Design and Implementation Considerations of POA Network Architecture in the Ethereum Blockchain

P. Shamili
Faculty of Computer Science and Engineering, SRM Institute of Science and Technology (SRM IST), Kattankulathur, Chennai, India.
E-mail: shamilip@srmist.edu.in

Dr.B. Muruganantham
Faculty of Computer Science and Engineering, SRM Institute of Science and Technology (SRM IST), Kattankulathur, Chennai, India.
E-mail: muruganb@srmist.edu.in

Received October 04, 2021; Accepted December 23, 2021
ISSN: 1735-188X
DOI: 10.14704/WEB/V19I1/WEB19358

Abstract

The upcoming technology in creating the distributed applications for different use cases in real world applications are Blockchain technologies. The application may be private or public depending upon the use cases. Most of the applications are built by using the Ethereum platform in blockchain for use of security purpose. The nodes in the distributed network shares all the data to the other nodes without any modification of the data in the blockchain technology. The transactions done by the nodes were trusted through the digital signatures. Here, no central control or the central system will be there to keep track of other nodes data. In this paper, we built the POA network by using the Ethereum blockchain platform. The network is called the Proof of Authority and its main purpose is to deploy the contracts especially with authority in the Ethereum blockchain platform. We also explained about the ERC20 tokens, the working process of using the POA network and the codes for connecting with the web page.

Keywords

Blockchain, Ethereum Platform, Distributed network, ERC-20 Token, Web3.

Introduction

The distributed database that stores all the records of the transactions among the nodes in the network named as Blockchain (Nakamoto, 2008). Each and every transaction, it is verified by the majority of the nodes in the network. It is the backbone for the digital crypto currency called Bitcoin (Nakamoto, 2008). One of the best examples for the

blockchain is the bitcoin. This technology came to the world through a paper "Bitcoin: a peer-to-peer electronic cash system" published in the year 2008 by Satoshi Nakamoto (Nakamoto, 2008). Here, in blockchain technology the transactions stored in the digital ledgers are distributed among the nodes in the network which are immutable. Some of the use cases in blockchain are land assets, supply chain management, etc.

In blockchain, we use a decentralized network instead of using the centralized network. In the peer-to-peer network, all nodes share their records to every node in the distributed network. A copy on one node's data will be shared to all the other nodes in the blockchain. Here, no central authority is given to any of the node. As it is decentralized (Lin, et al., 2018), anyone in the network can verify the other node whether it is valid or not. The node which is verified by the participant is called the minor. The minor is the participant (Nakamoto, 2008), who goes for mining work. Here, mining is the process of solving the mathematical crypto-puzzle. In blockchain, the bitcoin uses the cryptographic proofs for trusting the nodes to execute their transactions. Every transaction by the nodes were trusted through the digital signatures. Here, no central control or the central system to keep track of others data. Where, nodes are the computers that are connected to a distributed network. In blockchain, the data of one node is distributed among all other nodes in the world through distributed system. It is transparent to all the nodes that verify the data and are tamper proof (Nakamoto, 2008).

Transactions are verified and stored in the blocks are done by Miners. The main role of a miner is to solve the Crypto-Puzzle to verify the transaction that occurs between the two nodes in the distributed network and the reward for the verification of a transaction is given to the miner (Nakamoto, 2008). The blockchain builds the trust among the nodes in the distributed network by the following steps:

a) Distributed – the nodes in the network are connected and shares the data transparently in the real time with no central authority.
b) Secured – it is secured through the cryptographic method named SHA256 for accessing and verifying the data.
c) Transparent – because of the distributed network, the nodes can see all the transaction details of all the other nodes and can verify it easily.
d) Consensus based – with the consensus algorithm (Wang, et al., 2020), all the nodes in the network agree to follow the rules that made to verify the transactions whether valid or not.
e) Flexible – the users in the network evolve in pace with their business dealing.

f) Time saving – because of no central authority, no need to wait for the transactions to be validated and verified.

Ethereum a Blockchain Platform

Blockchain allows developers to build many of the Dapps by using the different platforms (Buterin, 2014). Here, we are going to build an application using the Ethereum platform. One of the most important platforms to execute the arbitrary codes to run on the programs in Ethereum (Buterin, 2014; Wood, 2014). With the help of smart contracts, the distributed infrastructure could be able to complete the different types of projects in Ethereum.

Ethereum allows us to build the decentralized (Liang, 2019) and the fault tolerant applications, which removes the intermediaries and provides the transparency to all the nodes in the network (Buterin, 2014).

A new tradable token can be created by using the Ethereum that can be used as the cryptocurrency. The wallets that are compatible with the Ethereum blockchain are used as a token coin as a standard API (Ethereum).

To build a blockchain based organisation (Wood, 2014), the nodes to be added in a network and should be agreed for the constraints. Smart contracts will execute the contracts automatically only when all the constraints are satisfied.

A clear concept of Ethereum should be known before entering into it. Now, we are going to learn few things about the Blockchain - Ethereum platform and the working of smart contracts (Wood, 2014).

1) Ethereum Virtual Machine (EVM)

Ethereum virtual machine are provided by a platform called Ethereum (Ganache), where the codes are executed by using the nodes in the distributed network. Here, the machine handles all the nodes information like address, pertaining to balance, current gas price and all about the block information. It is written in C++, Java, JavaScript, python, Solidity and many programming languages (2019). By using this, anyone on the network can execute the codes that are guaranteed and are deterministic. Which means the Smart Contracts.

2) Solidity Language

For smart contract, the language used by Ethereum is the Solidity language (2016). The programming language that are developed on top of the Ethereum virtual machine. The main purpose of solidity is to store all the states of the tokens, which are send and receive by the nodes. The programmers who are experienced in JavaScript, C++, Python could easily understand the solidity language also. If the solidity language are known well, then the program for Ethereum smart contract could be written easily.

