Project Documentation

Blockchain

A Blockchain is a public decentralized ledger that securely records transactions between parties anonymously, thus cutting out the middleman. The word “block” refers to the way data is stored; on blocks. Any transaction is broadcasted to all the nodes on the blockchain which have to verify the transaction.

Ethereum

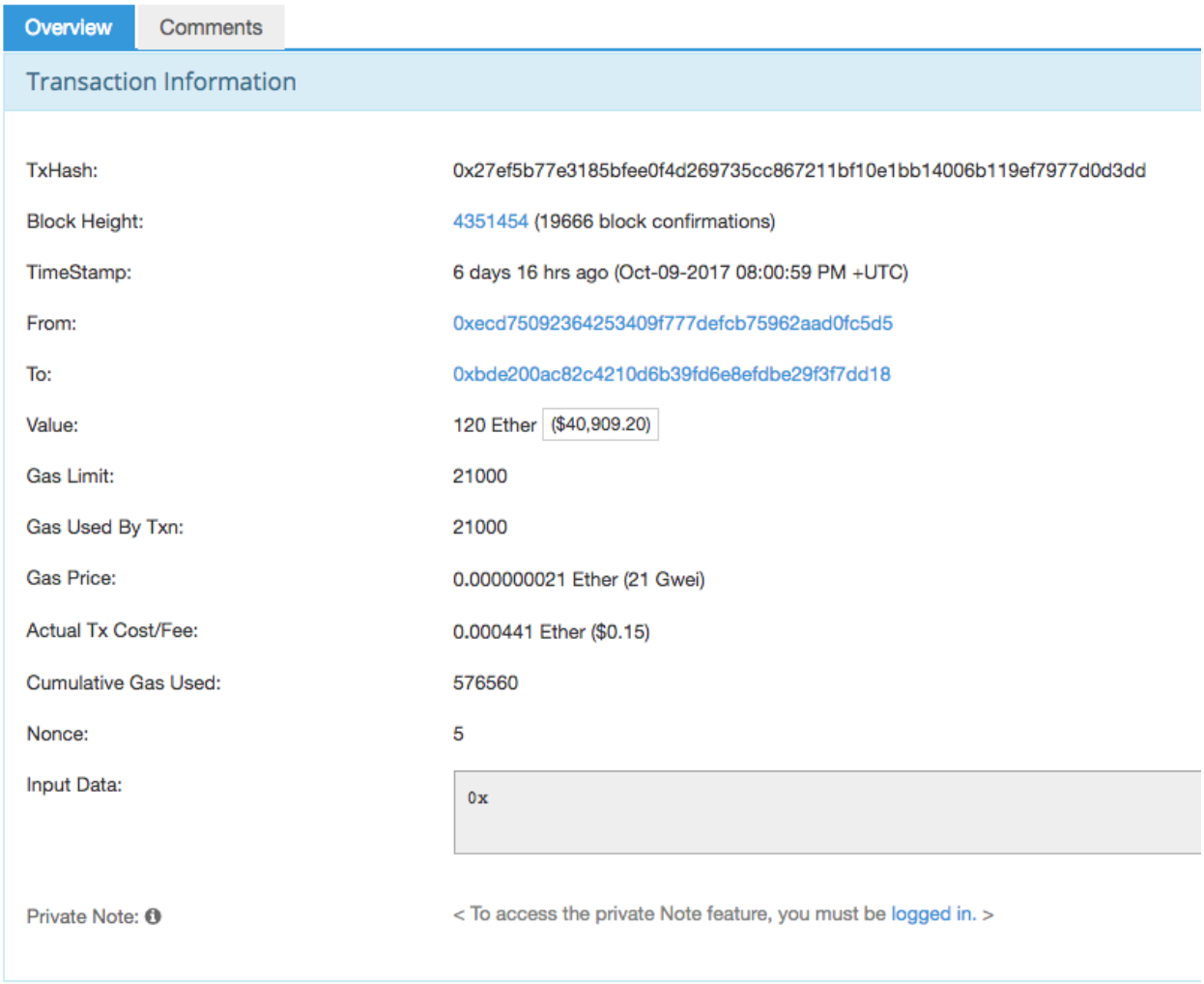
Ethereum is an open software platform based on blockchain technology that enables developers to build and deploy decentralized applications. Ethereum introduced its own digital currency, called Ether. Ether is largely known today as cryptocurrency or a “token”. This is important, because whenever developers execute a smart contract or transact using the Ethereum blockchain, they must include enough “gas”, aka ether to run the program. In other words, ether is what you pay when you program code on ethereum.

Fuel of the Blockchain

Ethereum blockchain is run by nodes that keep the blockchain state but also calculate new blocks. New blocks are needed to change Blockchain’s state e.g. move Ethereum from one account to another. Calculation of the new block is made by miners, to cover their effort transaction sender must pay a fee. Transaction fee depends on complexity of transaction sender wants to make, if it’s a regular “send Ether transaction” or more complex one like “create smart contract”. So the more complex transaction, the more Gas we need to pay for it’s execution on Blockchain. Main complexity factors are:

- operations performed by the smart contract's code e.g. arithmetical operations
- data that is stored on blockchain e.g. storing information in the smart contract or updating an amount of Ether on the account

Analysis of transaction in terms of Gas:



This transactions just sends Ether (120.0) from one account to the other. Following are the Gas related informations about this transaction:

- **Gas limit**—that's the maximum amount of Gas that user commits to the transaction. If transaction will need more Gas

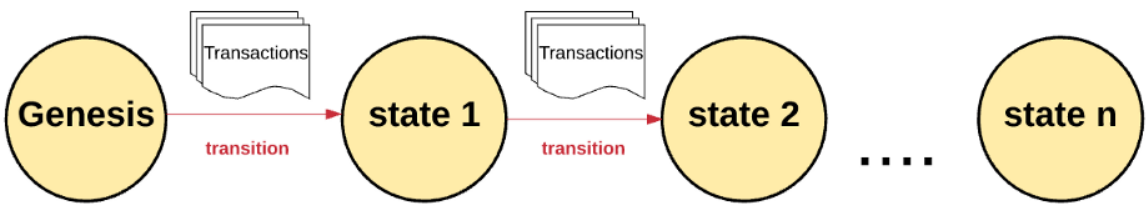
than it was defined in Gas limit, transaction will fail with “out of Gas” status (if your transaction goes out-of-gas you pay for it, even if it failed). Think about it as credit card blocking

