

BLOCK-CHAIN BASED E-VOTING SYSTEM WITH ACCESS CONTROL

A PROJECT REPORT

Submitted

*In the partial fulfillment of the requirements for
the award of the degree of*

Bachelor of Technology

in

Electronics and Communication Engineering

By

M. Srujitha – 171FA05244

I. Varshitha – 171FA05158

Under the esteemed guidance of

Dr. Sk. Jakeer Hussain

Professor, Dept. of ECE.



VIGNAN'S

Foundation for Science, Technology & Research

(Deemed to be University)

-Estd. u/s 3 of UGC Act 1956

(ACCREDITED BY **NAAC** WITH 'A' GRADE)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

**VIGNAN'S FOUNDATION FOR SCIENCE, TECHNOLOGY AND RESEARCH
(Deemed to be University)**

VADLAMUDI, GUNTUR – 522 213, INDIA

June-2021

CERTIFICATE

This is to certify that the project entitled **BLOCKCHAIN BASED E-VOTING SYSTEM WITH ACCESS CONTROL** is being submitted by **M. Nausheen, I. Varshitha, V. Harika, M. Srujitha** bearing **Regd. No. 171FA05105, 171FA05158, 171FA05207, 171FA05244** in partial fulfilment for the award of **IV** year II semester B. Tech degree in Electronics and Communication Engineering to Vignan's Foundation for Science Technology and Research, is a record of work carried out by them under the guidance of **Dr. Sk. Jakeer Hussain** of ECE Department.

Faculty guide

Dr. Sk. Jakeer Hussain

Professor

Head of the Department

Dr. T. Pitchaiah Ph.D., M.E.

Professor

DECLARATION

I hereby declare that the project work entitled “**BLOCK-CHAIN BASED E-VOTING SYSTEM WITH ACCESS CONTROL**” is being submitted to Vignan’s Foundation for Science, Technology and Research (Deemed to be University) in partial fulfillment for the award of B. Tech degree in Electronics and Communication Engineering. The work was originally designed and executed by us under the guidance of our supervisor Dr. Sk. Jakeer Hussain at Department of Electronics and Communication Engineering, Vignan’s Foundation for Science Technology and Research (Deemed to be University) and was not a duplication of work done by someone else. We hold the responsibility of the originality of the work incorporated into this thesis.

Signature of the candidate

M. Srujitha

171FA05244

I. Varshitha

171FA05158

ACKNOWLEDGEMENT

It's our privilege to express our deep sense of gratitude to our Guide, **Dr. Sk. Jakeer Hussain ,M.Tech, Ph.D**, Professor, VFSTR (Deemed to be University) for his able guidance and useful suggestions, which helped us in completing the project work, in time.

We would like to specially thank **Dr. T. Pitchaiah**, Head of the Department, ECE **Dr. G.N.V.R. Vikram & Mr. M. Sekhar** Project Coordinators, Department of ECE, VFSTR (Deemed to be University) for their help and support during the program.

We wish to express our gratitude to **Dr. M. Y. S. Prasad** Vice-Chancellor, VFSTR (Deemed to be University) for providing us the greatest opportunity to have a great exposure and to carry out the Project.

We would like to thanks all my faculty and Technical staff for their support to finish our work successfully.

We wish to express our heart full thanks to our parents for their support and encouragement throughout our life.

Name of the Students

M. Srujitha

171FA05244

I. Varshitha

171FA05158

ABSTRACT

Blockchain, which is the underlying technology of the first cryptocurrency ‘Bitcoin’, has drawn a lot of global attention in recent years. Its notable characteristics of the distributed ledger, trustless system, and immutability not only makes it a disruptive innovation in the electronic payment industry but also potential solutions for other areas that require trust establishment. Electronic voting (E-voting) scheme is a use-case where all attributes of blockchain can offer a mechanism for an open, fair and universally verifiable electoral process. Here, we review the requirements and then propose an E-voting system that utilizes the blockchain technology. The proposed system is empowered by Ethereum platform along with an attribute based access control scheme. The smart contract is deployed on “Metamask” which is an injected Web3 provider and tested for security and feasibility.

E-voting plays a significant role in social activities. The trust of the voting results and the privacy of each voter are always the most important concerns in designing secure e-voting system. Here, we design a decentralized trustless e-voting system using the smart contract. To guarantee the correctness of the voting result, the smart contract on blockchain is used to provide an audit of the votes casted and using the attribute based access control scheme enables any individual to check for their eligibility to vote.

In this project, we have designed one such smart contract for e-voting, where the voter can remotely participate in the election process while still maintaining anonymity, verifying their vote after the voting process and instant declaration of results as soon as the election process is completed. The attribute based access control scheme enables only persons with desired attributes to participate in the process by providing the necessary authorization.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Page No

1.1	Motivation	1
1.2	Objective	1
1.3	Technology Requirements	2

CHAPTER 2: LITERATURE REVIEW

2.1	Blockchain	4
2.1.1	Block	4
2.1.2	Blockchain	4
2.1.3	Cryptographic Hash Function	5
2.1.4	Hash Signature	5
2.1.5	Hashing	6
2.1.6	SHA-256	6
2.1.7	Properties of Hash Function	6
2.1.8	Proof of Work	7
2.1.9	Nonce	8
2.1.10	Gas	8
2.2	Attribute Based Access Control Scheme	8
2.3	Creation of new Block	9

CHAPTER 3: SOFTWARES USED AND FAMILIARIZATION

3.1	Ethereum	12
3.1.1	Ethereum Accounts	12
3.1.2	Ethers	13
3.1.3	Ethers Vs. Other Cryptocurrencies	13
3.1.4	Ethereum Transaction	13
3.1.5	Smart Contract	14
3.1.6	Benefits of Smart Contract	15
3.2	Remix- Ethereum IDE	15
3.2.1	Layout	16
3.2.2	File explorer	16
3.2.3	Compiler	16
3.2.4	Deploy & Run Transaction	17
3.2.5	Debugger	18
3.2.6	Statistical Analysis	19
3.2.7	Plugin Manager	19
3.2.8	Unit Testing	19
3.3	Metamask – Injected Web3 Provider	19
3.3.1	Features of Metamask	20
3.4	ERC20 Tokens	20
3.4.1	List of ERC standards	21

CHAPTER 4: DESIGN AND IMPLEMENTATION

4.1	Architecture of Attribute Based Access Control Scheme	25
4.2	Workflow Scenario	26
4.3	E-Voting Design	27

CHAPTER 5: CODE AND EXPLANATION

5.1	Solidity Code	30
5.2	Code Explanation	39
5.2.1	Context Contract	40
5.2.2	ERC20 Contract	40
5.2.3	Owned Contract	42
5.2.4	Ballot, RequestAt, RequestKey, AT, DO Contracts	42

CHAPTER 6: RESULTS AND ANALYSIS

6.1	Results	45
6.2	Analysis	57
6.2.1	Security Analysis	57

CHAPTER 7: ADVANTAGES, LIMITATIONS & APPLICATIONS

7.1	Advantages	60
7.2	Limitations	60
7.3	Applications	61

CONCLUSION AND FUTURE SCOPE

REFERENCES	63
-------------------	----

LIST OF FIGURES

Fig.No	Figure Name	Page. No
2.1	Interlinked blocks	4
2.2	Structure of block chain	5
2.3	Proof of Work	7
2.4	Creating new block	10
3.1	Ethereum logo	12
3.2	Ethereum transaction data fields	14
3.3	Remix IDE Layout	16
3.4	Deploy & Run Layout	18
3.5	Metamask Logo	20
4.1	Architecture of ABAC	25
4.2	Workflow	26
4.3	Voting System Design	27
6.1	Creation of Contract AT	45
6.2	Creation of Contract Ballot	46
6.3	Creation of Contract Context	46
6.4	Creation of Contract DO	47
6.5	Creation of ContractERC20	47
6.6	Creation of Contract Owned	48
6.7	Creation of Contract RequestAt	48
6.8	Creation of Contract RequestKey	49
6.9	Constructor Parameters for ERC20	49
6.10	Constructor Parameters for Ballot	49
6.11	Invoking setAddress1 Function	50
6.12	Registering voter1	50
6.13	Registering voter2	51
6.14	Registering voter3	51
6.15	Checking Attributes of Voter1	52
6.16	Checking Attributes of Voter2	52
6.17	Checking Attributes of Voter3	53
6.18	Requesting Key for Voter1	53
6.19	Requesting Key for Voter2	54
6.20	Requesting Key for Voter3	54
6.21	Voter1 Casting vote	55
6.22	Voter2 Casting vote	55
6.23	Voter3 Casting vote	56
6.24	Declaration of winner	56
6.25	Output incase of tie	57
6.26	Double voting error	58
6.27	Voting for second time	58

LIST OF ACRONYMS

AA	Attribute Authority
ABS	Attribute Based Signature
API	Application Program Interface
AT	Attribute Token(s)
DO	Data Owner
DU	Data User
ERC	Ethereum Request for Comments
ICO	Initial Coin Offerings
NFT	Non- Fungible Token
SD	Shared Data
SPDX	Software Package Data Exchange

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

Election is a formal process and allows members of a political body to vote and select individuals to become their leaders. This is a symbol of modern democracy. So, this process needs to be secure, fair and provide a convenient way for the citizens to access. The traditional paper ballot system is the most popular and easiest way. But, it is prone to malfunctioning and relies on trust that the officials are conducting the process honestly and fairly. In such cases, there is a chance that the results may get tampered without people's awareness.

Blockchain, a distributed append-only public ledger technology, was initially intended for the crypto-currencies. Emerging peer-to-peer (P2P) technology for distributed computing and decentralized data sharing. Due to the adoption of cryptography technology and centralized data storage, the blockchain can avoid the attacks that want to take control over the system. Ethereum, a transaction based state-machine, was presented to program the blockchain technologies.

1.2 OBJECTIVE

Here comes the solution of using blockchain technology to implement an e-voting system that ensures security, transparency and prevents the result from manipulation while preserving the voter's anonymity also allowing only people with desired attributes to vote. The code for this is written in solidity language on the ethereum platform.

This kind of E- voting schemes have been implemented by few countries and were successful to an extent. For example, due to its transparency and auditability features Sierra Leone a west African country conducted world first E-voting system using blockchain technology.

1.3 TECHNOLOGY REQUIREMENTS

1. Block Chain – Technology that acts as a database/ ledger
2. Ethereum – Blockchain based platform and operating system featuring smart contracts
3. Metamask – Injected Web3 Provider
4. Solidity – Language used to write the smart contracts

CHAPTER 2

LITERATURE REVIEW

2.1 BLOCKCHAIN

2.1.1 BLOCK:

In simplistic terms, a Block in a Blockchain consists of three main components.

1. **Data:** This could be any digital asset e.g. transactional records, medical records etc. that we want to preserve, and prevent any unintended tampering/alterations. The type of data (digital asset) stored in the block is determined by the type of Blockchain i.e. Bit Coin blockchain stores transactional records. Other blockchain might store voting records etc.

2. **“Hash” of the Current Block:** A unique fingerprint of the block, generated using a hashing algorithm, e.g. SHA256. If using the hash algorithm generates a different hash than the one recorded on the block, then we know that that block has been altered and should be discarded.

Hash of current Block = Hash (Hash of Previous Block + Data in Current Block + PoW).

3. **“Hash” of the Previous Block:** The Hash of the previous block along with the data of the current Block and the Proof-of-Work is used to generate the Hash of the current Block. i.e. This is how the Blocks become “linked” in a blockchain.

Note: The first Block of the Block chain is called in the **“Genesis Block”**.

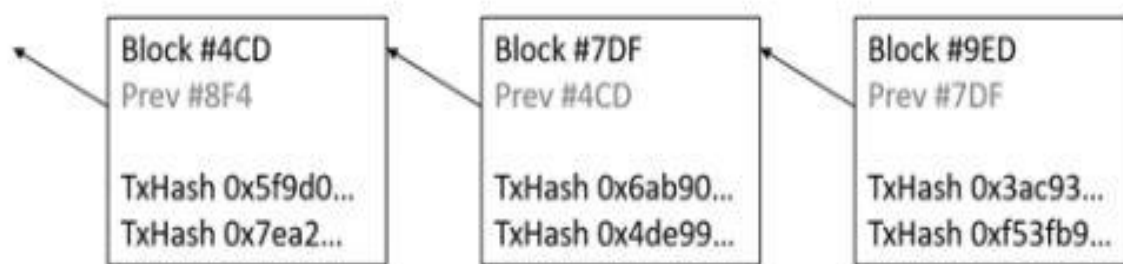


Fig 2.1: Interlinked Blocks

2.1.2 BLOCK CHAIN

. Most recently, the blockchain a paradigm shift, is transforming all the major application areas of IoT by enabling a decentralized environment with anonymous and trustful transactions. Combined with the blockchain technology, IoT systems benefit from the lower operational cost, decentralized resource management, robustness against threats and attacks, and so on. Therefore, the convergence of IoT and blockchain technology aims to overcome the significant challenges of realizing the IoT platform in the near future. Blockchain, a distributed

append-only public ledger technology, was initially intended for the crypto-currencies, e.g. Bitcoin. In 2008, Nakamoto introduced the concept of blockchain that has attracted much attention over the past years as an emerging peer-to-peer (P2P) technology for distributed computing and decentralized data sharing. Due to the adoption of cryptography technology and without a centralized control actor or a centralized data storage, the blockchain can avoid the attacks that want to take control over the system.

