PRACTICAL NO. 2

Cryptocurrency

Aim: To implement immutable public/private blockchain.

1. Concept of Bitcon:1

- Bitcoin is a decentralized digital currency, without a central bank or single administrator, that can be sent from user to user on the peer-to-peer bitcoin network without the need for intermediaries. Transactions are verified by network nodes through cryptography and recorded in a public distributed ledger called a blockchain_l[5]l
- The blockchain was invented by a person (or group of people) using the name Satoshi Nakamoto in 2008 to serve as the public transaction ledger of the cryptocurrency bitcoin.
- A blockchain is a decentralized, distributed, and oftentimes public, digital ledger consisting of records called blocks that is used to record transactions across many computers so that any involved block cannot be altered retroactively (with effect from a date in the past), without the alteration of all subsequent blocks.

2. Block: 1[2]1

- In a blockchain network, transactions are validated by a community of nodes and then recorded in a block. As shown in below figure a), a block is composed of a header and a body, in the latter of which the transaction data is stored.

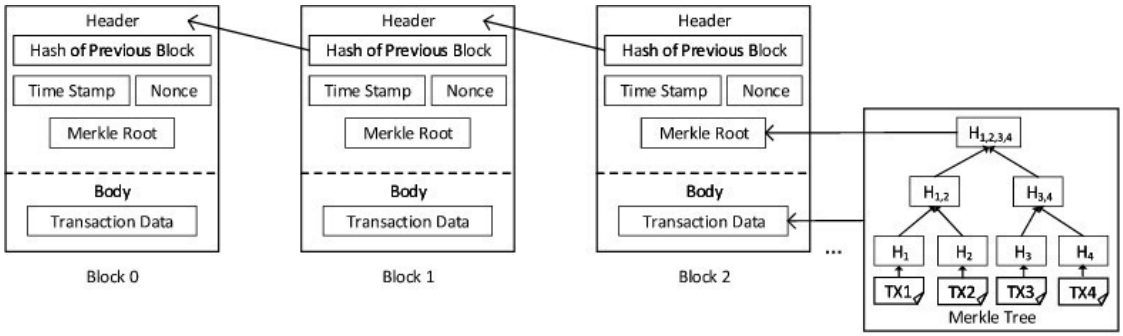


Fig. a) Blockchain Block Structure

- The block header contains the hash of the **previous block, a timestamp, Nonce** and the **Merkle root.**
- The **hash value** is calculated by passing the header of the previous block to a hash function. With the hash of the previous block stored in the current block, blockchain is thus growing with new blocks being created and linked to it. Moreover, this guarantees that tampering on the previous block will efficiently detected.
- The **timestamp** is to record the time when a block is created.
- **Nonce** is used in the creation and verification of a block.
- The **Merkle tree** is a binary tree with each leaf node labelled with the hash of one transaction stored in the block body, and the non-leaf nodes labelled with the concatenation of the hash of its child nodes.
- **Merkle root,** i.e., the root hash of a Merkle tree which is stored in the block body, and is used to reduce the efforts to verify the transactions in a block. Since a tiny change in one transaction can

produce a significantly different Merkle root, the verification can be completed by simply comparing the Merkle root instead of verifying all the transactions in the block.

- We denote a transaction as TX and take the 3-rd block, which only contains four transactions, as an example to illustrate the structure of a Merkle Tree.

3. Immutable Ledger: [3][1]

- Immutable ledger in blockchain refers to any records that have the ability to remain unchanged. It cannot be altered and hence the data cannot be changed with ease, thereby making sure that the security is quite tight. Immutability means that it is very difficult to make changes without collusion. The central idea behind the blockchain ledger is the security of data and the proof that data has not been changed or altered. Let us delve a bit deeper into the topic to understand more about blockchain immutability and the benefits it offers.

Understanding Blockchain Immutability:

- Immutability is defined as the ability of a blockchain ledger to remain unchanged, unaltered, and indelible. Each of the blocks of information like facts or transaction details is carried out with the help of a cryptographic principle or a hash value. Now, this hash value has an alphanumeric string generated by each block individually. Each of the blocks contains a hash value or digital signature for itself and for the previous one as well. This, in turn, makes sure that the blocks are retroactively coupled together and unrelenting. It is this functionality of blockchain technology that makes sure no one is able to interfere with the system or change the already saved data into the block.
- In this regard, it is also quite essential to know that blockchain is distributed and decentralized in nature. Here a consensus is made among the different storing a copy of the data. It is this consensus that makes sure the originality of data is righty maintained. Immutability is undoubtedly one of the most definitive features of blockchain technology and also brings out the best use cases of smart contracts that can be deployed. The concept can simply redefine the entire process of auditing of data to make it much more efficient, and cost-effective, along with bringing about more trust as well as integrity into the data.

