisli, mhamdaqa}@ru.is

***Abstract*—Electronic voting (e-voting) plays a crucial role in democratic countries. Unfortunately, implementing a trusted e-voting system is always a challenge. Scholars have been trying to propose solutions to improve e-voting systems. However, existing e-voting approaches still need to adopt new technologies to maintain their integrity and modernity. Unlike other technologies that were used in e-voting, blockchain has the potential for perfecting e-voting requirements and fostering trust in e-voting systems. The first part of this paper presents a state-of-the-art review of blockchain-based e-voting implementations. The second part proposes a blockchain-based e-voting system (TrustVote). TrustVote was realised using public and permissioned blockchain. This work results show that the permissioned blockchain-based implementation of TrustVote outperforms the identical version of the same system that uses public blockchain. The results also show that while several challenges and requirements still need to be addressed, using permissioned blockchain for implementing an e-voting system can help to satisfy many of the requirements for e-voting and pave the road for a new generation of e-voting systems that people can trust.**

***Index Terms*—Electronic Voting, Blockchain, Smart Contracts, Permissioned Blockchain, Hyperledger, Ethereum, Distributed Ledger.**

I. Introduction

Voting is an essential foundation for any modern democracy. At present, e-voting is the main method used in national elections in ten countries around the world. Moreover, 30 countries are in the process of implementing e-voting systems to be used to elect national representatives [1]. Recent elections witnessed an increasing number of electronically cast votes and people gradually started to have more faith in e-voting systems. For instance, Figure 1 shows the number of e-votes per election in Estonia (i.e., the first nation that legally employed e-voting in general elections over the Internet), since it was first introduced in 2005 [2].

Traditional voting systems have been criticized regularly for being slow and inefficient. To resolve this, there has been a high interest in employing electronic digital systems to assist votes' casting and counting, a process known as e-voting. The first generation of e-voting systems was centralized. These voting systems did not solve the trust problem, as they depended on a centralized third party to aid, record, and calculate votes. Moreover, these first-generation systems lacked tractability, verifiability, integrity, and voters' privacy.

978-1-7281-7091-6/20/$31.00 ©2020 IEEE

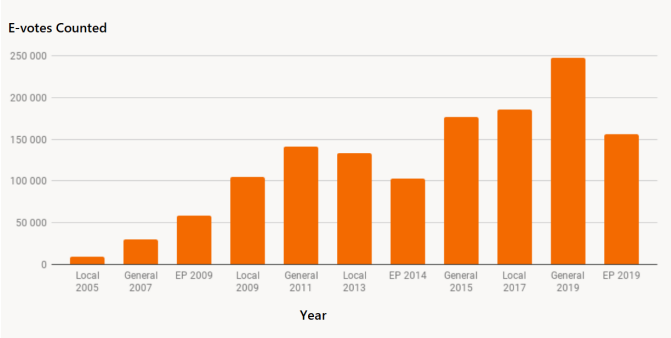

Fig. 1. Number of e-votes per election [2].

The goal of this paper is to evaluate the feasibility of achieving trusted elections and e-voting systems through utilizing blockchain as a service. In order to achieve the goal of this paper, we start by surveying the available blockchain-based e-voting systems. Based on the survey, critical e-voting requirements were extracted and listed. Finally, a proof-of-concept e-voting system was implemented twice; first using a public blockchain (i.e., Ethereum) and second using permissioned blockchain (i.e., Hyperledger). The two implementations were compared and a final remark was provided.

Contributions. This paper makes the following original contributions:

1) **A state-of-the-art review of blockchain-based e-voting systems that we used as a base to identify requirements for e-voting systems.**
2) **An e-voting system (TrustVote) that takes the defined requirements into consideration.** We propose a blockchain-based e-voting system named TrustVote. We implemented the system twice; using public and permissioned blockchain and compared the two alternatives.

Paper Outline. The remainder of this paper is organized as follows. Section II presents a state-of-the-art review of the existing blockchain-based e-voting systems. Section III shows the design of TrustVote. Section IV explains the implementation of TrustVote. Section V presents the evaluation and results. Finally, the conclusion is presented in Section VI.

II. State of the Art Review

In this section, a Systematic Literature Review (SLR) of the related blockchain-based e-voting systems is presented.

After that, the main requirements of the e-voting systems are collected and explained as they were listed in the literature. Finally, the reviewed e-voting systems are compared based on the explained requirements. Table I summarizes this comparison.

A. Blockchain-based e-voting in the literature

Multiple blockchain-based voting systems were proposed in the literature. In this review, we only include e-voting systems that provide online elections based on Blockchain and smart contracts.

