VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

**Information Security Analysis and Audit (BCSE353E)**

# Research Paper on
# Secure Online Transaction Using Cryptography

**Submitted to-**
Brindha S

**Submitted by-**
Ishita Agarwal (21BCE5060)
S. Caitlin (21BCE5737)
Shivansh Bansal (21BCE5742)

## PROPOSED SYSTEM IN DETAIL

The following is a comprehensively suggested cryptographic system for safe transactions:

- **Authentication and registration of users:** By providing their credentials (such as a username and password), users can sign up for the system. Passwords can be securely stored by hashing them with a robust hashing algorithm like bcrypt. For subsequent transactions, use a secure authentication method like JWT (JSON Web Tokens) to verify user identity.
- **Encryption of Transaction Data:** Utilize either symmetric or asymmetric encryption algorithms to encrypt sensitive transaction data. Equilibrium encryption: The transaction data is encrypted and decrypted with a shared secret key. Encryption asymmetrically: Use public-private key matches, where the source scrambles the information with the beneficiary's public key, and the beneficiary unscrambles it utilizing their confidential key.
- **Secure Means of Communication:** Utilizing protocols like HTTPS (HTTP over SSL/TLS), create a secure communication channel between the client and the server. During the transmission of data, SSL/TLS certificates can be used to guarantee encryption and data integrity.
- **Electronic Signatures:** Digital signatures can be used to check the authenticity of transactions and ensure the integrity of data. Encrypt the transaction data with the sender's private key and create a hash of it. Using the sender's public key, the recipient can verify the signature, ensuring that the transaction has not been tampered with.
- **The Blockchain System:** Use a blockchain-based system to safely store and verify transactions. The digital signature can be represented for each transaction as a block of relevant data. To generate a hash for each block, cryptographic hashing algorithms like SHA-256 are used. By chaining the blocks together, data immutability is ensured and each block's hash is dependent on the previous block. To validate and add new blocks to the blockchain, use a consensus mechanism like proof-of-work or proof-of-stake.
- **Hashing Operations:** Data are represented by unique, fixed-size hash values using hash functions. Hash functions are used in online transactions to check for tampering and ensure the integrity of the data. The sender incorporates the hash value of the data into the transaction when the data is sent. After receiving the data, the recipient independently calculates the hash value and compares it to the hash value that was transmitted. It verifies the data's integrity if the hash values match.
- **Key Leadership:** Maintaining the safety of online transactions necessitates good key management. Key management includes key generation, distribution, storage, and revocation. To reduce the likelihood of a key being compromised, symmetric encryption keys should be rotated periodically and securely shared between the client and the server. The safe storage of private keys and the appropriate distribution and management of public keys are necessary for asymmetric encryption keys.

- **Updates for the safety:** It is essential to regularly update the cryptographic algorithms, protocols, and software implementations in order to guarantee the continued safety of online transactions. This ensures that the system is always up to date with the most recent security practices and helps address any vulnerabilities that have recently been discovered. Online transactions can be protected from eavesdropping, data tampering, and impersonation attacks by combining these cryptographic techniques and best practices. This gives users confidence in the security of their interactions with online services and businesses.

