

SREE NARAYANA INSTITUTE OF TECHNOLOGY

Vadakkevila PO, Kollam -10

(Affiliated to University Of Kerala)



**DEPARTMENT OF COMPUTER APPLICATIONS
(MCA)**

CASE STUDY REPORT

“e-VOTING USING BLOCKCHAIN”

SREE NARAYANA INSTITUTE OF TECHNOLOGY

Vadakkevila PO, Kollam -10

(Affiliated to University of Kerala)



CASE STUDY REPORT

Year :

Semester :

Subject :

Name :

Adm.No : Re'g...No.....

Head of Department

Date

Faculty in Charge

Date.....

CONTENTS

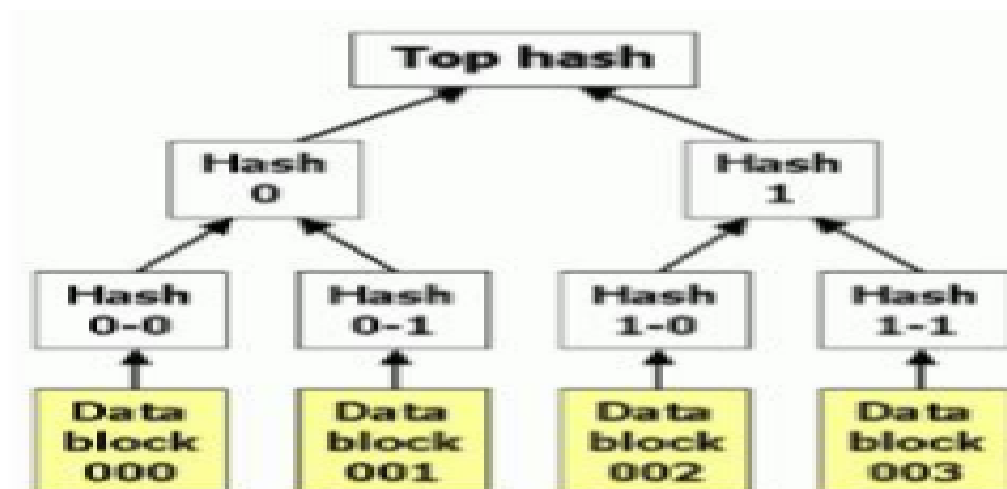
Sl no	Content	Page no
1	Abstract	4
2	Introduction	5
3	Motivation and Related work	7
4	Aim and Scope	11
5	Existing System	13
6	Proposed System	13
7	Project Design	14
7.1	Overall Project Planning	15
8	Sequence Diagram	35
8.1	Sequence Diagram for Virtual Tutor	35
9	E-R Diagram	36
9.1	E-R Diagram for Virtual Tutor	36
10	System Requirements	37
10.1	Software Requirements	37
11	Hardware Requirements	37
12	Conclusion	38
12.1	Future Enhancement	38
13	Reference	38

1.ABSTRACT

Online voting is a trend that is gaining momentum in modern society. It has great potential to decrease organizational costs and increase voter turnout. It eliminates the need to print ballot papers or open polling stations—voters can vote from wherever there is an Internet connection. Despite these benefits, online voting solutions are viewed with a great deal of caution because they introduce new threats. A single vulnerability can lead to large-scale manipulations of votes. Electronic voting systems must be legitimate, accurate, safe, and convenient when used for elections. Nonetheless, adoption may be limited by potential problems associated with electronic voting systems. Blockchain technology came into the ground to overcome these issues and offers decentralized nodes for electronic voting and is used to produce electronic voting systems mainly because of their end-to-end verification advantages. This technology is a beautiful replacement for traditional electronic voting solutions with distributed, non-repudiation, and security protection characteristics. The following article gives an overview of electronic voting systems based on blockchain technology. The main goal of this analysis was to examine the current status of blockchain-based voting research and online voting systems and any related difficulties to predict future developments. This study provides a conceptual description of the intended blockchain-based electronic voting application and an introduction to the fundamental structure and characteristics of the blockchain in connection to electronic voting. As a consequence of this study, it was discovered that blockchain systems may help solve some of the issues that now plague election systems. On the other hand, the most often mentioned issues in blockchain applications are privacy protection and transaction speed. For a sustainable blockchain-based electronic voting system, the security of remote participation must be viable, and for scalability, transaction speed must be addressed. Due to these concerns, it was determined that the existing frameworks need to be improved to be utilized in voting systems

2.INTRODUCTION

Blockchain technology that shines like a star after the entrance and widespread acceptance of Bitcoin [1], the very first cryptocurrency in peoples' everyday life, has become a trending topic in today's software world[2]. Blockchain technology originates from the underlying architectural design of the cryptocurrency bitcoin, where it was first introduced to the internet world and sooner became a promising technology due its high degree of transparency in the system it become an active field of research and study for its application various other fields For example, in Bitcoin, since the wallets are in a distributed structure, the total amount of coins and instant transaction volume in the world can be followed momentarily and clearly.



There is no need for a central authority to approve or complete the operations on this P2P-based system. Because of that, not only the money transfers but also all kinds of structural information can be kept in this distributed chain, and with the help of some crypto-logical methods, the system can be maintained securely. Like people's assets, marriage certificates, bank account books, medical information, etc., a lot of information can be recorded with this system with relevant modifications [3].

Ethereum coin (Ether), another cryptocurrency with multipurpose development environments, which emerged a few years after Bitcoin, distinguishes the blockchain in a real sense, revealing that this technology can produce software that can hold information that is structured as described above. The software programs enforced by smart contracts [4] (explained later) are written into the blockchain and are immutable. They cannot be (illegally) removed nor manipulated once written. Hence, they can work properly, autonomously and transparently forever, without any external stimuli [5]. Blockchain stores transactions in a block, the block eventually becomes completed as more transactions are carried out. Once complete it is then added in a linear, chronological order to the blockchain. The initial block in a blockchain is known as the

‘Genesis block’ or ‘Block 0’. The genesis block is usually hardcoded into the software; it is special in that it doesn’t contain a reference to a previous block. (‘Genesis Block’, 2015) Once the genesis block has been initialized ‘Block 1’ is created and when complete is attached to the genesis block. Each block has a transaction data part, copies of each transaction are hashed, and then the hashes are paired and hashed again, this continues until a single hash remains; also known as a merkle root (Figure 1). The block header is where the merkle root is stored. To ensure that a transaction cannot be modified each block also keeps a record of the previous blocks header, this means to change data you would have to A blockchain is designed to be accessed across a peer-to-peer network, each node/peer then communicates with other nodes for block and transaction exchange. Once connected to the network, peers start sending messages about other peers on the network, this creates a decentralised method of peer discovery. The purpose of the nodes within the network is to validate unconfirmed transactions and recently mined blocks, before a new node can start to do this it first has to carry out an initial block download. The initial block download makes the new node download and validate all blocks from block 1 to the most current blockchain,

once this is done the node is considered synchronised

Blockchain might be a suitable solution for e-voting projects. E-voting is being studied extensively, and many implementations are tested and even used for a while. However, very few implementations are reliable enough and are still in use. Of course, there are many successful examples of online polls and questionnaires, yet we cannot claim the same for online elections for governments and businesses. That's mainly because, official elections are essential elements of the democracy and democratic administrations, which are the most preferred administrative methodology in the modern world.