How to Achieve Immutability?

- As explained above, the hash value helps in securing each block of code in a separate manner. To understand how to achieve immutability, clarification of the concept of cryptographic hashing is essential. Nowadays, the generation of a cryptographic is not quite a dreadful task. It is because of the fact that modern programming languages come with an array of hash functions. With the help of these hash functions, it is just required to pass a set of bytes and the function will be returning a checksum signature. These functions always generate a string of length of 64 characters and we would always be getting the fixed string length regardless of the size of the input, which is referred to as a digital signature.
- The digital signature points to the exact data that the users input. But hash cannot be reverse-engineered which means that the users cannot make use of this output string for the purpose of finding the input data. This, in turn, results in the immutability of the blockchain ledger. In this system, each of the transactions is verified with the help of a blockchain network. It includes blocks of information embedded with timestamps and is secured by a hashing process. It links together and incorporates the hash of the last block. This mechanism plays a major role in developing the chronological chain which helps in joining each of the blocks.
- The meta-data of the last block is always included by hashing at the time of generating a new hash for it. This, in turn, helps in creating a link between the block and the chain, thereby making it unbreakable. Once this is done, none can alter or delete the data of the block which is placed in the

blockchain. It is because whenever anyone would be attempting to make a change, the modification is rejected by the subsequent block since the hash of the block would not be valid anymore.

4. Private/ Public Blockchain: 141

- **A public blockchain** is one where anyone is free to join and participate in the core activities of the blockchain network. Anyone can read, write, and audit the ongoing activities on a public blockchain network, which helps achieve the self-governed, decentralized nature often touted when blockchain is discussed.
- In a public blockchain, anyone is free to join and participate in the core activities of the blockchain network.
- In a **private blockchain** network participants can join only through an invitation where their identity or other required information is authentic and verified. The validation is done by the network operator(s) or by a clearly defined set protocol implemented by the network through smart contracts or other automated approval methods.
- Private blockchains control who is allowed to participate in the network. If the network is capable of mining, its private nature could control which users can execute the consensus protocol that decides the mining rights and rewards. Additionally, only select users might maintain the shared ledger. The owner or operator has the right to override, edit, or delete the necessary entries on the blockchain as required or as they see fit.

References:

1. https://en.wikipedia.org/wiki/Blockchain
2. https://www.researchgate.net/figure/The-structure-of-a-Blockchain-A-block-is-composed-of-a-header-and-a-body-where-a-header_fig1_337306138
3. https://www.solulab.com/what-is-immutable-ledger-in-blockchain-and-its-benefits/
4. https://www.investoped ia.com/news/public-private-permissioned-blockchains-compared/
5. https://artsandculture.google.com/entity/bitcoin/m0Sp0rrx?hl=en

Exercise:

1. Write a program to create the chain with Genesis block and adding block into blockchain and validating the chain for any alteration. (Part-I)

BlockChain1.js

Code:

```
//Part 1
//javascript program to create the blockchain with genesis block
//adding the block into blockchain and validate the blockchain for any alteration
// install crypto-js

const SHA256 = require('crypto-js/sha256')

class Block {
   constructor(index, previousHash = '', timestamp, data) {
      this.index = index;
```

```
        this.previousHash = previousHash;
        this.timestamp = timestamp;
        this.data = data;
        this.hash = this.calculateHash();
    }
    calculateHash() {
        //return sha256
        return SHA256(this.index + this.previousHash + this.timestamp + JSON.stringify(this.data)).toString();
    }
}

class Blockchain {
    constructor() {
        this.chain = [this.createGenesisBlock()];
    }

    createGenesisBlock() {
        return new Block(0, "0", "06/08/24", "Genesis Block");
    }
    //latest block
    getLatestBlock() {
        return this.chain[this.chain.length - 1];
    }
    //add block
    addBlock(newBlock) {
        newBlock.previousHash = this.getLatestBlock().hash;
        newBlock.hash = newBlock.calculateHash();
        this.chain.push(newBlock);
    }
    //validate blockchain
    isChainValid() {
        for (let i = 1; i < this.chain.length; i++) {
            const currentBlock = this.chain[i];
            const previousBlock = this.chain[i - 1];

            if (currentBlock.hash !== currentBlock.calculateHash()) {
                return false;
            }
            if (currentBlock.previousHash !== previousBlock.hash) {
                return false;
```

```
        }


    }
    return true;
  }
}
let Coin = new Blockchain();
console.log(JSON.stringify(Coin, null, 4));


console.log("Adding Block ");
Coin.addBlock(new Block(1,"06/09/2024",{amount: 4000}));
Coin.addBlock(new Block(2,"07/09/2024",{amount: 1000}));
console.log(JSON.stringify(Coin, null, 4));
console.log('IS chain valid: '+ Coin.isChainValid());
Coin.chain[2].data = {amount:2000};
console.log('IS chain valid: '+ Coin.isChainValid());
```