TIVI [5]. An online voting system that depends on biometric authentication. Its design basically takes the advantages of Blockchain technology, such as a ballot box and encryption mechanisms. This solution requires a picture of the face of the voter before casting a vote to check the voter identity. The anonymity of voters is provided in the mixing phase. To count the votes and tally the results in this system, a central party is required to shuffle the encrypted votes and decouple the voter's real-world identity from their voting key. Therefore, this solution uses the blockchain as an immutable database only. In this work, it is shown that votes can be counted on the blockchain without the need of any third party.

Agora voting system [7]. This solution has five phases. First, the configuration phase in which the administrator starts the election. Second, the voters fill and submit their ballots in the casting phase. Voters cast their ballots to the Bulletin Board. Third, the anonymization phase in which ballot casts on the Bulletin Board are anonymized using Neff shuffling. Fourth, the authorities decrypt the anonymized ballots and publish them along with proof of decryption correctness. Finally, the result is calculated and published on the Bulletin Board. Comparing with TrustVote, our proposed system has a simple architecture, and all of the calculations are done on the fly in the smart contracts.

Open Vote Network (OVN) [8]. This online solution is designed as a smart contract on Ethereum blockchain. It has an administrator who authenticates the voters and sets-up the election. Votes in this system are encrypted before casting, which guarantees individual confidentiality. It also achieves universal verifiability because it is a self-tallying solution. This system only supports elections with two options (yes, no). Furthermore, because of the mathematical tools that are used in this system, the maximum allowed voters' number is 50 and the system cannot scale more. Comparing with TrustVote, our system is not limited to a specific number of voters.

Follow My Vote (FMV) [6]. This system uses webcams and government-issued IDs in the authentication phase. After verifying voters' identity, a third trusted party allows eligible voters to cast their ballots. This system also allows voters to change their vote casts in the future. It provides each eligible voter with pass-phrases that are required when changing votes. The ballots are stored in the blockchain. One primary drawback of this system that it does not guarantee universal verifiability. In this work, universal verifiability is guaranteed and the uniqueness of votes is achieved.

Verify-Your-Vote (VYV) [4]. An online electronic voting solution that uses cryptographic primitives based on Elliptic-Curve Cryptography (ECC). VYV has several components, including a trusted server that is used to register eligible voters and authenticate them. It also has an external administrator who owns an account that manages election parameters. Finally, it has a tallying authority with an externally owned account. This solution has many tallying authorities as candidates to construct ballots and calculate the final election result. This solution was modeled with the ProVerif tool in order to formally prove some security features such as voters' authentication. Compared with TrustVote, Blockchain is not fully utilized in their system and they depend on a third party to calculate the results.

BroncoVote [3]. A blockchain-based voting solution that was proposed as a university-scaled framework. It also utilizes smart contracts to manage voters. This system has seven phases: (1) Initialization, in which the administrator deploys both Registrar Smart Contract (RSC) and Creator Smart Contract (CSC). RSC enables voters to register, and the CSC creates voting contracts for the registered voters. (2) Registration, in which only students and employees within a whitelisted domain are allowed to be voters. (3) Ballot creation, creator should send the ballot ID to all voters. (4) Loading the ballot, voters can load the ballot using the previously received ballot ID. (5) Voting phase, votes are encrypted and sent to a voting contract. (6) All encrypted votes are retrieved and (7) The results are published. This system was deployed on Ethereum's Testnet. Compared with TrustVote, system evaluation in BroncoVote is done in terms of gas cost per number of voting options, which does not show the overall system performance.

BitCongress [9]. An online voting platform to prevent double voting. It depends on Bitcoin Blockchain, Smart Contract Blockchain, and Counterparty. In the registration phase, it allows any Bitcoin address to register in the election; hence, eligibility is not guaranteed. In the tallying phase, if the election is of a large scale, it uses the Quota Borda system. This platform depends on many external technologies, while the same functionality can be achieved by utilizing smart contracts. TrustVote system architecture is simpler and achieves voters' eligibility.

A Privacy Preserving e-Voting Protocol for Permissioned Blockchain (DABSTERS) [10]. An e-voting protocol to balance voters' privacy and election transparency. It depends on the BlindCons consensus algorithm. It has five phases, the enrollment phase, in which verification of the eligibility of voters is done offline by Election Authorities (EAs). In the validation phase, the eligible voters' list is validated on the blockchain. Then, voters write their encrypted votes to the transactions and send them to the EAs to blind the votes' transactions. After blinding the transactions, they are sent to the blockchain to be verified. After that, reading the votes and tallying the results are done by the Tallying Authorities (TAs). Finally, the verification phase, in which voters verify the results. This work provides formal verification using ProVerif.