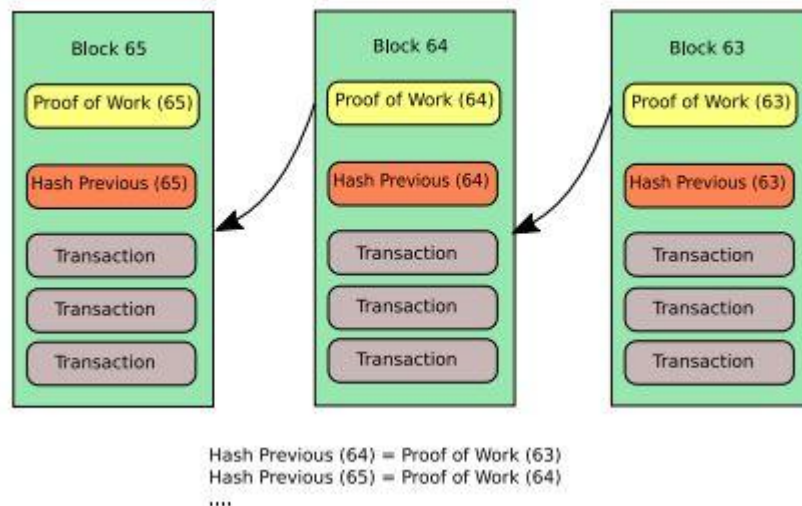


Fig 2.2: Structure of Blockchain

2.1.3 CRYPTOGRAPHIC HASH FUNCTION

A hash function, will take any transaction/data input and rehash it to produce an output of a fixed size. The process of using a given hash function to process a transaction is called hashing. The transactional output of that given hash function is what we call a hash. The basic characteristic of any given hash function lies in the size of its output. This is what gives us the different hash functions.

Simply, a hash is a function that converts an input of letters and numbers into an encrypted output of a fixed length. A hash is created using an algorithm and is essential to blockchain management in cryptocurrency.

2.1.4 HASH SIGNATURE

Every Block in the blockchain is “sealed” with a hash signature. This hash signature is generated by passing three components to any predefined hashing algorithm.

1. Data in the Current Block
2. Proof-of-Work (obtained through the “mining” process)

3. Hash of the previous Block

To verify the integrity of the Block, any user can simply check the “sealing” hash signature on the Block and compare it with the “hash” generated by passing the above three components through the hashing algorithm.

2.1.5 HASHING

Hashing in blockchain refers to the process of having an input item of whatever length reflecting an output item of a fixed length. If we take the example of blockchain use in cryptocurrencies, transactions of varying lengths are run through a given hashing algorithm, and all give an output that is of a fixed length. This is regardless of the length of the input transaction. The output is what we call a hash. A good example is bitcoin’s Secure Hashing Algorithm 256 (SHA-256).

2.1.6 SHA-256

Hashing using SHA-256 always gives an output result of a fixed length, which has a 256-bits length (the output is 32 bytes). This is always the case whether the transaction is just a single word or a complex transaction with huge amounts of data. What this means is that keeping track of a transaction becomes easier when you can recall/trace the hash. The size of the hash will depend on the hash function utilized, but the output of using a particular hashing algorithm will be of a specific size.

Example:

When you take a YouTube video of say 50 megabytes and hash it using SHA-256, the output will be a hash of 256-bits in length. Similarly, if you take a text message of 5 kilobytes, the output hash will still be 256-bits. The only difference between the two will be the hash pattern.

2.1.7 PROPERTIES OF HASH FUNCTIONS

1. Deterministic: A hash function needs to have a fixed or specific output. What this means is that it doesn’t matter what number of times you process a given input using a hash function, the result is always of the same length.

2. Quick Computation

3. Pre-image Resistance: One of the important properties of secure cryptographic hash functions is they are one-way. Let's take it this way: given a hash of a particular transaction, it should be virtually impossible or practically infeasible to determine the original input data using this output. This property lends a level of security to the blockchain. When given a particular hash, the only possible way of finding what the original input data is if you hashed all the possible combinations of inputs until you eventually hash the correct or corresponding input. However, because the input data is randomized, hashing it is practically impossible.

4. Randomized hashes: Hash functions produce different outputs for every input, even if the input data differs by only a digit or letter. For instance, the hash of the word "Alpha" should be completely different from the hash of the word "Alpha1". If the patterns were to be similar and differ only at the end, then deciphering them would be easy.

2.1.8 PROOF OF WORK

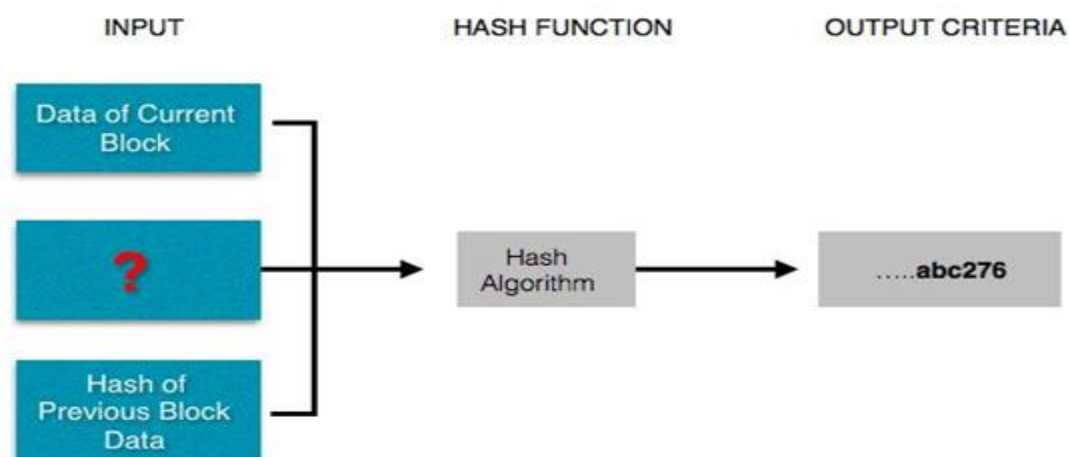


Fig 2.3 Proof of work

The ‘?’ in the above figure is the **Proof of Work** which is calculated while mining. The computation of the input string that generated a particular output/hash requires a great deal of computational power, and is called “mining”. Proof of Work (PoW) is the original consensus algorithm in a blockchain network. The algorithm is used to confirm the transaction and creates a new block to the chain. In this algorithm, miners, a group of people, compete against each other to complete the transaction on the network. The process of competing against each other is called **mining**. As soon as miners successfully created a valid block, he gets rewarded. The most famous application of Proof of Work (PoW) is the bitcoin.

For example, block is composed of a block number, data field, cryptographic hash associated with it and a nonce. The nonce is responsible for making the block valid. In the puzzle game, bitcoin software creates a challenge, and the game begins. This game involves all miners competing against each other to solve the challenges, and this challenge will take approximately 10 minutes to be completed. Every single miner starts trying to find the solution to that one nonce that will satisfy the hash for the block. At some specific point, one of those miners in the global community with higher speed and great hardware specs will solve the cryptography challenge and be the winner of the game. Now, the rest of the community will start verifying that block which is mined by the winner. If the nonce is correct, it will end up with the new block that will be added to the blockchain. The concept of generating a block provides a clear explanation of proof of work (PoW).

2.1.9 NONCE

A **nonce** is an abbreviation for "number only used once," which is a number added to a hashed or encrypted block in a blockchain that, when rehashed, meets the difficulty level restrictions. The nonce is the number that blockchain miners are solving for. When the solution is found, the blockchain miners are offered cryptocurrency in exchange.

2.1.10 GAS

There is a second type of token that is used to pay miners fees for including transactions in their block, it is called gas, and every smart contract execution requires a certain amount of gas to be sent along with it to entice miners to put it in the blockchain. Gas is a transaction pricing-based mechanism to mitigate spam and allocate resources on the network. Minors on the network defines the price of the gas and if a transaction is less the defined gas it will be declined.

2.2 ATTRIBUTE BASED ACCESS CONTROL SCHEME:

In the proposed scheme, ethereum smart contracts are created to define the interactions between data owner, data user, and attribute authorities. A data user presents its attributes to different attribute authorities, and after successful validation of attributes, obtains attribute tokens from respective attribute authorities. After collecting enough attribute tokens, a smart contract will be executed to issue secret key to the data user to access the requested object. Broadly speaking, attribute-based access control has been defined as an access control method

where subject requests to perform operations on objects are granted or denied based on assigned attributes of the subject, assigned attributes of the object, environment conditions, and a set of policies that are specified in terms of those attributes and conditions.

In this scheme, data owner generates a secret key and encrypt the shared data with the AES algorithm, and keeps the secret key with himself. Within ethereum's smart contracts, a data user presents attributes to respective authorities to obtain attribute tokens after successful validation. Upon collecting enough attribute tokens from multiple attribute authorities, the data user will receive the AES secret key capable of accessing the request data upon which the individual is allowed to vote.

2.3 CREATION OF NEW BLOCK

The blockchain structure is composed of a sequence of blocks, which are linked together by their hash values. In the blockchain network, a public ledger maintains the digitally signed transactions of the users in a P2P network. In general, a user has two keys: 1) a public key for other users for the encryption and 2) a private key to read an encrypted message. From the blockchain perspective, the private key is used for signing the blockchain transaction and the public key represents the unique address. Asymmetric cryptography is used to decrypt the message encrypted by the corresponding public key. At the initial stage, an user signs a transaction using its private key and broadcasts it to its peers. Once the peers receive the signed transaction, they validate the transaction and disseminate it over the network. All the parties who are involved in the transactions mutually validate the transaction to meet a consensus agreement. Once a distributed consensus is reached, the special node, called as miner, includes the valid transaction into a time-stamped block. The block, which is included by the miner, is broadcast back into the network. After validating the broadcast block, which contains the transaction, as well as hash-matching it with the previous block in the blockchain, the broadcast block is appended to the blockchain.

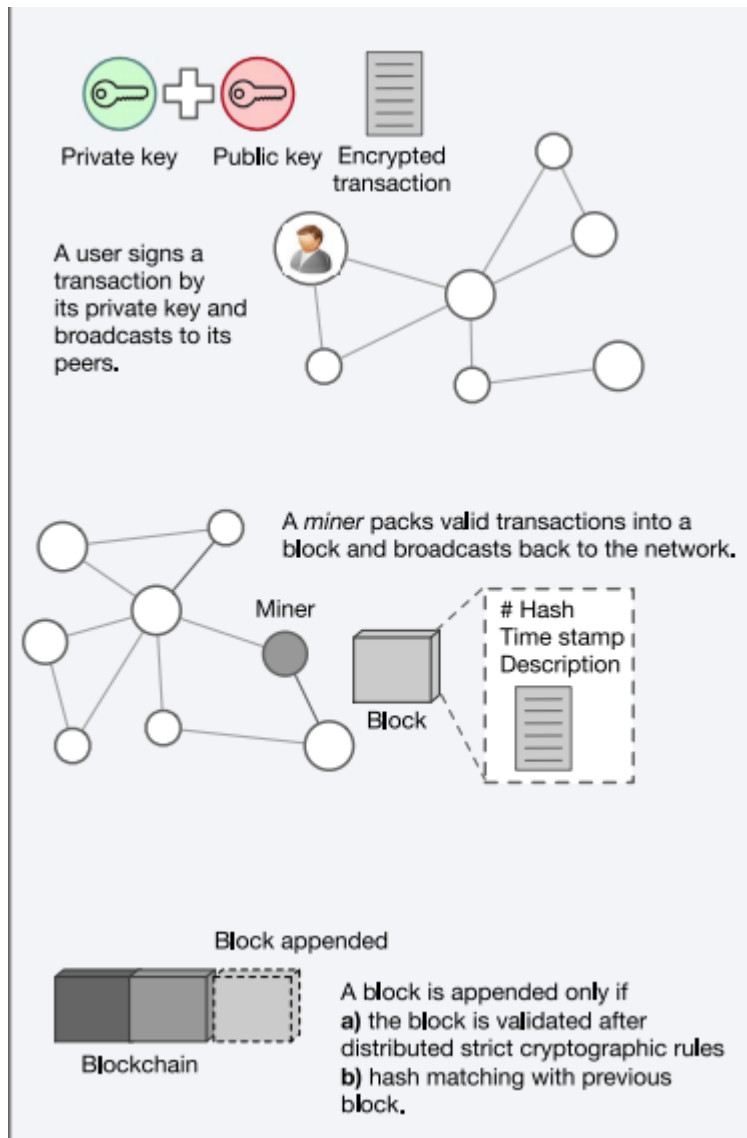


Fig 2.4 Creating New Block

CHAPTER 3

SOFTWARES USED AND FAMILIARIZATION

3.1 ETHEREUM:

An open-source, public blockchain-based distributed computing platform and operating system featuring smart contract (scripting) functionality. The Ethereum Virtual Machine (EVM) is provided by Ethereum which is a decentralized virtual machine for executing smart contract code on ethereum nodes. Ethereum network is permission less i.e. any node can join ethereum network if user downloads ethereum client to create account.



Fig 3.1 Ethereum Logo

3.1.1 ETHEREUM ACCOUNTS

Within Ethereum, there are two types of accounts: Externally Owned Accounts (EOA) and Contract Accounts, both of which are uniquely identified by a 20-byte hexadecimal string as their address. An EOA is controlled by its owner's private key, has an available ether balance, and can send transactions. For instance, send a message to another account for transferring ether or trigger the execution of a smart contract. It has no associated code with EOA account. While a contract account also has an ether balance, but contains the associated code which can be triggered by a transaction or from other smart contract. Ether is the official cryptocurrency used in Ethereum platform.