- **Gas used by transaction**—that’s actual amount of Gas that was used during execution. Sometimes it’s hard to predict how much Gas transaction will cost, so the actual cost of transaction is computed afterwards. Sender is charged for used Gas and the difference is returned to the sender.
- **Gas price**—that’s the Gas price in ETH that sender defined at transaction creation.
- **Actual Tx Cost**—gas used by transaction * gas price (in ETH)
- **Cumulative gas used**—The total amount of gas used when this transaction was executed in the block (gas used by previous transactions and this one together)

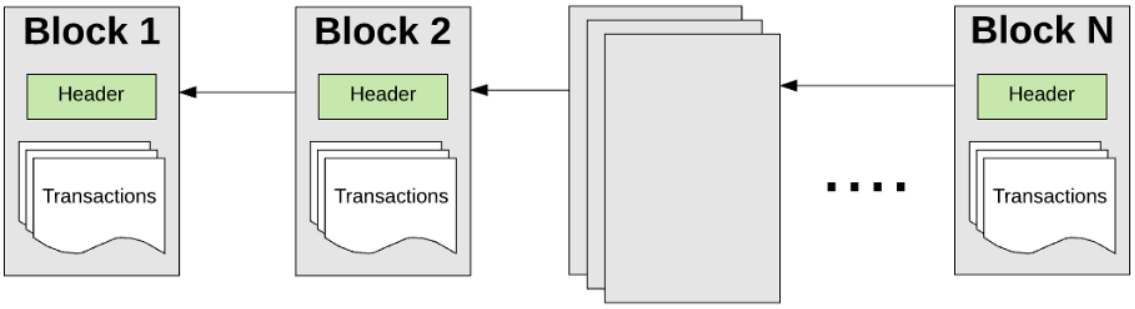
Those where transaction related information regarding Gas and its usage. Each block on the blockchain may contain one or more transaction. Blocks have their gas properties:

- **Gas Limit**—maximum amount of Gas that might be used for all transactions. Sum of all transactions’ Gas limit values.
- **Gas Used**—total amount of Gas used during execution of all transactions

States of Ethereum Blockchain



The state of Ethereum has millions of transactions. These transactions are grouped into “blocks”. A block contains a series of transactions, and each block is chained together with its previous block.



Proof of Work

To cause a transition from one state to the next, a transaction must be valid. For a transaction to be considered valid, it must go through a validation process known as **mining.** Mining is when a group of nodes (i.e. computers) expend their compute resources to create a block of valid transactions.

Any node on the network that declares itself as a miner can attempt to create and validate a block. Lots of miners from around the world try to create and validate blocks at the same time. Each miner provides a mathematical “proof” when submitting a block to the blockchain, and this proof acts as a guarantee: if the proof exists, the block must be valid.

For a block to be added to the main blockchain, the miner must prove it faster than any other competitor miner. The process of validating each block by having a miner provide a mathematical proof is known as a **“proof of work”.**

A miner who validates a new block is rewarded with a certain amount of value for doing this work. What is that value? The Ethereum blockchain uses an intrinsic digital token called “Ether.” Every time a miner proves a block, new Ether tokens are generated and awarded.

Smart contract

Smart contract is just a phrase used to describe computer code that can facilitate the exchange of money, content, property, shares, or anything of value. When running on the blockchain a smart contract becomes like a self-operating computer program that automatically executes when specific conditions are met. Because smart contracts run on the blockchain, they run exactly as programmed without any possibility of censorship, downtime, fraud or third party interference.

“It is a special kind of the blockchain account, that can not only keep Ether but also computer program with it’s state”

The Ethereum Virtual Machine

The Ethereum Virtual Machine (EVM) is a Turing complete software that runs on the Ethereum network. It enables anyone to run any program, regardless of the programming language given enough time and memory. The Ethereum Virtual Machine makes the process of creating blockchain applications much easier and efficient than ever before. Instead of having to build an entirely original blockchain for each new application, Ethereum enables the development of potentially thousands of different applications all on one platform.

Benefits of Ethereum decentralized Platform

Because decentralized applications run on the blockchain, they benefit from all of its properties.

- **Immutability** – A third party cannot make any changes to data.
- **Corruption & tamper proof** – Apps are based on a network formed around the principle of consensus, making censorship impossible.
- **Secure** – With no central point of failure and secured using cryptography, applications are well protected against hacking attacks and fraudulent activities.
- **Zero downtime** – Apps never go down and can never be switched off.

Solidity – The Language for smart contracts

Solidity is a contract-oriented, high-level language for implementing smart contracts. It was influenced by C++, Python and JavaScript and is designed to target the Ethereum Virtual Machine (EVM).
Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features.

*The entire documentation can be found on
-https://solidity.readthedocs.io/en/v0.4.24/index.html

Blockchain Use Case in Healthcare

What is the problem?

When it comes to patient data management, there are two main issues in the healthcare industry. First, each patient is unique therefore there is no such thing as a common disease or common treatment strategy. What works on a patient might not work on the other due to inter-individual variability. Hence, access to complete medical records is essential in order to adapt the treatment and provide personalised care. Healthcare is

becoming more-and-more patient-centred. Second, sharing information among the medical community is a major challenge.

Still today, doctors use social networks to communicate and share patient data. This type of medical data is sensitive and should always go through secured networks when divulgated. Moreover, the lack of secure structure to share data is an important obstacle for scientific advances. Indeed, medical records are kept in very different locations, and there is no common database. Allowing the researchers to access the data could heavily contribute to scientific advances worldwide especially when it comes to rare diseases or minorities.