3.MOTIVATION AND RELATED WORKS

Our main motivation in this project is to provide a secure voting environment and show that a reliable e-voting scheme is possible using blockchain. Because, when e-voting is available for everyone who has a computer, or a mobile phone, every single administrative decision can be made by people and members; or at least people's opinion will be more public and more accessible by politicians and managers. This will eventually lead humanity to the true direct democracy [6]. It's important for us since elections can easily be corrupted or manipulated especially in small towns, and even in bigger cities located in corrupt countries. Plus, large-scale traditional elections are very expensive in the long term, especially if there are hundreds of geographically distributed vote centers and millions of voters [7]. Also, the voter turn-out at the voting centers is relatively low as the person might not be staying at the address his name is enrolled in the list, or he might be out for vacation or any other work. E-voting will be able solve these problems, if implemented carefully. The concept of e- voting is significantly older than blockchain. So that, all known examples so far used means of centralized computation and storage models.

Estonia is a very good example, since the government of Estonia is one of the first to implement a fully online and comprehensive e-voting solution [8]. The concept of e-voting was started to be debated in the country in 2001 and officially started by the national authorities in the summer of 2003 [9]. Their system is still in use, with many improvements and modifications on the original scheme. As reported, it is currently very robust and reliable. They use smart digital ID cards and personal card readers (distributed by the government) for person-wise authentication [10]. For citizens to attend the elections by listing the candidates and casting a vote, there is a special web portal as well as an equivalent desktop app. So that, anyone having a computer and Internet connection and also his/her ID car

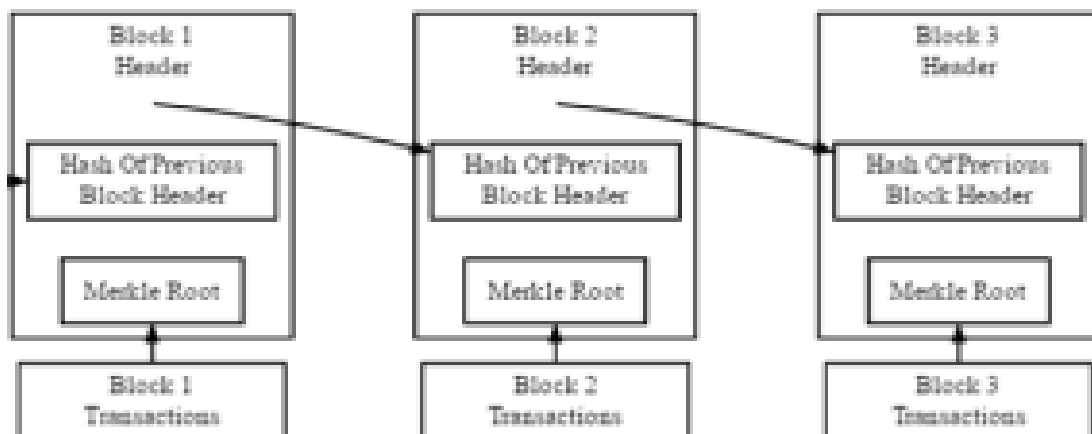


Figure-2 Simplified bitcoin blockchain(source-bitcoin.org)

PROBLEM STATEMENT AND PROJECT RELEVANCE

Current voting systems like ballot box voting or electronic voting suffer from various security threats such as [DDoS attacks](#), polling booth capturing, vote alteration and manipulation, malware attacks, etc, and also require huge amounts of paperwork, human resources, and time. This creates a sense of distrust among existing systems.

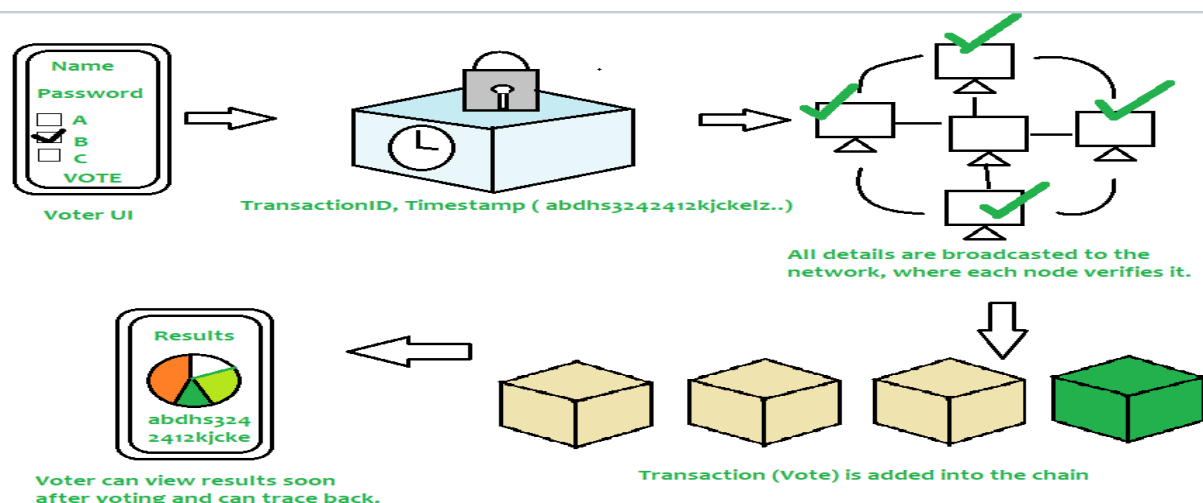
Some of the disadvantages are:

- Long Queues during elections
- Security Breaches like data leaks, vote tampering.
- Lot of paperwork involved, hence less eco-friendly and time-consuming.
- Difficult for differently-abled voters to reach polling booth.
- Cost of expenditure on elections is high.
- Solution :

Using blockchain, voting process can be made more secure, transparent, immutable, and reliable. How? Let's take an example. Suppose you are an eligible voter who goes to polling booth and cast vote using EVM (Electronic Voting Machine). But since it's a circuitry after all and if someone tampers with microchip, you may never know that did your vote reach to person for whom you voted or was diverted into another candidate's account? Since there's no tracing back of your vote. But, if you use blockchain- it stores everything as a transaction that will be explained soon below; and hence gives you a receipt of your vote (in a form of a transaction ID) and you can use it to ensure that your vote has been counted securely. Now suppose a digital voting system (website/app) has been launched to digitize process and all confidential data is stored on a single admin server/machine, if someone tries to hack it or snoop over it, he/she can

change candidate's vote count- from 2 to 22! You may never know that hacker installs malware or performs clickjacking attacks to steal or negate your vote or simply attacks central server. To avoid this, if system is integrated with blockchain- a special property called immutability protects system. Consider SQL, PHP, or any other traditional database systems. You can insert, update, or delete votes. But in a blockchain you can just insert data but cannot update or delete. Hence when you insert something, it stays there forever and no one can manipulate it- Thus name immutable ledger. But Building a blockchain system is not enough. It should be decentralized i.e if one server goes down or something happens on a particular node, other nodes can function normally and do not have to wait for victim node's recovery.