All the ether balances and values are represented in units of wei: 1 ether is $1e18$ wei. The Ethereum platform has a smart contract running environment, which is known as the Ethereum Virtual Machine (EVM). Each mining node in the ethereum network runs EVM as part of the validation procedure and later perform the same results and store the data. Node receives, broadcast, verifies, and executes the transaction in ethereum network. Every operation in EVM has a specific cost, which is counted by the amount of gas. The sender of the transaction needs to pay for ether for the operations and the total transaction cost is estimated as:

$$\text{Ether} = \text{Gas Used} * \text{Gas Price}.$$

3.1.2 ETHERS

Ethereum is not owned by anyone. All of the programs and services linked with the network require computing power and that power is not free. Ether is the solution to the issue of payment a digital asset-bearer like a bond or other security. You can call it the cryptocurrency of the ethereum network. Just like cash, it doesn't require a third party to process or approve transactions. It should be considered as fuel for the apps on the decentralized ethereum network.

For instance, say there is an app on the ethereum network that allows you to create, modify, and delete simple notes. In order to complete any of these tasks, the app requires processing power via the network. To cover the cost of this power, you likely need to pay a marginal fee anytime you wish to make any changes to your existing notes. Ether is the token by which you make this payment. It is, in a sense digital oil in that it allows the network to process the changes you've made. As a type of fuel, it then makes sense that ether transaction fees will be different depending upon how much fuel is required for the service.

3.1.3 ETHERS VS. OTHER CRYPTOCURRENCIES

Each particular action on the ethereum network or in a decentralized app requires a different amount of computational power and time. The greater the power and time required, the higher the ether fee for the action to be completed. In this way, ether is different from a digital currency like bitcoin. There are also other ways that it differs. For instance, many digital currencies have hard caps or maximum numbers of tokens or coins that can be mined. With ether, there are no limits. A total of 18 million ether are mined each year. 12 million went to the Ethereum Foundation, a collective of developers and analysts who work to enhance the ethereum network and the underlying technology. Five ether tokens (ETH) are allotted to the miners that verify transactions on the network every 12 seconds.

3.1.4 ETHEREUM TRANSACTION

The Ethereum transaction is a data package which enables user to transfer ether from one account to another account. In addition, it can also trigger the execution of the code in the smart contract through one transaction. One unique Ethereum transaction consists of the following data field: Nonce, represents for the transaction sequence number from the sender. Receiver address, which contains the receiver account information. Gas price, the price user

offer to pay. Gas limit, maximum amount of gas allowed for one transaction. Amount, the total ether balance transferred to the destination address. V, R, and S, together makes up the Elliptic Curve Digital Signature Algorithm (ECDSA) for sender's signature. Data, optional additional data fields which can be put into any data.

Nonce
Receiver Address
Gas Price
Gas Limit
Amount
V
R
S
Data

Fig 3.2 Ethereum Transaction Data Fields

3.1.5 SMART CONTRACT

Smart contracts are simply programs stored on a blockchain that run when predetermined conditions are met. They typically are used to automate the execution of an agreement so that all participants can be immediately certain of the outcome, without any intermediary's involvement or time loss. They can also automate a workflow, triggering the next action when conditions are met.

Smart contracts work by following simple "if/when...then..." statements that are written into code on a blockchain. A network of computers executes the actions when predetermined conditions have been met and verified. These actions could include releasing funds to the appropriate parties, registering a vehicle, sending notifications, or issuing a ticket. The blockchain is then updated when the transaction is completed. That means the transaction cannot be changed, and only parties who have been granted permission can see the results.

Within a smart contract, there can be as many stipulations as needed to satisfy the participants that the task will be completed satisfactorily. To establish the terms, participants must determine how transactions and their data are represented on the blockchain, agree on the "if/when...then..." rules that govern those transactions, explore all possible exceptions, and

define a framework for resolving disputes. Then the smart contract can be programmed by a developer.

3.1.6 BENEFITS OF SMART CONTRACT

1. **Speed, Efficiency and Accuracy:** Once a condition is met, the contract is executed immediately. Because smart contracts are digital and automated, there's no paperwork to process and no time spent reconciling errors that often result from manually filling in documents.
2. **Trust and Transparency:** Because there's no third party involved, and because encrypted records of transactions are shared across participants, there's no need to question whether information has been altered for personal benefit.
3. **Security:** Blockchain transaction records are encrypted, which makes them very hard to hack. Moreover, because each record is connected to the previous and subsequent records on a distributed ledger, hackers would have to alter the entire chain to change a single record.
4. **Savings:** Smart contracts remove the need for intermediaries to handle transactions and, by extension, their associated time delays and fees.

Applications: Reimagine dispute resolution, Build a Resilient supply chain, Bring trust to trade finance, Reinvent bank letters of guarantee.

3.2 REMIX-ETHEREUM IDE:

Remix IDE is an open source web and desktop application. It fosters a fast development cycle and has a rich set of plugins with intuitive GUIs. Remix is used for the entire journey of contract development as well as being a playground for learning and teaching Ethereum. Remix IDE is part of the Remix Project which is a platform for development tools that use a plugin architecture. It encompasses sub-projects including Remix Plugin Engine, Remix Libs, and of course Remix-IDE.

Remix IDE is a powerful open source tool that helps you write Solidity contracts straight from the browser. It is written in JavaScript and supports both usage in the browser, in the browser but run locally and in a desktop version. Remix IDE has modules for testing, debugging and deploying of smart contracts and much more.

3.2.1 LAYOUT

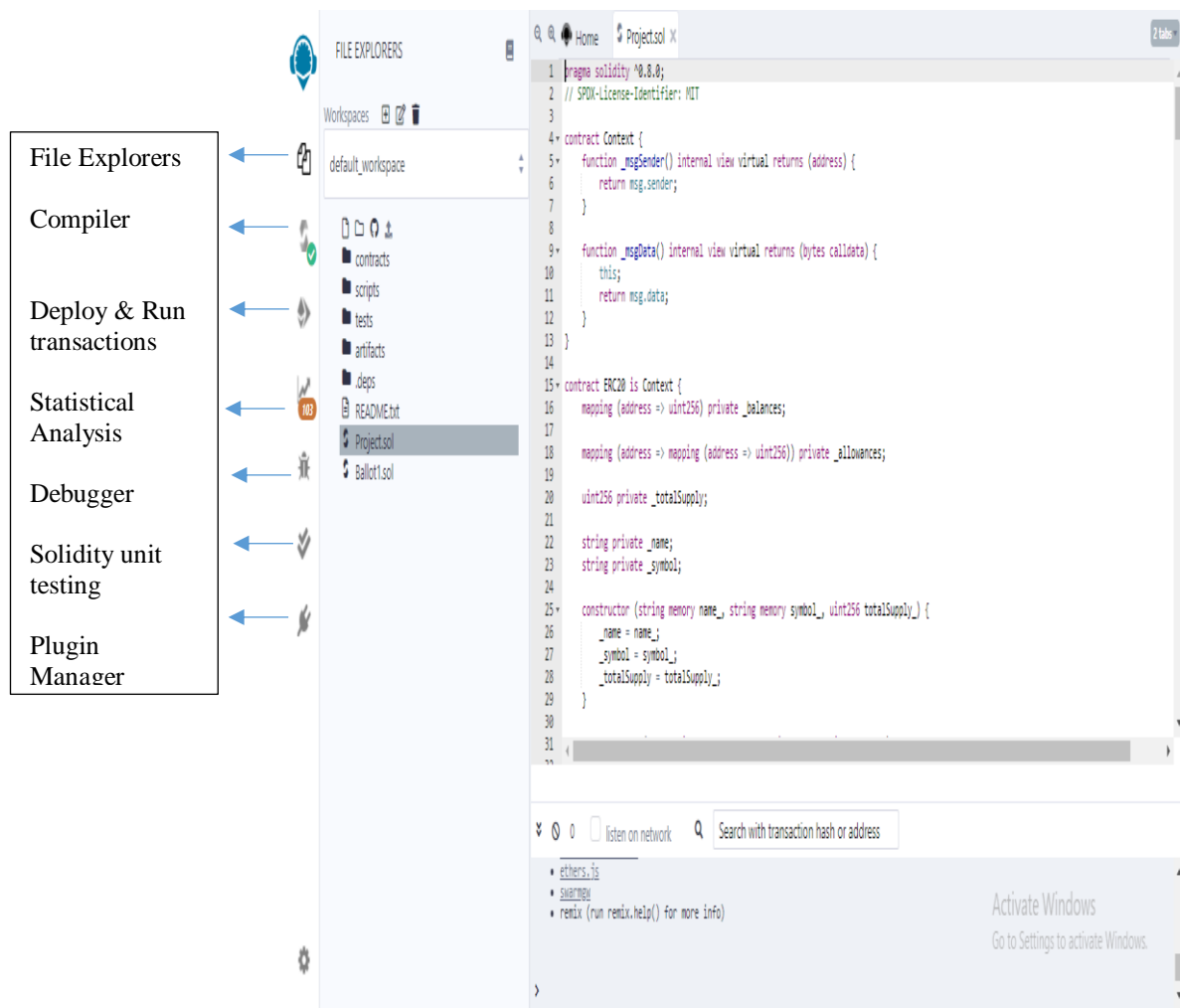


Fig 3.3 Remix IDE Layout

3.2.2 FILE EXPLORER

File Storage: By default, Remix IDE stores files in Workspaces which are folders in your browser's local storage.

Workspace: Workspaces should help clean up & organize your Files Explorer by allowing you to separate your projects. The Files Explorer's Workspaces all have a contracts folder, a scripts folder, a tests folder, and a README.txt.

3.2.3 COMPILER

Compiling is triggered when you click the compile button. If you want the file to be compiled each time the file is saved or when another file is selected - check the auto compile checkbox. It has compiler, Language, EVM version fields along with check boxes for auto

compile, enable optimisation and hide warnings. It also displays the name of the contract currently being compiled.

3.2.4 DEPLOY & RUN TRANSACTION

The Deploy & Run module allows you to send transactions to the current environment. To use this module, you need to have a contract compiled. It has the following fields:

1. **Environment:**

- a. **JavaScript VM:** All the transactions will be executed in a sandbox blockchain in the browser. This means nothing will be persisted when you reload the page. The JsVM is its own blockchain and on each reload it will start a new blockchain, the old one will not be saved.
- b. **Injected Provider:** Remix will connect to an injected web3 provider. Metamask is an example of a provider that inject web3.
- c. **Web3 Provider:** Remix will connect to a remote node. You will need to provide the URL to the selected provider: geth, parity or any Ethereum client.

2. Account: The list of accounts associated with the current environment and their associated balances. On the JsVM, you have a choice of 5 accounts. If using Injected Web3 with Metamask, you need to change the account in Metamask.

3. Gas Limit: This sets the maximum amount of gas that will be allowed for all the transactions created in Remix.

4. Value: This sets the amount of ETH, WEI, GWEI etc that is sent to a contract or a payable function. The payable functions have a red button. The Value field is always reset to 0 after each transaction execution. The Value field is not for gas.

DEPLOY AND RUN TRANSACTIONS

Environment: JavaScript VM

Account: 0xca3...a733c (100 ether)

Gas limit: 3000000

Value: 0 wei

Ballot

Deploy uint8 _numProposals

or

At Address Load contract from Address

Fig 3.4 Deploy & Run Layout

You will see the functions in the contract. The functions buttons can have different colour buttons. Functions that are constant or pure functions in Solidity have a blue buttons. Clicking one of this type does not create a new transaction. So clicking will not cause state changes - it will only return a value stored in the contract - so it won't cost you anything in gas fees. Functions that change the state of the contract AND that do not accept Ether are called non-payable functions and have an orange button. Clicking on them will create a transaction and thus cost gas. Functions that have red buttons are payable functions in Solidity. Clicking one of these will create a new transaction and this transaction can accept a value. The value is put in in the Value field which is under the Gas Limit field.

3.2.5 DEBUGGER

The Debugger shows the contract's state while stepping through a transaction. It can be used on transactions created on Remix or by providing a transaction's address. The latter assumes that you have the contract's source code or that you have input the address of a verified contract. To start a debugging session either: Click the debug button in the Terminal when a successful or failed transaction appears there. The Debugger will be activated and will gain the focus in the Side Panel or activate the Debugger in the Plugin Manager and then click the bug in the icon panel. To start the debugging session, input the address of a deployed transaction - while having the source code in the editor and then click the Start debugging button. The

debugger will highlight the relevant code in the Editor. If you want to go back to editing the code without the Debugger's highlights, then click the Stop Debugging button.

3.2.6 STATISTICAL ANALYSIS

Static code analysis is a process to debug the code by examining it and without actually executing the code. Solidity Static Analysis plugin performs static analysis on Solidity smart contracts once they are compiled. It checks for security vulnerabilities and bad development practices, among other issues. It can be activated from Remix Plugin Manager.

3.2.7 PLUGIN MANAGER

In Remix IDE you only load the functionality you need. Controlling which plugins are active or inactive happens in the Plugin Manager. This plugin architecture has made it possible to integrate tools made by the Remix team with tools made by external teams. This architecture also allows Remix or just parts of Remix to be integrated into other projects.

3.2.8 UNIT TESTING

You can write sufficient unit tests to ensure that your contract works as expected under different scenarios. Although, Remix injects a built-in assert library which can be used for testing. Apart from this, Remix allows usage of some special functions in the test file to make testing more structural. They are as:

- `beforeEach()` - Runs before each test
- `beforeAll()` - Runs before all tests
- `afterEach()` - Runs after each test
- `afterAll()` - Runs after all tests

3.3 METAMASK- INJECTED WEB3 PROVIDER:

Metamask is a software cryptocurrency wallet used to interact with the Ethereum blockchain. It allows users to access their Ethereum wallet through a browser extension or mobile app, which can then be used to interact with decentralized applications. Metamask and other "web3" focused applications aim to decentralize control over personal data and increase user privacy. Metamask does not control any user data, and all data is encrypted on the user's browser and protected by the users Metamask password. It enables

users to execute Ethereum dApps in their internet browser directly without running a full Ethereum node.