Finally, blockchain also addresses the notion of data ownership. Today, the patient cannot claim full ownership of his medical records because giving him complete control over the information would also allow him to change certain information or even delete parts of it. This could have repercussions on both his health and public health. The downside of not letting the patient own his data is that he doesn't always have control over who is using and sharing it.

How can Blockchain help?

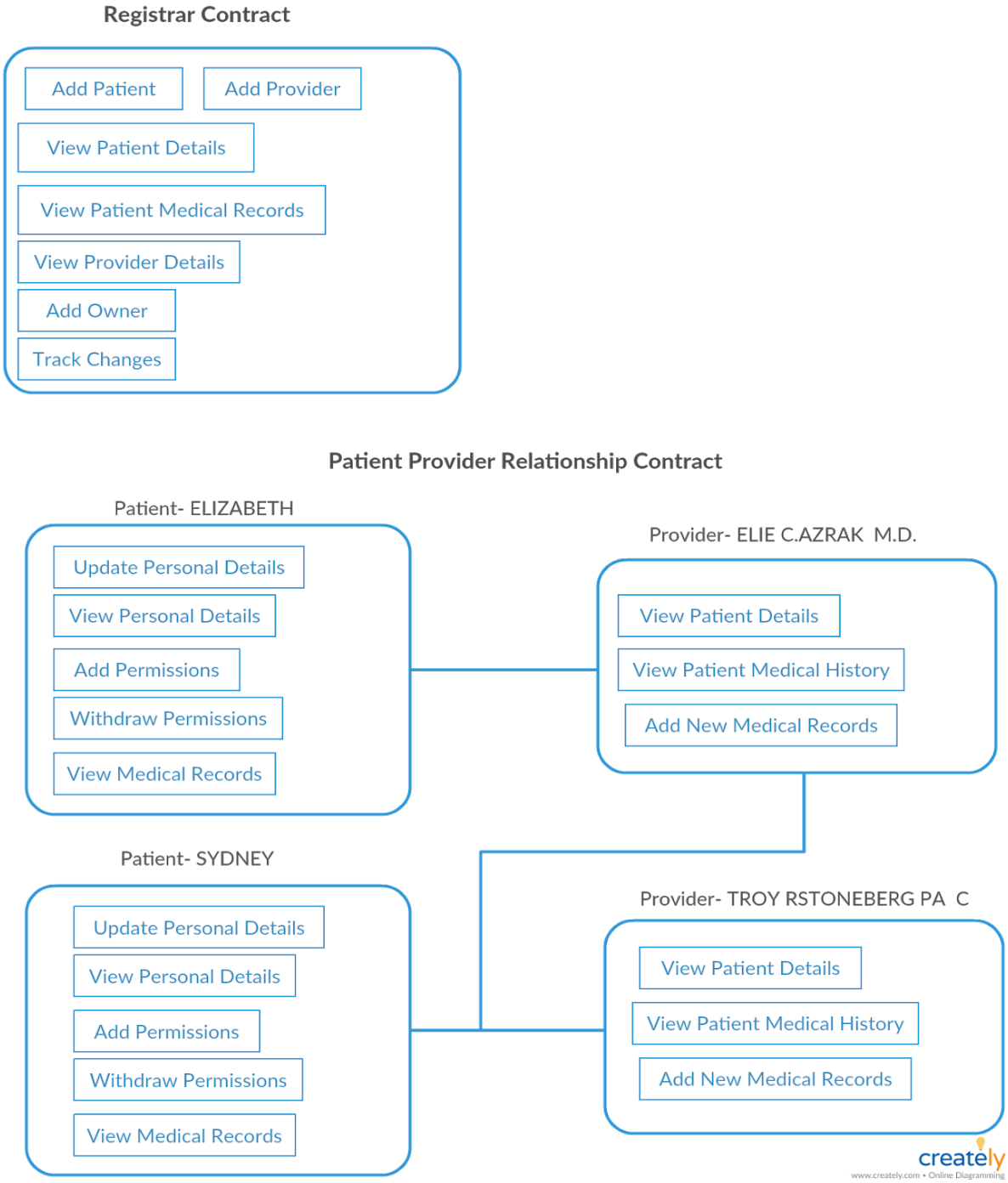
Blockchain can provide a structure for data sharing as well as security. This is how: healthcare providers collect information from the patient such as name, date of birth, procedures performed and prescriptions. The data is stored in the organisation's existing databases and/or on cloud computing systems. A hash is created from each source of data and is redirected to the blockchain along with the patient's public ID. Smart contracts are used to manage patient data access.

Through an API, healthcare stakeholders can query the blockchain that provides the location where the data can be found without revealing patient identity.

If needed, the patient can share his full medical record (with or without identifiable data) to any stakeholder. The patient can decide to whom he gives access to and on which conditions.

One of the main advantages of this technology, is that it allows the patient to control the access he gives to his medical records. The patient defines through a smart contract the condition on which his data can be accessed on the blockchain. In fact, all this will be done through an API and the patient will set the conditions on his profile.

Implementation



Registrar Contract (RC)

Add Patient

```
68  function addpatient(address publickey,string _fname,string _lname,uint _zipcode,uint _phoneno,uint _ssn) onlyOwner public returns(uint)
69  {
70      var a = pdetails[publickey];
71      require(keccak256("created")!= keccak256(a.stat));
72      a.stat="created";
73      a.fname = _fname;
74      a.lname = _lname;
75      a.zipcode = _zipcode;
76      a.phoneno = _phoneno;
77      a.ssn = _ssn;
78      ++pidcounter;
79      a.mid=pidcounter;
80      fetchaddr[pidcounter]=publickey;
81      return pidcounter;
82  }
```

The Registrar can create a new patient account after checking that it isn't already created and assigns a patient ID to the account along with all the other credentials.After creation of account the pid counter(no. of patients) gets incremented by one and also the created account address gets mapped to the assigned pid so that the account can be accessed directly by its id.