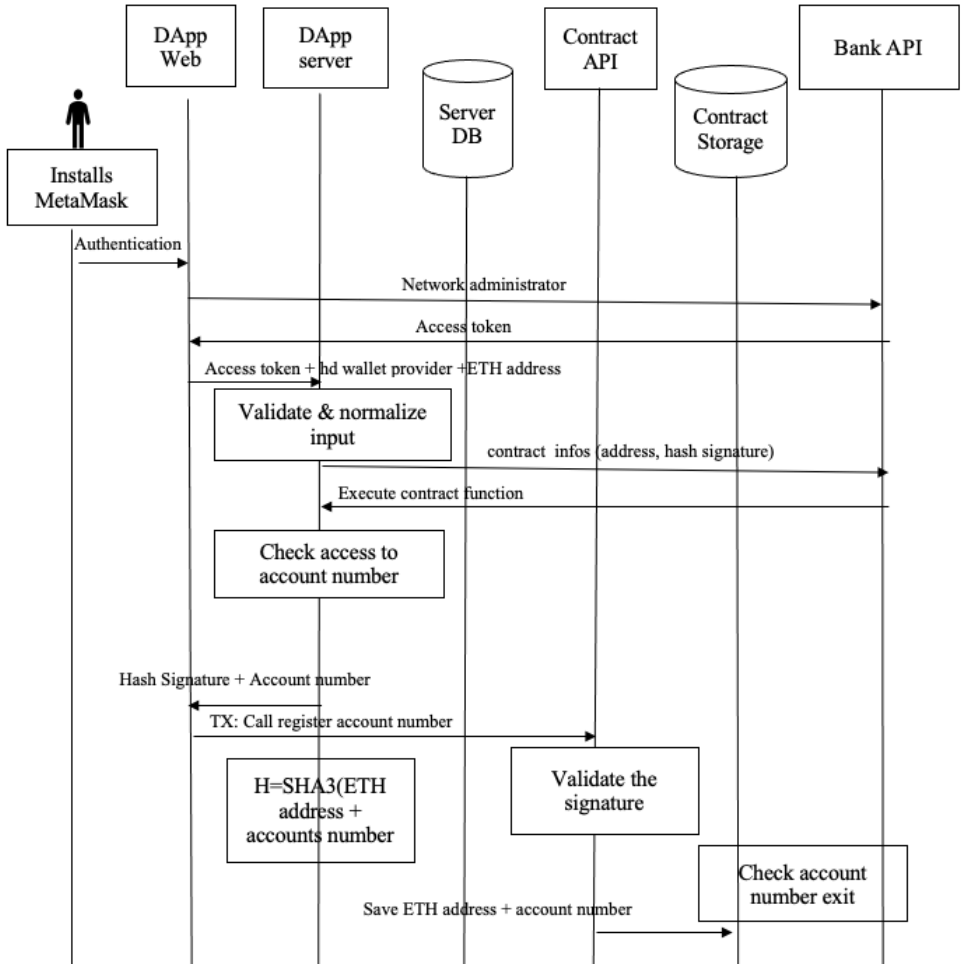
3) GAS

In Ethereum blockchain (2013), the miners process the smart contracts and results in adding the new block to the Ethereum blockchain. The rewards given to the miners for their efforts, through the smart contracts on the Ethereum virtual machine requires some amount of payments named as GAS (2013). The amount of gas needed to spend for the miners were created in their smart contracts. If the smart contract gets complicated, the gas fee also gets increased.

Toolkits for Implementing the Ethereum Smart Contract are Follows

- **Node.js:** The program for running the JavaScript for server side environment (2020). For testing the functionality for the Ethereum smart contract we need Node.js (2020).
- **Truffle:** The Ethereum development framework, which allows us to write and test the smart contracts (Ganache). It is written in the JavaScript language and has the compilers for Solidity language.
- **Ganache CLI:** It is also named as TestRPC. The client (Ethereum remote procedure call) within the network (Ganache).
- **Web3.js:** The javascript API (2019), that communicates through RPC calls.
- **Parity**: the Ethereum client which is secure and fast for managing the accounts, tokens, etc …
- **Visual Studio Code:** The code editor (2019), where any other editor can implement by using it.

Sequence diagram for use case bank application



The above sequence diagram explains about use case of a banking application by using PoW (Nakamoto, 2008). In comparison with the other Dapps, PoA (Proof of Authority) (2018) is the process that completes in one-step verification. Our Dapp client and server integrate with the accounts for API services (plaid.com). To authenticate the user (Hu, 2019), the client side uses the service widget. The access token is returned from the plaid to client, when the authentication is successfully completed. With the plaid access token, the user fills the data about bank account number, account holder details, etc… and submits it by using the server. Here, the server contains the web app and the parity nodes that are connected to the Blockchain (Liang, et al., 2015). To deploy the PoA contract (2018), the nodes are made run under the Ethereum account. It needs the account to be unlocked. By using the trailing space, the user account is validated and normalized by the server. Then, the bank account number is fetched by using the access token by the server. Later, it verifies that the account number given by the user is present in the list returned by the plaid or not.

Now, the server combines all the user's details of eth address + bank's name + account number into a single stream of hash (SHA-3) function (Manzoor, et al., 2019). With the user's signs with their private key and the hash (to unblock the users account). The digital signature, normalized bank account number and the name all returned to the client. With the Metamask (2020), the user signs the transaction and invokes the contracts.

The eth address is being verified by the contract that the address already exists or not. If exists, the transaction is rejected by the contract. If its new, then combines the parameters in order the server did and compute the new hash of SHA3. To verify the account that belongs to the user, it checks for the digital signature. If not match, rejects the transaction. If matched, then stores the transaction into the blockchain.

Implementation of Ethereum

1. Build Ethereum DAPP

In Ethereum blockchain, to interact with the smart contracts, the Web3 is used. Where Web3 is a JavaScript library. When we develop the decentralized application, the Web3 library plays a major role in interaction with the user and the smart contracts. With this we are directly interacting with the API of Ethereum as shown in figure 1. Here, the Web3 makes the process easier for us.

The two main ways to interact with the smart contracts are:

1. Allows us to call the API (eth_call) or to read the data from the smart contracts.
2. Allows us to modify the data in the smart contracts

