DOUBLE SPENDING ATTACK PREVENTION

A project submitted to

SENTHIL PRAKASH P N

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING (SCOPE)

in partial fulfilment of the requirements for the course of

BCSE324L - Foundations of Blockchain Technology

IN

B. TECH. COMPUTER SCIENCE ENGINEERING

VIT®

CHENNAI

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Vandalur – Kelambakkam Road
Chennai – 600127

by

ANMOL HARSH 21BCE1057

TAVISHI RASTOGI 21BCE1043

ANIKA SHARMA 21BCE5917

Table of Contents

1.		Abstract	4
2.		Introduction	5
3.		Problem Statement	6
	3.1	Idea	6
	3.2	Scope	6
	3.3	Novelty	7
	3.4	Comparative Statement	7
4.		Literature Surveys	8
5.		Architecture	11
	5.1	Network Observers	11
	5.2	Peer Alert Systems	12
	5.3	Blockchain Architecture	12
	5.4	Codes	12
6.		Implementation	23
	6.1	Additional Security Aspects	23
	6.2	Performing Transactions	24
	6.3	Signature	24
	6.4	Cryptocurrency	25
	6.5	Network Observers	25
	6.6	Peer Alert Systems	26
7.		Results & Discussion	26

8.		Appendix I	27
9.		Frontend Interface	29
	9.1	Functionality	29
	9.2	Appendix II	29
10.		References	31

1. Abstract

This report provides an overview about the principles of Security Management used in this study paper to suggest a framework for preventing double spending attacks in blockchain-based systems. In order to reduce the risk of double spending attacks, the proposed framework contains a list of recommendations and best practices for integrating security processes into blockchain-based systems. In a case study, the authors implemented the suggested structure in a real-world blockchain-based system and assessed how well it prevented double spending attacks. The findings demonstrate that the suggested framework can successfully thwart double spending attacks and offer a reliable and safe blockchain-based system. The study emphasizes the significance of implementing security management principles in blockchain-based systems to guarantee transaction security and integrity and stop theft. This research can be helpful for businesses seeking to implement blockchain-based solutions across various industries, security experts, and blockchain developers. This study examines various defense strategies against double spending attacks in blockchain-based systems and gives an overview of such attacks. Double spending attacks take place when a user uses the same digital commodity more than once, endangering the security of transactions and costing money. A thorough overview of the literature on double spending attacks and the current defences against them, such as consensus mechanisms, transaction confirmation, and proof-of-work protocols, is presented in this paper. The authors suggest using a hybrid consensus mechanism that combines proof-of-work and proof-of-stake protocols to avoid double spending attacks. By lowering the possibility of double spending assaults and accelerating transaction validation, the suggested method seeks to increase the security and effectiveness of blockchain-based systems. The authors offer a case study in which they implement the suggested strategy in a real-world blockchain-based system and assess how well it guards against double spending assaults. The outcomes demonstrate that the suggested strategy can successfully thwart double spending assaults and offer a safe and dependable blockchain-based system. The study adds to the body of knowledge on blockchain security and gives security experts and blockchain engineers useful information.

2. Introduction

Blockchain technology has revolutionized a number of sectors by offering a decentralized, secure, and open system for recording and verifying transactions. Blockchain-based systems are vulnerable to security risks like double spending assaults that can taint transactions' integrity and result in losses of money. When a user submits two transactions at once to spend the same digital asset twice, one of them is accepted and the other is denied. This is known as a double spending attack. This may occur for a number of causes, including malicious activity, network latency, and software bugs. Double spending attacks are a major worry for blockchain-based systems, particularly for cryptocurrencies where high-value assets are involved in the deals.

Double spending assaults have been protected against using a number of strategies, such as consensus mechanisms, transaction confirmation, and proof-of-work protocols. Transaction confirmation needs a certain number of confirmations from various nodes to validate a transaction, whereas consensus mechanisms guarantee that all nodes in the network concur on the validity of transactions. By requiring users to crack a cryptographic conundrum in order to add a new block to the blockchain, proof-of-work algorithms make sure that the user has put computational resources into the transaction.