Add Provider

```
211
212  function addprovider(address publickey,string _pname,uint _pid) onlyOwner public
213  {
214      var t=prodetails[publickey];
215        t.pname=_pname;
216        t.pid=_pid;
217        fetchpaddr[_pid]=publickey;
218
219  }
220
```

The Registrar can create a new provider account and assign a provider ID to it along with all other credentials.The created account address gets mapped to the assigned pid so that the account can be accessed directly by its id.

View Patient Details

```
124     function oviewdetails(uint _mid) onlyOwner public constant returns(string,string,uint,uint,uint)
125     {
126
127       address ad=fetchaddr[_mid];
128       var a=pdetails[ad];
129       return (a.fname,a.lname,a.zipcode,a.phoneno,a.ssn);
130
131     }
132
133
```

The Registrar can fetch the patient details using the patient id.This function uses the modifier onlyOwner(defined in the contract itself),so it reverts any request which comes from a non-owner account.

View Patient Medical Records

```
191
192     function oviewmedicalhistory(uint _mid,uint _index) onlyOwner public constant returns(string,uint,string,uint,uint,string,uint,uint,string)
193     {
194       address ad=fetchaddr[_mid];
195       var a=pmhistory[ad][_index];
196       return (a.provider,a.providerid,a.claimtype,a.cfs_id,a.age,a.adjd_date,a.cca_nbr,a.claimamount,a.description);
197     }
198
```

The Registrar can fetch the patient medical records using the patient id.This function uses the modifier onlyOwner,so it reverts any request which comes from a non-owner account.

View Provider Details

```
220
221     function viewprovider(uint _pid) onlyOwner public returns(string,uint)
222     {
223         var ad=fetchpaddr[_pid];
224         return(prodetails[ad].pname,prodetails[ad].pid);
225     }
226
```

The Registrar can fetch the provider details using the provider id.This function uses the modifier onlyOwner,so it reverts any request which comes from a non-owner account.

Add Owner

```
19
20      function addOwnership(address newOwner) onlyOwner public {
21        owner1 = newOwner;
22      }
23  }
24
```

The existing owner can provide ownership rights of the contract to a new account and so both the parties have the same privileges to function as a registrar.

Track Changes

```
1   function _trackchanges()
2   {
3       $("#tb2 tr").remove();
4       document.getElementById("d1").style.display = 'none';
5               instancecreator();
6               console.log(x);
7               var i;
8               var table = document.getElementById("tb2");
9
10              var a="";
11              var b="";
12              var j=0;
13              for (i = x; i >= 2; i--) {
14                  var c = myInstance.oviewdetails.call($("#cid").val(), i);
15                  if (c[0] != "" && c[1] != "" && c[2] != "" && c[3] != [] &&
16                      !(new String(c[2]).valueOf() == new String(a).valueOf() &&
17                      new String(c[3]).valueOf() == new String(b).valueOf()))
18                  {
19                      console.log(c);
20                      var row = table.insertRow(j);
21                      var cell1 = row.insertCell(0);
22                      var cell2 = row.insertCell(1);
23                       a=c[2];
24                       b=c[3];
25                      cell1.innerHTML = a;
26                      cell2.innerHTML = b;
27
28                      j++;
29                  }
30              }
31                      var header = table.createTHead();
32                      var row = header.insertRow(0);
33                      var cel1 = row.insertCell(0);
34                      var cel2 = row.insertCell(1);
35                      cel1.innerHTML = "<b>ZIPCODE</b>"
36                      cel2.innerHTML = "<b>PHONE NO.</b>"
37              document.getElementById("t2").style.display='block';
38  }
39
```

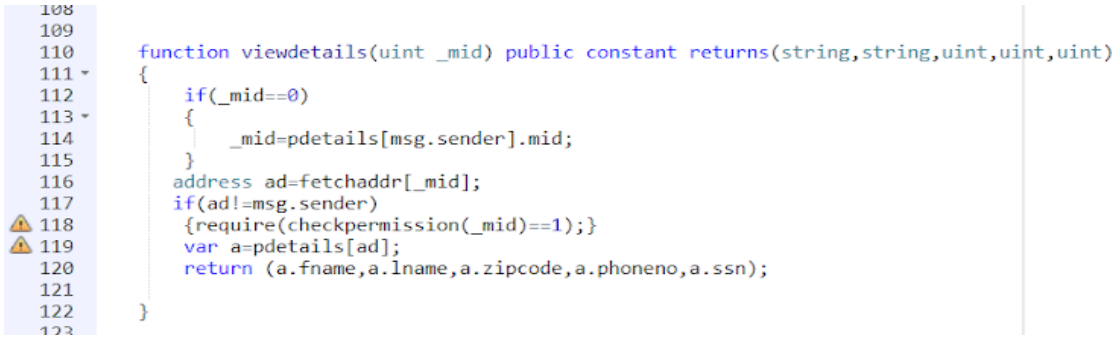
Since blockchain is immutable therefore all the updates are made in the new blocks.This property can be exploited to backtrack all the changes made by a user by iterating over each block and checking whether changes corresponding to a specified user are present in this block.

This functionality can be implemented by using the oviewdetails() function.This is done by calling it with a block number(variable i) successively while iterating over the blockchain.

Patient Provider Relationship Contract

- **Patient**

View Personal Details

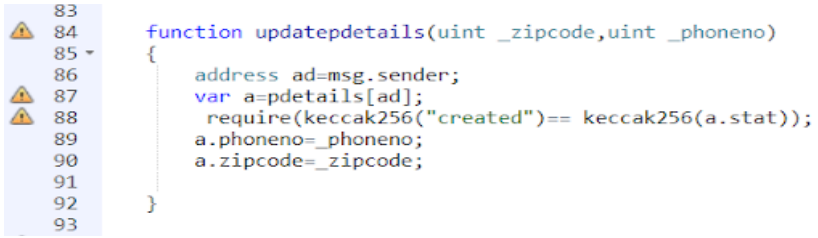


```
function viewdetails(uint _mid) public constant returns(string,string,uint,uint,uint)
{
    if(_mid==0)
    {
        _mid=pdetails[msg.sender].mid;
    }
   address ad=fetchaddr[_mid];
   if(ad!=msg.sender)
    {require(checkpermission(_mid)==1);}
    var a=pdetails[ad];
    return (a.fname,a.lname,a.zipcode,a.phoneno,a.ssn);

}
```

The patient can view his/her personal details using the patient id.The function is called every time with mid(member id) equal to 0 which is used to signify that the function is being called by the patient itself.Every time this condition is met,the pid of the patient(msg.sender) is fetched using the pdetails mapping and it is used to view the patient details.