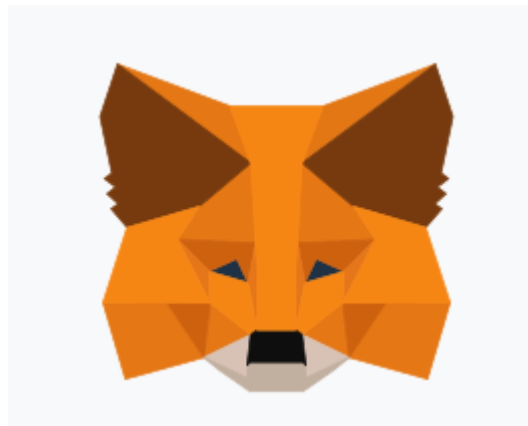


Fig 3.5 Metamask Logo

Developers	-	ConsesSys Software Inc.
Operating Systems	-	iOS & Android
Platform	-	Browser Extension, Mobile App
Type	-	Cryptocurrency Wallet

3.3.1 FEATURES OF METAMASK

Metamask allows users to – 1. Store and manage account keys

2. Broadcast Transactions

3. Send and receive ethereum based cryptocurrencies and tokens

4. Securely connect to decentralized applications through a compatible web browser or the mobile app's built-in browser.

3.4 ERC20 TOKENS:

The popular cryptocurrency and blockchain system Ethereum is based on the use of tokens, which can be bought, sold, or traded. In the Ethereum system, tokens represent a diverse range of digital assets, such as vouchers, IOUs, or even real-world, tangible objects. Essentially, Ethereum tokens are smart contracts that make use of the Ethereum blockchain. One of the most significant tokens is called ERC-20, which has emerged as the technical standard used for all smart contracts on the Ethereum blockchain for token implementation and

provides a list of rules that all Ethereum-based tokens must follow.. Plenty of well-known digital currencies use the ERC-20 standard, including Maker (MKR), Basic Attention Token (BAT), Augur (REP), and OmiseGO (OMG).

ERC-20 is similar, in some respects, to bitcoin, Litecoin, and any other cryptocurrency; ERC-20 tokens are blockchain-based assets that have value and can be sent and received. The primary difference is that instead of running on their own blockchain, ERC-20 tokens are issued on the Ethereum network. ERC-20 is similar, in some respects, to bitcoin, Litecoin, and any other cryptocurrency; ERC-20 tokens are blockchain-based assets that have value and can be sent and received. The primary difference is that instead of running on their own blockchain, ERC-20 tokens are issued on the Ethereum network.

Consequently, this particular token empowers developers of all types to accurately predict how new tokens will function within the larger Ethereum system. This simplifies the task set forth for developers; they can proceed with their work knowing that each and every new project won't need to be redone every time a new token is released, as long as the token follows the rules. This compliance is also necessary; it ensures compatibility between the many different tokens issued on Ethereum.

Fortunately, the vast majority of token developers have fallen in line with ERC-20 rules, meaning that most of the tokens released through Ethereum initial coin offerings are ERC-20 compliant. If you are planning on purchasing any digital currency that's issued as an ERC-20 token, you must also have a wallet that is compatible with these tokens. Luckily, because ERC-20 tokens are so popular, there are many different options for wallets.

ERC-20 defines six different functions for the benefit of other tokens within the Ethereum system. These are generally basic functionality issues, including the method in which tokens are transferred and how users can access data regarding a particular token. Taken together, this set of functions and signals ensures that Ethereum tokens of different types will all uniformly perform in any place within the Ethereum system. As such, nearly all of the digital wallets which support the ether currency also support ERC-20-compliant tokens.

3.4.1 LIST OF ERC STANDARDS

1. ERC20 - Most popular token standards.

- Used in most of the ICO's.
- Fungible token standard.
- Allows the implementation of standard API within a smart contract.
2. ERC165 - A standard for a method, instead of tokens.
- Covers how interfaces are identified.
- States how any contract can publish the interfaces after implementation.
- States how to detect when a contract implements ERC-165.
- Covers the way to detect when a smart contract uses any given interface.
3. ERC721 - A standard for non-fungible tokens.
- Allows the implementation of standard API for NFTs in smart contract.
- Offers functionality to track and transfer NFTs.
4. ERC223 - Prevents accidental burns of tokens, a bug in ERC20.
- Developer can either accept or decline tokens arriving at their smart contract addresses.
- Rejected transactions will fail but won't burn the tokens.
- Not in use, still in EIP phase.
5. ERC621 - An extension of ERC20 standard.
- Uses 2 functions 'increaseSupply' and 'decreaseSupply'.
- Not in use, still in EIP phase.
6. ERC777 - Reduces friction in cryptotransactions.
- Not in use, still in EIP phase.
- Gets rid of the double transaction verification of ERC20.
- Lowers transaction overhead.
- Allows users to reject incoming tokens from a blacklisted address.
7. ERC827 - An extension of ERC20.
- Wallets and exchanges can reuse tokens.
- Token holder can transfer tokens while also allowing 3rd party to spend

it.

Not in use, still in EIP phase.

8. ERC884 - Allows companies to use blockchain to maintain share registries.
- Identity verification and mandatory whitelisting of token holders.
- Only whole value of tokens, no partial value.
- Recording of information regulators mandate.
- Not in use, still in EIP phase.

CHAPTER 4

DESIGN AND IMPLEMENTATION

4.1 ARCHITECTURE OF ATTRIBUTE BASED ACCESS CONTROL

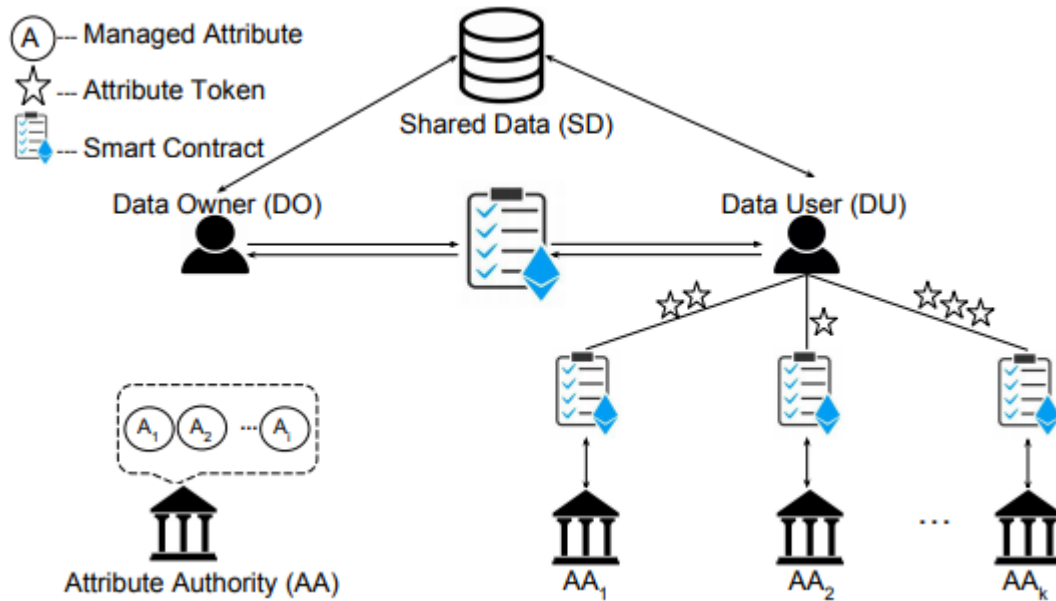


Fig 4.1 Architecture of ABAC

- **Data Owner (DO):** A DO is an entity (e.g., person, organization, or process) who owns the data to be shared. A DO actively specifies access policies for the data it shares.
- **Data User (DU):** A DU is an entity who wants to access data shared by DOs. A DU actively seeks access authorizations from DOs.
- **Shared Data (SD):** An SD is a piece of data owned by a DO, and can be accessed passively by authorized DUs.
- **Attribute Token (AT):** An AT is a credential representing an attribute that a DU possesses.
- **Attribute Authority (AA):** An AA is a pre-verified and authorized node in Ethereum who issues ATs to qualified DUs who possess the corresponding attributes.

In the proposed architecture, after a DU has been validated for possessing a particular set of attributes by an AA, the AA will then issue the corresponding set of ATs to the DU. This validation process is carried out in the context of smart contracts. Consequently, satisfaction of the access policy associated with an SD is now represented by the collection of the corresponding ATs. Once a DU meets the access control policy imposed by the DO of the SD,

the smart contract between the DU and the DO is executed for the DU to receive the AES secret key (which is encrypted with DU's public key) from the DO to access the SD.

4.2 WORKFLOW SCENARIO

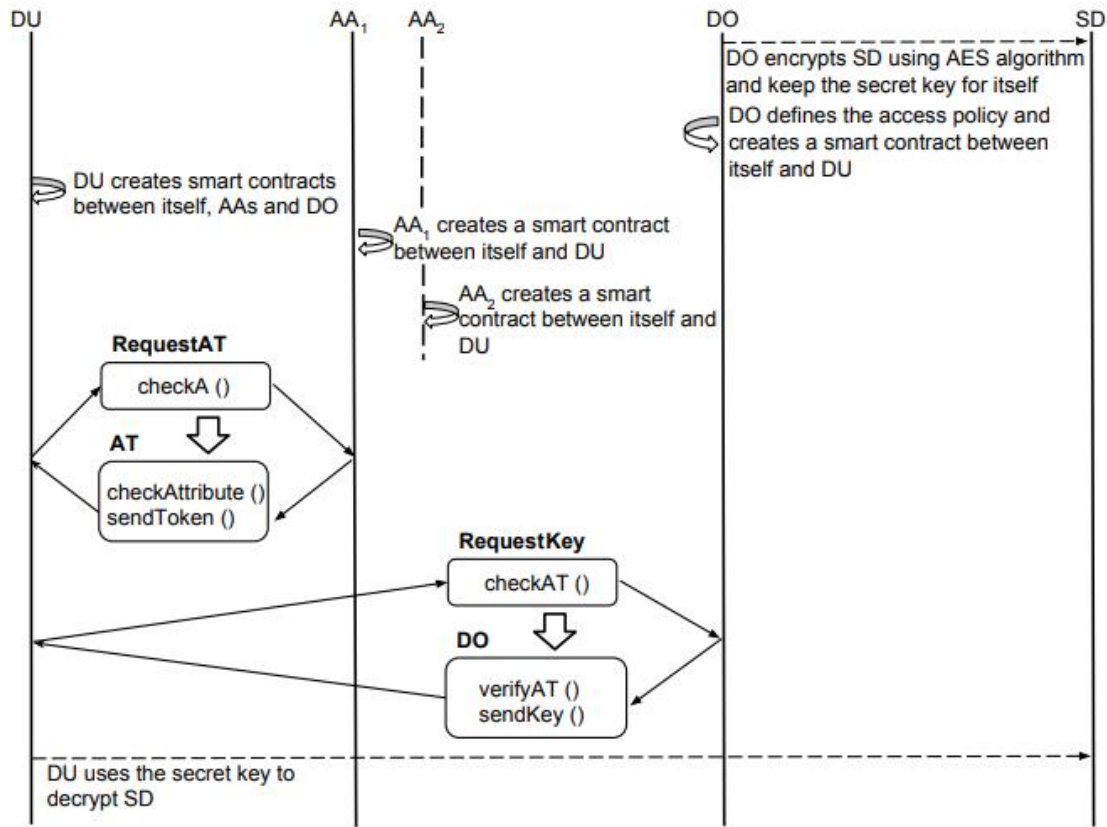


Fig 4.2 Workflow

- AAs, DO, and DU register their respective EOA accounts with Ethereum, so that they may participate in Ethereum blockchain network.
- Using a standard symmetric encryption algorithm, such as AES-256, the DO generates a secret key to encrypt the SD, and uploads the encrypted data to a shared database. In addition, the DO defines the access policy for the SD and creates smart contract DO to be executed between itself and the DU.
- A DU creates smart contract RequestAT with each AA, which contains the function checkA that requests validation of its attributes, and returns the corresponding ATs upon successful validation using the sendToken function. The DU also creates smart contract RequestKey with the DO, which contains function checkAT that the DU holds enough ATs to send the AES secret key.
- AA1 creates a smart contract **AT** between itself and the DU, which contains function checkAttribute for validating the DU's attributes, and the function sendToken for

granting ATs. Similarly, AA2 creates another smart contract AT between itself and the DU.

- Now, the DU wanting to access the SD executes its smart contract RequestAT to request validation of its attributes by invoking function checkA. In turns, checkA triggers the execution of the smart contract AT by invoking the checkAttribute function of AT. Upon successful validation, the sendToken function is invoked to return the associated ATs to be saved in the “balance” of DU’s EOA account.
- After accumulating enough “balance,” the DU will executes smart contract RequestKey to invoke its function checkAT for validating its ATs so as to obtain the AES secret key (encrypted with DU’s public key) by triggering smart contract DO and invoking its sendKey function.
- At the end, the DU decrypts the AES secret key (which was encrypted with its public key) with his private key, and uses the AES secret key to access the SD.