Output:

```
PS D:\SyMCA Sem3\BlockChain\Prac_2> node "d:\SyMCA Sem3\BlockChain\Prac_2\BlockChain1.js"
{
    "chain": [
        {
            "index": 0,
            "previousHash": "0",
            "timestamp": "06/08/24",
            "data": "Genesis Block",
            "hash": "23d56565b7790a4ca2aace34397925ce29ddf2ead96395e86abee77a1734b908"
        }
    ]
}
Adding Block
{
    "chain": [
        {
            "index": 0,
            "previousHash": "0",
            "timestamp": "06/08/24",
            "data": "Genesis Block",
            "hash": "23d56565b7790a4ca2aace34397925ce29ddf2ead96395e86abee77a1734b908"
        },
        {
            "index": 1,
            "previousHash": "23d56565b7790a4ca2aace34397925ce29ddf2ead96395e86abee77a1734b908",
            "timestamp": {
                "amount": 4000
            },
            "hash": "2406b2506cbba4b1bf9b6b1046a54e69973023fc3adf331f6484cc3293e9fcad"
        },
        {
            "index": 2,
            "previousHash": "2406b2506cbba4b1bf9b6b1046a54e69973023fc3adf331f6484cc3293e9fcad",
            "timestamp": {
                "amount": 1000
            },
            "hash": "f032d93ded528ab98e3099c49648dadd8755e480014b50b933772b250af890aa"
        }
IS chain valid: true
IS chain valid: false
```

2.Write a program to implementing proof of work for blockchain. (Part-II)

BlockChain2.js

Code:

```
//Part 2
//javascript program to create the blockchain with genesis block
//adding the block into blockchain and validate the blockchain for any alteration
// install crypto-js
//proof of work

const SHA256 = require('crypto-js/sha256')

class Block {
    constructor(index, previousHash = '', timestamp, data) {
        this.index = index;
        this.previousHash = previousHash;
        this.timestamp = timestamp;
        this.data = data;
        this.hash = this.calculateHash();
        this.nonce = 0;

    }
    calculateHash() {
        //return sha256
        return SHA256(this.index + this.previousHash + this.timestamp + JSON.stringify(this.data) + this.nonce).toString();
    }
    //proof of work
    mineBlock(difficulty) {
        while (this.hash.substring(0, difficulty) !== Array(difficulty + 1).join("0")) {
            this.nonce++;
            this.hash = this.calculateHash();
        }
        console.log("Block mined: " + this.hash);
        console.log("Nonce: " + this.nonce);
    }

}

class Blockchain {
    constructor() {
        this.chain = [this.createGenesisBlock()];
        //set difficulty
        this.difficulty = 7;
    }


    createGenesisBlock() {
        return new Block(0, "0", "06/08/24", "Genesis Block");
    }
    //latest block
    getLatestBlock() {
        return this.chain[this.chain.length - 1];
    }
    //add block
    addBlock(newBlock) {
        newBlock.previousHash = this.getLatestBlock().hash;
        this.chain.push(newBlock);
        newBlock.mineBlock(this.difficulty);

    }
    //validate blockchain
    isChainValid() {
        for (let i = 1; i < this.chain.length; i++) {
            const currentBlock = this.chain[i];
            const previousBlock = this.chain[i - 1];

            if (currentBlock.hash !== currentBlock.calculateHash()) {
                return false;
            }
            if (currentBlock.previousHash !== previousBlock.hash) {
                return false;
            }
```

```
        }
        return true;
    }
}
let Coin = new Blockchain();
console.log(JSON.stringify(Coin, null, 4));

console.log("Adding Block ");
Coin.addBlock(new Block(1, "06/09/2024", { amount: 4000 }));
Coin.addBlock(new Block(2, "07/09/2024", { amount: 1000 }));

console.log(JSON.stringify(Coin, null, 4));
console.log('IS chain valid: ' + Coin.isChainValid());

Coin.chain[2].data = { amount: 2000 };
console.log('IS chain valid: ' + Coin.isChainValid());
```