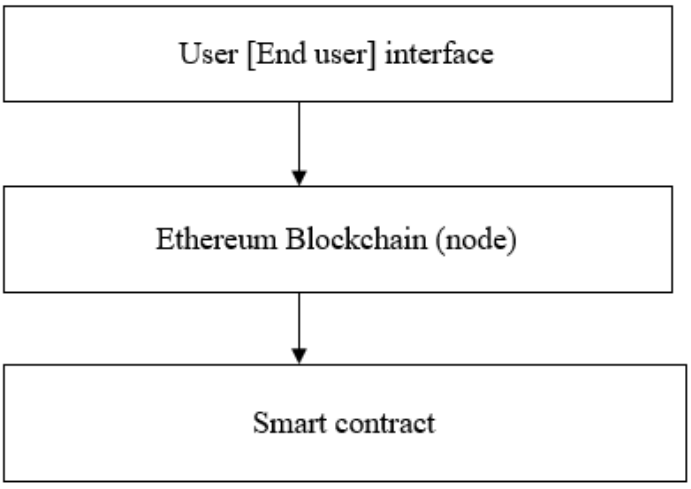


Figure 1 Decentralized application architecture

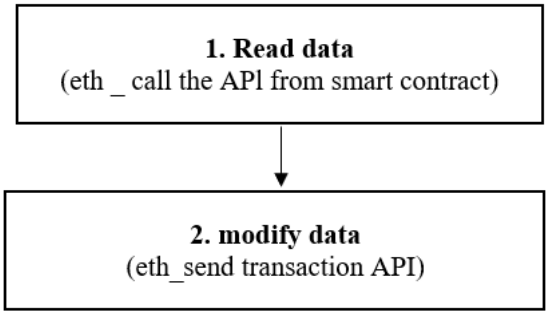


Figure 2 Interaction of web3 and smart contract

Functions of Smart Contract in Ethereum

1. Smart contract address
2. Smart contract function
3. Function arguments
4. Ether value [transaction API, return API, call API]
5. Other transaction parameter [gas, gas limit]

In the transaction API, the block mining time is 15 seconds and we need to wait for it. The functions done within 15 seconds calls the function, read the function (Ethereum Community, 2016), read from the blockchain and finally returns the transaction shown in figure 2. In the earlier days, the Web development used a protocol called REST API. Now, in Ethereum API, we are using the JSON-RPC protocols. In our JSON-RPC, we are having the single end- point, means that with one end-point we can use other multiple end- points.

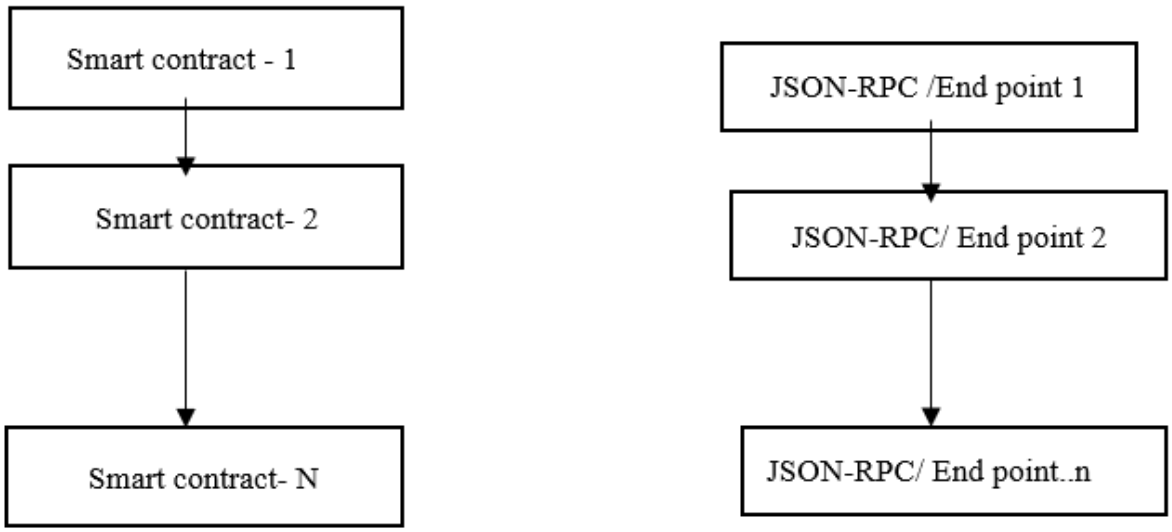


Figure 3 Comparison of smart contract with API and RPC

The smart contracts could be created and deployed into the blockchain (Lin, et al., 2018). For this, we need to modify the Ethereum platform to add a new block. But, it is not feasible for us to modify it easily as shown in figure 3. Instead of modifying it, we are using the JSON-RPC protocols which allows us to dynamically create many of the end points. This problem between the REST API and the web API are specified to certain amount of data in our smart contract are not that much easy. So, here we are using the Web3 a JavaScript library. We can install it simply from the NPM package manager of Node.js (2020).

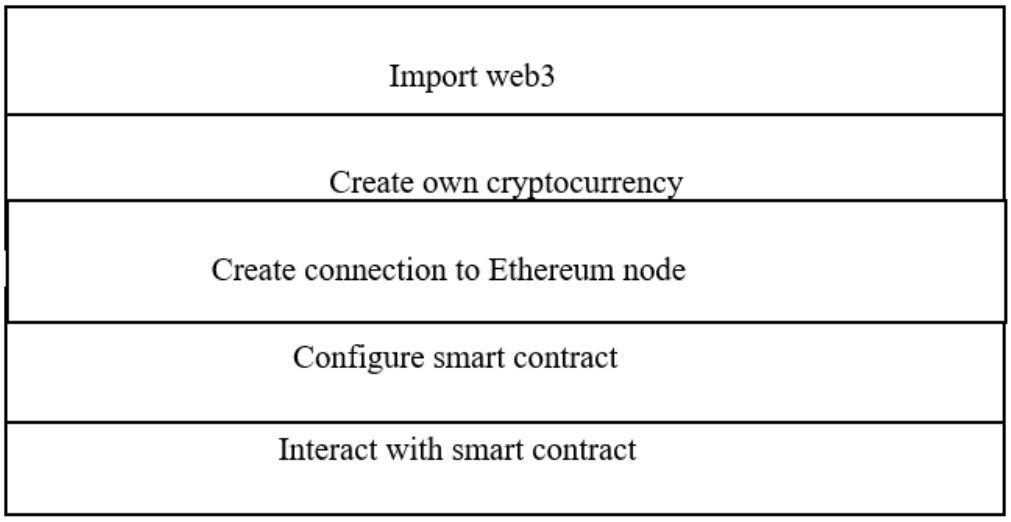


Figure 4 Flow of interaction with web3 and smart contract

Where Web3 is an isomorphic library, used as a backend with node.js in the frontend and in the web browser, we need to access the Ethereum node ie., Main Ethereum network. All these are to be run in the Ethereum blockchain software. It has many of the nodes. In order to execute the web3, we need to get access to any these nodes.

