Practical No:02 Roll No:16

Cryptocurrency

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)

Code:-

```
const SHA256=require('crypto-js/sha256');
class Block
{
    constructor(index,timestamp,data,previousHash='')
    {
        this.index=index;
        this.timestamp=timestamp;
        this.data=data;
        this.previousHash=previousHash;
        this.hash=this.calculateHash();
    }
    calculateHash()
    {
        return
SHA256(this.index+this.previousHash+this.timestamp+JSON.stringify(this.data)+this.nonce).
toString();

    }
}
class Blockchain
{
    constructor()
    {
        this.chain=[this.createGenesisBlock()];
    }
    createGenesisBlock()
    {
        return new Block(0,"02/11/2021","Genesis Block","0");

    }
    getLatestBlock()
    {
        return this.chain[this.chain.length-1];
    }
    addBlock(newBlock)
    {
        newBlock.previousHash=this.getLatestBlock().hash;
        newBlock.hash=newBlock.calculateHash();
        this.chain.push(newBlock);
    }
    isChainValid()
    {
        for(let i=1;i<this.chain.length;i++)
```

```
        {
            const currenytBlock = this.chain[i];
            const previousBlock=this.chain[i-1];
            if(currenytBlock.hash!==currenytBlock.calculateHash())
            {
                return false;
            }
            if(currenytBlock.previousHash !== previousBlock.hash)
            {
                return false;
            }

        }
        return true;
    }
}
let MyCoin =new Blockchain();
console.log("Adding Blocks...");
MyCoin.addBlock(new Block(1,"05/08/2022",{amount:4000}));
MyCoin.addBlock(new Block(2,"07/08/2022",{amount:1000}));
console.log(JSON.stringify(MyCoin,null,4));
console.log('Is blockchain valid?'+MyCoin.isChainValid());
```

Output:-

```
akshay@akshay-HP-Laptop-15s-du2xxx:~/Finolex/Sem3/BlockChain/practical2$ node Blockchain1.js
Adding Blocks...
{
    "chain": [
        {
            "index": 0,
            "timestamp": "02/11/2021",
            "data": "Genesis Block",
            "previousHash": "0",
            "hash": "e017a178e601ed29b73480b738c3d543fe2aa3d794de3083e6a80ebb2e6eb96c"
        },
        {
            "index": 1,
            "timestamp": "05/08/2022",
            "data": {
                "amount": 4000
            },
            "previousHash": "e017a178e601ed29b73480b738c3d543fe2aa3d794de3083e6a80ebb2e6eb96c",
            "hash": "c08a0d4ca97a630ed75fa6ecdc163f8a1842aa7f9ee51f6f4c07fcb02bfd16ad"
        },
        {
            "index": 2,
            "timestamp": "07/08/2022",
            "data": {
                "amount": 1000
            },
            "previousHash": "c08a0d4ca97a630ed75fa6ecdc163f8a1842aa7f9ee51f6f4c07fcb02bfd16ad",
            "hash": "59c7657f8cb4d604ed42a6d2ef962653f144c6dea0950c3c5351528abf349a38"
        }
    ]
}
Is blockchain valid?true
akshay@akshay-HP-Laptop-15s-du2xxx:~/Finolex/Sem3/BlockChain/practical2$
```

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

Code:

```
//proof of work addeded
      this.chain=[this.createGenesisBlock()];
      this.difficulty=4;
  }
  createGenesisBlock()
  {
      return new Block(0,"02/11/2021","Genesis Block","0");

  }
  getLatestBlock()
  {
      return this.chain[this.chain.length-1];
  }
  addBlock(newBlock)
  {
      newBlock.previousHash=this.getLatestBlock().hash;
      newBlock.mineBlock(this.difficulty);
      this.chain.push(newBlock);

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

```
        }

      }
      return true;
    }
}
let MyCoin =new Blockchain();
console.log("Mining The Bloc1...");
MyCoin.addBlock(new Block(1,"05/08/2022",{amount:4000}));
console.log("Mining The Bloc2...");
MyCoin.addBlock(new Block(2,"07/08/2022",{amount:1000}));
console.log(JSON.stringify(MyCoin,null,4));
```

Output:-