Output:

```
PS D:\SyMCA Sem3\BlockChain\Prac_2> node "d:\SyMCA Sem3\BlockChain\Prac_2\BlockChain2.js"
{
    "chain": [
        {
            "index": 0,
            "previousHash": "0",
            "timestamp": "06/08/24",
            "data": "Genesis Block",
            "hash": "7bca51baa1e6cad79cdb1f700a96ce81388c0a2f0a80109ed5438c6e4dcbaf9e",
            "nonce": 0
        }
    ],
    "difficulty": 6
}
Adding Block
Block mined: 000000a214b088966f312b6315d4827418cbefa1f4318cd00ce5148aabba4d6d
Nonce: 7078484
```

3.Write a program to add multiple transactions into block and give reward to miner for successful mining of block in blockchain. (Part-III)

BlockChain3.js

Code:

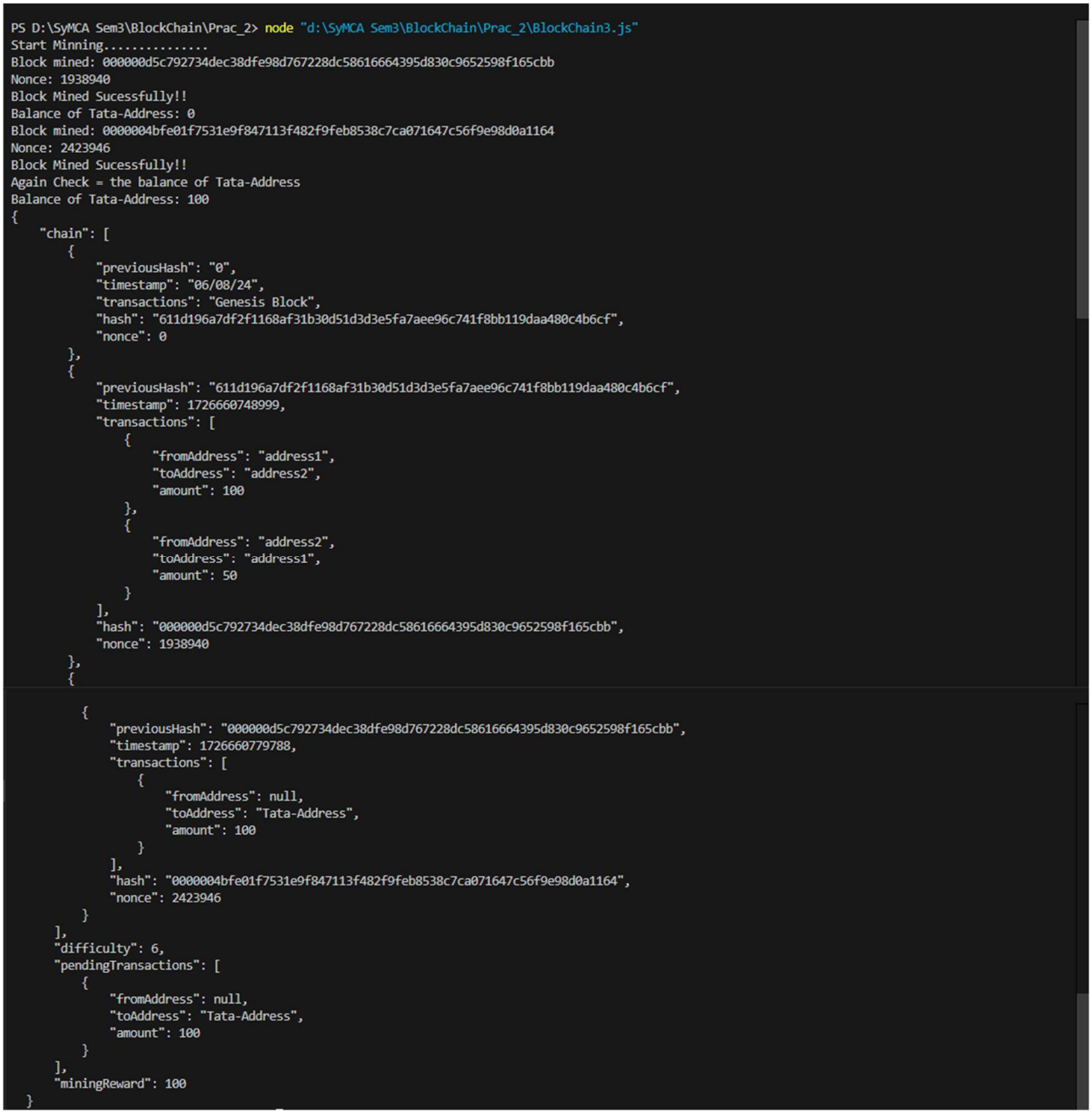
```
//Part 3

//javascript program to create the blockchain with genesis block

//adding the block into blockchain and validate the blockchain for any alteration

// install crypto-js

//proof of work

//multiple transaction in block and giving reward to minner who sucessfully solve the puzzle



const SHA256 = require('crypto-js/sha256')



class Transaction {

    constructor(fromAddress, toAddress, amount) {

        this.fromAddress = fromAddress;

        this.toAddress = toAddress;

        this.amount = amount;
```

```
    }

}
class Block {
    constructor(previousHash = '', timestamp, transactions) {
        this.previousHash = previousHash;
        this.timestamp = timestamp;
        this.transactions = transactions;
        this.hash = this.calculateHash();
        this.nonce = 0;

    }
    calculateHash() {
        //return sha256
        return SHA256(this.previousHash + this.timestamp + JSON.stringify(this.transactions) +
this.nonce).toString();
    }
    //proof of work
    mineBlock(difficulty) {
        while (this.hash.substring(0, difficulty) !== Array(difficulty + 1).join("0")) {
            this.nonce++;
            this.hash = this.calculateHash();
        }
        console.log("Block mined: " + this.hash);
        console.log("Nonce: " + this.nonce);
    }

}

class Blockchain {
    constructor() {
        this.chain = [this.createGenesisBlock()];
        //set difficulty
        this.difficulty = 6;
        this.pendingTransactions = [];