- You can vote anytime/anywhere (During Pandemics like COVID-19 where it's impossible to hold elections physically)
- Secure
- Immutable
- Faster
- Transparent



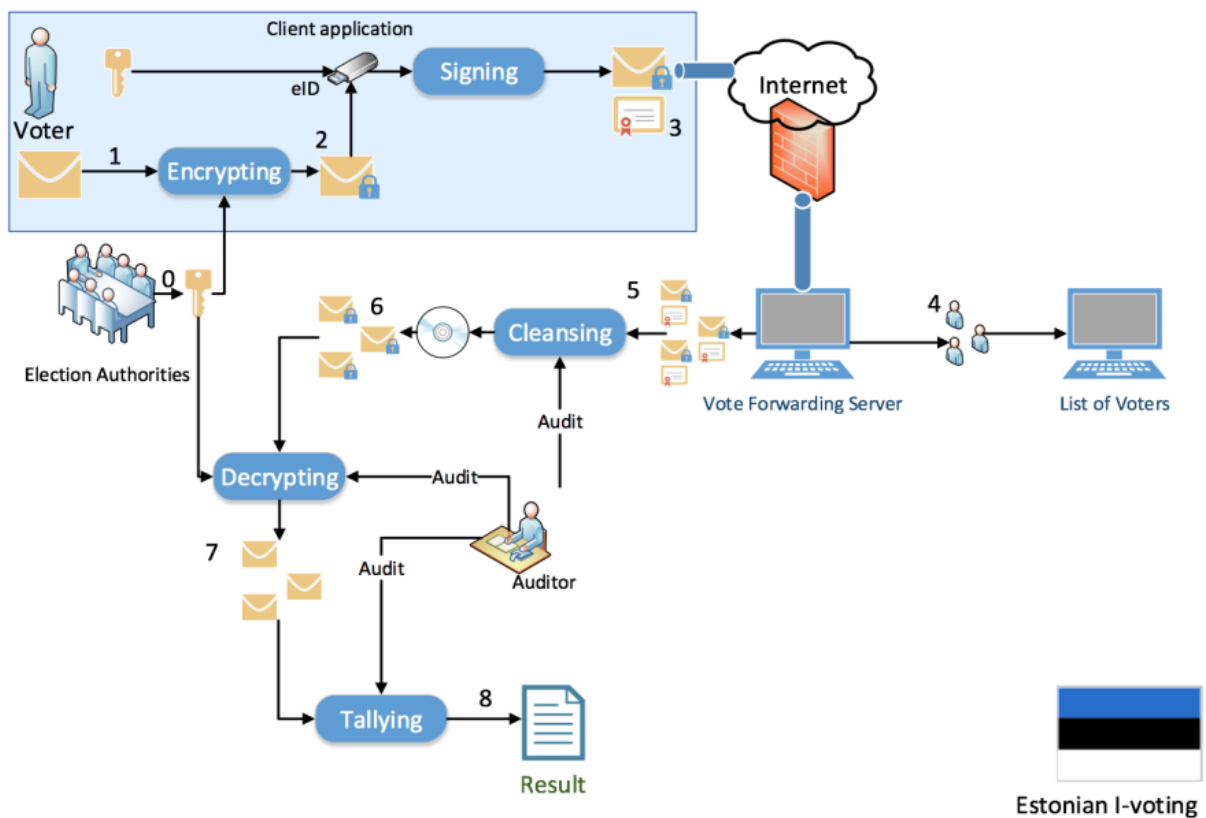
According to above diagram, voter needs to enter his/her credentials in order to vote. All data is then encrypted and stored as a transaction. This transaction is then broadcasted to every node in network, which in turn is then verified. If network approves transaction, it is stored in a block and added to chain. Note that once a block is added into chain, it stays there forever and can't be updated. Users can now see results and also trace back transaction if they want

4.AIM AND SCOPE

A number of digital voting systems are currently in use in countries around the world. We researched some of these systems to familiarise ourselves with current implementations, particularly Estonia. Estonia has had electronic voting since 2005 and in 2007 was the first country in the world to allow online voting. In the 2015 parliamentary election 30.5% of all votes were made through the nation's i-voting system (Vabariigi Valimiskomisjon, 2016). The basis of this system is the national ID card that all Estonian citizens are given. These cards contain encrypted files that identify the owner and allow the owner to carry out a number of online and electronic activities including online banking services, digitally signing documents, access their information on government databases and i-voting. (Electronic ID Card, no date) In order to vote, the voter must enter their card into a card reader and then access the voting website on the connected computer. They then enter their PIN number and a check is made to see if they are eligible to vote. Once confirmed, they are able to cast/change their vote up until four days before election day. The voter may also use a mobile phone to identify themselves for i-voting if they do not have a card reader for their computer. However, this process requires a specialised SIM card for the phone. (Estonian Ministry of Foreign Affairs, 2015) When a voter submits their vote, the vote is passed through the publicly accessible vote forwarding server to the vote storage server where it is encrypted and stored until the online voting period is over. Then the vote has all identifying information cleaned from it and is transferred by DVD to a vote counting server which is disconnected from all networks. This server decrypts and counts the votes and then outputs the results. Each stage of this process is logged and audited. During the 2013 Local Election, researchers observed and studied the i-voting process and highlighted a number of potential security risks with the system. One such risk is the possibility of

on the client side machine that monitors the user placing their vote and then later changing their vote to a different candidate. Another possible risk is for an attacker to directly infect the servers though malware being placed on the DVDs used to set up the servers and transfer the votes. (Springall et al., 2014) However, this report has also come under criticism from the Estonian Information Systems Authority.

(Veldre, 2014)



5.EXISTING SYSTEM

An online voting system is a platform that allows organizational members to cast their votes electronically, which can be through a website, mobile app, or any internet-connected device. You can conduct various types of elections through an online voting system. For example, you can use it for a simple majority vote, where the option or the candidate with the most votes wins. You can also use it for a more complex voting system like proportional representation, where each vote holds weight according to the voter's preference.