Update Personal Details

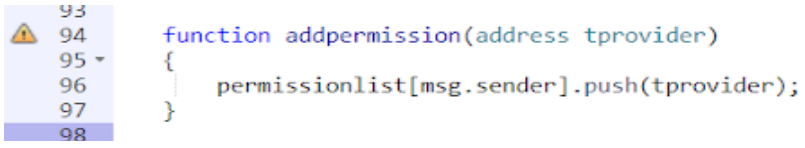


```
function updatepdetails(uint _zipcode,uint _phoneno)
{
    address ad=msg.sender;
    var a=pdetails[ad];
     require(keccak256("created")== keccak256(a.stat));
    a.phoneno=_phoneno;
    a.zipcode=_zipcode;

}
```

The patient can update his or her details once the account is created.

Add Permissions

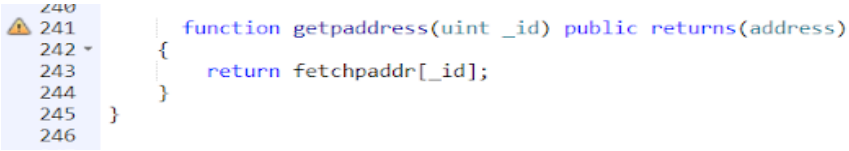


```
function addpermission(address tprovider)
{
    permissionlist[msg.sender].push(tprovider);
}
```

It allows the patient to control the access he gives to his medical records by granting permissions to various providers.A list is maintained corresponding to every patient account which is used to store the account

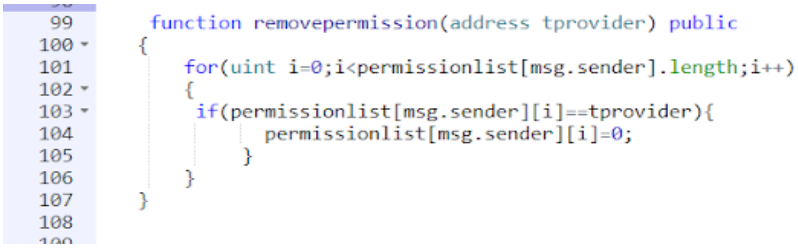
address of the providers which are allowed to access the patient personal details and their medical records.

Withdraw Permissions



```
241     function getpaddress(uint _id) public returns(address)
242     {
243         return fetchpaddr[_id];
244     }
245 }
246
```

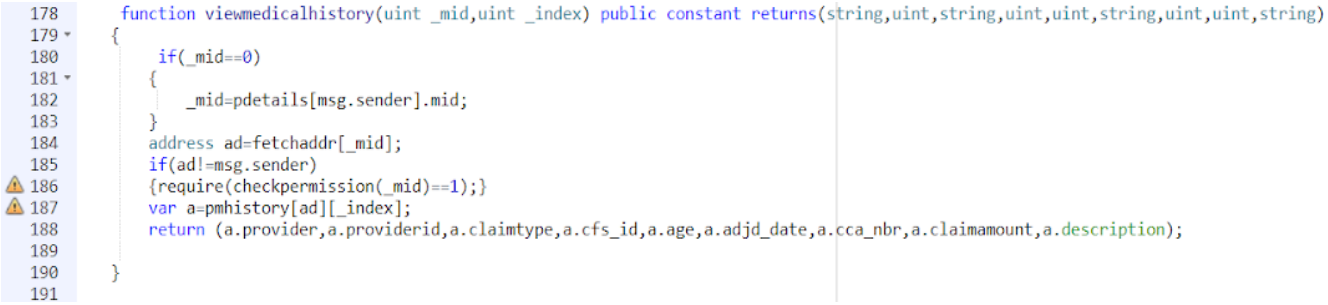
The patient can fetch the address of any provider using the provider id.



```
99     function removepermission(address tprovider) public
100    {
101        for(uint i=0;i<permissionlist[msg.sender].length;i++)
102        {
103         if(permissionlist[msg.sender][i]==tprovider){
104             permissionlist[msg.sender][i]=0;
105           }
106        }
107    }
108
```

The above obtained address can be used to remove the access permissions for a particular provider.

View Medical Records



```
178    function viewmedicalhistory(uint _mid,uint _index) public constant returns(string,uint,string,uint,uint,string,uint,uint,string)
179    {
180        if(_mid==0)
181        {
182            _mid=pdetails[msg.sender].mid;
183        }
184        address ad=fetchaddr[_mid];
185        if(ad!=msg.sender)
186        {require(checkpermission(_mid)==1);}
187        var a=pmhistory[ad][_index];
188        return (a.provider,a.providerid,a.claimtype,a.cfs_id,a.age,a.adjd_date,a.cca_nbr,a.claimamount,a.description);
189
190    }
191
```

The patient can view his/her medical records using the patient id.The function is called every time with mid(member id) equal to 0 which is used to signify that the function is being called by the patient itself.Every time this condition is met,the pid of the patient(msg.sender) is fetched using the pdetails mapping and it is used to view the patient details.

- **Provider**

View Patient Details

```
108
109
110     function viewdetails(uint _mid) public constant returns(string,string,uint,uint,uint)
111     {
112         if(_mid==0)
113         {
114             _mid=pdetails[msg.sender].mid;
115         }
116        address ad=fetchaddr[_mid];
117        if(ad!=msg.sender)
118         {require(checkpermission(_mid)==1);}
119         var a=pdetails[ad];
120         return (a.fname,a.lname,a.zipcode,a.phoneno,a.ssn);
121
122     }
123
```

The provider can view patient details using the patient id.Everytime the function is called, it checks for permissions in the corresponding patient's list by using the checkpermission() to determine whether the given provider has access rights or not.