        this.miningReward = 100;
    }
```

```
createGenesisBlock() {
    return new Block("0", "06/08/24", "Genesis Block");
}
//latest block
getLatestBlock() {
    return this.chain[this.chain.length - 1];
}
//add mine pending transaction
minePendingTransaction(miningRewardAddress) {
    let block = new Block(this.getLatestBlock().hash, Date.now(), this.pendingTransactions)
    block.mineBlock(this.difficulty);
    this.chain.push(block);
    console.log("Block Mined Sucessfully!!");
    this.pendingTransactions = [new Transaction(null, miningRewardAddress, this.miningReward)];

}
//create transactions
createTransaction(transaction) {
    this.pendingTransactions.push(transaction);
}

//get balance of address
getBalanceOfAddress(address) {
    let balance = 0;
    for (const block of this.chain) {
        for (const trans of block.transactions) {
            if (trans.fromAddress == address) {
                balance -= trans.amount;
            }
            if (trans.toAddress == address) {
                balance += trans.amount;
            }
        }
    }
    return balance;
}


//validate blockchain
```

```
  isChainValid() {
    for (let i = 1; i < this.chain.length; i++) {
      const currentBlock = this.chain[i];
      const previousBlock = this.chain[i - 1];

      if (currentBlock.hash !== currentBlock.calculateHash()) {
        return false;
      }
      if (currentBlock.previousHash !== previousBlock.hash) {
        return false;
      }

    }
    return true;
  }
}
let Coin = new Blockchain();
Coin.createTransaction(new Transaction('address1', 'address2', 100));
Coin.createTransaction(new Transaction('address2', 'address1', 50));

console.log("Start Minning.....");

Coin.minePendingTransaction('Tata-Address');
console.log("Balance of Tata-Address: " + Coin.getBalanceOfAddress('Tata-Address'));
Coin.minePendingTransaction('Tata-Address');
console.log('Again Check = the balance of Tata-Address');
console.log("Balance of Tata-Address: " + Coin.getBalanceOfAddress('Tata-Address'));


console.log(JSON.stringify(Coin, null, 4));

// console.log('IS chain valid: ' + Coin.isChainValid());

// Coin.chain[2].data = { amount: 2000 };
// console.log('IS chain valid: ' + Coin.isChainValid());
```