TABLE I
EVALUATION OF THE SURVEYED BLOCKCHAIN-BASED E-VOTING SYSTEMS

Requirement	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
Flexibility	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
Coercion-resistance	No	No	No	No	No	No	No	No	No
Consistency	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Integrity	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Auditability	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Identity Verification	Yes	No	No	Yes	Yes	No	Yes	Yes	No
Eligibility	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No
Voter Secrecy	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
Tamper-resistance	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Uniqueness	No	No	Yes	No	N/A	Yes	No	No	Yes
Universal Verifiability	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
Individual Verifiability	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Compared with TrustVote, it is a prototype and does not provide any performance evaluation or implementation evaluation. Smart contracts are used in TrustVote to replace the TAs.

Platform-independent Secure Blockchain-based Voting System [11]. This system is implemented on Hyperledger Fabric and depends on Paillier cryptosystem, proof of knowledge consensus, and Ring Signatures for voters' privacy. The eligibility of voters in this system is not achieved because there is no registration phase. In addition, this system does not guarantee coercion resistance. Compared with TrustVote, their evaluation was only for the encryption and decryption algorithms, which does not show the overall system performance.

To summarize, we find that most of the surveyed e-voting systems (1) are using the blockchain as a database only to record votes and (2) concerning the time spent in encryption and decryption algorithms in their systems' evaluation, which does not reflect the whole system performance. TrustVote is utilizing blockchain not only for storing votes in immutable ledger but also for calculating votes, tallying, and publishing the results without the need of a third party to publish the results.

B. E-voting requirements and design considerations

The following list of requirements is explained for a viable e-voting system to be effective and trusted. These critical requirements are identified based on the presented review.

(i) **Flexibility.** An election should have the flexibility to support ballots with various complexity of all poll types such as (Multiple items selection, Yes/No voting, and Priority voting)
(ii) **Coercion-resistance.** An election system should prevent coerced voting.
(iii) **Consistency.** All voters in the election should have the same voting procedure.
(iv) **Integrity.** E-voting system should assure the accuracy of votes.
(v) **Auditability.** The voting procedure from the start of the election to generating the results is auditable after the election.
(i) **Identity Verification.** An election system should provide secure authentication via an identity verification service.
(ii) **Eligibility.** An election system should only allow eligible individuals to vote in an election and prevent any ineligible voters from voting.
(iii) **Voter Secrecy.** An election system should not allow tracing back from votes to respective voters.
(iv) **Tamper-resistance.** An election system should prevent any third party from tampering with any vote.
(v) **Uniqueness.** An election system should accept one vote from each voter and prevent any vote duplication.
(vi) **Universal Verifiability.** Anyone who is included in the election has the ability to verify the whole election's procedure, results, and outcomes. This also includes spectators who can see the voting process on the blockchain.
(vii) **Individual Verifiability.** An election system should allow any voter to verify that his/her vote is counted in the final result of the election.

Table I shows an a comparison between the reviewed blockchain-based e-voting systems based on the identified e-voting requirements.

III. TRUSTVOTE SYSTEM DESIGN

In this section, an overview of TrustVote is provided, then TrustVote's design is explained, TrustVote's smart contract is defined, and the architecture of TrustVote system is presented.

A. TrustVote overview

TrustVote is a decentralized e-voting system that is based on blockchain and smart contracts. The goal of TrustVote is to facilitate managing electronic elections and to minimize humans' errors. A high level of TrustVote structure is provided in Figure 2. It mainly consists of two types of nodes (i.e., Managerial and District nodes), and depends on the government identity identification service to authenticate voters. TrustVote also depends on the administrators of the election to initialize and create the Election Creation Smart Contract (ECSC) in which the administrators initialize the aforementioned nodes and the voters' names. The rest is done by the smart contracts and the blockchain. Next, all the components and processes of TrustVote system are described in detail.

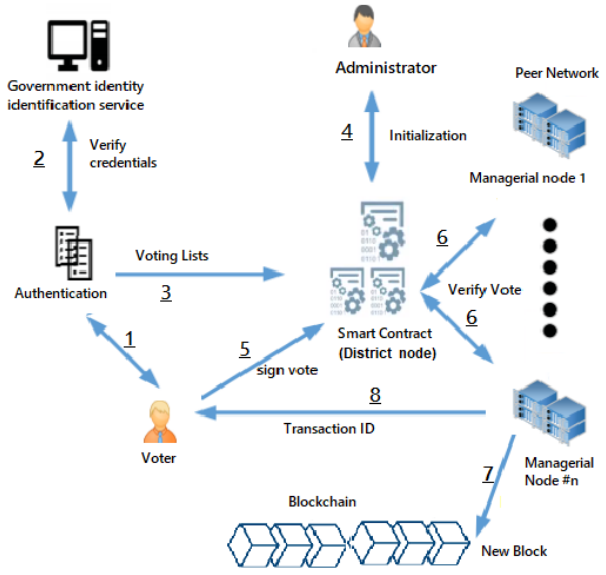

Fig. 2. TrustVote structure and processes.