```
akshay@akshay-HP-Laptop-15s-du2xxx:~/Finolex/Sem3/BlockChain/practical2$ node Blockchain2.js
Mining The Bloc1...
Block Mined!0000081c499ff3b62daf7fa80ed70d17d7ed6481e8374126d8a9f60e3e34ccb5
Mining The Bloc2...
Block Mined!0000cbf745c34731a0267f55595f440e84f5027b8b98615f243329bcb7b88f4b
{
    "chain": [
        {
            "index": 0,
            "timestamp": "02/11/2021",
            "data": "Genesis Block",
            "previousHash": "0",
            "hash": "e017a178e601ed29b73480b738c3d543fe2aa3d794de3083e6a80ebb2e6eb96c",
            "nonce": 0
        },
        {
            "index": 1,
            "timestamp": "05/08/2022",
            "data": {
                "amount": 4000
            },
            "previousHash": "e017a178e601ed29b73480b738c3d543fe2aa3d794de3083e6a80ebb2e6eb96c",
            "hash": "0000081c499ff3b62daf7fa80ed70d17d7ed6481e8374126d8a9f60e3e34ccb5",
            "nonce": 33828
        },
        {
            "index": 2,
            "timestamp": "07/08/2022",
            "data": {
                "amount": 1000
            },
            "previousHash": "0000081c499ff3b62daf7fa80ed70d17d7ed6481e8374126d8a9f60e3e34ccb5",
            "hash": "0000cbf745c34731a0267f55595f440e84f5027b8b98615f243329bcb7b88f4b",
            "nonce": 77855
        }
    ],
    "difficulty": 4
}
```

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

Code:

```
//giving reward to miners and multiple transaction can be added
const SHA256=require('crypto-js/sha256');
class Transaction
{
  constructor(fromAddress,toAddress,amount)
  {
     this.fromAddress=fromAddress;
     this.toAddress=toAddress;
     this.amount=amount;
  }
}
class Block
{
  constructor(timestamp,transactions,previousHash='')
  {

     this.timestamp=timestamp;
     this.transactions=transactions;
     this.previousHash=previousHash;
     this.hash=this.calculateHash();
     this.nonce=0;
  }
  calculateHash()
  {
     return SHA256(this.index+this.previousHash+this.timestamp+
     JSON.stringify(this.data)+this.nonce).toString();

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

    }
    createGenesisBlock()
    {
        return new Block(0,"02/11/2021","Genesis Block","0");

    }
    getLatestBlock()
    {
        return this.chain[this.chain.length-1];
    }
    minePendingTransaction(miningRewardAddress)
    {
        let block = new Block(Date.now(),this.pendingTransaction);
        block.mineBlock(this.difficulty);
        console.log("Block mined Successfully");
        this.chain.push(block);
        this.pendingTransaction=[new
Transaction(null,miningRewardAddress,this.miningReward)];
    }
    createTransaction(transaction)
    {
        this.pendingTransaction.push(transaction);
```

```
}
getBalancedOfAddress(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;
}
isChainValid()
{
    for(let i=1;i<this.chain.length;i++)
    {
        const currenytBlock = this.chain[i];
        const previousBlock=this.chain[i-1];
        if(currenytBlock.hash!==currenytBlock.calculateHash())
        {
            return false;
        }
        if(currenytBlock.previousHash !== previousBlock.hash)
        {
            return false;
        }

    }
    return true;
```

```
  }
}
let MyCoin =new Blockchain();
MyCoin.createTransaction(new Transaction('address1','address2',200));
console.log("\n Starting the mining by miner....");
MyCoin.minePendingTransaction('Tata-address');
console.log("\n Balance of Tata-address is= "+MyCoin.getBalancedOfAddress('Tata-address'));

console.log(JSON.stringify(MyCoin,null,4));
```

Output:

```
 Starting the mining by miner....
Block Mined!00006555adef94a6889801a237b3c5126c1ad7c854ecc8807b2190f14398f78a
Block mined Successfully

 Balance of Tata-address is= 0
{
    "chain": [
        {
            "timestamp": 0,
            "transactions": "02/11/2021",
            "previousHash": "Genesis Block",
            "hash": "f131e1ff46f7e98c8ffbe75f7b4d0304574f06a19072932158245d4de0fdff5e",
            "nonce": 0
        },
        {
            "timestamp": 1666144838952,
            "transactions": [
                {
                    "fromAddress": "address1",
                    "toAddress": "address2",
                    "amount": 200
                }
            ],
            "previousHash": "",
            "hash": "00006555adef94a6889801a237b3c5126c1ad7c854ecc8807b2190f14398f78a",
            "nonce": 4334
        }
    ],
    "difficulty": 4,
    "pendingTransaction": [
        {
            "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)

Code:

Blockchain.js

```
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;
}
calculateHash()
{
return SHA256(this.fromAddress+this.toAddress+this.amount).toString();
}
signTransaction(signingKey)
{
if(signingKey.getPublic('hex')!==this.fromAddress)
{
throw new Error('You cannot sign transactions for other wallets');
}
const hashTx=this.calculateHash();
const sig=signingKey.sign(hashTx,'base64');
this.signature=sig.toDER('hex');
}
isValid()
{
if(this.fromAddress==null)return true;
if(!this.signature || this.signature.length==0)
{
throw new Error('No signature in this transaction!');
}
```

```
const publicKey=ec.keyFromPublic(this.fromAddress,'hex');
return publicKey.verify(this.calculateHash(),this.signature);
}
}
class Block
{
  constructor(timestamp,transactions,previousHash='')
  {
  this.timestamp=timestamp;
  this.transactions=transactions;
  this.previousHash=previousHash;
  this.hash=this.calculateHash();
  this.nonce=0;
  }
  calculateHash()
  {
  return

SHA256(this.previousHash+this.timestamp+JSON.stringify(this.transactions)+this.no
nce).toString();
  }
  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);
  }
  hasValidTransactions()
  {
  for(const tx of this.transactions)
  {
  if(!tx.isValid())
  {
  return false;
```

```
    }
    }
    return true;
    }
}
class Blockchain
{
    constructor()
    {
    this.chain=[this.createGenesisBlock()];
    this.difficulty=4;
    this.pendingTransaction=[];
    this.miningReward=100;

    }
    createGenesisBlock()
    {
        return new Block(0,"02/11/2021","Genesis Block","0");
    }
    getLatestBlock()
    {
        return this.chain[this.chain.length-1];
    }
    minePendingTransaction(miningRewardAddress)
    {
        const rewardTx=new
        Transaction(null,miningRewardAddress,this.miningReward);
        this.pendingTransaction.push(rewardTx);
        let block = new
        Block(Date.now(),this.pendingTransaction,this.getLatestBlock().hash);
        block.mineBlock(this.difficulty);
        console.log("Block mined Successfully");
        this.chain.push(block);
        this.pendingTransaction=[];
    }
    addTransaction(transaction)
    {
```

```
if(!transaction.fromAddress||!transaction.toAddress)
{
   throw new Error('Transaction must include from and to address! ');
}
if(!transaction.isValid())
{
   throw new Error("Cannot add invalid transaction to chain!");
}
   this.pendingTransaction.push(transaction);
}
getBalancedOfAddress(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;
}
isChainValid()
{
   for(let i=1;i<this.chain.length;i++)
   {
      const currenytBlock = this.chain[i];
      const previousBlock=this.chain[i-1];
      if(currenytBlock.hash!==currenytBlock.calculateHash())
      {
```

```
            return false;
        }
        if(currenytBlock.previousHash !== previousBlock.hash)
        {
            return false;
        }
    }
    return true;
  }

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

Main.js:-

```
const {Blockchain,Transaction}=require("./Blockchain");
const EC=require('elliptic').ec;
const ec=new EC('secp256k1');
const
myKey=ec.keyFromPrivate('046992ca6c22bf0bbeaf55eb86d663b60158690502016b
5c6c1c5f7e0d52fa3a701155ce1de50b4014e532b95737f62be92f2164a593170102a4
413e3189b5ef3d');
const myWalletAddress=myKey.getPublic('hex');
let MyCoin =new Blockchain();

const tx1=new Transaction(myWalletAddress,'public key goes here',10);
tx1.signTransaction(myKey);
MyCoin.addTransaction(tx1);
console.log("\n Starting the miner....");
MyCoin.minePendingTransaction(myWalletAddress);
console.log("\n Balance of myWalletAddresss
is="+MyCoin.getBalancedOfAddress(myWalletAddress));
console.log("\n Is chain valid= "+MyCoin.isChainValid());
MyCoin.chain[1].transactions[0].amount=15;
console.log("\n Is chain valid= "+MyCoin.isChainValid());
```

KeyGenerator.js:

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

Output-

```
PS D:\MCA\sem3\Blockchain\Practical\Practical2> node main.js

 Starting the miner....
Block Mined!0000449a0610ad43a51137e5db4d9c9d49f9f9271fb308a3124c0ea59c2fd258
Block mined Successfully

 Balance of myWalletAddresss is= 90

 Is chain valid= true

 Is chain valid= false
PS D:\MCA\sem3\Blockchain\Practical\Practical2>
```