4.3 E-VOTING DESIGN:

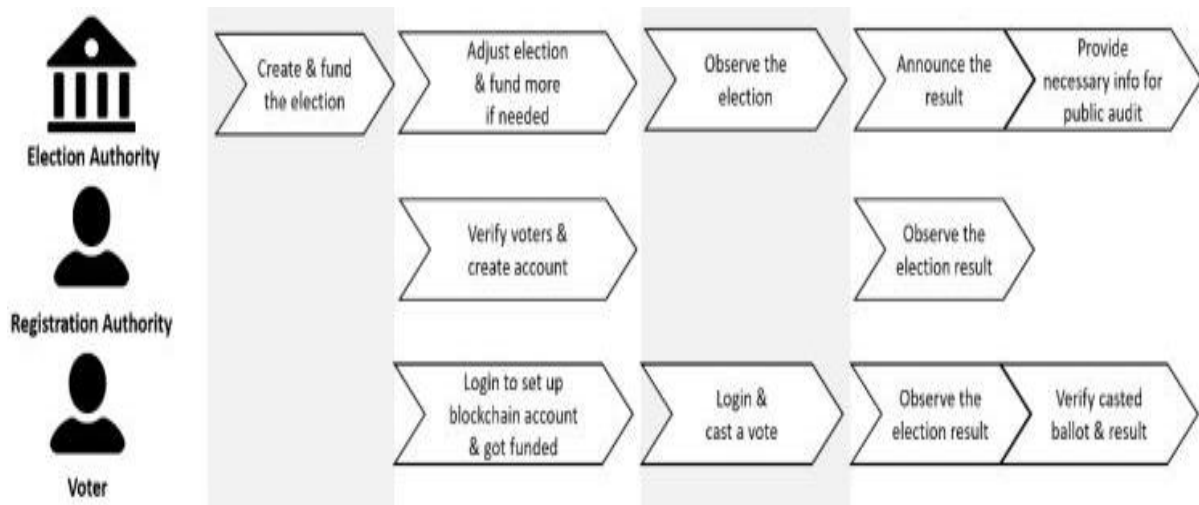


Fig 4.3 Voting System Design

Here, the voting system is divide based on people as:

Election Authority – Those that act as the body conducting elections and funding the elections.

Registration Authority - The person who is responsible for registering the voters. This Person Is referred to as chairperson in the implementation and every ballot

has a chairperson.

Voters - The people who participate in the election and cast their vote.

The election authority initiates and funds the elections and the registration authority verifies the voters and create blockchain accounts to allow voters cast their vote. The voters login into their respective accounts at the time of election and get funded by the authority. The voters then cast their votes from the respective accounts. After the process is complete the Election authority announces the result instantaneously. They also provide information for public audit. Now, Users/voters can use their private key to access the data and verify if their vote is properly casted to the respective candidate.

CHAPTER 5
CODE AND EXPLANATION

5.1 SOLIDITY CODE

```
pragma solidity ^0.8.0;
// SPDX-License-Identifier: MIT

contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes calldata) {
        this;
        return msg.data;
    }
}

contract ERC20 is Context {
    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowances;

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    constructor (string memory name_, string memory symbol_, uint256 totalSupply_) {
        _name = name_;
        _symbol = symbol_;
        _totalSupply = totalSupply_;
    }

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
```

```
function name() public view virtual returns (string memory) {  
    return _name;  
}
```

```
function symbol() public view virtual returns (string memory) {  
    return _symbol;  
}
```

```
function decimals() public view virtual returns (uint8) {  
    return 18;  
}
```

```
function totalSupply() public view virtual returns (uint256) {  
    return _totalSupply;  
}
```

```
function balanceOf(address account) public view virtual returns (uint256) {  
    return _balances[account];  
}
```

```
function transfer(address recipient, uint256 amount) public virtual returns (bool) {  
    _transfer(_msgSender(), recipient, amount);  
    return true;  
}
```

```
function allowance(address owner, address spender) public view virtual returns (uint256) {  
    return _allowances[owner][spender];  
}
```

```
function approve(address spender, uint256 amount) public virtual returns (bool) {  
    _approve(_msgSender(), spender, amount);  
}
```



```
    return true;
}
```

```
function transferFrom(address sender, address recipient, uint256 amount) public virtual
returns (bool) {
    _transfer(sender, recipient, amount);

    uint256 currentAllowance = _allowances[sender][_msgSender()];
    require(currentAllowance >= amount, "ERC20: transfer amount exceeds allowance");
    _approve(sender, _msgSender(), currentAllowance - amount);

    return true;
}
```

```
function increaseAllowance(address spender, uint256 addedValue) public virtual returns
(bool) {
    _approve(_msgSender(), spender, _allowances[_msgSender()][spender] + addedValue);
    return true;
}
```

```
function decreaseAllowance(address spender, uint256 subtractedValue) public virtual
returns (bool) {
    uint256 currentAllowance = _allowances[_msgSender()][spender];
    require(currentAllowance >= subtractedValue, "ERC20: decreased allowance below
zero");
    _approve(_msgSender(), spender, currentAllowance - subtractedValue);

    return true;
}
```

```
function _transfer(address sender, address recipient, uint256 amount) internal virtual {
    require(sender != address(0), "ERC20: transfer from the zero address");
    require(recipient != address(0), "ERC20: transfer to the zero address");
```

```

    _beforeTokenTransfer(sender, recipient, amount);

    uint256 senderBalance = _balances[sender];
    require(senderBalance >= amount, "ERC20: transfer amount exceeds balance");
    _balances[sender] = senderBalance - amount;
    _balances[recipient] += amount;

    emit Transfer(sender, recipient, amount);
}

function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

    _beforeTokenTransfer(address(0), account, amount);

    _totalSupply += amount;
    _balances[account] += amount;
    emit Transfer(address(0), account, amount);
}

function _burn(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: burn from the zero address");

    _beforeTokenTransfer(account, address(0), amount);

    uint256 accountBalance = _balances[account];
    require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
    _balances[account] = accountBalance - amount;
    _totalSupply -= amount;

    emit Transfer(account, address(0), amount);
}

```

```

function _approve(address owner, address spender, uint256 amount) internal virtual {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
}

function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual
{ }
}

contract Owned {
    address public owner;
    constructor() { owner = msg.sender; }
    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }
}

contract Ballot{
    struct Voter {
        uint weight;
        bool voted;
        uint vote;
    }
    struct Proposal{
        uint votecount;
    }

    Proposal[] proposals;

    address chairperson;

```

```
address ReqAtAddress;  
address ReqKeyAddress;  
address addressofAT;  
address addressofDO;  
address addressofAA;
```

```
mapping(address=>Voter)voters;  
uint len;
```

```
constructor(uint _numProposals) {  
    for(uint i=0;i<_numProposals;i++)  
    {  
        proposals.push(Proposal({ votecount: 0}));  
    }  
    len= _numProposals;  
    chairperson= msg.sender;  
    voters[chairperson].weight =1;  
  
}
```

```
function register(address toVoter) public {  
    if(msg.sender != chairperson || voters[toVoter].voted)return;  
    require(voters[toVoter].weight==0);  
    voters[toVoter].weight=1;  
    voters[toVoter].voted=false;  
}
```

```
function setAddress1(address _ReqAtAddress, address _ReqKeyAddress, address  
_addressofAt, address _addressofDO, address _addressofAA) external {  
    ReqAtAddress= _ReqAtAddress;  
    ReqKeyAddress = _ReqKeyAddress;  
    addressofAT = _addressofAt;  
    addressofDO = _addressofDO;  
    addressofAA = _addressofAA;
```

```

}

function callReqAt() public returns(bool){
    RequestAT A = RequestAT(ReqAtAddress);
    return A.checkA(addressofAA,addressofAT);
}

function callReqKey() public returns(bool){
    RequestKey R = RequestKey(ReqKeyAddress);
    return R.checkAt(addressofDO,addressofAA);
}

function Vote(uint toProposal) public {

    Voter storage sender = voters[msg.sender];
    require(!sender.voted || toProposal>=len,"CAN'T VOTE");
    require(sender.weight != 0);
    sender.voted = true;
    sender.vote = toProposal;
    proposals[toProposal].votecount += sender.weight;
}

function winningProposal() public view returns(uint8 _winningProposal, uint256
winningVotecount){
    for(uint8 prop=0;prop<len;prop++){
        if(proposals[prop].votecount>winningVotecount){
            winningVotecount = proposals[prop].votecount;
            _winningProposal=prop;
        }
    }

    for(uint8 i=0;i<len;i++){
        if(i!=_winningProposal)require(proposals[i].votecount !=
proposals[_winningProposal].votecount,"TIE");
    }
}

```

```

    }
    assert(winningVotecount>0);
}
}

```

```

contract RequestAT {
    function checkA(address addressofAA, address addressofAT) external returns (bool) {
        AT my_at= AT(addressofAT);
        if(my_at.checkAttribute(1,"Registered","Permit")== true){
            return my_at.sendToken(addressofAA,msg.sender, 2);
        }
        return false;
    }
}

```

```

contract RequestKey {
    function checkAt(address addressofDO, address addressofAA) external returns (bool){
        DO my_ap = DO(addressofDO);
        if(my_ap.verifyAT(addressofAA,msg.sender,2,"Permit")==true){
            return
my_ap.sendKey(msg.sender,"4008b1066bd42a3625dbd4fc4e95275533e49b51590d9856c969
96e6063357ae");
        }
        return false;
    }
}

```

```

contract AT is ERC20("UD Token","UD",100), Owned{

    string Symbol;
    string Name;
    uint8 Decimals;
    uint256 Totalsupply;

```

```

mapping(address => uint) balances;
mapping(uint256 => Data) CheckAttribute;
mapping (address => bool) frozenAccount;

event Sendtoken(address from, address to, uint tokens);
event FrozenFunds(address target, bool frozen);

```

```

struct Data{
    uint256 AttributeID;
    string attribute;
    string approve;
}

```

```

function AttributeToken(address addressofAA) public{
    Symbol = symbol();
    Name = name();
    Decimals = decimals();
    Totalsupply = totalSupply();
    balances[addressofAA] = Totalsupply;
}

```

```

function checkAttribute(uint256 AttributeID, string memory attribute, string memory
approve) public returns (bool success){
    CheckAttribute[AttributeID] = Data(AttributeID, attribute, approve);
    return true;
}

```

```

function sendToken(address from, address to, uint tokens) public returns (bool success){
    require(!frozenAccount[to]);
    emit Sendtoken(from, to, tokens);
    return true;
}

```

```

function freezeAccount(address target, bool freeze) onlyOwner public {

```

```

        frozenAccount[target] = freeze;
        emit FrozenFunds(target, freeze);
    }
}

contract DO is ERC20("UD Token","UD",100){
    uint256 allowed;
    event Sendkey(address from, address to, bytes encryptedKey);
    event VerifyAT(address to, uint tokens, bytes approve);

    struct AESData{
        bytes encryptedKey;
    }

    function sendKey(address to, bytes memory encryptedKey) public returns (bool success){
        emit Sendkey (msg.sender, to, encryptedKey);
        return true;
    }

    function verifyAT(address from, address to, uint tokens, bytes memory approve) public
returns (bool success){
        allowed= allowance(to,from);
        allowed = tokens;
        require(balanceOf(to) >= balanceOf(from));
2    emit VerifyAT(to, tokens, approve);
        return true;
    }
}

```

5.2 CODE EXPLANATION:

- Initially the compiler version is specified using the “pragma solidity” statement. ‘^0.8.0’ specifies that the code will not compile for earlier versions than 0.8.0 and also on versions starting from 0.9.0.
- The SPDX license is provided. Software Package Data Exchange (SPDX) is a file format used to document information on the software licenses under which a given piece of computer software is distributed.

5.2.1 CONTEXT CONTRACT

- It provides information about the current execution context, including the sender of the transaction and its data. While these are generally available via `msg.sender` and `msg.data`, they should not be accessed in such a direct manner, since when dealing with meta-transactions the account sending and paying for execution may not be the actual sender.

5.2.2 ERC20 CONTRACT

- This used to implement all the standard functions. The constructor sets the values for name, symbol and totalsupply. All these values are immutable, they can only be set once during construction. The default value for decimals is 18. To select a different value overload it and modify.
- The `name()` and `symbol()` functions return the name of the token and symbol of the token, which is usually a shorter version of the name. `decimals()` function returns the no. of decimals used to get its user representation. Tokens usually opt for a value of 18, imitating the relationship between Ether and Wei.
- Function `totalSupply()` returns the amount of tokens in existence. The function `balanceOf(address account)` returns the amount of tokens owned by account.
- The function `transfer(address recipient, uint256 amount)` moves `amount` tokens from the caller's account to `recipient`. It also returns a boolean value indicating whether the operation succeeded. It emits a "Transfer" event.
- The function `allowance(address owner, address spender)` returns the remaining number of tokens that 'spender' will be allowed to spend on behalf of 'owner' through "transferFrom". This is zero by default. This value changes when "approve" or "transferFrom" are called.
- The function `approve(address spender, uint256 amount)` sets 'amount' as the allowance of 'spender' over the caller's tokens. It returns a boolean value indicating whether the operation succeeded. Changing an allowance with this method brings the risk that someone may use both the old and the new allowance by unfortunate transaction ordering. One possible solution to mitigate this race condition is to first reduce the spender's allowance to 0 and set the desired value afterwards. It emits an Approval event.
- The function `transferFrom(address sender, address recipient, uint256 amount)` moves 'amount' tokens from 'sender' to 'recipient' using the allowance mechanism, 'amount' is then deducted from the caller's allowance. It returns a boolean value indicating whether the operation succeeded. Emits a "Transfer" event.