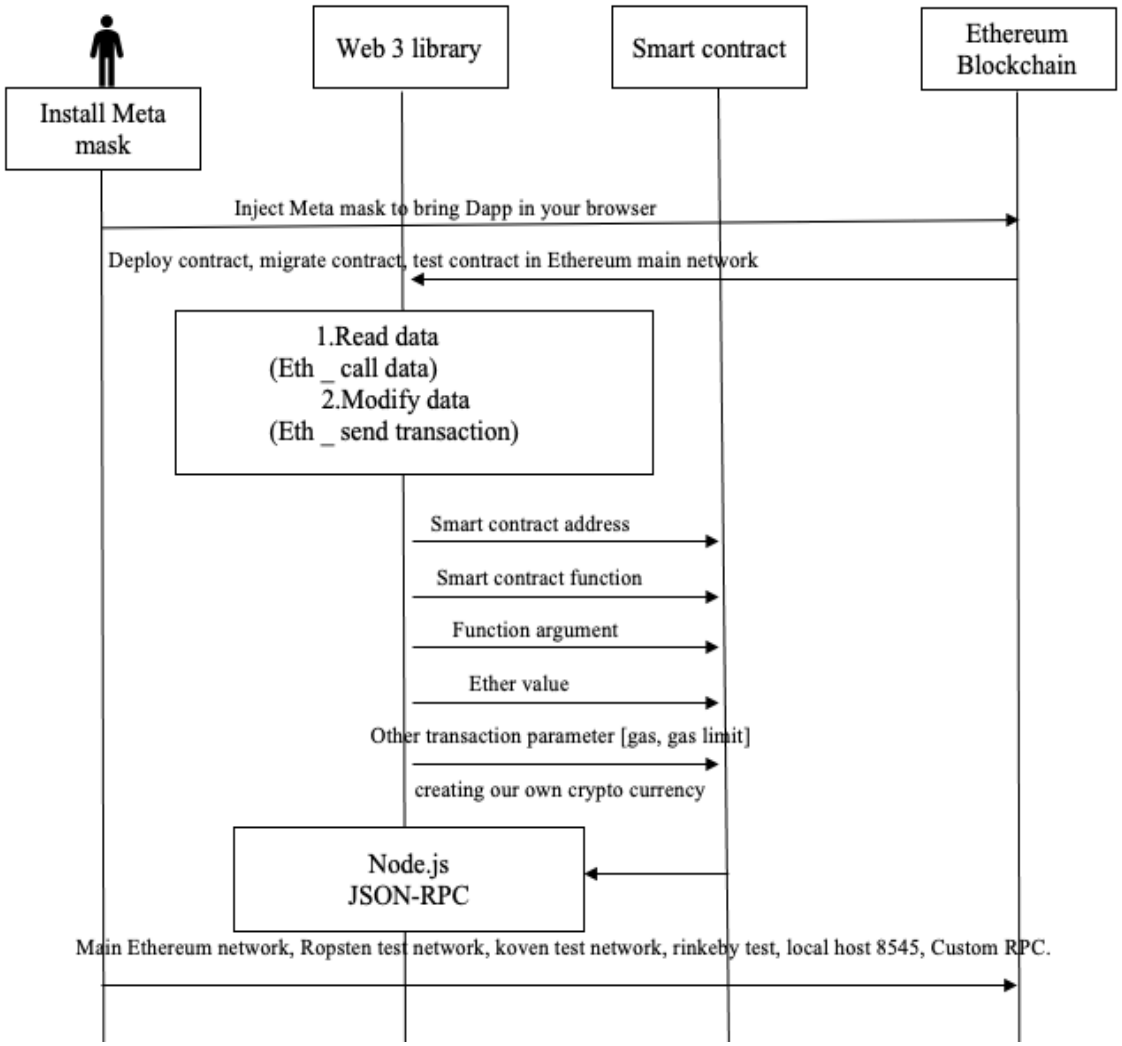
In Ethereum blockchain, it is difficult to have high computational power and storage. For that, we need ERC20 (Vogelsteller, & Buterin, 2015) token and it is highly expensive. Instead of using ERC20 token (Vogelsteller, & Buterin, 2015), we can use Infura, which interconnects us with the Ethereum nodes. It is free for its usage and the access to Ethereum can be given by creating the new account in it. The API gives us the access to get interacted with the URL's. This is to be connected with the infura. When we are connecting with the Ethereum with the smart contract, we use main network or like Ropsten, Rinkpy, Koven. We can also use the local host with Ganache. By using this, we don't need Web3 (2018) in contact with ganache or with any Infura.

The steps to be followed for the Ethereum blockchain platform:

1. Creating the ERC20 token (Vogelsteller, & Buterin, 2015; Entriken, Shirley, Evans, & Sachs, 2018).
2. Importing the Web3 and creating a new connection to the Ethereum node.

Configuring the contracts by passing an interface, which describes all the functions that is from outside the blockchain. We name it as API and we also need the provider address of our smart contract. The instance is passed to the Web3 and after this process shown in sequence diagram, the objects are created and interacted with the smart contracts as shown in figure 4.

Sequence Diagram



2. Our Own Crypto-currency in Ethereum

ERC20 Token

Ether, the native crypto-currency of the Ethereum blockchain. With this, we can create our own crypto-currency or token which are allowed by the Ethereum blockchain. ERC20 ("Solidity", 2019) are the standard that specifies how the tokens behave and how they are compatible with the other platforms, i.e: Crypto-currency exchange. Where ERC is defined as Ethereum Request for Command and 20 is the proposal identifier.

The Ethereum keeps track of account balances of the users who own their Ether. Ethereum is a platform, which allows us to create our own token without creating the new blocks.

By using the smart contract, the Ethereum Tokens are created. This standard shows that how the smart contract should work.

Example

We want to create a new token named "My Token" with the symbol (we can give as our wish) "MTK". There will be 10,000,000 of the tokens in existence.

Now, at the initial state the token keeps track of the basic token attributes. Like it records the name, symbol, cryptocurrency exchange and how many remaining tokens are there, etc… These tokens can be transferred from one account to another account payments like crypto-currency exchange. They are purchased in the Crowd sale like ICO (Initial Coin Offerings).

Crowd Sale

One of the most popular way is to hold a crowd sale or the Initial Coin Offering (ICO). The way for a company to raise its capital for their business by creating their own ERC20 token (Dafflon, Baylina, & Shababi, 2017) and can be purchased by the Investors with their Ethers.

Working of ERC20 Tokens

1. Wallet transfers: sending tokens from one account to another account.
2. Buying and selling the crypto-currency exchanges.
3. Purchasing the tokens in the Crowd sale.