6.PROPOSED SYSTEM

When we talk about E-Voting, we recognize that it has been discussed for years in several ways that how to use the electronic technology to make people participate in the election digitally by registering their votes electronically. [2, 3]

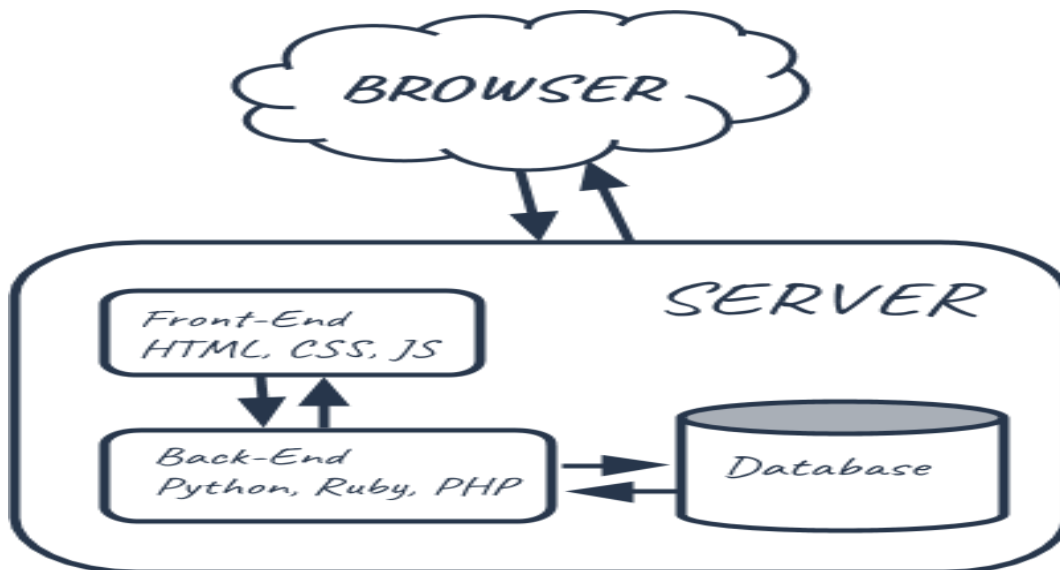
Only some countries are already using e-voting systems in elections, India use voter verified paper audit trail (VVPAT) machines for both Municipal and National elections in which voter can verify for whom they voted, and the stored results can be audited multiple times.⁴

Country like Estonia where 99% of public services are now online have been using internet voting i.e. I-voting platform since 2005 for all its elections. [5, 6] Before implementing any system there are security measures, constraints and requirements which need to be looked after.

7.PROJECT DESIGN

OVERALL PROJECT PLANNING AND IMPLEMENTATION

What is a Blockchain?

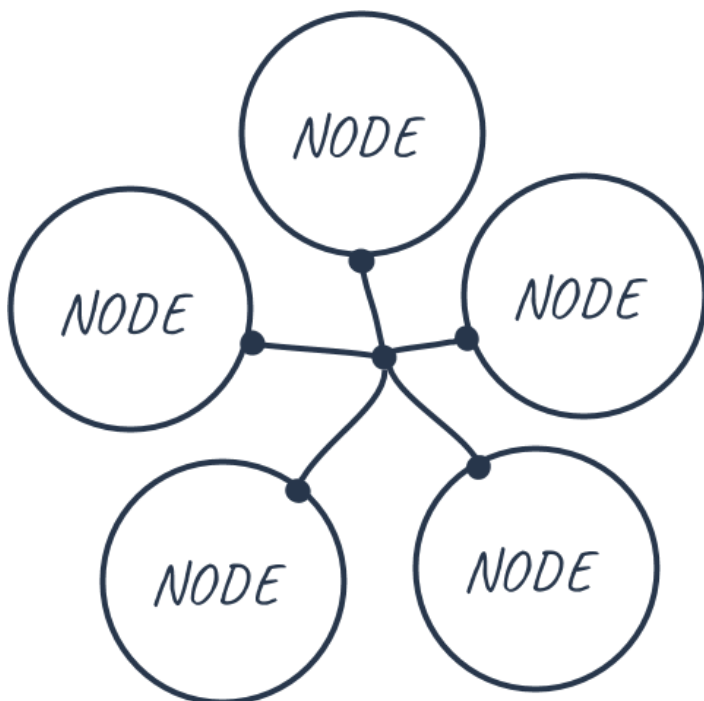


Normally when you interact with a web application, you use a web browser to connect to a central server over a network. All the code of this web application lives on this central server, and all the data lives in a central database. Anytime you transact with your application, must communicate with this central server on the web.

If we were to build our voting application on the web, we'd run into a few problems:

1. The data on the database could be changed: it could be counted more than once, or removed entirely.
2. The source code on the web server could also be changed at any time.

We don't want to build our app on the web. We want to build it on the blockchain where anyone connected to the network can participate in the election. We want to ensure that their votes are counted, and that they are only counted once. So let's take a look at how that works.



Instead of a centralized database, all the transaction data that is shared across the nodes in the blockchain is contained in *bundles of records called blocks, which are chained together* to create the public ledger. This public ledger represents all the data in the blockchain. All the data in the public ledger is secured by cryptographic hashing, and validated by a consensus algorithm. Nodes on the network participate to ensure that all copies of the data distributed across the network are the same. That's one very important reason why we're building our voting application on the blockchain, because we want to ensure that our vote was counted, and that it d

What would it look like for a user of our application to vote on the blockchain? Well, for starters, the user needs an account with a wallet address with some Ether, Ethereum's cryptocurrency. Once they connect to the network, they cast their vote and pay a small transaction fee to write this transaction to the blockchain. This transaction fee is called "gas". Whenever the vote is cast, some of the nodes on the network, called miners, compete to complete this transaction. The miner who completes this transaction is awarded the Ether that we paid to vote.

As a recap, when I vote, I pay a gas price to vote, and when my vote gets recorded, one of the computers on the network gets paid the my Ether fee. I in turn am confident my vote was recorded accurately forever.

So it's also important to note that voting on the blockchain costs Ether, but just seeing a list of candidates does not. That's because reading data from the blockchain is free, but writing to it is not.

What is a Smart Contract?