- The event *Transfer(address indexed from, address indexed to, uint256 value)* is emitted when ‘value’ tokens are moved from one account ‘from’ to another ‘to’. The ‘value’ may be zero.
- The event *Approval(address indexed owner, address indexed spender, uint256 value)* is emitted when the allowance of a ‘spender’ for an ‘owner’ is set by a call to “approve”. ‘value’ is the new allowance.
- An “Approval” event is emitted on calls to “transferFrom”. This allows applications to reconstruct the allowance for all accounts just by listening to said events. Other implementations of the EIP may not emit these events, as it isn’t required by the specification.
- Finally, the non-standard *decreaseAllowance(address spender, uint256 subtractedValue)* and *increaseAllowance(address spender, uint256 addedValue)* functions have been added to mitigate the well-known issues around setting allowances.
- *increaseAllowance(address spender, uint256 addedValue)* function automatically increases the allowance granted to ‘spender’ by the caller. This is an alternative to “approve” that can be used as a mitigation for problems. It emits an “Approval” event indicating the updated allowance. The requirement is that ‘spender’ cannot be the zero address.
- *decreaseAllowance(address spender, uint256 subtractedValue)* function automatically decreases the allowance granted to ‘spender’ by the caller. This is an alternative to “approve” that can be used as a mitigation for problems. It emits an “Approval” event indicating the updated allowance. The requirements are that ‘spender’ cannot be the zero address and spender must have allowance for the caller of at least ‘subtractedValue’.
- The function *_transfer(address sender, address recipient, uint256 amount)* moves tokens ‘amount’ from ‘sender’ to ‘recipient’. This is internal function is equivalent to transfer, and can be used to implement automatic token fees, slashing mechanisms, etc. It emits a “Transfer” event. The requirements are ‘sender’ cannot be the zero address, ‘recipient’ cannot be the zero address and ‘sender’ must have a balance of at least ‘amount’.
- The function *_mint(address account, uint256 amount)* creates ‘amount’ tokens and assigns them to ‘account’, increasing the total supply. It emits a “Transfer” event with ‘from’ set to the zero address. The requirement is that ‘account’ cannot be the zero address.
- The function *_burn(address account, uint256 amount)* destroys ‘amount’ tokens from ‘account’, reducing the total supply. It emits a “Transfer” event with ‘to’ set to the zero address. The requirements are the ‘account’ cannot be the zero address and ‘account; must have at least ‘amount’ tokens.

- The function *_approve(address owner, address spender, uint256 amount)* sets ‘amount’ as the allowance of ‘spender’ over the ‘owners’ token. This internal function is equivalent to ‘approve’ and can be used to set automatic allowances for certain subsystems. It emits an “Approval” event. The requirements are ‘owner’ and ‘spender’ cannot be the zero address.
- The function *_beforeTokenTransfer(address from, address to, uint256 amount)* is called before any transfer of tokens. This includes minting and burning. The conditions are when ‘from’ and ‘to’ are both non-zero, ‘amount’ of ‘from’ tokens will be transferred to ‘to’. Also, when ‘from’ is zero, ‘amount’ tokens will be minted for ‘to’ and when ‘to’ is zero, ‘amount’ of ‘from’ tokens will be burned and ‘from’ and ‘to’ are never both zero.

5.2.3 OWNED CONTRACT

- This contract defines a modifier which is used by the derived contracts. “_” this symbol is called the place holder which is where the implementation goes. If the owner calls this function it is executed and otherwise, an exception is thrown.

5.2.4 BALLOT, REQUESTAT, REQUESTKEY, AT AND DO CONTRACTS

- This contract implements voting process. The “Voter” structure represents each voter taking part in the election process. The structure “Proposal” represents the candidates standing to compete for the election. Chairperson is the registration authority who verifies the voters. The constructor takes input the no. of persons standing for the election and assign their individual vote counts to zero.
- The *register()* function is used to register the voters for the election process. It also assigns the weight of each vote to one for all the voters and voted flag to false or 0. The registration is done only if the voted has not already voted.
- The function *setAddressI()* is used to read the addresses of other contracts and uses them later to call other functions or in passing the function parameters.
- The function *callReqAt()* is used to call the *RequestAt* contract. The function *checkA()* is called which inturn calls the contract *AT* and one of its functions *checkAttribute()*. This function checks for the attributes of the voter and returns a boolean value. If the value is true i.e. the voter has necessary attributes, the attribute authority sends tokens to the user. If not boolean false is returned.
- Now, the function *callReqKey()* makes a call to the *RequestKey* contract and its function *checkAt()*. Here, the control goes to contract *DO* and *verifyAT()* function is invoked, which is

used to verify if the voter has enough tokens. If the return value is true, then the encrypted key is send to the user/voter by invoking the *sendKey()* function.

CHAPTER 6

RESULTS AND ANALYSIS

6.1 RESULTS:

We were able to show our project Blockchain based E-Voting System is in complete working condition as stated. Registered voters were able to cast their vote and un-registered voters were barred by the smart contract. Also the voters were allowed to vote only once. Also attribute checking is performed once the voter is registered in order to improve secured voting and reduce any kind of malpractices. Once the attributes are verified the tokens are sent to the voters from the attribute authority.

Voters can be divided into three sets. One set of voters are those who belongs to that voting center and are voting for the first time, known to be valid voters. Second are those who doesn't belong to that particular voting center, known to be invalid voters. Third are those who are voting for the second (or more) time(s).

Step -1: Initially to start the voting process all the smart contracts are deployed on remix as shown in the following figures. Here the chairperson is used to deploy all the contracts. The account is created on Metamask environment and connected to the smart contract.








Fig 6.1 Creation of contract AT

creation of Ballot pending...

<https://ropsten.etherscan.io/tx/0x4a5105e8c858907833f0922beeb34ec873d10be94336cdb5ed8e39cd345909a7>

 [block:10294294 txIndex:8] from: 0x506...e7f59 to: Ballot.(constructor) value: 0 wei data: 0x608...00003 logs: 0 hash: 0x4a5...909a7


status	true Transaction mined and execution succeed
transaction hash	0x4a5105e8c858907833f0922beeb34ec873d10be94336cdb5ed8e39cd345909a7 
from	0x506166938fc2c2ba205848ac01125979b932e7f59 
to	Ballot.(constructor) 
gas	820873 gas 
transaction cost	820873 gas 
hash	0x4a5105e8c858907833f0922beeb34ec873d10be94336cdb5ed8e39cd345909a7 
input	0x608...00003 
decoded input	{ "uint256 _numProposals": "3" } 
decoded output	- 
logs	[]  
value	0 wei 













Activate Windows
Go to Settings to activate Windows.

Fig 6.2 Creation of Ballot contract

creation of Context pending...

<https://ropsten.etherscan.io/tx/0xb08e51c4802b8812d889b211c3b527293a1859a05dd1cb960fb01f773c68567a>

 [block:10294296 txIndex:4] from: 0x506...e7f59 to: Context.(constructor) value: 0 wei data: 0x608...10033 logs: 0 hash: 0xb08...8567a

status	true Transaction mined and execution succeed
transaction hash	0xb08e51c4802b8812d889b211c3b527293a1859a05dd1cb960fb01f773c68567a 
from	0x506166938fc2c2ba205848ac01125979b932e7f59 
to	Context.(constructor) 
gas	67066 gas 
transaction cost	67066 gas 
hash	0xb08e51c4802b8812d889b211c3b527293a1859a05dd1cb960fb01f773c68567a 
input	0x608...10033 
decoded input	{ } 
decoded output	- 
logs	[]  
value	0 wei 

Activate Windows
Go to Settings to activate Windows.

Fig 6.3 Creation of Context contract

creation of DO pending...

<https://ropsten.etherscan.io/tx/0x6878b265243fb44620398d2fd0913d7e492dd4f0ab13493e56d4c10ee9aef6ac>

✓ [block:10294300 txIndex:0] from: 0x506...e7F59 to: DO.(constructor) value: 0 wei data: 0x608...10033 logs: 0 hash: 0x687...ef6ac	
status	true Transaction mined and execution succeed
transaction hash	0x6878b265243fb44620398d2fd0913d7e492dd4f0ab13493e56d4c10ee9aef6ac
from	0x506166938fc2c8a205848ac01125979b932e7F59
to	DO.(constructor)
gas	1446028 gas
transaction cost	1446028 gas
hash	0x6878b265243fb44620398d2fd0913d7e492dd4f0ab13493e56d4c10ee9aef6ac
input	0x608...10033
decoded input	{}
decoded output	-
logs	[]
value	0 wei

Activate Win
Go to Settings

Fig 6.4 Creation of DO contract

creation of ERC20 pending...

<https://ropsten.etherscan.io/tx/0xca35f109045e21cd3556cf9b263d0213874f2955edc06948b2a9cb28ce021792>

✓ [block:10294301 txIndex:44] from: 0x506...e7F59 to: ERC20.(constructor) value: 0 wei data: 0x608...00000 logs: 0 hash: 0xca3...21792	
status	true Transaction mined and execution succeed
transaction hash	0xca35f109045e21cd3556cf9b263d0213874f2955edc06948b2a9cb28ce021792
from	0x506166938fc2c8a205848ac01125979b932e7F59
to	ERC20.(constructor)
gas	1227458 gas
transaction cost	1227458 gas
hash	0xca35f109045e21cd3556cf9b263d0213874f2955edc06948b2a9cb28ce021792
input	0x608...00000
decoded input	{ "string name_": "UD Token", "string symbol_": "UD", "uint256 totalSupply_": "100" }
decoded output	-
logs	[]
value	0 wei

Activate Win
Go to Settings to

Fig 6.5 Creation of ERC20 contract

creation of Owned pending...

<https://ropsten.etherscan.io/tx/0xf3da1b373fc3dc25b04b21a710cba49f0cb4dbc2da14fde6ef79326680a4b7a5>

✓ [block:10294302 txIndex:15] from: 0x506...e7f59 to: Owned.(constructor) value: 0 wei data: 0x608...10033 logs: 0 hash: 0xf3d...4b7a5

status	true Transaction mined and execution succeed
transaction hash	0xf3da1b373fc3dc25b04b21a710cba49f0cb4dbc2da14fde6ef79326680a4b7a5
from	0x506160930fc2cba205048ac01125979b932e7f59
to	Owned.(constructor)
gas	129921 gas
transaction cost	129921 gas
hash	0xf3da1b373fc3dc25b04b21a710cba49f0cb4dbc2da14fde6ef79326680a4b7a5
input	0x608...10033
decoded input	{}
decoded output	-
logs	[]
value	0 wei

Activate Windows
Go to Settings to activate Windows.

Fig 6.6 Creation of Owned contract

creation of RequestAT pending...

<https://ropsten.etherscan.io/tx/0x48f4f96347cb493a3f3a7db0683a1d6c720b8ea200f6190ff35e5ad5dbbd1837>

✓ [block:10294307 txIndex:5] from: 0x506...e7f59 to: RequestAT.(constructor) value: 0 wei data: 0x608...10033 logs: 0 hash: 0x48f...d1837

status	true Transaction mined and execution succeed
transaction hash	0x48f4f96347cb493a3f3a7db0683a1d6c720b8ea200f6190ff35e5ad5dbbd1837
from	0x506160930fc2cba205048ac01125979b932e7f59
to	RequestAT.(constructor)
gas	299829 gas
transaction cost	299829 gas
hash	0x48f4f96347cb493a3f3a7db0683a1d6c720b8ea200f6190ff35e5ad5dbbd1837
input	0x608...10033
decoded input	{}
decoded output	-
logs	[]
value	0 wei

Activate Windows
Go to Settings to activate Windows.

Fig 6.7 Creation of RequestAt contract



Fig 6.8 Creation of RequestKey contract

While deploying the contracts, if any of them have constructors the necessary parameters must be passed in manually which is shown in the following figures.

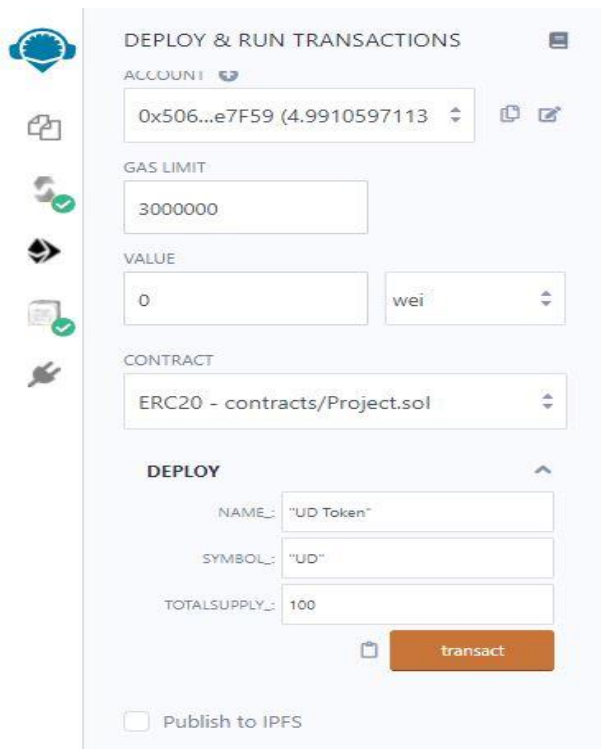


Fig 6.9 Constructor parameters for ERC20

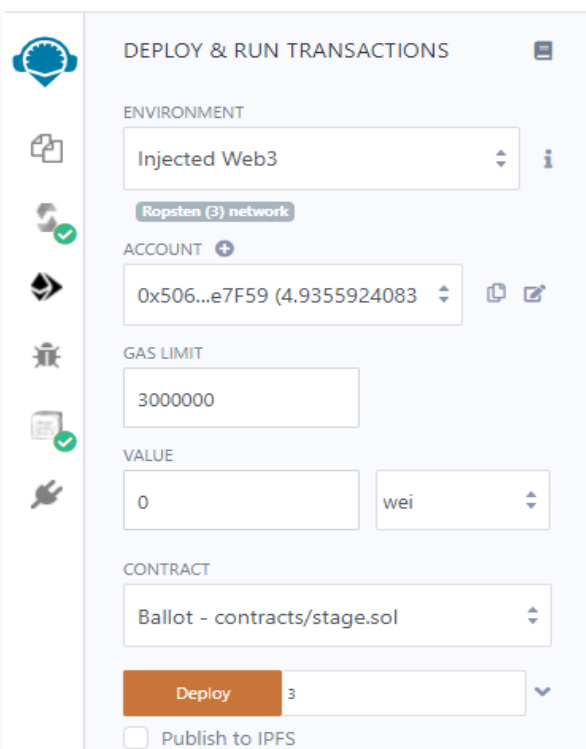


Fig 6.10 Constructor parameters for Ballot

The setAddress function is used to pass the address of all the deployed contracts into ballot for invoking further. The Transaction is as follows:

transact to Ballot.setAddress1 pending ...