The ERC20 (Vogelsteller, & Buterin, 2015) specification dictated interface that the smart contract must respond to. It specifies that the structure and the functions of the smart contract should have. Also it provides certain events that the token must have like Transfer event. For an example in the codes given below, the ERC20 standard specifies the implementation of the transfer function. The smart contract is required and that governs how we can send the ERC20 token from our wallet to another wallet that are shown in code given below,

Code for Smart Contract

Step 1: contract ERC 20 Token {
Step 2: function transfer (address to, uint256 value) **public returns**
{
require [balanceOf] $msg.sender \geq value$;
balanceOf [To] $\pm$ $value$;
Transfer ($msg.sender, to, value$);
return true;
}
//…
}

The above function implements the ERC20 (Dafflon, Baylina, & Shababi, 2017; Hale, 2017) standard:

1. Whether the function exists or not
2. It accepts the correct arguments or not
3. If the user don't have enough token for transfer, then it gets fail
4. If the user has the enough token, then the transfer occurs.
5. Selling of tokens event is triggered
6. If true, it returns the correct value.

A look into smart contract that how does it works as follows:

A. Stored the token name publicly as "DAPP Token" (Fröwis, Fuchs, & Böhme, 2019).
B. Stores the token symbol for currency exchange as string public symbols as "DAPP".
C. Total supply of the token in existence is stored as uint256 public total Supply.
D. For token mapping, (Wang, et al., 2020) it uses the solidity mapping to store the balance of each account as (address => uint256) public balance of (Ethereum Community, 2016; Ethereum Foundation, 2018).

E. A transfer function is implemented to allow the users to send the token from one account to another account.
F. Implements an approve function, so that the users are allowed to spend their tokens like crypto-currency exchange.

3. Creating the Decentralized Application

Contract Compilation

Here, the contracts are written and executed in the Solidity language and the extension is.sol file (Visual Studio Code, 2019; "Solidity", 2019). Now, we are going to create a truffle file by providing the command as truffle compile**.** Then the Migrations.sol (Rahimian, Eskandari, & Clark, 2019) file helps the process of deployment. If suppose we use Truffle box, we will be having multiple files. Now the truffle file gets compiled.

Running the Migration File

To deploy the contracts in the Ethereum, the Migrations files are used. These files are responsible for doing the deployment tasks, and the assumptions over time to time. Our project evolves that the new migration scripts on the blockchain network. As we compiled the truffle previously, now the command to run the migration is truffle migrate**.** It will run all the migration files in the migrations directory. In the simplest form, the migrations are simply the deployment scripts. If our migration runs successfully in the previous steps, then the Truffle migrate will starts executing from the last migration where it left. Now runs the newly created migrations. Truffle migrate won't work if no new migration exists. We can use the command --reset to run all the migration files from the very first beginning. Make sure that the Ganache is installed and running properly before executing the migration commands as we use here a local testing as Ganache (Ganache).

Testing Our Contracts

Truffle test command is given to test that our contracts are running perfectly or not.

The two different ways are there to test the truffle. They are:

1) In our application, we are using the JavaScript and Typescript for executing.
2) For exercising our contracts, we use the Solidity language. Both styles of tests have their advantages and drawbacks. See the next two sections for the explanation.

Truffle and Metamask

Make sure that the truffle is compiled and deployed before interacting with the smart contracts and then to interact with them by using the Web3 (Ganache) client side JavaScript. Here, we recommend that using truffle-contract library is much easier and robust. The simplest way to interact with the Dapps in a browser is MetaMask (2020). To connect with the Ethereum network, we use an extension for Chrome or Firefox without running the whole node on the browsers. The metamask can connect to main network and any of the test networks like Ropsten, Rinkeby, Kovan or a local Ethereum development called Ganache or Truffle Develop.

We can use our Dapp the same way that node interacts with the main network as we use development with truffle. MetaMask can be installed by using the following steps:

- For chrome, go to chrome web store and click add to chrome button.
- For firefox, go to firefox add-ons page and click add to firefox button.

The MetaMask is installed and create the user account and get the 12 seed phrase while doing the initial steps. Now MetaMask is connected with Ganache and it runs on 127.0.0.1:7545 Before proceeding, first check MetaMask Web3 (2020) instance and extension are configured properly with Ganache. After all the circumstances are verified, perform the following codes.

Code for Checking Instance for Web3

Step 1: $Is\ there\ is\ an\ injected\ web3\ instance$
Step 2: $if\ (typesofweb3\ !\ ==' undefined')\ \{$
App.web3Provider = $Web3.current\ Provider;$
Web3= $newWeb3(web3.currentProvider);$
Step 3:
}else {
Step 4: $if\ no\ injected\ Web3\ instance\ is\ detected, fallback\ to\ Ganache.$
App.web3Provider =new
Web3 = $Providers.HttpProvider('http: 127.0.0.1: 7545');$
Web3=$newWeb3(App.web3Provider);$
}

Connecting to the MetaMask Network

Here we are going to connect the MetaMask to the blockchain with the Ganache. Click (Nodejs, 2020) the networks to add what type of network we want to do. It may be main network or the test network. In the menu bar "New RPC URL" enter https://127.0.0.1:7545 and save it. Due to the local host running in MetaMask, our network will be named as "Private Network". After this, close the tab and move to the accounts page. Which means "Ganache". In Ganache, each accounts are given with 10 ethers and total of 100 ethers. For the deployment of smart contracts, the gas fee will be charged from the account. Since we have deployed our smart contract to the network, it is paid for it. Now, click on the icon in the upper-right corner to create the new accounts. When we launch Ganache, we will be provided with 10 accounts.

MetaMask along with Truffle Develop

The command line application that are used for testing purpose and to run the temporary blockchain (Metamask, 2020; Ganache) is called the Truffle Develop. It runs on the local host 127.0.0.1:7545. We suggest you to use 127.0.0.1:7545 instead of 127.0.0.1. Now, we want to use Truffle develop and to run it by default on 127.0.0.1:7545. So, we want to modify the codes. The New RPC URL entered by using the MetaMask in https://127.0.0.1:7545. Now, using Ganache cli with MetaMask, the only difference is that Ganache CLI runs by default on https://127.0.0.1:8545. So, we want to edit Web3 codes in MetaMask, when entering the "New RPC URL", enter http://127.0.0.1:8545.

The state of the block in Ganache has been changed. After the computation, the block number is changed. In our side, it is changed to Block 4. Initially, the original account will have 100 ethers. Now, it becomes less due to the transaction cost of the migrations.

Packet Management by Using NPM

Through the integration of npm, the truffle comes standard. It means that we can distribute and use the contracts, Dapps and even more Ethereum enabled libraries through npm (Nodejs, 2020). The following two directories are more important in Truffle package:

1. Contract.
2. Build (created by truffle and hold build/contracts files).

The Contract directory includes our projects raw solidity contracts and the build directory includes /build/contract and that holds the artifacts in the form of.json files (Nodejs, 2020). With this file, we can import all those contracts within the solidity codes. The.json build file allows us to interact with our contracts from the JavaScript, which could be used in our Dapp, migrations and scripts.