View Patient Medical History

```
178     function viewmedicalhistory(uint _mid,uint _index) public constant returns(string,uint,string,uint,uint,string,uint,uint,string)
179     {
180         if(_mid==0)
181         {
182             _mid=pdetails[msg.sender].mid;
183         }
184         address ad=fetchaddr[_mid];
185         if(ad!=msg.sender)
186         {require(checkpermission(_mid)==1);}
187         var a=pmhistory[ad][_index];
188         return (a.provider,a.providerid,a.claimtype,a.cfs_id,a.age,a.adjd_date,a.cca_nbr,a.claimamount,a.description);
189
190     }
191
```

The provider can view patient medical records using the patient id.Everytime the function is called, it checks for permissions in the corresponding patient's list by using the checkpermission() to determine whether the given provider has access rights or not.

Add New Medical Records

```
function _updatemedicalhistory(uint _mid,string _claimtype,uint _cfs_id,string _adjd_date,uint _age,uint _cca_nbr,string _diseaseid,uint _claimamount,string _description)
        address ad=fetchaddr[_mid];
        require(checkpermission(_mid)==1);

        mhistory a;
        a.mid=_mid;
        a.provider=prodetails[msg.sender].pname;
        a.providerid=prodetails[msg.sender].pid;
        a.provideraddr=msg.sender;
        a.claimtype=_claimtype;
        a.cfs_id=_cfs_id;
        a.age=_age;
        a.adjd_date=_adjd_date;
        a.cca_nbr=_cca_nbr;
        a.diseaseid=_diseaseid;
        a.claimamount=_claimamount;
        a.description=_description;
       pmhistory[ad].push(a);

    }
```

The provider can add new medical records of a patient who have given permission to them.A list of medical records is maintained for each patient and the new records are inserted into it.

Deploying the Smart Contract :

1. Compiling the smart contract Using Remix IDE.

Remix is a powerful, open source tool that helps you write Solidity contracts straight from the browser. Written in Javascript, Remix supports both usage in the browser or locally.
Remix also supports testing, debugging and deploying of smart contracts and much more.

2.Creating a blockchain network on the local system(using Ganache)

Ganache is a personal blockchain for Ethereum development you can use to deploy contracts, develop your applications, and run tests. The Ganache screen will show some details about the server, and also list out a number of accounts. Each account is given 100 ether.By default it hosts the RPC server at HTTP://127.0.0.1:7545

3.Deploying the contract using MyEtherWallet

It is an interface for deploying and interacting with the contract.For deploying the contract ,it asks for contract's Bytecode and the private key of the account through which to deploy the contract .The bytecode can be obtained by viewing the details of the contract in the remix IDE.

4.Interacting with the contract

For interacting with the contract ,MyEtherWallet asks for the contract address and the ABI of the contract.The ABI can be obtained in the remix IDE and the contract address can be seen in the transactions log in Ganache.

Creating web interface for interacting with the contract using Web3.js

Web3.js

This is the Ethereum compatible JavaScript API which implements the Generic JSON RPC spec. It's available on npm as a node module.

1. Set a provider using RPC server of Ganache.

```
if (typeof web3 !== 'undefined') {
    web3 = new Web3(web3.currentProvider);
 } else {
    web3 = new Web3(new
Web3.providers.HttpProvider("http://localhost:7545"));
}
```

2. Defining the account address and the ABI

web3.eth.defaultAccount = account; //account= account through which we have to interact with the contract.

var abi = [contract abi here];

3. Creating an object of the contract

var myContract = web3.eth.contract(abi);

4. Creating an instance of the deployed contract using contract object

myInstance = myContract.at(-- contract address --);

5. Calling the functions of the contract

a)Functions which do not cause any transaction

Syntax - myContractInstance.myMethod.call(param1 [, param2, ...] [, transactionObject] [, defaultBlock] [, callback]);

Parameters

- String|Number|BigNumber - (optional) Zero or more parameters of the function. If passing in a string, it must be formatted as a hex number, e.g. "0xdeadbeef" If you have already created BigNumber object, then you can just pass it too.
- Object - (optional) The (previous) last parameter can be a transaction object, see web3.eth.sendTransaction parameter 1 for more.
- Number|String - (optional) If you pass this parameter it will not use the default block set with web3.eth.defaultBlock.
- Function - (optional) If you pass a callback as the last parameter the HTTP request is made asynchronous.

Return

- The data values of the different variables.

b)Functions that cause transaction

Syntax - myContractInstance.myMethod.(param1 [, param2, ...] [, transactionObject] [, defaultBlock] [, callback]);

Return

- Transaction hash

Web3.js documentation - https://github.com/ethereum/wiki/wiki/JavaScript-API

Private Key Validation

```
2  app.get('/validate', function (req, res) {
3      const priv = req.param('pkey');
4      const rsv = eutil.ecsign(eutil.sha256("message"), eutil.toBuffer(priv));
5      const pubKey = eutil.ecrecover(eutil.sha256("message"), rsv.v, rsv.r, rsv.s);
6      const addrBuf = eutil.pubToAddress(pubKey);
7      const addr = eutil.bufferToHex(addrBuf);
8      console.log(addr);
9      res.send(addr);
10 });
11
```

When any of the stakeholders try to interact with the contract using the web interface,the private key of the user is used to retrieve the account address and matched against the user's account address to validate the user.If both the addresses leads to a match then the user is logged in successfully else results in warning message.

- The above lines of code uses ethereumjs-util module.
- Sign a message using the function ecsign() passing the hash of the message and the private key.
- Recover the public key using the function ecrecover() passing the hash of the message and the signature object.
- The public key is converted to the account address using the pubtoAddress() function.
- The obtained address is matched against the user's account address.

Scalability problem

The main scalability problems in the ethereum decentralized applications can be categorized as:

- The time taken to put a transaction in the block.
- The time taken to reach a consensus.