## MODULES IN PROJECT AND DETAIL DESCRIPTION

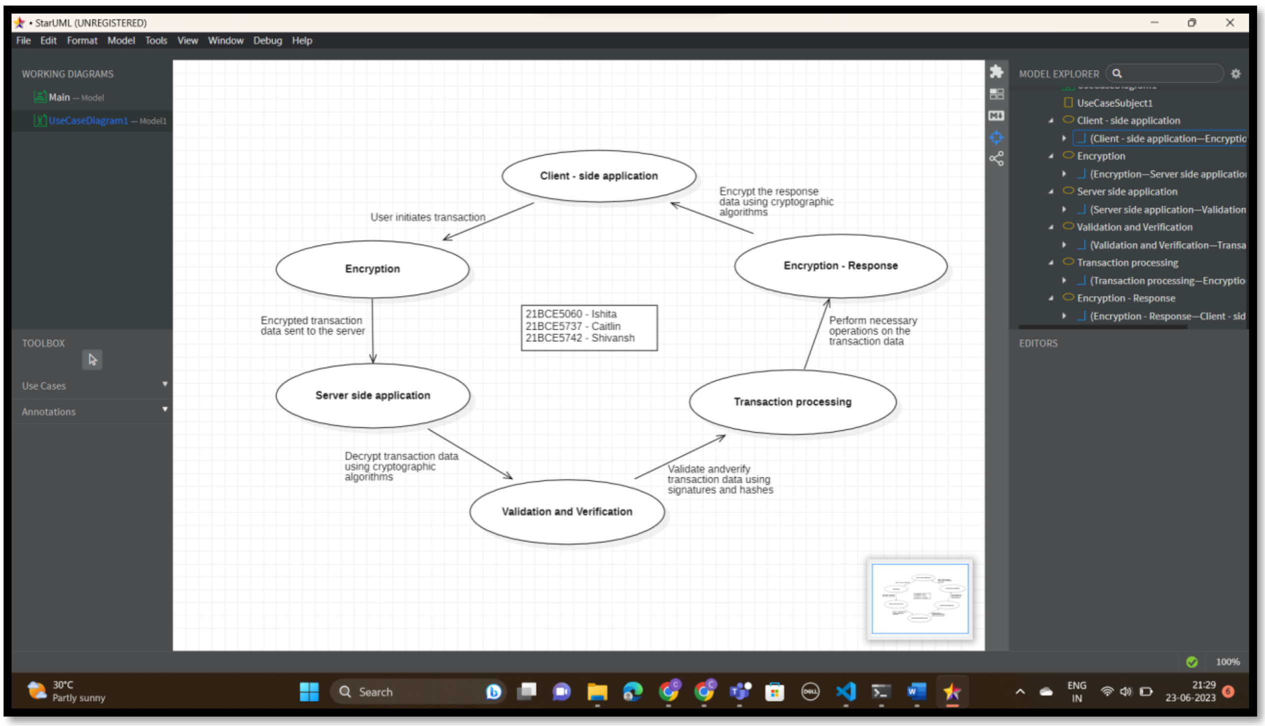
- **Crypto**: Node.js's built-in crypto module offers cryptographic features. It contains a variety of algorithms for digital signatures, encryption, decryption, and hash functions. This module can be used to make digital signatures, encrypt and decrypt data, and generate hashes.

- **Jsonwebtoken**: JSON Web Tokens (JWTs) can be created and verified with the help of the jsonwebtoken module. Secure information transmission between parties is made possible by JWTs. To enforce authentication, sign and verify tokens, and guarantee the integrity of data, you can make use of this module.

- **Bcrypt**: Hashing and salting passwords are done with the bcrypt module. By employing a one-way hash function and adding a salt to ward off rainbow table attacks, it provides a safe method for storing passwords. This module safeguards client passwords and touchy data.

- **Tls**: Using the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols, the tls module in Node.js makes it possible for secure communication to take place over the network. Secure communication channels can be enforced, encrypted connections can be established, and secure servers and clients can be created with this tool.