B. *E-voting as a smart contract*

One way to elaborate the design of a smart contract is to depict it as a business process model that consists of the contract roles, activities, and transactions.

1) Election Roles: The roles in TrustVote are explained as follows:

(i) **Election administrators:** This role includes several trusted organizations. They start the election process and manage their life-cycle. Managing the election includes: (1) Specifying the type of the election, (2) Creating the election, (3) Determining the lifetime of the election, and (4) Assigning the permissioned nodes (i.e., the District and Managerial nodes).

(ii) **Voters:** Voters are the eligible participants for specific elections. Voter role includes: (1) Self-authentication at the beginning of the voting process, (2) Casting a vote, and (3) Verifying the cast vote after the election is over.

(iii) **District node:** In this system, each voting district is considered a district node that has a software agent to interact with the private blockchain and manage the smart contracts on that node. Once an election is created, one ballot asset is distributed for each registered district node. Permission to interact with the ballots is granted for each corresponding district nodes when the ballot assets are created.

(iv) **Managerial node:** Each institution, with permissioned access to the network, hosts a managerial node. A managerial node provides the same blockchain functionality as a district node but does not allow for voting through it. These nodes ensure blockchain integrity verification. All of them can individually tally and verify results.

2) Election Process: The election process is represented as a set of transactions. This work's proposed election process is divided into four main activities as follows:

(i) **Election creation:** Election administrators start the process by creating election ballots. Then, administrators define the Election Creation Smart Contract (ECSC). This ECSC contains the list of all candidates and all voting districts. After that, a set of Ballot Smart Contracts (BSCs) are created and deployed on the blockchain by the ECSC. These BSCs contains the list of candidates for each voting district. Therefore, each BSC includes the corresponding voting district as a parameter in it. Figure 3 shows the election as a smart contract in TrustVote.

(ii) **Voter registration:** Voters' registration is done by the administrators. Defining a deterministic list of eligible voters is the first thing that administrators should do when they create an election. In TrustVote, voters are authenticated and authorized via a government identity verification service. For each eligible voter, a corresponding card is created using the voter electronic ID, PIN, and the voting district the voter belongs to. Voters then can retrieve the cards electronically with a One-Time Password (OTP) login with those entities.

(iii) **Tallying results:** Tallying results in TrustVote is done on the fly in the smart contracts. Each BSC tally votes and store its tallying in its own storage for the corresponding districts only. Finally, each BSC publishes the final result for their corresponding districts when the election lifetime is over.

(iv) **Verifying vote:** TrustVote sends the transaction ID of each vote to his/her corresponding owner so that the voter can verify that his/her vote was counted and correctly counted. Each voter, who wants to verify his/her own vote, can go to the government authority in which he/she should verify his/her identity using his/her electronic ID and its corresponding PIN. Then he/she should present his/her transaction ID so that the government authority can search his/her transaction ID on the blockchain. This can be easily done using a blockchain explorer to locate the transaction with the corresponding transaction ID.

3) Election transactions: transactions are the third essential components to define a smart contract. Each vote is stored as a transaction on the private blockchain if it reaches consensus. Each transaction has an ID that is sent to the corresponding voter after his/her vote is recorded in the blockchain. Receiving the transaction ID will help the voter to verify his/her vote. Besides the ID, each transaction has all the information about the vote, such as the location where the vote was cast and whom was voted for. Each vote transaction can only be appended onto the blockchain by its corresponding ballot smart contract if all the nodes in the corresponding district agreed upon the correctness and verification of the transaction data.

C. *TrustVote Process*

An example of an election in this system is as follows:

1) Eligible voters can use any computer in the corresponding voting district to cast their votes as the One-Time Password (OTP) for the corresponding voter has information about

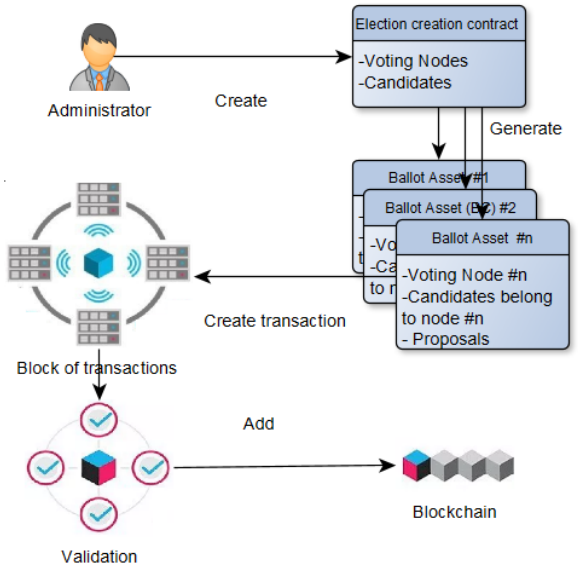

Fig. 3. Election as a smart contract.

the voter such as his/her electronic ID, PIN, and corresponding voting district. Furthermore, each voter also has to present his/her ID and PIN at the voting district.