Lite-server Installation and Configuration

Now, we start our local web server and start using Dapp. The Lite-server library is used to serve our static files. Let us look in to the codes for our project the Lite-server to which and where the files to be included in the directory. Here, we added the./src directory for the website files and for contracts, we added./build/contracts directory. The dev command is to chech the scripts in the package.json (Rahimian, Eskandari, & Clark, 2019), file in our project root directory. The script code tells the npm to run our local lite-server (Nodejs, 2020). When we give the command **npm run dev**, the server starts running and the local host for our project will be displayed. we will be directed to get the tokens from the crowd sale (ICO) as we discussed in the topic ERC20 tokens (Hale, 2017; Victor, & Lüders, 2019; Fröwis, Fuchs, & Böhme, 2019). Immediately, the MetaMask got popped up for requesting us to provide approval to connect with it or not. Here, without approval we will not be able to interact with our Dapp. Then the confirmation page of MetaMask (2020) appears. After confirming the page by providing out ethers in the account, we get the tokens and the ethers gets reduced and we got one token as Contract Iteration.

Proof of Authority Implementation

1. Proof of Authority (POA)

The consensus algorithm (Wang, et al., 2020; POA Network, 2018), which gives the power to limited number of validators to validate the transactions in the blockchain network. All the transactions are added to the blocks in the blockchain after the validation of the blocks by the authorized validators (POA Network, 2018). The proof of authority works according to the schema, where the validators are given the responsibility for creating the new blocks of transaction and that will be added to the blockchain. Also, by using the majority or the unanimous vote of block generators or without validating and depending on the configuration chosen for the blockchain, a new block can be directly accepted. The advantage of using the proof of authority is by using the limited number of validators with the less energy consumption.

Compared to Proof of Work (POW) mechanism, there is no need to go for solving the mathematical puzzle like the mining and it needs low energy consumption. Here, we use the Ethereum blockchain platform for proof of authority by using the Oracle network as an independent validator. In the oracle network, there is approximately 1.5 million transactions per day are occurring. Here, the blocks are generated with an average time of about 5 seconds. This is the fastest network, where all types of applications are build using consensus algorithm in this network. With the proper time and response, the Ethereum Dapps were easily migrated and run on the oracle network. Because, it is more compatible in the Ethereum platform.

2. Overview Design of Proof of Authority

The participants (nodes) in the network agreed to follow the POA consensus algorithm (Wang, et al., 2020) to maintain the transaction records. To maintain this, the three constraints to be followed:

a) **To keep the ledger consensus:** Bitcoin applies the proof of work protocol, where the mathematical crypto-puzzle are solved by the miners. They verify the transactions and publish the new block in the blockchain (Lin, et al., 2018; Wang, et al., 2020). Similarly, in proof of authority (Feng, et al., 2019), if anyone have to publish a block, there should have the highest trust in the group of transaction. Then only the block could be accepted and published. Here, the verification of any of the block is open to all the participants in the network (Wang, et al., 2020).

b) **To motivate participants to publish blocks:** In bitcoin, reward will be given to the participants who solves the mathematical crypt-puzzle. But in proof of authority, the trustworthiness is used. So, no reward will be given to the participants. It is based on the reputation, that shows how the participants trust each other could provide better service. In the overall network, the participant who gets more trust will publish the new block and gets the higher trust rate.

c) **To prevent blocks from being modified:** In bitcoin, the open network has the access to every machine that client has installed on their machine. It consumes high computational power, when they try to modify the transactions (Wang, et al., 2020). This becomes an attack called 51% attack. The proof of authority follows permissioned blockchain (Khan, et al., 2019), where all the participants are agreed to use their public and private keys stored locally and all other participants copy others public key. Because of having this public key, if any of the malicious events occurs, it could be immediately identified. The integrity of the data from the transactions are protected by using the Merkle tree concept.

Transaction Filter

In the above paragraph, we discussed about the modification of the transactions by the malicious nodes (Manzoor, et al., 2019), here we are going to discuss how to prevent those transactions. For this, we use filter transactions before packaging them up. In Algorithm 1: filter transaction is used to filter them up. Initially, when the transaction occurs, it will be authenticated for its user's digital signature. Next is to check whether the verifier and the provider appeared belongs to the same block or not. For an example, let us assume that the transaction occurs between two nodes. The transaction Signature (Sig-x10) has been verified. If any of the transaction with same signature (Sig-10) comes, it will replace with the former ones that were used. To reduce the impact of bad-mouthing attack, (Khan, et al., 2019) from the same pair of transactions only one of them will be kept in the block. If the number of filtered transactions reaches the threshold λ, then the remaining set of transactions will be packaged to conduct the alternative block, meanwhile the other transactions would remain in the memory pool for next round.

Algorithm 1: For Transaction Filter

Input T=$\{t_0, t_1, t_{2,} \dots t_n\}$;, where t_n is the number of the set.
Output Filter Transaction
Step1: $filterednumber = 0$
Filtered$Transactions = \{t_0, t_1, t_{2,} \dots t_n\}$;
Step 2: $for\ t_0\ in\ T\ do$
Step 3: $if\ the\ rater\ and\ the\ provider\ of\ t_n$
Have appeared in $filteredTransactions\ then$
Update ($filteredTransactions$);
Else
Step 4: $if\ filterenumber \leq \lambda, then$
Filtered$Transactions.append(t_n)filterenumber$
Filtered$number \pm 1$;
else:
break;
else;
return verifying Drop(t_n)

Block Verification

Before appending to blockchain, the blocks received from the participants are verified. This process is shown in the Algorithm 2: initially, the participant's signature should be authenticated (Manzoor, et al., 2019). So, the transactions in the current block are verified by the signature. Next, based on the checked transactions, the rank lists will be recalculated. Later, the rank list provided will be checked that whether the sender of the block is there in the list or not.

Algorithm 2: Block Verification Algorithm

Input: currentBlock, $ranking\ List\ =\ \{\}$
Output: block validity
Step 1: $if\ !verify\ (currentBlock.Signature) then$
return false;
rankingList = $calculateRankingList(currentBlock.Transaction\ List)$;
Step2:
$if\ (currentBlock.Signature)\ is\ not\ matched\ to\ he\ public\ key\ of\ the\ rankingList.Top$
then, r
return false;
return true:

Implementation of Proof of Authority Network

The upcoming screenshots will show that we deployed in the POA network with the sample Dapp token with ICO (Victor, & Lüders, 2019) smart contract which allows us to get the tokens and to send the tokens. Here, we are using the truffle framework to develop and the language used is Solidity programming language (Nodejs, 2020).