The time taken to put a transaction in the block

In ethereum, a transaction goes through when a miner puts the transaction data in the blocks that they have mined. So suppose A wants to send 4 ETH to B, she will send this transaction data to the miners, the miner will then put it in their block and the transaction will be deemed complete.
However, as ethereum becomes more and more popular, this becomes more time-consuming. Plus, there is also the small matter of transactions fees. You see, when miners mine a block, they become temporary dictators of that block. If you want your transactions to go through, you will have to pay a toll to the miner in charge. This “toll” is called transaction fees.
The higher the transaction fees, the faster the miners will put them up in their block. While this is okay for people who have a huge repository of bitcoins, it might not be the most financially viable options.
If you pay the lowest possible transaction fees, then you will have to wait for a median time of 13 mins for your transaction to go through.
More often than not, the transactions had to wait until a new block was mined (which is 10 mins in bitcoin), because the older blocks would fill up with transactions. Bitcoin has a size limit of 1 mb which severely limits its transaction carrying capacity.

What about Ethereum?

Theoretically speaking, Ethereum is supposed to process 1000 transactions per second. However, in practice, Ethereum is limited by 6.7 million gas limit on each block.

What does this mean for Blockchain scalability?

Since each block has a gas limit, the miners can only add transactions whose gas requirements add up to something which is equal to or less than the gas limit of the block.Therefore number of transactions going through is limited.

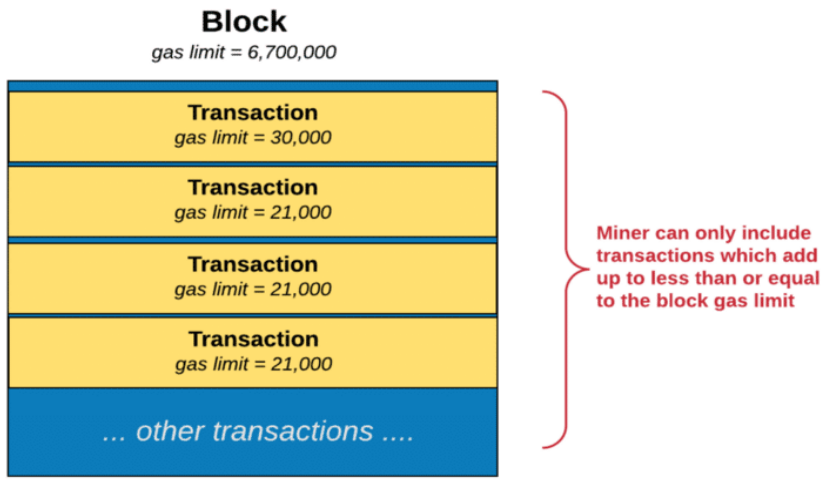


Image courtesy: Hackernoon

The Time Taken To Reach A Consensus

Currently, ethereum is structured as a peer-to-peer network. The participants, aka the nodes, are not given any extra special privileges. The idea is to create an egalitarian network. There is no central authority and nor is there any hierarchy. It is a flat topology.
Now, if there is no central entity, how would everyone in the system get to know that a certain transaction has happened? Suppose A sent 3 ETH to B. The nodes nearest to A will get to know of this, and then they will tell the nodes closest to them, and then they will tell their neighbors, and this will keep on spreading out until everyone knows.
Remember, the nodes follow a trustless system. What this means is, just because node A says that a transaction is valid doesn't mean that node B will believe it to be so. Node B will do their own set of calculations to see whether the transaction is actually valid or not. This means, that every node

must have their own copy of the blockchain to help them do so. As you can imagine, this makes the whole process very slow.
The problem is, that unlike other pieces of technology, the more the number of nodes increases in a cryptocurrency network, the slower the whole process becomes. Consensus happens in a linear manner, meaning, suppose there are 3 nodes A, B and C.For consensus to occur, first A would do the calculations and verify and then B will do the same and then C.However, if there is a new node in the system called “D”, that would add one more node to the consensus system, which will increase the overall time period. As cryptocurrencies has become more popular, the transaction times have gotten slower.

Solutions to the Blockchain scalability issues

Ethereum has come up with a host of solutions which have either already been or are going to be implemented.

- Block Size Increase.
- Sharding.
- Proof Of Stake

Block size Increase

The main problem of ethereum has been the limited blocksize.
Implementation of this has been anything but in fact, this has given birth to a lot of debate in the Ethereum community with sides passionately arguing both for and against the block size increase.
Arguments against block size increase -

- It will cause increased centralization: Since the network size will increase, the amount of processing power required to mine will increase as well. This will take out all the small mining pools and give mining powers exclusively to the large scale pools. This will in turn increase centralization which goes against the very essence of bitcoins.

- Firstly, the main thing that is hindering Ethereum's scalability is the speed of consensus among nodes. Increasing the block size will still not solve this problem. In fact, as the number of transactions per block increases, the number of calculations and verifications per node will increase as well.

Arguments for the block size increase -

- Increased block size will mean increase transactions per block which will in turn decrease the amount of per transaction fees and hence the cost for running the decentralised application will be low.

Proof Of Stake

One of biggest things happening right now is Ethereum's shift from proof of work to proof of stake.

- Proof of work: This is the protocol that most cryptocurrencies like Ethereum and Bitcoin have been following so far. This means that miners "mine" cryptocurrencies by solving crypto-puzzles using dedicated hardware.
- Proof of stake: This protocol will make the entire mining process virtual. In this system we have validators instead of miners. The way it works is that as a validator, you will first have to lock up some of your ether as stake. After doing that you will then start validating blocks which basically means that if you see any blocks that you think can be appended to the blockchain, you can validate it by placing a bet on it. When and if, the block gets appended, you will get a reward proportional to the stake you have invested. If, however, you bet on the wrong or the malicious block, the stake that you have invested will be taken away from you.

How does this help in Blockchain scalability?

Makes 51% attack harder: 51% attack happens when a group of miners gain more than 50% of the world's hashing power. Using proof of stake negates this attack.