2) After successful authentication, the corresponding smart contract is automatically called, and the election ballots are displayed to the voter. These election ballots represent the items that the voter can vote on. After the voter finishes his/her voting selection, then the selected vote should be signed using the voter OTP.
3) Then the signed vote transaction is verified by the corresponding district node, from which the voter is interacting with the smart contract. If the vote transaction was verified, then the aforementioned district node transfers the verified vote transaction to other district nodes to be verified by their consensus algorithm. After the vote transaction is verified by all district nodes, it will be recorded on the private blockchain.
4) Consensus for a specific vote transaction is reached only if the majority of district nodes agreed on the correctness of the vote data. After that, the voter receives the transaction ID for the corresponding verified and counted transaction of his/her vote. This transaction ID is sent to the corresponding voter in a form of QR-code. Then, the smart contract functionality will add the verified vote to the given ballot and denies the voter from voting again. In the end, the functionality of the smart contract is employed to generate the election result when the election lifetime is over and for each voting district. All of these steps are visually presented in Figure 2.
5) Verified transactions that were received before the time limit of the ongoing block has been reached, will be added to the block and when the time limit of the block is reached then it will be appended to the blockchain. After that, the new coming verified transactions will be added to a new block and so on.

IV. TrustVote System Implementation

This section focuses on the implementation details for TrustVote system. The same system was implemented on public blockchain; namely, Ethereum and on permissioned blockchain; namely, Hyperledger fabric. To limit any coerced voting, voters in TrustVote will have to vote in a supervised environment.

A. TrustVote implementation on Hyperledger

The model components of this work implementation from both Hyperledger Fabric and Chaincode is described as follows.

1) System model: TrustVote model contains the following:
Orderer (O): Is the entity from Hypeledger Fabric that handles consensus, and orders the events that happened on the blockchain correctly. This work implementation uses Apache Kafka[1] to set up a multinode cluster. Moreover, Apache Zookeeper[2] is used to manage the cluster nodes to enable highly reliable distributed coordination.
Peer (P): Is another entity from Hyperledger Fabric, which handles sending and receiving transactions to the blockchain and from other peers. Peers maintain the validity of the blockchain by validating the actions of other peers.
Certificate Authority (CA): The last entity from Hyperledger Fabric which is utilized. This node handles the construction and verification of participants on the blockchain, using standard PKI[3] methods.
Managerial Nodes(N_M): Are nodes that participate in consensus and maintain the integrity of the blockchain; they do so by hosting both a P and an O.
District Nodes (N_D): Behave exactly like N_M but also serve the role of allowing voters to cast votes on ballots.

Next, we will elaborate on the models and functionalities of a novel ballot and election smart contract for the proposed election as a smart contract, without the integration of a government identity verification service.

2) System Implementation: As stated earlier this section we used Fabric, and Hyperledger Composer to implement this system. In order to develop smart contracts in Composer, structures were created of the three classes, which are *Participants*, *Assets*, and *Transactions*. Participants are actors that interact with the smart contract. Assets are the logical entities that participants can own and trade with other participants. Lastly, the Transactions are the methods the participants utilize to move assets between themselves. For this implementation, the next concepts were created.
Participants: *Person* that is a representation of the voters in the system they belong to some district, *District* which

[1]https://kafka.apache.org/
[2]https://zookeeper.apache.org/
[3]Public Key Infrastructure

holds the ballots the voters vote on, and lastly *Election Administrators* that represent the people who initiate elections.
Assets: *Election* representing an ongoing election, it has some election administrator and a number of ballots. A *ballot* is owned by a district and has a number of proposals. Lastly *Proposal*, which has a description of the matter being voted on, the date when it was created, and a count of how many votes this proposal has.
Transactions: *Vote*, this transaction is called by voters referencing a given proposal, and *createElection* which is called by election administrators, and has a series of items that will be voted on.