## ARCHITECTURE DIAGRAM:

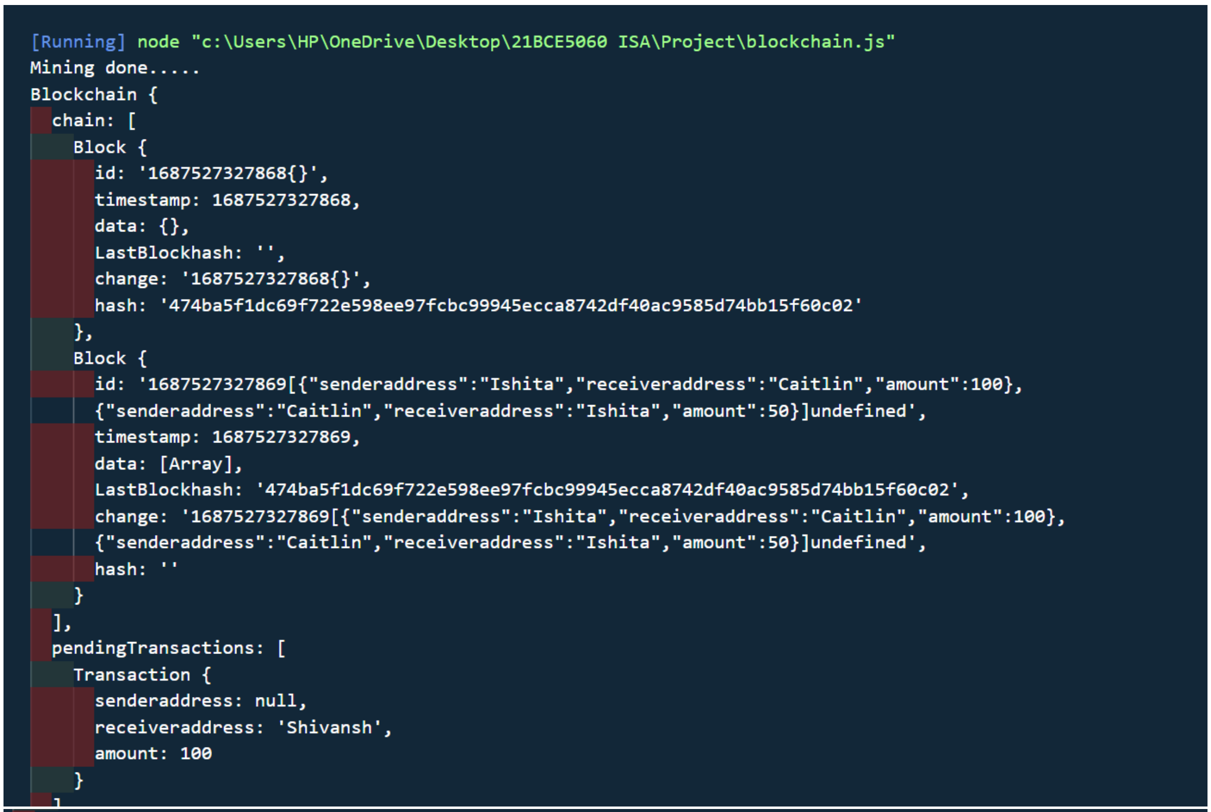

Client - side application
User initiates transaction
Encrypt the response data using cryptographic algorithms
Encryption
Encryption - Response
Encrypted transaction data sent to the server
21BCE5060 - Ishita
21BCE5737 - Caitlin
21BCE5742 - Shivansh
Perform necessary operations on the transaction data
Server side application
Transaction processing
Decrypt transaction data using cryptographic algorithms
Validate andverify transaction data using signatures and hashes
Validation and Verification

In this architecture diagram:
The client-side application initiates a transaction by generating the transaction data.
The transaction data is encrypted using cryptographic algorithms on the client-side and sent to the server.
The server-side application receives the encrypted transaction data and decrypts it using the appropriate cryptographic algorithms.
The server-side application performs validation and verification of the transaction data, including checking digital signatures and hashes to ensure integrity and authenticity.
The server-side application processes the transaction, performing necessary operations based on the transaction data.
The response data is encrypted using cryptographic algorithms on the server-side and sent back to the client.
The client-side application receives the encrypted response data and decrypts it using the appropriate cryptographic algorithms.
Throughout the transaction flow, cryptographic techniques such as encryption, digital signatures, and hashing are used to ensure the confidentiality, integrity, and authenticity of the data. This architecture provides a secure framework for online transactions, protecting sensitive information and maintaining the trustworthiness of the transaction process.