<https://ropsten.etherscan.io/tx/0x066e25f9e94c0c70958917347cfc213e5cea4491ac3bb7528bcac607baa9770>

✓ [block:10294318 txIndex:13] from: 0x506...e7f59 to: Ballot.setAddress1(address,address,address,address,address) 0x985...B4ff4 value: 0 wei data: 0x8dc...86a93 logs: 0 hash: 0x066...a9770

status	true Transaction mined and execution succeed
transaction hash	0x066e25f9e94c0c70958917347cfc213e5cea4491ac3bb7528bcac607baa9770
from	0x506166930fc2c2ba205840ac01125979b932e7f59
to	Ballot.setAddress1(address,address,address,address,address) 0x985Ce071E69b7d40fcAC1e411aCc7049A4EB4ff4
gas	135126 gas
transaction cost	135126 gas
hash	0x066e25f9e94c0c70958917347cfc213e5cea4491ac3bb7528bcac607baa9770
input	0x8dc...86a93
decoded input	{ "address_ReqAtAddress": "0x93E71F95B11728C6ED59C78046b104266f796a31", "address_ReqKeyAddress": "0xc1280aC080a78e95d50386f768837A09F09e683", "address_addressofAT": "0x289cd0FcD559ecC99cA1E87f36fa035A57803E6", "address_addressofDO": "0x4D04dD9F2309C30625fb4058C7d49af2c524F057", "address_addressofAA": "0x17140aCe34f7144c39478A4DA42c830A8E8a86A93" }
decoded output	-
logs	[]
value	0 wei

Fig 6.11 Invoking setAddress function

Step-2: After deploying all the contracts, now the voters must be registered. The 1st person deploying the contracts is considered as chairperson. Here we have registered 3 people which is shown in the below figure.

transact to Ballot.register pending ...

<https://ropsten.etherscan.io/tx/0xd41a80bf9e589fb4dcd423fca2b60206b2cb8e116b0ee14f9637daf756714ebd>

✓ [block:10294325 txIndex:8] from: 0x506...e7f59 to: Ballot.register(address) 0x985...B4ff4 value: 0 wei data: 0x442...255e6 logs: 0 hash: 0xd41...14ebd

status	true Transaction mined and execution succeed
transaction hash	0xd41a80bf9e589fb4dcd423fca2b60206b2cb8e116b0ee14f9637daf756714ebd
from	0x506166930fc2c2ba205840ac01125979b932e7f59
to	Ballot.register(address) 0x985Ce071E69b7d40fcAC1e411aCc7049A4EB4ff4
gas	51175 gas
transaction cost	48988 gas
hash	0xd41a80bf9e589fb4dcd423fca2b60206b2cb8e116b0ee14f9637daf756714ebd
input	0x442...255e6
decoded input	{ "address_toVoter": "0xc043B45C81752b50ecE848E4913219f585a255e6" }
decoded output	-
logs	[]
value	0 wei

Fig 6.12 Registering voter1

transact to Ballot.register pending ...

<https://ropsten.etherscan.io/tx/0xf192a0878e6636dfbed4ecd725631a30b7fb16bbacd33acd4aab51f67208156d>

✓ [block:10294331 txIndex:9] from: 0x506...e7f59 to: Ballot.register(address) 0x985...84ff4 value: 0 wei data: 0x442...f2c1b logs: 0 hash: 0xf19...8156d

status	true Transaction mined and execution succeed
transaction hash	0xf192a0878e6636dfbed4ecd725631a30b7fb16bbacd33acd4aab51f67208156d
from	0x506166930fc2c2ba205848ac01125979b932e7f59
to	Ballot.register(address) 0x985Ce071E69b7d40fcAC1e411aCc7049A4EB4ff4
gas	51175 gas
transaction cost	48988 gas
hash	0xf192a0878e6636dfbed4ecd725631a30b7fb16bbacd33acd4aab51f67208156d
input	0x442...f2c1b
decoded input	{ "address toVoter": "0xd87066A44c6a8036eff3a2c60e7e6D82d75f2C10" }
decoded output	-
logs	[]
value	0 wei

Activate Windows
Go to Settings

Fig 6.13 Registering voter2

transact to Ballot.register pending ...

<https://ropsten.etherscan.io/tx/0x58d8abe55fc846d3fe8ee0fc94ebabc9c7266c900d36afb1eb8ce98d33503366>

✓ [block:10294334 txIndex:20] from: 0x506...e7f59 to: Ballot.register(address) 0x985...84ff4 value: 0 wei data: 0x442...7d6f0 logs: 0 hash: 0x58d...03366

status	true Transaction mined and execution succeed
transaction hash	0x58d8abe55fc846d3fe8ee0fc94ebabc9c7266c900d36afb1eb8ce98d33503366
from	0x506166930fc2c2ba205848ac01125979b932e7f59
to	Ballot.register(address) 0x985Ce071E69b7d40fcAC1e411aCc7049A4EB4ff4
gas	51175 gas
transaction cost	48988 gas
hash	0x58d8abe55fc846d3fe8ee0fc94ebabc9c7266c900d36afb1eb8ce98d33503366
input	0x442...7d6f0
decoded input	{ "address toVoter": "0x6c61Ff8B8c8988c7C2464563fA9DC13E3727d6F0" }
decoded output	-
logs	[]
value	0 wei

Activate Windows
Go to Settings

Fig 6.14 Registering voter3

Step-3: Once the voters are registered they are checked for their attributes by calling the necessary function(`callReqAt`) which is shown in the following figures. While checking for the attributes the `msg.sender` should be the voter.

transact to `Ballot.callReqAt` pending ...

<https://ropsten.etherscan.io/tx/0xcd67827d6ceef902eb62f0db0023b243fd1d16160d49a68021641fe2fcd0583f>

✓ [block:10294335 txIndex:29] from: 0xcB4...255e6 to: `Ballot.callReqAt()` 0x985...B4ff4 value: 0 wei data: 0x496...7bb1c logs: 1 hash: 0xcd6...0583f

status	true Transaction mined and execution succeed
transaction hash	0xcd67827d6ceef902eb62f0db0023b243fd1d16160d49a68021641fe2fcd0583f
from	0xcB43045C81752b5DccE848E4913219f585a255e6
to	<code>Ballot.callReqAt()</code> 0x985Ce071E69b7d40fcAC1e411aCc7049A4EB4ff4
gas	112630 gas
transaction cost	111879 gas
hash	0xcd67827d6ceef902eb62f0db0023b243fd1d16160d49a68021641fe2fcd0583f
input	0x496...7bb1c
decoded input	{}
decoded output	-
logs	[{ "from": "0x289cd0FEcD559eCc99cA1E87f36fa035A57803E6", "topic": "0xfe5123cf6f119ad6ed886775d8eb77d77e915c52e9a0427669e5a109eb089395", "event": "Sendtoken", "args": { "0": "0x17140aCe34f714Ac39478A4DA42c830AE8a86A93", "1": "0x985Ce071E69b7d40fcAC1e411aCc7049A4EB4ff4", "2": "2", "from": "0x17140aCe34f714Ac39478A4DA42c830AE8a86A93", "to": "0x985Ce071E69b7d40fcAC1e411aCc7049A4EB4ff4", "tokens": "2" } }]
value	0 wei

Fig 6.15 Checking attributes of voter1

transact to `Ballot.callReqAt` pending ...

<https://ropsten.etherscan.io/tx/0xe0f021568e58b5aa1a75d3080853b2d05afd8e9100dec5d7d696792b1e8935f0>

✓ [block:10294342 txIndex:2] from: 0xd87...f2C1B to: `Ballot.callReqAt()` 0x985...B4ff4 value: 0 wei data: 0x496...7bb1c logs: 1 hash: 0xe0f...935f0

status	true Transaction mined and execution succeed
transaction hash	0xe0f021568e58b5aa1a75d3080853b2d05afd8e9100dec5d7d696792b1e8935f0
from	0xd87066A44c6a8036eff3a2c60e7e6D8d75f2C1B
to	<code>Ballot.callReqAt()</code> 0x985Ce071E69b7d40fcAC1e411aCc7049A4EB4ff4
gas	56693 gas
transaction cost	56693 gas
hash	0xe0f021568e58b5aa1a75d3080853b2d05afd8e9100dec5d7d696792b1e8935f0
input	0x496...7bb1c
decoded input	{}
decoded output	-
logs	[{ "from": "0x289cd0FEcD559eCc99cA1E87f36fa035A57803E6", "topic": "0xfe5123cf6f119ad6ed886775d8eb77d77e915c52e9a0427669e5a109eb089395", "event": "Sendtoken", "args": { "0": "0x17140aCe34f714Ac39478A4DA42c830AE8a86A93", "1": "0x985Ce071E69b7d40fcAC1e411aCc7049A4EB4ff4", "2": "2", "from": "0x17140aCe34f714Ac39478A4DA42c830AE8a86A93", "to": "0x985Ce071E69b7d40fcAC1e411aCc7049A4EB4ff4", "tokens": "2" } }]
value	0 wei

Fig 6.16 Checking attributes of voter2

transact to Ballot.callReqAt pending ...

<https://ropsten.etherscan.io/tx/0xf5b0e48cc0bd1c8fbb084777a25f5dacad6e5875ed4920c9b4257c1deda4e848>

✓ [block:10294350 txIndex:8] from: 0x6c6...7d6f0 to: Ballot.callReqAt() 0x985...84ff4 value: 0 wei data: 0x496...7bb1c logs: 1 hash: 0xf5b...4e848

status	true Transaction mined and execution succeed
transaction hash	0xf5b0e48cc0bd1c8fbb084777a25f5dacad6e5875ed4920c9b4257c1deda4e848
from	0x6c61ff808c8988c7c2464563f49DC13E3727d6F0
to	Ballot.callReqAt() 0x985Ce071E69b7d40fcAC1e411aCc7849A4E84ff4
gas	56693 gas
transaction cost	56693 gas
hash	0xf5b0e48cc0bd1c8fbb084777a25f5dacad6e5875ed4920c9b4257c1deda4e848
input	0x496...7bb1c
decoded input	{}
decoded output	-
logs	[{ "from": "0x2089cd0fEc0559eCc99cA1E87f36fa035A57803E6", "topic": "0xfe5123cf6f119ad6ed886775d0eb77d77e915c52e9a0427669e5a109eb089395", "event": "Sendtoken", "args": { "0": "0x17140aCe34f714Ac39478A4DA42c830AE8a86A93", "1": "0x985Ce071E69b7d40fcAC1e411aCc7849A4E84ff4", "2": "2", "from": "0x17140aCe34f714Ac39478A4DA42c830AE8a86A93", "to": "0x985Ce071E69b7d40fcAC1e411aCc7849A4E84ff4", "tokens": "2" } }]
value	0 wei

Activate Windows
Go to Settings to activate W

Fig 6.17 Checking attributes of voter3

Step-4: After checking the attributes of the voters, if they posses the attributes tokens are sent by the attribute athoutrity. Now to access the encrypted key RequestKey function is called and msg.sender should be the voter same like in the above case. The transactions are depicted as follows:

transact to Ballot.callReqKey pending ...

<https://ropsten.etherscan.io/tx/0xd4607551cf570a475d741b26a599d3002bef3bee34ae64c544e9612f33a0ba32>

✓ [block:10294338 txIndex:11] from: 0xc84...255e6 to: Ballot.callReqKey() 0x985...84ff4 value: 0 wei data: 0xba5...f6e15 logs: 2 hash: 0xd46...0ba32

status	true Transaction mined and execution succeed
transaction hash	0xd4607551cf570a475d741b26a599d3002bef3bee34ae64c544e9612f33a0ba32
from	0xc843845C81752b50ccE848E4913219f585a255e6
to	Ballot.callReqKey() 0x985Ce071E69b7d40fcAC1e411aCc7849A4E84ff4
gas	75659 gas
transaction cost	75486 gas
hash	0xd4607551cf570a475d741b26a599d3002bef3bee34ae64c544e9612f33a0ba32
input	0xba5...f6e15
decoded input	{}
decoded output	-
logs	[{ "from": "0x4084d0F2309C38625fb4058C7d49af2c524f057", "topic": "0x8a42a398408a615c6659da74210b7d467bfdd7a92b76622063380b487d0c2b1", "event": "VerifyAll", "args": { "0": "0x985Ce071E69b7d40fcAC1e411aCc7849A4E84ff4", "1": "2", "2": "0x9865726d6974", "to": "0x985Ce071E69b7d40fcAC1e411aCc7849A4E84ff4", "tokens": "2", "approve": "0x5065726d6974" } }, { "from": "0x4084d0F2309C38625fb4058C7d49af2c524f057", "topic": "0x217a7e9da95d494c6a2f84b6baf75229e889826562fdd5ed270a4fb3f0fc0", "event": "Sendkey", "args": { "0": "0x8c1280aC890a78c95d50386f768837A08E09e503", "1": "0x985Ce071E69b7d40fcAC1e411aCc7849A4E84ff4", "2": "0x4084d0F2309C38625fb4058C7d49af2c524f057", "tokens": "2", "approve": "0x5065726d6974" } }, { "from": "0x8c1280aC890a78c95d50386f768837A08E09e503", "to": "0x985Ce071E69b7d40fcAC1e411aCc7849A4E84ff4", "encryptedkey": "0x343030306231303036626434326133363235646264346663346539353237353533336534396235313539306439383536633936393936653630363333393876345" }]
value	0 wei

Activate Wi
Go to Settings

Fig 6.18 Requesting key for voter1

3335376165", "from":
3335376165" } } 1. 10

Fig 6.19 Requesting key for voter2

35376165" "from":
Activate Wi
35376165" } } } }
Go to Settings

Fig 6.20 Requesting key for voter3

Step-5: Now, the voters cast their vote using the Vote function. The transactions are depicted as follows.

transact to Ballot.Vote pending ...