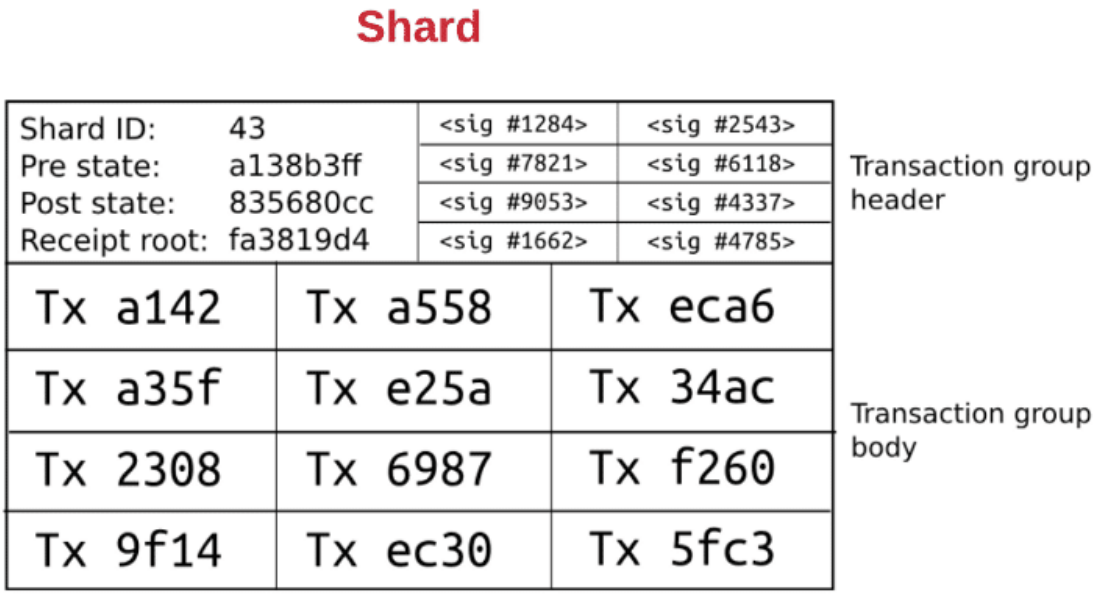
- Malicious-free validators: Any validator who has their funds locked up in the blockchain would make sure that they are not adding any wrong or malicious blocks to the chain, because that would mean their entire stake invested would be taken away from them.
- Block creation: Makes the creation of newer blocks and the entire process faster.Introducing proof-of-stake is going to make the blockchain a lot faster because it is much more simple to check who has the most stake then to see who has the most hashing power. This makes coming to a consensus much more simple.
- Scalability: Makes the blockchain scalable by introducing the concept of "sharding".Proof-of-stake makes the implementation of sharding easier. In a proof-of-work system it will be easier for an attacker to attack individual shards which may not have high hashrate.

Sharding

The biggest problem that Ethereum is facing is the speed of transaction verification. Each and every full node in the network has to download and save the entire blockchain. What sharding does is that it breaks down a transaction into shards and spreads it among the network. The nodes work on individual shards side-by-side. This in turn decreases the overall time taken.

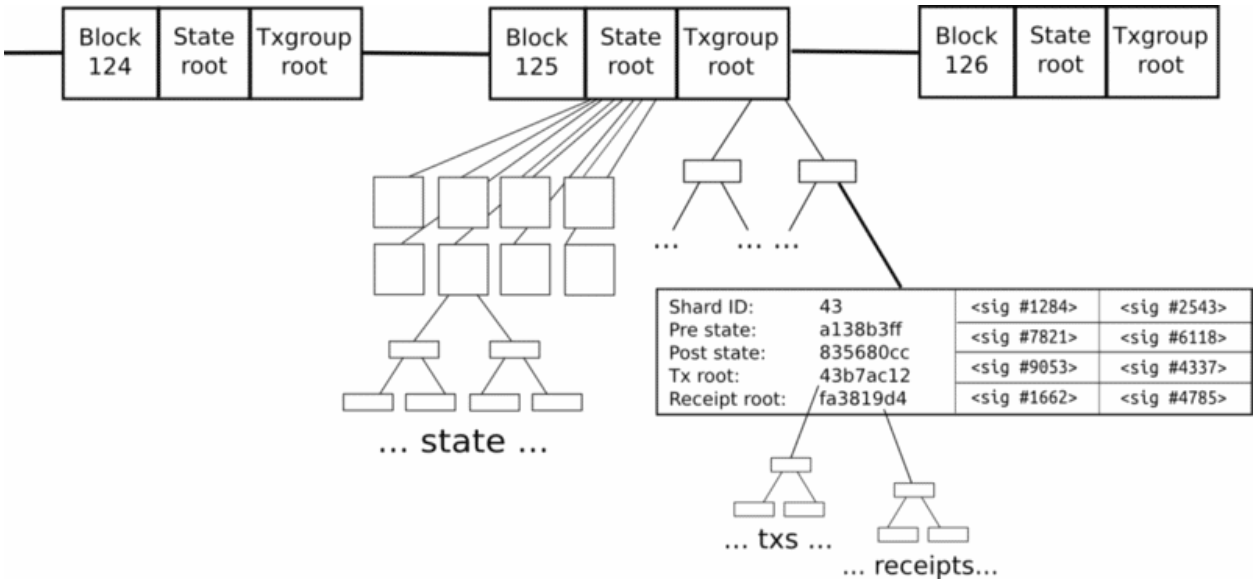
The transactions are arranged in a block to create one level of interaction and make the whole process scalable.Ethereum suggests that they change this into two levels of interaction to make it more scalable.

The First Level



The first level is the transaction group. Each shard has its own group of transaction.

The Second Level



There is the normal block chain, but now it contains two primary roots:

- The state root
- The transaction group root

The state root represents the entire state, and as we have seen before, the state is broken down into shards, which contain their own substates.
The transaction group root contains all the transaction groups inside that particular block.