Listing 1 shows the Assets in the system Business Network Archive (BNA) file that was created for TrustVote.

```
asset Election identified by electionId {
 o String electionId
 o Ballot[] ballots
}
asset Ballot identified by ballotId {
 o String ballotId
 o Proposal[] proposals
 --> Person chairperson
 --> District votingDistrict
}
asset Proposal identified by proposalId {
 o String proposalId
 o String description
 o Integer voteCount
 o DateTime creationDate
}
```

Listing 1. TrustVote BNA file on Hyperledger.

B. TrustVote implementation on Etherum

The same underlying functions were implemented as in TrustVote Hyperledger, but using Solidity and smart contracts. The next Listings show the main functionalities that were implemented.
Election Creation Smart Contract (ECSC) functionality: Takes in a list of candidates and districts along with the address of the wallet of the creator and the number of hours the election will take. The contract then creates a single smart contract for each district provided and puts the address of each smart contract created into the deployed Ballots array. ECSC functionality is shown in Listing 2 and Listing 3 in both Hyperledger and Ethereum consecutively.
Ballot Smart Contract (BSC) functionality: Which sets the manager of the ballot smart contract by default to the address of the wallet which created the election, the voting district of the smart contract to the district which the ECSC provided and then proceeds to fill the Candidates struct with the list of candidates provided and the number of votes for each candidate to 0. The BSC also stores the time of the creation of the contract along with the time when the contract is to expire, as shown in Listing 4. The same functionality was implemented using Hyperledger Chaincode.

```
async function createElection(createElection)
    {
const registry = await getAssetRegistry(
   orgNamespace + '.Election')
const factory = getFactory()
identificaiton
let newElection = factory.newResource(
   orgNamespace, 'Election', uuid())
newElection.ballots = []
let date = new Date()
await getAllDistricts().then((districts) => {
 districts.forEach((district) => {
  let newBallot = factory.newResource(
   orgNamespace, 'Ballot', uuid())
  newBallot.chairperson = createElection.
   creator
  newBallot.votingDistrict = district
  newBallot.proposals = []
  createElection.items.forEach((item) => {
   let newProposal = factory.newResource(
   orgNamespace, 'Proposal', uuid())
   newProposal.description = item
   newProposal.voteCount = 0
   newProposal.creationDate = date
   newBallot.proposals.push(newProposal)
  })
  newElection.ballots.push(newBallot)
 })
})
await registry.add(newElection)
```

Listing 2. ECSC functionality in Hyperledger.

```
function createElection(createElection){
 const registry =
 await getAssetRegistry(orgNamespace +
 '.Election')
 const factory = getFactory()
 let newElection =
 factory.newResource(orgNamespace,
 'Election', uuid())
 newElection.ballots = []
 let date = new Date()
 await getAllDistricts().then((districts)
 =>{ districts.forEach((district) => {
   let newBallot =
   factory.newResource(orgNamespace,
   'Ballot', uuid())
   newBallot.chairperson =
   createElection.creator
   newBallot.votingDistrict = district
   candidate in each ballot
   newBallot.proposals = []
   createElection.items.forEach((item)
   => { let newProposal =
   factory.newResource(orgNamespace,
   'Proposal', uuid())
    newProposal.description = item
    newProposal.voteCount = 0
    newProposal.creationDate = date
    newBallot.proposals.push(newProposal)
   })
   newElection.ballots.push(newBallot)})
 })
 await registry.add(newElection)}
```

Listing 3. ECSC functionality in Etherum.

Casting a vote: It allows voters to vote. The requirement for a voter to vote is that the mapping of the address of the voter is set to its default, false. If that is the case, the function guarantees that the election time limit has not been reached. If both requirements are satisfied, the contract retrieves the index of which candidate was voted for and increases his vote count by one and sets the mapping to true, so that the voter can never vote again in this particular election as shown in Listing 4 and Listing 5. Similar functionality was implemented in Ethereum.

```
struct Candidates {
bytes32 name;
uint voteCount;
uint creationDate;
uint expirationDate;}
Candidates[] public candidates;
address public manager;
bytes32 public votingDistrict;
mapping(address => bool) public voters;
modifier restricted() {
require(msg.sender == manager);
_;}
constructor (bytes32[] candidateNames,
bytes32 district, address creator,
uint amountOfHours) public {
manager = creator;
votingDistrict = district;
for (uint i = 0;
i < candidateNames.length; i++) {
candidates.push(Candidates({
name: candidateNames[i],
voteCount: 0,
creationDate: now,
expirationDate: now + amountOfHours }));  } }
```

Listing 4. BSC functionality.

```
async function vote(vote) {
 let proposal = vote.proposal
 let voter = vote.voter
 if (voter.alreadyVoted.includes(proposal.
    proposalId)) {
  throw new Error("Already voted on this
    proposal.") }
 var time_elapsed = (Date.now() - proposal.
    creationDate) / 1000
 if (time_elapsed > 30) {
  throw new Error("Proposal has timed out.") }
 proposal.voteCount += 1
 voter.alreadyVoted.push(proposal.proposalId)
 const personRegistry = await getAssetRegistry
    (orgNamespace + '.Person')
 await personRegistry.update(voter)
 const proposalRegistry = await
    getAssetRegistry(orgNamespace + '.Proposal
    ')
 await proposalRegistry.update(proposal)}
```

Listing 5. Casting a vote in TrustVote.