Output:



```
PS D:\SyMCA Sem3\BlockChain\Prac_2> node "d:\SyMCA Sem3\BlockChain\Prac_2\BlockChain3.js"
Start Minning.....
Block mined: 000000d5c792734dec38dfe98d767228dc58616664395d830c9652598f165cbb
Nonce: 1938940
Block Mined Sucessfully!!
Balance of Tata-Address: 0
Block mined: 0000004bfe01f7531e9f847113f482f9feb8538c7ca071647c56f9e98d0a1164
Nonce: 2423946
Block Mined Sucessfully!!
Again Check = the balance of Tata-Address
Balance of Tata-Address: 100
{
    "chain": [
        {
            "previousHash": "0",
            "timestamp": "06/08/24",
            "transactions": "Genesis Block",
            "hash": "611d196a7df2f1168af31b30d51d3d3e5fa7aee96c741f8bb119daa480c4b6cf",
            "nonce": 0
        },
        {
            "previousHash": "611d196a7df2f1168af31b30d51d3d3e5fa7aee96c741f8bb119daa480c4b6cf",
            "timestamp": 1726660748999,
            "transactions": [
                {
                    "fromAddress": "address1",
                    "toAddress": "address2",
                    "amount": 100
                },
                {
                    "fromAddress": "address2",
                    "toAddress": "address1",
                    "amount": 50
                }
            ],
            "hash": "000000d5c792734dec38dfe98d767228dc58616664395d830c9652598f165cbb",
            "nonce": 1938940
        },
        {
            {
                "previousHash": "000000d5c792734dec38dfe98d767228dc58616664395d830c9652598f165cbb",
                "timestamp": 1726660779788,
                "transactions": [
                    {
                        "fromAddress": null,
                        "toAddress": "Tata-Address",
                        "amount": 100
                    }
                ],
                "hash": "0000004bfe01f7531e9f847113f482f9feb8538c7ca071647c56f9e98d0a1164",
                "nonce": 2423946
            }
        ],
        "difficulty": 6,
        "pendingTransactions": [
            {
                "fromAddress": null,
                "toAddress": "Tata-Address",
                "amount": 100
            }
        ],
        "miningReward": 100
    }
```

4.Write a program to sign the transaction with private key and verify the signed transactions for blockchain. (Part-IV)

KeyGenerator.js

Code:

```
//generate the keys
//private and public key
const EC = require('elliptic').ec;
const ec = new EC('secp256k1');
const key = ec.genKeyPair();
const publicKey = key.getPublic('hex');
```

```
const privateKey = key.getPrivate('hex');

console.log();

console.log("Public Key: ",publicKey);
console.log();
console.log("Private Key: ",privateKey);
```

Key.txt:

Keys of From :
Public Key:
04615e95fa13e28c3ca8312dc9a188c90ff64883249251cf0a69722d3a9eeae6530fd3216dd4b138e91748875a58fa4f87f616464a1
25ea43152c3ff27bb2ea01e
Private Key: 62a8423e598a3894ffa33e21f3c4c662645f2b2e4726f04313d67eb873745f37

Keys of To :
Public Key:
0415ae88d502bfe5691d1d4789ff9fa91abab4a0f4cc8a6209ec6a4418f3ecc774aa72b817cb292f00077ccf9bb6f95ef815aedaf051
12157b96b9e5b884a75e4d
Private Key: e0ede27a5594c8afdffc6993435031b73549a1c6117b940e4d2c9b2d04a498ec