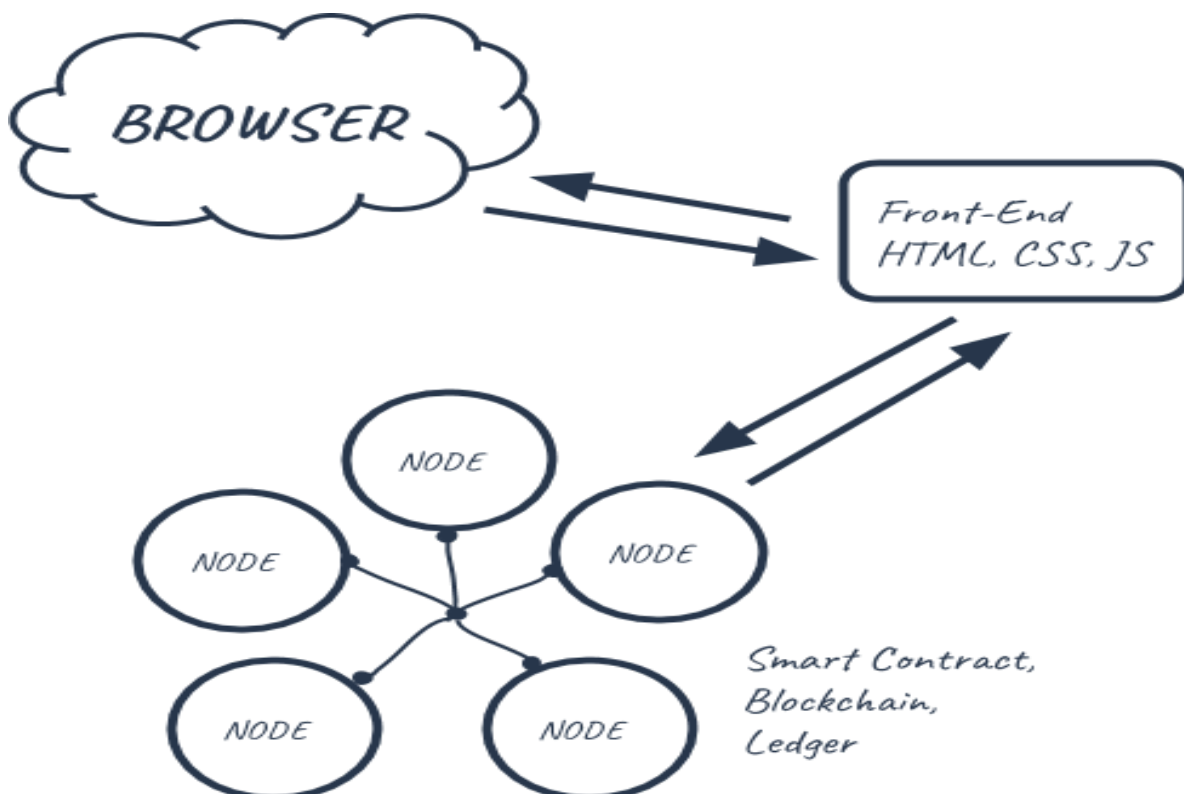
That's how the voting process works, but how do we actually code our app? Well, the Ethereum blockchain allows us to execute code with the Ethereum Virtual Machine (EVM) on the blockchain with something called a smart contract.

Smart contracts are where all the business logic of our application lives. This is where we'll actually code the decentralized portion our app. Smart contracts are in charge of reading and writing data to the blockchain, as well as executing business

logic. Smart contracts are written in a programming language called [Solidity](#), which looks a lot like Javascript. It is a full blown programming language that will allow us to do many of the same types of things Javascript is capable of, but it behaves a bit differently because of its use case, as we'll see in this tutorial.

The function of smart contracts on the blockchain is very similar to a microservice on the web. If the public ledger represents the database layer of the blockchain, then smart contracts are where all the business logic that transacts with that data lives.

Also, they're called smart contracts because they represent a covenant or agreement. In the case of our voting dApp, it is an agreement that my vote will count, that other votes are only counted once



We'll have a traditional front-end client that is written in HTML, CSS, and Javascript. Instead of talking to a back-end server, this client will connect to a local Ethereum blockchain that we'll install. We'll code all the business logic about our dApp in an Election smart contract with the Solidity programming language. We'll deploy this smart contract to our local Ethereum blockchain, and allow accounts to start voting.

Now we've seen what a blockchain is and how it works. We've seen why we want to build our voting dApp on the blockchain instead of the current web. And we've seen that we want to code our dApp by writing a smart contract that will be deployed to the Ethereum blockchain. Now let's jump in and start programming!

What We'll Be Building

Here is a demonstration of the voting dApp that we'll be building.

Election Results

#	Name	Votes
1	Candidate 1	0
2	Candidate 2	0

Select Candidate

Candidate 1

Vote

Your Account: 0x2191ef87e392377ec08e7c08eb105ef5448eced5

We'll build a client-side application that will talk to our smart contract on the blockchain. This client-side application will have a table of candidates that lists each candidate's id, name, and vote count. It will have a form where we can cast a vote for our desired candidate. It also shows the account we're connected to the blockchain with under "your account".

Installing Dependencies

The accompanying video footage for this portion of the tutorial begins at [8:53](#).

In order to build our dApp, we need a few dependencies first.

Node Package Manager (NPM)

The first dependency we need is [Node Package Manager](#), or NPM, which comes with Node.js. You can see if you have node already installed by going to your terminal and typing:

```
$ node -v
```

Truffle Framework

The next dependency is the [Truffle Framework](#), which allows us to build decentralized applications on the Ethereum blockchain. It provides a suite of tools that allow us to write smart contracts with the Solidity programming language. It also enables us to test our smart contracts and deploy them to the blockchain. It also gives us a place to develop our client-side application.

You can install Truffle with NPM in your command line like this:

```
$ npm install -g truffle
```

Ganache

The next dependency is [Ganache](#), a local in-memory blockchain. You can install Ganache by [downloading it from the Truffle Framework website](#). It will give us 10 external accounts with addresses on our local Ethereum blockchain. Each account is preloaded with 100 fake ether.

Metamask

The next dependency is the [Metamask extension for Google Chrome](#). In order to use the blockchain, we must connect to it (remember, I said the block chain is a network). We'll have to install a special browser extension in order to use the Ethereum block chain. That's where metamask comes in. We'll be able to connect to our local Ethereum blockchain with our personal account, and interact with our smart contract.

We're going to be using the Metamask chrome extension for this tutorial, so you'll also need to install the google chrome browser if you don't have it already. To install Metamask, search for the Metamask Chrome plugin in the Google Chrome web store. Once you've installed it, be sure that it is checked in your list of extensions. You'll see the fox icon in the top right hand side of your Chrome browser when it's installed. Reference the video walk through if you get stuck!

The dependency is optional, but recommended. I recommend installing syntax highlighting for the [Solidity](#) programming language. Most text editors and IDEs don't have syntax highlighting for Solidity out of the box, so you'll have to install a package to support this. I'm using [Sublime Text](#), and I've downloaded the ["Ethereum" package](#) that provides nice syntax highlighting for Solidity.

Smoke Test - Step 1

The accompanying video footage for this portion of the tutorial begins at [11:40](#). You can download the code for this portion of the tutorial [here](#). Feel free to use these as a reference point if you get stuck!