V. Evaluation and Results

In this section, TrustVote is evaluated in two ways. First, we evaluate the two implementations of TrustVote, by comparing the transaction execution time and second, by validating this system adherence to the requirements in Section V-B.

A. Experimental evaluation

In this evaluation, the impact of the implementation decision is studied, whether it is on public or permissioned blockchain, on the performance of TrustVote. Particularly, the performance is compared in terms of the execution time of key functions of the system, such as creating an election.

1) Hyperledger Fabric Setup: To evaluate the performance of this work implementation on a permissioned blockchain we started a local Hyperledger fabric blockchain was started, using a single laptop with Ubuntu 18.04, 16Gb of RAM, Intel i7-7600u running at 3.9GHz. The blockchain hosted a single smart-contract. The Hyperledger fabric components, that were described before, were all dockerized, and they were as follows, 2 peers, 2 orderers, 4 kafkas, 3 zookeepers, and 1 certificate authority. Hyperledger fabric version 1.4 was used. For all tests, an election with 5 districts was used, and with 20 items in an election.

2) Ethereum Setup: In order to setup Ethterum, we used the same underlying functions within TrustVote but using Solidity. We also used the same machine as when testing the Fabric system and we setup a local Ethereum testnet within the development framework Truffle[4]. For all tests, we used an election which has 5 districts, and 20 items in an election.

3) Experimental Results: The execution time for TrustVote main functions was measured on both Ethereum and Hyperledger using the aforementioned private setup. Figure 4 shows the results of the main functions in TrustVote; namely, (1) Election creation, (2) Casting a vote, and (3) Tallying votes. The results show that the permissioned blockchain performs with respect to time better than public blockchains in all of these three functions. Thus, permissioned blockchain should scale better as the number of participants increases.

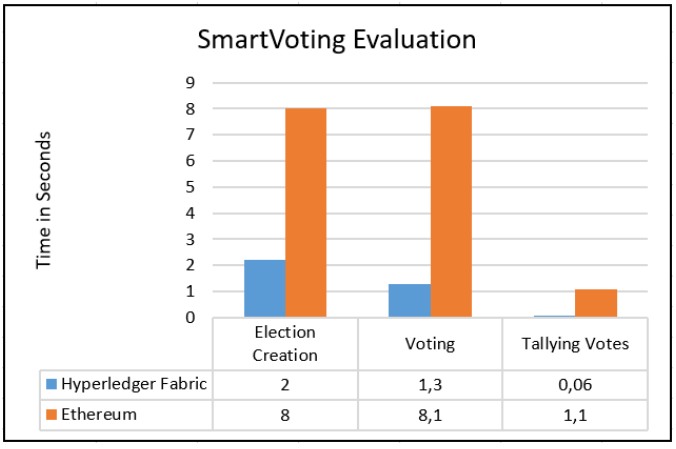

Fig. 4. TrustVote transaction execution time on two blockchain frameworks.

[4]https://github.com/trufflesuite/truffle

TABLE II
REQUIREMENTS OF TRUSTVOTE

Requirement	Met	Description
Flexibility	Yes	Dependent on each smart-contracts functionality.
Coercion-resistance	No	We only limited it by introducing a supervised environment at a voting center to cast their vote.
Consistency	Yes	This is guaranteed from using the blockchain.
Integrity	Yes	This is guaranteed from using the blockchain.
Auditability	Yes	This is guaranteed from using the blockchain and that we share the vote transaction ID with the corresponding voter so any voter can audit the process that his/her vote went through.
Identity Verification	Yes	The card system of Fabric is issued based on the electronic ID and PIN of each participant so their identity is confirmed using the card .
Eligibility	Yes	In order to perform transaction on TrustVote, a voter must first acquire a card for encryption/decryption. besides, eligible voters' lists are first generated and deployed with each district smart contract. Hence, if a voter is identified and he/she is not in the eligible list, he/she is not allowed to vote.
Voter Secrecy	No	In the blockchain one can read the history of a single vote to an identifier of a voter, this could be a random key and therefore be pseudonymous but not fully anonymous
Tamper-resistance	Yes	This property is provided by the blockchain
Uniqueness	Yes	This is provided via using OTP cards and after casting the first vote, the voter will be prevented by our system to vote again. Only one vote is accepted from each voter.
Universal Verifiability	Yes	All entities that have access to the Fabric can read the history, this can be multiple trusted organizations.
Individual Verifiability	No	This can be fulfilled by a public API hosted by one of the institutions maintaining the Fabric. Besides, since we are sharing the voter transaction ID with the corresponding voter, he/she can still use the API to trace his/her vote and verify that his/her vote was counted correctly.