BlockChain.js

```
//Part 4 full implementation
//javascript program to create the blockchain with genesis block
//adding the block into blockchain and validate the blockchain for any alteration
// install crypto-js
//proof of work
//multiple transaction in block and giving reward to minner who sucessfully solve the puzzle
//signing and verifing the transactions

const EC = require('elliptic').ec;
const ec = new EC('secp256k1');
const SHA256 = require('crypto-js/sha256')


class Transaction {
    constructor(fromAddress, toAddress, amount) {
        this.fromAddress = fromAddress;
        this.toAddress = toAddress;
        this.amount = amount;
    }

    //calculate hash
    calculateHash() {
        //return sha256
        return SHA256(this.fromAddress, this.toAddress, this.amount).toString();
    }

    //sign the transaction
    signTransaction(signingKey) {
        if (signingKey.getPublic('hex') !== this.fromAddress) {
            throw new Error("Invalid transaction sender");
        }
        const hashTx = this.calculateHash();
        const sig = signingKey.sign(hashTx, 'base64');
        this.signature = sig.toDER('hex');

    }

    //block contains only valid transactions
    isValid() {
        //reward
        if (!this.fromAddress) return true;
        //signature is empty
        if (!this.signature || this.signature.length == 0) {
            throw new Error("No Signature in this transaction");
        }
        //verify the tranasction
        const publicKey = ec.keyFromPublic(this.fromAddress, 'hex');
        return publicKey.verify(this.calculateHash(), this.signature);

    }

}
```

```
class Block {
    constructor(previousHash = '', timestamp, transactions) {
        this.previousHash = previousHash;
        this.timestamp = timestamp;
        this.transactions = transactions;
        this.hash = this.calculateHash();
        this.nonce = 0;

    }
    calculateHash() {
        //return sha256
        return SHA256(this.previousHash + this.timestamp + JSON.stringify(this.transactions) + this.nonce).toString();
    }


    //valid
    hasValidTransactions() {
        for (let transaction of this.transactions) {
            if (!transaction.isValid()) {
                return false;
            }
        }
        return true;
    }

    //proof of work
    mineBlock(difficulty) {
        while (this.hash.substring(0, difficulty) !== Array(difficulty + 1).join("0")) {
            this.nonce++;
            this.hash = this.calculateHash();
        }
        console.log("Block mined: " + this.hash);
        console.log("Nonce: " + this.nonce);
    }

}

class Blockchain {
    constructor() {
        this.chain = [this.createGenesisBlock()];
        //set difficulty
        this.difficulty = 4;
        this.pendingTransactions = [];
        this.miningReward = 100;
    }


    createGenesisBlock() {
        return new Block("0", "06/08/24", "Genesis Block");
    }
    //latest block
    getLatestBlock() {
        return this.chain[this.chain.length - 1];
    }
    //add mine pending transaction
    minePendingTransaction(miningRewardAddress) {
        let block = new Block(this.getLatestBlock().hash, Date.now(), this.pendingTransactions)
        block.mineBlock(this.difficulty);
        this.chain.push(block);
        console.log("Block Mined Sucessfully!!");
        this.pendingTransactions = [new Transaction(null, miningRewardAddress, this.miningReward)];

    }
    //add transactions
    addTransaction(transaction) {
        if (!transaction.isValid()) {
            throw new Error("transaction is invalid");
        }
        if (!transaction.fromAddress || !transaction.toAddress) {
            throw new Error("Transaction should have from and to addresses");
        }
        this.pendingTransactions.push(transaction);
    }

    //get balance of address
    getBalanceOfAddress(address) {
```

```
        let balance = 0;
        for (const block of this.chain) {
            for (const trans of block.transactions) {
                if (trans.fromAddress == address) {
                    balance -= trans.amount;
                }
                if (trans.toAddress == address) {
                    balance += trans.amount;
                }
            }
        }
        return balance;
    }


    //validate blockchain
    isChainValid() {
        for (let i = 1; i < this.chain.length; i++) {
            const currentBlock = this.chain[i];
            const previousBlock = this.chain[i - 1];
            if (!currentBlock.hasValidTransactions()) {
                return false;
            }
            if (currentBlock.hash !== currentBlock.calculateHash()) {
                return false;
            }
            if (currentBlock.previousHash !== previousBlock.hash) {
                return false;
            }


        }
        return true;
    }
}

module.exports.Blockchain = Blockchain;
module.exports.Transaction = Transaction;
```