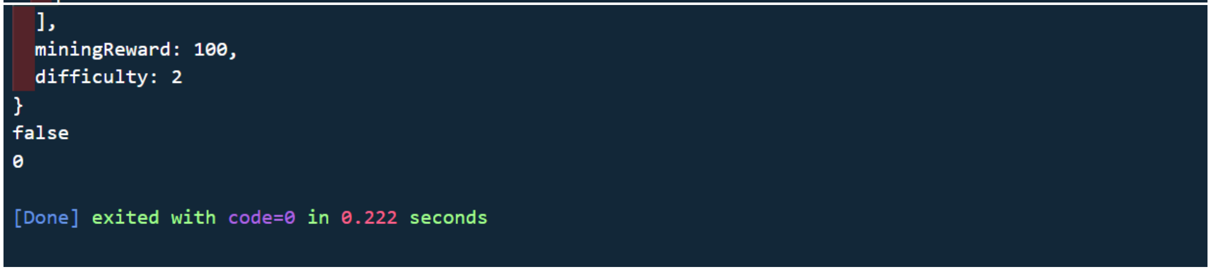
## IMPLEMENTATION:

```
const crypto=require('crypto');
const secret="ishita";
class Transaction{
    constructor(senderaddress,receiveraddress,amount){
        this.senderaddress=senderaddress;
        this.receiveraddress=receiveraddress;
        this.amount=amount;
    }
}

class Block{
    constructor(id,timestamp,data,LastBlockhash){
    this.id=id;
    this.timestamp=timestamp;
    this.data=data;
    this.LastBlockhash=LastBlockhash;
    this.change=0;
    this.hash=this.createHash();
    }
    createHash(){
return crypto.createHmac('sha256',secret)
.update(this.change=this.id=this.timestamp+JSON.stringify(this.data)+this.Last
Blockhash)
.digest('hex')
    }
    //rewarding minors
    mineTheBlock(difficulty){
        let hash="";
        while(hash.substring(0,difficulty)=="0".repeat(difficulty)){
            this.change++;
            hash=this.createHash();
        }
        console.log("Mining done.....",hash);
        this.hash=hash;
    }

}
class Blockchain{
    constructor(){
        this.chain=[this.createGenesisBlock()];
        this.pendingTransactions=[];
        this.miningReward=100;
        this.difficulty=2;
    }
    createTransaction(transaction){
        this.pendingTransactions.push(transaction);
    }
    createGenesisBlock(){
        return new Block(0,Date.now(),{},"")
    }
    getLastBlockHash(){
        return this.chain[this.chain.length-1].hash;
    }
    addBlock(index,mineraddress){
        let block=new Block(index,Date.now(),this.pendingTransactions);
        block.LastBlockhash=this.getLastBlockHash();
        block.mineTheBlock(this.difficulty);
        this.chain.push(block);

        this.pendingTransactions=[new
Transaction(null,mineraddress,this.miningReward)];
    }
    getBalance(address){
        let balance=0;
        for(let index=0;index<this.chain.length;index++){
            const block=this.chain[index];
            for(let index=0;index<block.data.length;index++){
                const transaction=block.data[index];
                if(address==transaction.senderaddress){
                    balance-=transaction.amount;
                }
                if(address==transaction.receiveraddress)
                balance+=transaction.amount;
            }
        }
        return balance;
    }
    isValidBlockChain(){
        for(let index=1;index<this.chain.length;index++){
            const Block=this.chain[index];
            const newHash=Block.createHash();
            if(newHash!=Block.hash)return false;
            if(this.chain[index-1].hash!=Block.LastBlockhash)return false;
        }
        return true;
    }
}
let kwkcoin=new Blockchain();
/*kwkcoin.addBlock(new Block(1,Date.now(),{balance:100}));
kwkcoin.addBlock(new Block(2,Date.now(),{balance:200}));
kwkcoin.addBlock(new Block(3,Date.now(),{balance:300}));*/

kwkcoin.createTransaction(new Transaction("Ishita","Caitlin",100));
kwkcoin.createTransaction(new Transaction("Caitlin","Ishita",50));
kwkcoin.addBlock(1,"Shivansh");
//kwkcoin.chain[2].data.balance=400;
console.log(kwkcoin);
console.log(kwkcoin.isValidBlockChain());
console.log(kwkcoin.getBalance("Shivansh"));
```

## OUTPUT:



```
[Running] node "c:\Users\HP\OneDrive\Desktop\21BCE5060 ISA\Project\blockchain.js"
Mining done.....
Blockchain {
  chain: [
    Block {
      id: '1687527327868{}',
      timestamp: 1687527327868,
      data: {},
      LastBlockhash: '',
      change: '1687527327868{}',
      hash: '474ba5f1dc69f722e598ee97fcbc99945ecca8742df40ac9585d74bb15f60c02'
    },
    Block {
      id: '1687527327869[{"senderaddress":"Ishita","receiveraddress":"Caitlin","amount":100},
      {"senderaddress":"Caitlin","receiveraddress":"Ishita","amount":50}]undefined',
      timestamp: 1687527327869,
      data: [Array],
      LastBlockhash: '474ba5f1dc69f722e598ee97fcbc99945ecca8742df40ac9585d74bb15f60c02',
      change: '1687527327869[{"senderaddress":"Ishita","receiveraddress":"Caitlin","amount":100},
      {"senderaddress":"Caitlin","receiveraddress":"Ishita","amount":50}]undefined',
      hash: ''
    }
  ],
  pendingTransactions: [
    Transaction {
      senderaddress: null,
      receiveraddress: 'Shivansh',
      amount: 100
    }
```



```
  ],
  miningReward: 100,
  difficulty: 2
}
false
0

[Done] exited with code=0 in 0.222 seconds
```