B. Requirement mapping

We validated TrustVote conformance to the requirements outlined in Section II-B, to check if this system fulfills these requirements. Table II. shows that TrustVote meets most of the requirements. Two requirements (i.e., Voter Secrecy and Coercion-resistance) are not achieved. We proposed solutions for these unsatisfied requirements in the same table.

Final remark. Employing TrustVote in elections will result in overall better governance through supporting liquid democracy [12] and maintaining trust between the citizens and the governance. TrustVote also uses modern technologies (i.e., Blockchain and Smart Contracts), which can speed up the process of voting, limit the need to queue up at polling stations and the chance to follow an insufficient voting process. Moreover, implementing TrustVote in actual elections will not require governments to fully rebuild their systems, but remodel the available resources and systems to fit TrustVote technologies. Still, implementing TrustVote introduces the transparency in the election process, which might need to be addressed in new laws that provide regulations on implementing such transparency methods in official elections.

VI. CONCLUSION

In this paper, a state-of-the-art review of blockchain-based e-voting solutions is presented. Based on the survey, we identified a set of essential requirements that an e-voting system should satisfy. This paper also proposed an e-voting system (TrustVote) that was realised twice; using Hyperledger permissioned blockchain, and using Ethereum public blockchain. To evaluate TrustVote, first, we compared the two implementations of TrustVote based on the execution time of the different transactions. Then, we validated TrustVote against the defined requirements. This paper evaluation results showed that utilizing private permissioned blockchain for implementing e-voting systems can address many of the challenges and satisfy many of the requirements of e-voting systems. Particularly, private and permissioned blockchain solutions have the advantage of better performance in terms of the number and execution time of transactions and adhere better to privacy and governance constraints.

REFERENCES

[1] R. M. Alvarez, G. Katz, and J. Pomares, "The impact of new technologies on voter confidence in latin america: Evidence from e-voting experiments in argentina and colombia," *Journal of Information Technology & Politics*, vol. 8, no. 2, pp. 199–217, 2011.

[2] Valimised, "Statistics about internet voting in estonia," Available at https://www.valimised.ee/en/archive/statistics-about-internet-voting-estonia.

[3] G. G. Dagher, P. B. Marella, M. Milojkovic, and J. Mohler, "Broncovote: Secure voting system using ethereum's blockchain," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy, ICISSP 2018, Funchal, Madeira - Portugal, January 22-24, 2018*, P. Mori, S. Furnell, and O. Camp, Eds. SciTePress, 2018, pp. 96–107.

[4] M. Chaieb, S. Yousfi, P. Lafourcade, and R. Robbana, "Verify-your-vote: A verifiable blockchain-based online voting protocol," in *EMCIS*, ser. Lecture Notes in Business Information Processing, M. Themistocleous and P. R. da Cunha, Eds., vol. 341. Springer, 2018, pp. 16–30.

[5] Smartmatic, "Tivi: Accessible and verifiable online voting," Available at http://www.smartmatic.com/voting/online-voting-tivi/.

[6] "Follow my vote," Available at https://followmyvote.com.

[7] "Agora: Bringing our voting systems into the 21st century," agora.vote, p. 46, 2017.

[8] P. McCorry, S. F. Shahandashti, and F. Hao, "A smart contract for boardroom voting with maximum voter privacy," *IACR Cryptology ePrint Archive*, vol. 2017, p. 110, 2017.

[9] "Bit congress," Available at http://cryptochainuni.com/wp-content/uploads/ BitCongress-Whitepaper.pdf.

[10] M. Chaieb, M. Koscina, S. Yousfi, P. Lafourcade, and R. Robbana, "Dabsters: A privacy preserving e-voting protocol for permissioned blockchain," in *International Colloquium on Theoretical Aspects of Computing*. Springer, 2019, pp. 292–312.

[11] B. Yu, J. K. Liu, A. Sakzad, S. Nepal, R. Steinfeld, P. Rimba, and M. H. Au, "Platform-independent secure blockchain-based voting system," in *International Conference on Information Security*. Springer, 2018, pp. 369–386.

[12] F. Þ. Hjálmarsson, G. K. Hreiðarsson, M. Hamdaqa, and G. Hjálmtỳsson, "Blockchain-based e-voting system," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 983–986.