Main.js

```
//Part 4 full implementation
//javascript program to create the blockchain with genesis block
//adding the block into blockchain and validate the blockchain for any alteration
// install crypto-js
//proof of work
//multiple transaction in block and giving reward to minner who sucessfully solve the puzzle
//signing and verifing the transactions

const {Blockchain, Transaction} = require('./BlockChain');
const EC = require('elliptic').ec;
const ec = new EC('secp256k1');
const SHA256 = require('crypto-js/sha256')

const myKey = ec.keyFromPrivate('62a8423e598a3894ffa33e21f3c4c662645f2b2e4726f04313d67eb873745f37');
//wault address
const myWaultAddress = myKey.getPublic('hex');

const myKey1 = ec.keyFromPrivate('e0ede27a5594c8afdffc6993435031b73549a1c6117b940e4d2c9b2d04a498ec');
//wault address
const myWaultAddress1 = myKey1.getPublic('hex');
let Coin = new Blockchain();

//create transaction
const tx1 = new Transaction(myWaultAddress,myWaultAddress1,70);
//authenticate transaction
tx1.signTransaction(myKey);
//minePendingTransaction
Coin.addTransaction(tx1);


console.log("Start Minning.....");

Coin.minePendingTransaction(myWaultAddress);
console.log("Balance of Tata-Address: " + Coin.getBalanceOfAddress(myWaultAddress));
```

```
Coin.minePendingTransaction(myWaultAddress);
console.log("Balance of Tata-Address: " + Coin.getBalanceOfAddress(myWaultAddress));
// Coin.minePendingTransaction(myWaultAddress);
// console.log('Again Check = the balance of Tata-Address');
// console.log("Balance of Tata-Address: " + Coin.getBalanceOfAddress('Tata-Address'));

console.log(JSON.stringify(Coin, null, 4));

// console.log('IS chain valid: ' + Coin.isChainValid());
// Coin.chain[2].data = { amount: 2000 };
// console.log('IS chain valid: ' + Coin.isChainValid());
```

```
PS D:\SyMCA Sem3\BlockChain\Prac_2> node "d:\SyMCA Sem3\BlockChain\Prac_2\Main.js"
Start Minning.............
Block mined: 00008ce11f2ef39a513b7fde4cbf84e04dfa5086ecab474c37259e8383dd115e
Nonce: 270573
Block Mined Sucessfully!!
Balance of Tata-Address: -70
Block mined: 0000094f93528a65a7e001ecddf1e28c367bf8d5baece903834ba010c7a8b642
Nonce: 11882
Block Mined Sucessfully!!
Balance of Tata-Address: 30
{
    "chain": [
        {
            "previousHash": "0",
            "timestamp": "06/08/24",
            "transactions": "Genesis Block",
            "hash": "611d196a7df2f1168af31b30d51d3d3e5fa7aee96c741f8bb119daa480c4b6cf",
            "nonce": 0
        },
        {
            "previousHash": "611d196a7df2f1168af31b30d51d3d3e5fa7aee96c741f8bb119daa480c4b6cf",
            "timestamp": 1726661320723,
            "transactions": [
                {
                    "fromAddress": "04615e95fa13e28c3ca8312dc9a188c90ff64883249251cf0a69722d3a9eeae6530fd3216dd4b138e91748875a58fa4f87f616464a125ea4315
2c3ff27bb2ea01e",
                    "toAddress": "0415ae88d502bfe5691d1d4789ff9fa91abab4a0f4cc8a6209ec6a4418f3ecc774aa72b817cb292f00077ccf9bb6f95ef815aedaf05112157b96b
9e5b884a75e4d",
                    "amount": 70,
                    "signature": "30450220671a3ba4ee42bf4dbde82a8625af1c71024e5adfecd9432db899440e2aee8ec502210091edf422ace589a03f5f475fec567c3513b3cae
4aaee72a47fe7108209eedf79"
                }
            ],
            "hash": "00008ce11f2ef39a513b7fde4cbf84e04dfa5086ecab474c37259e8383dd115e",
            "nonce": 270573
        },
        {
            "previousHash": "00008ce11f2ef39a513b7fde4cbf84e04dfa5086ecab474c37259e8383dd115e",
            "timestamp": 1726661328689,
            "transactions": [
                {
                    "fromAddress": null,
                    "toAddress": "04615e95fa13e28c3ca8312dc9a188c90ff64883249251cf0a69722d3a9eeae6530fd3216dd4b138e91748875a58fa4f87f616464a125ea43152c
3ff27bb2ea01e",
                    "amount": 100
                }
            ],
            "hash": "0000094f93528a65a7e001ecddf1e28c367bf8d5baece903834ba010c7a8b642",
            "nonce": 11882
        }
    ],
    "difficulty": 4,
    "pendingTransactions": [
        {
            "fromAddress": null,
            "toAddress": "04615e95fa13e28c3ca8312dc9a188c90ff64883249251cf0a69722d3a9eeae6530fd3216dd4b138e91748875a58fa4f87f616464a125ea43152c3ff27bb2
ea01e",
            "amount": 100
        }
    ],
    "miningReward": 100
}
```

Conclusion: Implemented immutable public/private blockchain.