<https://ropsten.etherscan.io/tx/0xf4485285cce91e3eb4807b67165f798d3057138977ab072b94e78b004bc1fb51>

✓ [block:10294358 txIndex:24] from: 0xcB4...255e6 to: Ballot.Vote(uint256) 0x985...B4ff4 value: 0 wei data: 0xdcd...00002 logs: 0 hash: 0xf44...1fb51

status	true Transaction mined and execution succeed
transaction hash	0xf4485285cce91e3eb4807b67165f798d3057138977ab072b94e78b004bc1fb51
from	0xcB43B45C81752b5DecE848E4913219f585a255e6
to	Ballot.Vote(uint256) 0x985Ce071E69b7d40fcAC1e411aCc7049A4EB4ff4
gas	93017 gas
transaction cost	93017 gas
hash	0xf4485285cce91e3eb4807b67165f798d3057138977ab072b94e78b004bc1fb51
input	0xdcd...00002
decoded input	{ "uint256 toProposal": "2" }
decoded output	-
logs	[]
value	0 wei

Activate V
Go to Setting

Fig 6.21 Voter1 casting vote

transact to Ballot.Vote pending ...

<https://ropsten.etherscan.io/tx/0xe3959e4870d2f9d4afff1e44e169fd80b5291bda1ecc255f081c6cf64c9a2011>

✓ [block:10294362 txIndex:3] from: 0xdB7...f2C18 to: Ballot.Vote(uint256) 0x985...B4ff4 value: 0 wei data: 0xdcd...00002 logs: 0 hash: 0xe39...a2011

status	true Transaction mined and execution succeed
transaction hash	0xe3959e4870d2f9d4afff1e44e169fd80b5291bda1ecc255f081c6cf64c9a2011
from	0xdB7066A44c6a8036eff3a2c60e7e6D82d75f2C1B
to	Ballot.Vote(uint256) 0x985Ce071E69b7d40fcAC1e411aCc7049A4EB4ff4
gas	75917 gas
transaction cost	75917 gas
hash	0xe3959e4870d2f9d4afff1e44e169fd80b5291bda1ecc255f081c6cf64c9a2011
input	0xdcd...00002
decoded input	{ "uint256 toProposal": "2" }
decoded output	-
logs	[]
value	0 wei

Activate V

Fig 6.22 Voter2 casting vote

transact to Ballot.Vote pending ...

<https://ropsten.etherscan.io/tx/0x6b496587e0223565ec7437c88e97896b71ff8c0d4d0ff013469f0acec9097a23>

✓	[block:10294367 txIndex:13] from: 0x6c6...7d6f0 to: Ballot.Vote(uint256) 0x985...84ff4 value: 0 wei data: 0xcd...00001 logs: 0 hash: 0x6b4...97a23		
status	True Transaction mined and execution succeed		
transaction hash	0x6b496587e0223565ec7437c88e97896b71ff8c0d4d0ff013469f0acec9097a23		
from	0x6c61ff8b8c8988c7c2464563fa9dc13e3727d6f0		
to	Ballot.Vote(uint256) 0x985ce071e69b7d40fcac1e411aCc7049A4EB4ff4		
gas	93017 gas		
transaction cost	93017 gas		
hash	0x6b496587e0223565ec7437c88e97896b71ff8c0d4d0ff013469f0acec9097a23		
input	0xcd...00001		
decoded input	{ "uint256 toProposal": "1" }		
decoded output	-		
logs	[]		
value	0 wei		

Activate Win

Fig 6.23 Voter3 casting vote

Step-6: After the voting process is completed invoking the winningProposal function results in declaration of the winner whose transaction is as follows.

call to Ballot.winningProposal

CALL [call] from: 0x6c61ff8b8c8988c7c2464563fa9dc13e3727d6f0 to: Ballot.winningProposal() data: 0x609...ff1bd

transaction hash	call0x6c61ff8b8c8988c7c2464563fa9dc13e3727d6f00x985ce071e69b7d40fcac1e411aCc7049A4EB4ff40x609ff1bd
from	0x6c61ff8b8c8988c7c2464563fa9dc13e3727d6f0
to	Ballot.winningProposal() 0x985ce071e69b7d40fcac1e411aCc7049A4EB4ff4
hash	call0x6c61ff8b8c8988c7c2464563fa9dc13e3727d6f00x985ce071e69b7d40fcac1e411aCc7049A4EB4ff40x609ff1bd
input	0x609...ff1bd
decoded input	{}
decoded output	{ "0": "uint8: _winningProposal 2" }
logs	[]

Activate Win

Fig 6.24 Declaration of winner

In case of tie between any candidates, the contract returns a string displaying the following message:

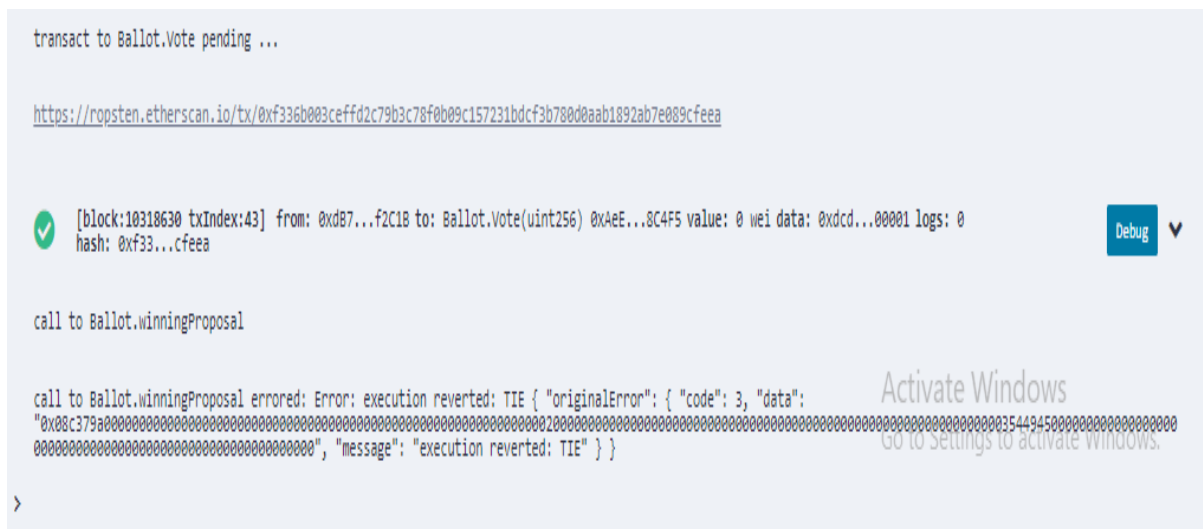


Fig 6.25 Output incase of TIE

6.2 ANALYSIS:

- Polling officers are not needed to check the voters on paper as the checking task is done by our smart contract itself
- There is no requirement for marking the fingers with ink of people who have casted their vote because our smart contract won't allow people second time for voting process and prevents unauthorized voters also.
- Our contract makes it quicker to count the total votes casted and whom because the data is continuously updated after every vote.

6.2.1 SECURITY ANALYSIS:

- Before any unintended transaction a pop-up window comes up stating that the transaction would fail given the unsatisfactory conditions. Here we have tried to vote second time for a single user and the pop-up is as follows which displays “CAN’T VOTE”:

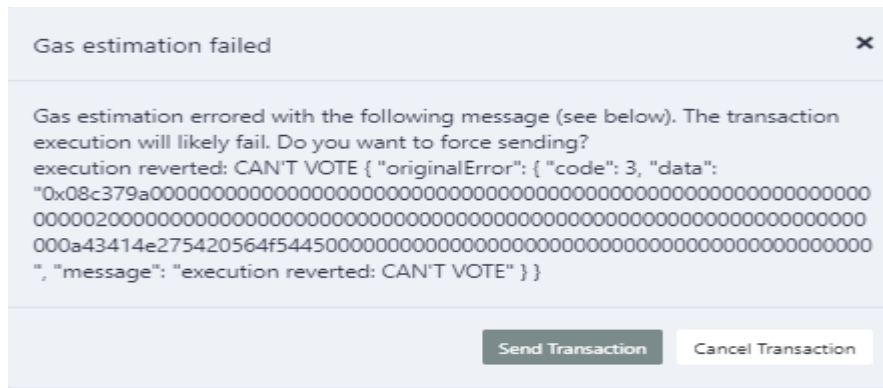


Fig 6.26 Double voting error

- When voters try to vote twice the transaction fails and doesn't allow them to vote the second time. The transaction details are as follows:



Fig 6.27 Voting for the second time

- Also, only chairperson is allowed to register the voters and if not the transaction is reverted.
- Since we are using blockchain technology, the anonymity of the voter is maintained since no one except the voter knows their account details or the private key.

CHAPTER 7

ADVANTAGES, LIMITATIONS AND APPLICATIONS

7.1 ADVANTAGES:

The following are the advantages of using blockchain technology and ABAC for Electronic Voting Process:

- Since blockchain maintains a distributed ledger, it is impossible to alter the votes.
- Since voters use Metamask accounts in order to participate in the election, their identity is not revealed thus maintaining anonymity.
- The entire process is quicker than the paper ballot system.
- Results are declared instantaneously after the election process is complete.
- Process for voting would be very easy.
- The saving of considerable printing stationery and transport of large volumes of electoral material.
- Only voters with particular attributes take part in the election process.
- The attributes can be changed according to the requirements and used in different scenarios.
- Reduction in polling time, resulting in fewer problems in electoral preparations, law and order, candidates' expenditure, etc.
- User friendly. Voter can exercise his vote within less time when compared to the existing voting process.
- Reduces man power. As there is no requirement of manual verification number of polling officials can be reduced when compared to the existing process.
- Auditing the votes is possible so that users/voters can verify their votes making the process highly transparent.

7.2 LIMITATIONS:

- Since smart contracts are being used gas limit is a major setback in case of very high number of candidates standing for the election.
- Blockchain technology requires mining every transaction which requires high computational power computers.
- Initial cost for the network setup is high.
- Since the entire process is stored at every node, it requires large amount of memory.
- The transactions might get reverted due to gas estimation problems.
- People with no educational background may find the process difficult.
- Since only few people are required to supervise the process, it may lead to unemployment.

7.3 APPLICATIONS:

- It can be used in companies, industries, Schools, Colleges or any Private or Government Sectors as it is fool proof and provide accurate results.
- It can be used in taking group decisions where voting is confidential.

CONCLUSION AND FUTURE SCOPE

CONCLUSION

In this project we have shown the implementation of a system that minimizes the possibility of rigging in elections and reduces the need to do manual work. Blockchain based E-Voting System with Access Control is programmed using solidity language for remix IDE software. This system reduces the complications in e-voting machines and voting process is made more transparent. It reduces the burden of the polling officials in identifying the voter. In the existing voting process manual verification of the voters is performed using voter card or aadhar card photo, where images are not clear and is a tedious process and sometimes can lead to wrong conclusions. Instead here we have used Metamask accounts and attribute based control which are used for voter verification with high accuracy.

There is no requirement for marking the fingers with ink for the people who have casted their vote because our smart contract won't allow people second time for voting process. Our code makes it quicker to count the total votes casted for each contestant because the data is continuously updated after every vote. This voting system is cost effective when compared to present system, this system is convenient to use and consumes less time.

FUTURE SCOPE

- Here the contract is implemented with less number of voters and candidates. In real life scenario, the numbers will be pretty high. In such cases one can concentrate on how to reduce the ether/wei consumption per transaction.
- The smart contract designed here is for a single ballot. One can work on integrating multiple ballots and updating the vote counts based on all the ballots.
- One can use the ABS scheme along with ABAC which enables signing with certain criteria like that the signers should possess some specific attributes, rather than just a private key.
- Our increasingly decentralized society motivates the need of multiple authorities to take part in attribute-based access control rather than a single authority.
- Intelligent agents can be integrated with smart contracts to implement the E-Voting system.

REFERENCES

1. Votereum: An Ethereum-based E-voting system, Linh Vo-Cao-Thuy, Khoi Cao-Minh, Chuong Dang-Le-Bao, Tuan A. Nguyen.
2. A Secure Decentralized Trustless E-Voting System Based on Smart Contract, Jiazhuo Lyu Zoe L. Jiang, Xuan Wang, Zhenhao Nong, Man Ho Au, Junbin Fang.
3. Towards the intelligent agents for blockchain e-voting system, Michał Pawlaka , Aneta Poniszewska-Maranda ´ a, Natalia Kryvinskab,c.
4. Blockchain Technologies for the Internet of Things: Research Issues and Challenges Mohamed Amine Ferrag , Makhlouf Derdour, Mithun Mukherjee, Abdelouahid Derhab, Leandros Maglaras and Helge Janicke.
5. Attribute Based Signature Scheme, Guo Shaniqng, Zeng Yingpei.
6. Attribute-based Multi-Signature and Encryption for EHR Management: A Blockchain-based Solution Hao Guo, Wanxin Li, Ehsan Meamari, Chien-Chung Shen, Mark Nejad.
7. Secure Attribute- Based Signature Scheme with Multiple Authorities for Blockchain in Electronic Health Records Systems, Rui Guo, Huixian Shi, Qinglan Zhao, Dong Zheng.
8. Blockchain based E-Voting, Friðrik Þ. Hjalmarsson, Gunnlaugur K. Hreiðarsson, Mohammad Hamdaqa, Gísli Hjalmtýsson.
9. Secure Digital Voting System based on Blockchain Technology, Kashif Mehboob Khan¹ , Junaid Arshad² and Muhammad Mubashir Khan.