Ganache

ACCOUNTS

BLOCKS

TRANSACTIONS

LOGS

SEARCH FOR BLOCK NUMBERS OR TX HASHES

CURRENT BLOCK

0

GAS PRICE

20000000000

GAS LIMIT

6721975

NETWORK ID

5777

RPC SERVER

HTTP://127.0.0.1:7545

MINING STATUS

AUTOMINING

MNEMONIC

candy maple cake sugar pudding cream honey rich smooth crumble sweet treat

HD PATH

m/44'/60'/0'/0/account_index

ADDRESS

0x627306090abaB3A6e1400e9345bC60c78a8BEf57

BALANCE

100.00

ETH

TX COUNT

0

INDEX

0

ADDRESS

0xf17f52151EbEF6C7334FAD080c5704D77216b732

BALANCE

100.00

ETH

TX COUNT

0

INDEX

1

ADDRESS

0xC5fdf4076b8F3A5357c5E395ab970B5B54098Fef

BALANCE

100.00

ETH

TX COUNT

0

INDEX

2

ADDRESS

0x821aEa9a577a9b44299B9c15c88cf3087F3b5544

BALANCE

100.00

ETH

TX COUNT

0

INDEX

3

ADDRESS

0x0d1d4e623D10F9FBA5Db95830F7d3839406C6AF2

BALANCE

100.00

ETH

TX COUNT

0

INDEX

4

Ganache gave us 10 accounts preloaded with 100 fake Ether (this isn't worth anything on the main Ethereum network). Each account has a unique address and a private key.

Now let's create a project directory for our dApp in the command line like this:

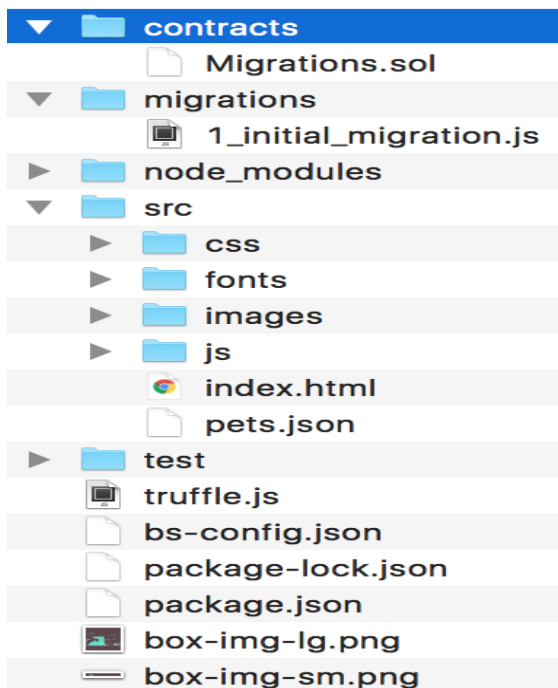
```
$ mkdir election
```

```
$ cd election
```

Now that we're inside our project, we can get up and running fast with a [Truffle box](#). We'll be using the [Pet Shop box](#) for this tutorial. From within your project directory, install the pet shop box from the command line like this:

```
$ truffle unbox pet-shop
```

Let's see what the pet shop box gave us:



contracts directory: this is where all smart contracts live. We already have a Migration

contract that handles our migrations to the blockchain.

migrations directory: this is where all of the migration files live. These migrations are similar to other web development frameworks that require migrations to change the state of a database. Whenever we deploy smart contracts to the blockchain, we are updating the blockchain's state, and therefore need a migration.

node_modules directory: this is the home of all of our Node dependencies.

src directory: this is where we'll develop our client-side application.

test directory: this is where we'll write our tests for our smart contracts.

truffle.js file: this is the main configuration file for our Truffle project

Now let's start writing our smart contract! This smart contract will contain all the business logic of our dApp. It will be in charge reading from and write to the Ethereum blockchain. It will allow us to list the candidates that will run in the election, and keep track of all the votes and voters. It will also govern all of the rules of the election, like enforcing accounts to only vote once. From the root of your project, go ahead and create a new contract file in the contracts directory like this:

```
$ touch contracts/Election.sol
```

Let's start by creating a "smoke test" that will ensure that we've set up our project properly, and that we can deploy the contract to the blockchain successfully. Open the file and start with the following code:

```
pragma solidity
0.4.2;

contract Election
{
    // Read/write
    candidate
        string public
    candidate;

    // Constructor
    function
    Election () public {
        candidate
        = "Candidate 1";
    }
}
```

Let me explain this code. We start by declaring the solidity version with the `pragma solidity` statement. Next, we declare the smart contract with the "contract" keyword, followed by the contract name. Next, we declare a state variable that will

store the value of the candidate name. State variables allow us to write data to the blockchain. We have declared that this variable will be a string, and we have set its visibility to `public`. Because it is public, solidity will give us a getter function for free that will allow us to access this value outside of our contract. We'll see that in action later in the console!

Then, we create a constructor function that will get called whenever we deploy the smart contract to the blockchain. This is where we'll set the value of the candidate state variable that will get stored to the blockchain upon migration. Notice that the constructor function has the same name as the smart contract. This is how Solidity knows that the function is a constructor.

Now that we've created the foundation for the smart contract, let's see if we can deploy it to the blockchain. In order to do this, we'll need to create a new file in the migrations directory. From your project root, create a new file from the command line like this:

```
$ touch migrations/2_deploy_contracts.js
```

Notice that we number all of our files inside the migrations directory with numbers so that Truffle knows which order to execute them in. Let's create a new migration to deploy the contract like this:

```
var Election =  
artifacts.require("./Election.sol");  
  
module.exports =  
function(deployer) {  
  
    deployer.deploy(Election  
    );  
  
};
```

First, we require the contract we've created, and assign it to a variable called "Election". Next, we add it to the manifest of deployed contracts to ensure that it gets deployed when we run the migrations. Now let's run our migrations from the command line like this:

```
$ truffle migrate
```

Now that we have successfully migrated our smart contract to the local Ethereum blockchain, let's open the console to interact with the smart contract. You can open the truffle console from the command line like this:

```
$ truffle console
```

Now that we're inside the console, let's get an instance of our deployed smart contract and see if we can read the candidate's name from the contract. From the console, run this code:

```
Election.deployed().then(function(instance) { app = instance
})
```

Here `Election` is the name of the variable that we created in the migration file. We retrieved a deployed instance of the contract with the `deployed()` function, and assigned it to an `app` variable inside the promise's callback function. This might look a little confusing at first, but you can reference [the console demonstration in the video at 21:50](#) for further explanation.

Now we can read the value of the candidate variable like this:

```
app.candidate()
// => 'Candidate
1'
```

Congratulations! You've just written your first smart contract, deployed to the blockchain, and retrieved some of its data.

The accompanying video footage for this portion of the tutorial begins at [27:11](#). You can download the code for this portion of the tutorial [here](#). Feel free to use these as a reference point if you get stuck!