Running the Dapp in Development Blockchain

For running the Dapp in the development blockchain, we installed Ganache. The command to install it using the npm is "npm install ganache-cli" and also runs using the client-side application and the development with server-side. In this paper, for implementing the POA network, we are using Infura for development of Dapps. Initially, we have to sign-up for getting the API key from Infura. After getting it, we have to paste the key in the code from the.env file.

Installing all the Dependencies Needed

We want to install the following dependencies to our project in order to deploy the Dapp to our POA network.

1. Truffle HD Wallet
2. Dot environment (.env)

1. Truffle HD Wallet

The library that allows truffle to sign the transaction and broadcast them to the POA network is Truffle HD Wallet. So, we can deploy our Dapp whenever we want to do the transactions.

Using the command "npm install truffle-hdwalletprovider –save dev", we installed the truffle HD Wallet provider. It uses the mnemonic seed phrase in order to generate our account. So that we can deploy our Dapp in the POA network.

2. Dot Environment

To install the.env file, the command used here is "npm install dotenv –save dev", Here, we now installed both the dependencies and we have to save.env file by applying infura API key and the mnemonics seed phrase.

After creating the truffle HD Wallet provider with the mnemonic seed phrase, the new wallet is created and stored it in the dotenv file with the mnemonic seed phrase. It can be read with the node global object processed. In the package.json file, we have all the packages needed for our project. And we have to keep track of the address, as we use it further in POA network

Setting Up Our POA Network

Our POA network will be connected by using the truffle.js file by using the truffle HD Wallet provider and the.env dependencies. In the truffle.js file, we apply the mnemonic seed and the infura API key. And also, we need to specify the RPC end-point of the infura shown in figure 5. In order to deploy the smart contract, we need to specify the network in POA, the gas limit and the gas price.

POA Network – Truffle Migrate

To migrate the truffle, the command used is "truffle migrate–network poa" and it is shown in figure 5.

```
✔ ~/code/deploy_poa_network [master L|⚑ 3]
11:17 $ truffle migrate --network poa
Using network 'poa'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
  ... 0x3b098e7182b5abbf8263388f73273838f6fc62a04ccf4d6f30d959256e077aaa
  Migrations: 0x2ff02686642bd80d3aff3c25e4e5c6eba70da23d
Saving successful migration to network...
  ... 0x1cd79ac83265e19499136e8c6cd710b09d1f526246dae1cc36383e3662221de1
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying SimpleStorage...
  ... 0x3c904a32585cf3283b5ea46154825e404cacdccbfd2bd6d117bd0d9b600d1d36
  SimpleStorage: 0x889ac76595fa2950e2d848764548399dae0d6fc8
Saving successful migration to network...
  ... 0x81989975e017fc41dd4311c22df93515d3cc3e9e53431b78b8322a1cc42f661c
Saving artifacts...
✔ ~/code/deploy_poa_network [master L|✚ 3⚑ 3]
```

Figure 5 Screenshot to migrate truffle into POA network

Here, by we used the metamask inject along with the client application, to connect the Ethereum HD Wallet, which supports the POA network (2018). We now select the POA network as our main network account shown in figure 6 and can be deployed by using our smart contract.

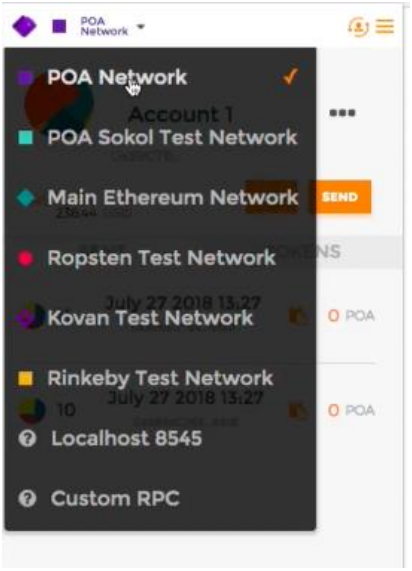


Figure 6 POA network

We installed a nifty wallet, by using the chrome extension. Which is used to buy some tokens by using the smart contracts to deploy in our POA network shown in figure 6. When we deploy the event, the nifty wallet will pop up for the confirmation of transaction. After the confirmation, it got updated successfully with the POA address account (2018). For the verification of the successful transaction, we open our nifty wallet and copy the transaction hash value. Paste it in the POA network blockchain and verify it as shown in figure 7.