Now that everything is set up properly, let's continue building out the smart contract by listing out the candidates that will run in the election. We need a way to store multiple candidates, and store multiple attributes about each candidate. We want to keep track of a candidate's id, name, and vote count. Here is how we will model the candidate:

```
contract Election
{
    // Model a
    Candidate
    struct
    Candidate {
        uint id;
        string
        name;
        uint
        voteCount;
    }
```

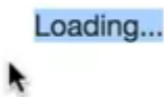


```
}
```

We have modeled the candidate with a [Solidity Struct](#). Solidity allows us to create our own structure types as we've done for our candidate here. We specified that this struct has an id of unsigned integer type, name of string type, and voteCount of unsigned integer type. Simply declaring this struct won't actually give us a candidate. We need to instantiate it and assign it to a variable before we can write it to storage.

The next thing we need is a place to store the candidates. We need a place to store one of the structure types that we've just created. We can do this with a [Solidity mapping](#). A mapping in Solidity is like an associative array or a hash, that associates key-value pairs. We can create this mapping like this:

Election Results



Loading...

Notice that your application says "Loading...". That's because we're not logged in to the blockchain yet! In order to connect to the blockchain, we need to import one of the accounts from Ganache into Metamask. You can [watch me set up Metamask in the video at 1:09:05](#).

Once you're connected with Metamask, you should see all of the contract and account data loaded.

Election Results

#	Name	Votes
1	Candidate 1	0
2	Candidate 2	0

Your Account: 0x0d80d9ce33320b0f1a0ec0bd75723a06a6fa9d28

```
// Render
candidate Result

var
candidateTemplate =
"<tr><th>" + id +
"</th><td>" + name +
"</td><td>" + voteCount +
"</td></tr>"

candidatesResults.append(ca
ndidateTemplate);

// Render
```

```
candidate ballot option
```

```
var
```

```
candidateOption = "<option  
value='" + id + "' >" +  
name + "</ option>"
```

```
candidatesSelect.append(can  
didateOption);
```

```
});
```

```
}
```

```
return
```

```
electionInstance.voters (App  
.account);
```

```
)).then(function (hasVoted)
```

```
{
```

```
// Do not allow
```

```
a user to vote
```

```
if (hasVoted) {
```

```
$('form').hide();
```

```
}
```

```
loader.hide();
```

```
content.show();
```

}

submitted:

$$=$$

```
)).then(function(result) {
```

```
    // Wait for
```

```
votes to update
```

```
$("#content").hide();
```

```
$("#loader").show();
```

```
}).catch(function(err) {
```

```
    console.error(err);
```

```
    });
```

```
}
```

First, we query for the candidateId in the form. When we call the vote function from our smart contract, we pass in this id, and we provide the current account with the function's "from" metadata. This will be an asynchronous call. When it is finished, we'll show the loader and hide the page content. Whenever the vote is recorded, we'll do the opposite, showing the content to the user again.

Now your front-end application should look like this:

Election Results

#	Name	Votes
1	Candidate 1	2
2	Candidate 2	0

Select Candidate

Candidate 1

Vote

Your Account: 0xc5fdf4076b8f3a5357c5e395ab970b5b54098fef

MetaMask Notification

CONFIRM TRANSACTION

Private Network

Account 2

C5fdf4...8Fef

100.000 ETH

114639.00 USD

>

4e7192...B60c

Amount

0 ETH

0.00 USD

Gas Limit

94378

UNITS

Gas Price

20

GWEI

Max Transaction Fee

0.001887 ETH

2.16 USD

Max Total

0.001887 ETH

2.16 USD

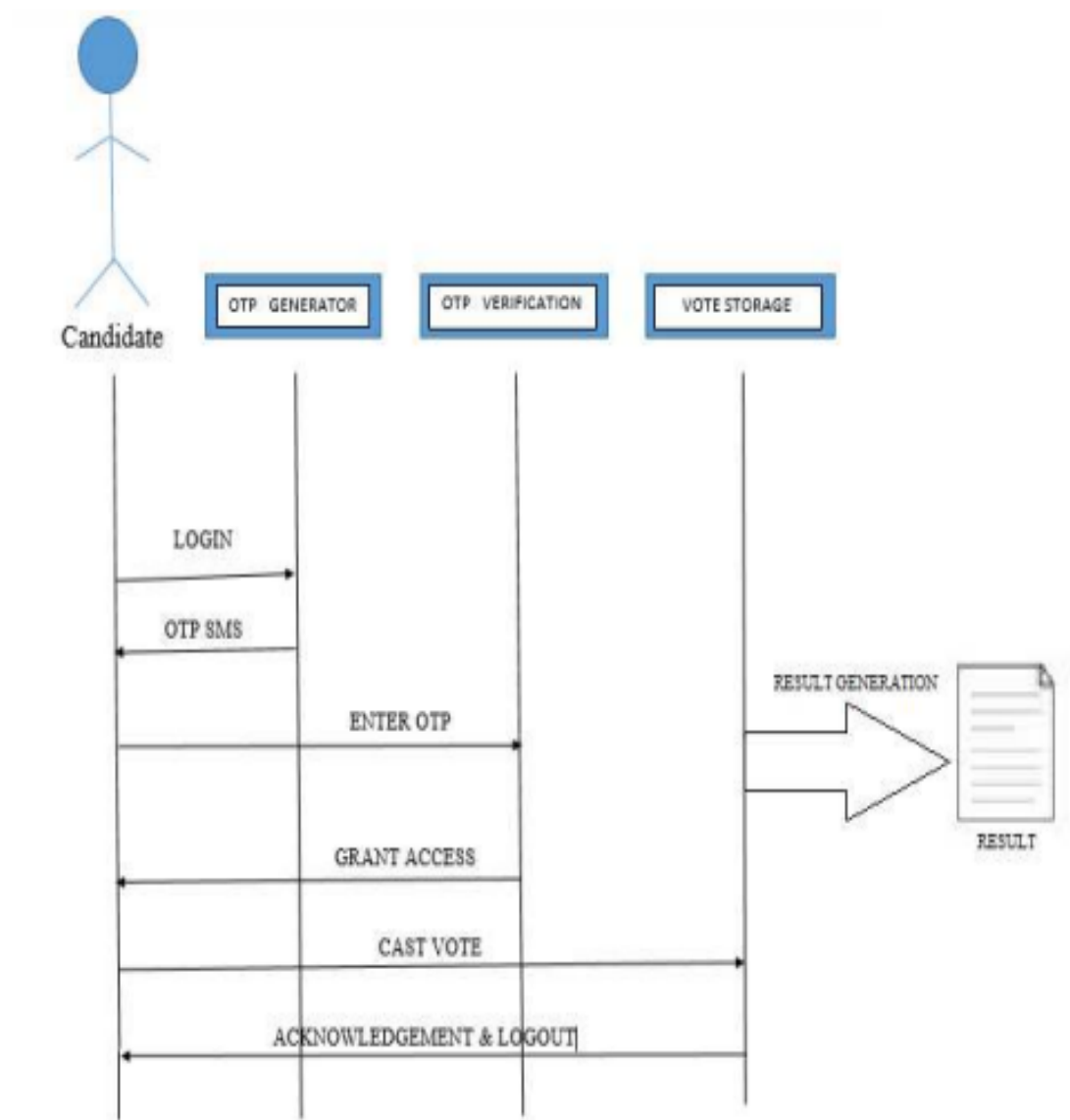
Data included: 36 bytes

RESET

SUBMIT

REJECT

8. SEQUENCE DIAGRAM



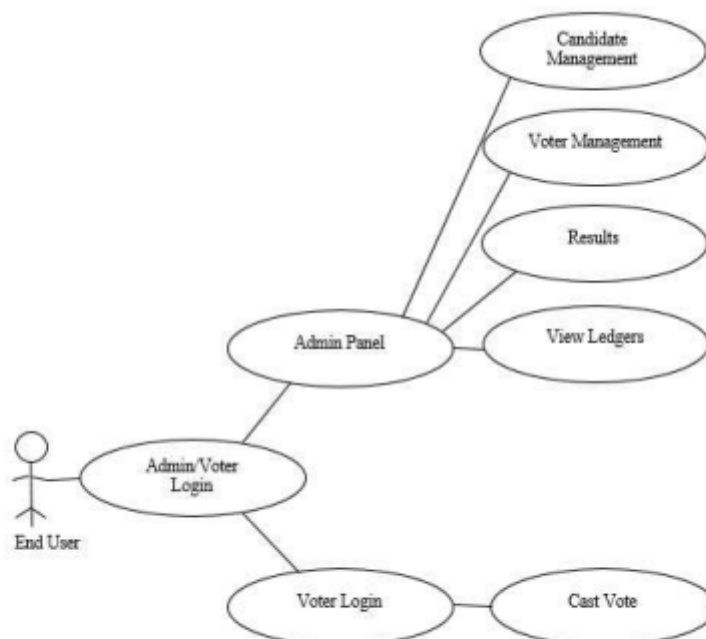
9.E-R DIAGRAM

The entity-relationship diagram is a data modeling technique that graphically represent an information system entities and relationships between those entities .An ER- diagram is a conceptual and representational model of data, which is used to represent the system framework infrastructure. The ER-diagram contains following elements:

- Entities
- Relationship
- Attributes

ENTITY RELATIONSHIP DIAGRAM FOR VIRTUAL TUTOR

Caption:-E-R Diagram is used for shows the relationship of entity sets stored in database of VIRTUAL TUTOR.



10.SOFTWARE REQUIREMENTS

PLATFORM:	WINDOWS 10
FRONT END	SOLIDITY,REACT JS
BACKEND:	ETHERIUMNETWORK

11.HARDWARE REQUIREMENTS

PROCESSOR :	INTEL CORE i5/8th
DISPLAY:	14.1” COLORMONITOR
MEMORY SIZE:	1.99GB RAM HARDDISK:
	250GB
KEYBOARD:	104 STANDARDS
CLOCK SPEED :	2.24GHZ

12.CONCLUSION AND FUTURE ENHANCEMENTS

- CONCLUSION

In this paper, we introduced a unique, blockchain-based electronic voting system that utilizes smart contracts to enable secure and cost-efficient election while guaranteeing voters privacy By comparison to previous work, we have shown that the blockchain technology offers a new possibility for democratic countries to advance from the pen and paper election scheme, to a more cost- and time- efficient election scheme, while increasing the security measures of the todays scheme and offer new possibilities of

transparency.

E-voting is still a controversial topic within both political and scientific circles. Despite the existence of a few very good examples, most of which are still in use; many more attempts were either failed to provide the security and privacy features of a traditional election or have serious usability and scalability issues [8]. On the contrary, blockchain-based e-voting solutions, including the one we have implemented using the smart contracts and the Ethereum network, address (or may address with relevant modifications) almost all of the security concerns, like privacy of voters, integrity, verification and non-repudiation of votes, and transparency of counting. Yet, there are also some properties that cannot be addressed solely using the blockchain, for example authentication of voters (on the personal level, not on the account level) requires additional mechanisms to be integrated, such as use of biometric factors [13].

Blockchain technology has lot of promise, but in its current state its require lot more research and currently might not reach till its full potential. There needs a concerted effort in the core blockchain technology to improve its support for more complex applications.

13. REFERENCES

- S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system", [Online]. Available: <https://bitcoin.org/bitcoin.pdf>.
- Ali Kaan Koç, Emre Yavuz, Umut Can Çabuk, Gökhan Dalkılıç "Towards Secure E-Voting Using Ethereum Blockchain"
- G. Wood, "Ethereum: a secure decentralised generalised transaction ledger", Ethereum Project Yellow Paper, vol. 151, pp. 1-32, 2014.
- C.D. Clack, V.A. Bakshi, and L. Braine, "Smart contract templates: foundations, design landscape and research directions", Mar 2017, arXiv:1608.00771.
- E. Maaten, "Towards remote e-voting: Estonian case", Electronic Voting in Europe-Technology, Law, Politics and Society, vol. 47, pp. 83-100, 2004.

- U.C. Çabuk, A. Çavdar, and E. Demir, "E-Demokrasi: Yeni Nesil Doğrudan Demokrasi ve Türkiye'deki Uygulanabilirliği", [Online] Available: https://www.researchgate.net/profile/Umut_Cabuk/publication/308796230_E-Democracy_The_Next_Generation_Direct_Democracy_and_Applicability_in_Turkey/links/5818a6d408aee7cdc685b40b/E-Democracy-The-Next-Generation-DirectDemocracy-and-Applicability-in-Turkey.pdf
- "Final report: study on eGovernment and the reduction of administrative burden (SMART 2012/0061)", 2014, [Online]. Available: <https://ec.europa.eu/digital-single-market/en/news/finalreport-study-egovernment-and-reduction-administrative-burdensmart-20120061>
- F. Hao and P.Y.A. Ryan, Real-World Electronic Voting: Design, Analysis and Deployment, CRC Press, pp. 143-170, 2017.
- N. Braun, S. F. Chancellery, and B. West. "E-Voting: Switzerland's projects and their legal framework–In a European context", Electronic Voting in Europe: Technology, Law, Politics and Society. Gesellschaft für Informatik, Bonn, pp.43-52, 2004.
- Nir Kshetri, Jeffrey Voas, "Blockchain-Enabled E-Voting".
- P. McCorry, S.F. Shahandashti, and F. Hao, "A smart contract for boardroom voting with maximum voter privacy", International Conference on Financial Cryptography and Data Security. Springer, Cham, pp. 357-375, 2017.
- U.C. Çabuk, T. Şenocak, E. Demir, and A. Çavdar, "A Proposal on initial remote user enrollment for IVR-based voice authentication systems", Int. J. of Advanced Research in Computer and Communication Engineering, vol 6, pp.118-123, July 2017.
- Y. Takabatake, D. Kotani, and Y. Okabe, "An anonymous distributed electronic voting system using Zerocoin", IEICE Technical Report, pp. 127-131, 2016.