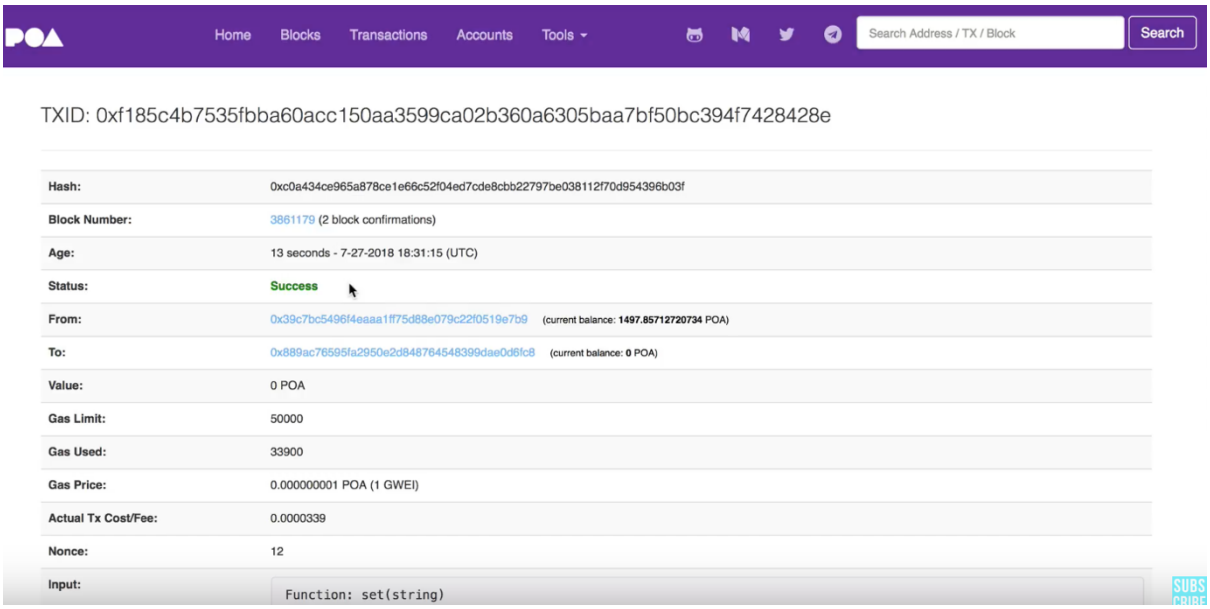


Figure 7 Transaction details in our POA network

Conclusion

In this paper, we implemented the Proof of Authority (POA) network in the Ethereum blockchain platform by using the smart contract. We also proposed the POA network by comparing it with the other networks by using the Metamask like Rinkeby, Ropsten etc… The other networks are public and open to everyone. Here, in this POA network, it is also public with few of the authority that is given only to the authorized participants, who have the permission to access the network. In the future work, the Scalability, network issues it can be resolved and implemented in the game modelling for scalability and the decentralized finance for using the tokens in the financial sectors.

References

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, 21260.

Buterin, V. (2014). Ethereum white paper: a next generation smart contract & decentralized application platform. *First Version*, *53*.

Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, *151*, 1-32.

Vogelsteller, F., & Buterin, V. (2015). ERC-20 token standard. *Ethereum Foundation (Stiftung Ethereum), Zug, Switzerland*.

Entriken, W., Shirley, D., Evans, J., & Sachs, N. (2018). EIP 721: ERC-721 non-fungible token standard. *Ethereum Improvement Proposals*.

Dafflon, J., Baylina, J., & Shababi, T. (2017). EIP 777: A New Advanced Token Standard. https://eips.ethereum.org/EIPS/eip-777

Hale, T. (2017). Resolution on the EIP20 API Approve / Transfer From multiple withdrawal attack, #738. https://github.com/ethereum

Victor, F., & Lüders, B.K. (2019). Measuring ethereum-based erc20 token networks. *In International Conference on Financial Cryptography and Data Security*, Springer, 113-129.

Fröwis, M., Fuchs, A., & Böhme, R. (2019). Detecting token systems on ethereum. *In International conference on financial cryptography and data security*, *Springer*, 93-112.

Rahimian, R., Eskandari, S., & Clark, J. (2019). Resolving the multiple withdrawal attack on erc20 tokens. *In IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 320-329.

Liang, W., Di Wang, K., Shi, J., Li, L., & Karagiannidis, G.K. (2019). Distributed sequential coalition formation algorithm for spectrum allocation in underlay cognitive radio networks. *IEEE Access*, *7*, 56803-56816.

Hu, Y., Li, L., Zhang, H., Liang, W., & Gao, A. (2019). Wireless powered D2D communication security using LSTM in emergency communication system. *In 28th Wireless and Optical Communications Conference (WOCC)*, 1-5.

Liang, W., Jiang, Y., Peng, L., Xie, Y., & Xiao, W. (2015). A Fixed Blocking Based Dispersed Information Hiding Algorithm for Multiple Carriers. *Journal of Computational and Theoretical Nanoscience*, *12*(10), 3722-3726.

Lin, C., He, D., Huang, X., Khan, M.K., & Choo, K.K.R. (2018). A new transitively closed undirected graph authentication scheme for blockchain-based identity management systems. *IEEE Access*, *6*, 28203-28212.

Wang, E.K., Liang, Z., Chen, C.M., Kumari, S., & Khan, M.K. (2020). PoRX: A reputation incentive scheme for blockchain consensus of IIoT. *Future Generation Computer Systems*, *102*, 140-151.

Manzoor, A., Shah, M.A., Khattak, H.A., Din, I.U., & Khan, M.K. (2019). Multi-tier authentication schemes for fog computing: Architecture, security perspective, and challenges. *International Journal of Communication Systems*, e4033.

Khan, T., Alam, M., Akhunzada, A., Hur, A., Asif, M., & Khan, M.K. (2019). Towards augmented proactive cyberthreat intelligence. *Journal of Parallel and Distributed Computing*, *124*, 47-59.

Feng, Q., He, D., Zeadally, S., Khan, M.K., & Kumar, N. (2019). A survey on privacy protection in blockchain system. *Journal of Network and Computer Applications*, *126*, 45-58.

Wang, E. K., Sun, R., Chen, C. M., Liang, Z., Kumari, S., & Khan, M.K. (2020). Proof of X-repute blockchain consensus protocol for IoT systems. *Computers & Security*, *95*.

Ethereum: https://ethereum.org

Ethereum virtual machine [EVM] Ganache truffle framework.

Nodejs, 2020. [Online]. https://nodejs.org/en

Ethereum, 2018. Ethereum (ETH) Blockchain Explorer. https://etherscan.io

Metamask, 2020. [Online]. https://metamask.io

Visual Studio Code, 2019. [Online]. https://code.visualstudio.com

Solidity, 2019. [Online]. https://solidity.readthedocs.io/en/v0.5.11

Web3ETHlibrary, [Online]. https://web3js.readthedocs.io/en/v1.2.0/web3-eth.html

Ethereum Foundation, 2013. [online]. EVM Operation Codes Gas Cost. https://bit.ly

Ethereum Community, 2016. [online]. Ethereum homestead documentation.

Ethereum Foundation, 2018. [online]. Solidity documentation release 0.4.24.

POA Network for Ethereum ecosystem [online] documentation release 2018. https://www.poa.network/

Authors Profile

Ms.P. Shamili has received M. Tech in Computer Science Engineering from the SRM Institute of Science and Technology in 2019 SRM IST, Chennai. B.E in Jeppiaar Engineering College, 2017. She is specialized on Bitcoin Cryptocurrency Analysis and Blockchain Technology Novel Application and she is currently (2021) is a Research Scholar in Computer Science Department and the domain is Blockchain Technology and Distributed Computing in ETH of SRM Institute of Science and Technology.

Dr.B. Muruganantham currently is a Assistant Professor, Faculty of Engineering and Technology in Computer science and Engineering Department SRM IST, Chennai. Completed B.E in Manonmaniam Sundaranar University, 1994, M.Tech in SRM Institute of Science and Technology, 2006. Completed Ph.D in SRM Institute of Science and Technology, 2018. Research interest area are Blockchain Technology, Data Structures, Data Mining, Cloud Computing, IoT, SOA, Semantic web, Ontology, Data warehousing, Service Oriented Architecture, Web Services, Internet of Things, Machine Learning, Deep Learning.