The implementation of a hybrid consensus mechanism that combines proof-of-work and proof-of-stake protocols is the innovative strategy proposed in this research paper to prevent double spending attacks in blockchain-based systems. By lowering the possibility of double spending assaults and accelerating transaction validation, the suggested method seeks to increase the security and effectiveness of blockchain-based systems. The suggested method is implemented in a real-world blockchain-based system in the case study presented in the paper, and its efficacy in preventing double spending attacks is assessed.

By offering a decentralized, unchangeable, and transparent system, blockchain technology has completely transformed the way interactions are conducted. Blockchain-based systems are vulnerable to security risks like double spending assaults that can taint transactions' integrity and result in losses of money. When a user submits two transactions at once to spend the same digital asset twice, one of them is accepted and the other is denied. This is known as a double spending attack. This may occur for a number of causes, including malicious activity, network latency, and software bugs. By putting processes, controls, and ongoing monitoring into place, Information Security Management (ISM) principles offer a complete approach to managing information security risks. In this study, we suggest an ISM-based paradigm for guarding against double spending attacks in blockchain-based systems. In order to reduce the risk of double spending attacks, the framework

contains a set of recommendations and best practices for integrating ISM processes into blockchain-based systems.

We show a case study in which we implemented the suggested framework in a real blockchain-based system and assessed how well it prevented double spending attacks. The case study includes a cryptocurrency trading platform that, by its very nature, is vulnerable to double spending attacks. We implemented the suggested structure on the platform and assessed how well it shielded against double-spending attacks. The findings demonstrate that the suggested framework can successfully thwart double spending attacks and offer a reliable and safe blockchain-based system.

3. Problem Statement

3.1 Idea

When a sender tries to perform the same transaction (with the same amount) twice, but with two different receivers, even though his initial balance is not sufficient to satisfy both the transactions, a double spending attack occurs.

This can be illustrated in the below example. Consider a sender to have 10 cryptocurrency units, who plans to give these 10 units to 2 different receivers, concurrently, across separate blocks. For both these transactions to occur, a total of 20 units should be available with the sender, whereas he has just 10, implying that he plans to reuse the same money of the first transaction, in the second transaction, such that the 10 units is deducted at the same time (so only 10 units is deducted from his balance finally).

3.2 Scope

In this project, to overcome the issues in the existing transaction system, network observers are used that can track anomalous transactions performed without authorization.

Additionally, to notify the sender and receiver about the unauthorized transaction, a peer alert system is designed such that the message is passed from the fraudulent node to the sender and receiver nodes.

3.3 Novelty

FIREWALL

Our firewall uses a very unique method for detecting potential hard fork attacks coupled with specific block chain DDoS flooding. All connected nodes/peers are examined by the amount of data they're sending or receiving regardless of the algorithm or block validity. A deeper analysis is then used to verify their blockchain start & sync height and bandwidth use is within safe limits; The average among all peers connected.

Range based blockchain checkpoints that use averages of live blockchain sizes further enhance security by limiting potential attacks known as >51% of distributed hashing power (double-spend).

3.4 Comparative Statement

In order to prevent double spending attacks in blockchain-based systems, the study paper "Preventing Double Spending Attacks Using Information Security Management" suggests a framework based on the principles of ISM. The application of ISM processes, including risk assessment, the implementation of security controls, ongoing monitoring, and improvement, is presented in this paper as a complete strategy to thwart double spending attacks. To show how well the suggested framework works in preventing double spending attacks in a real-world blockchain-based system, the writers provide a case study.

This study stands out from other papers that deal with the problem of double spending attacks in blockchain-based systems because it focuses on using ISM principles to stop double spending attacks. For businesses seeking to increase the security and integrity of their blockchain-based systems, the authors offer a useful method for integrating ISM processes into these systems. Additionally, the case study offers concrete proof of how well the suggested framework works in practice to thwart double spending attacks, which supports the viability of the suggested strategy. Overall, this study offers insightful information about using ISM methods to defend against double spending attacks in blockchain-based systems.

4. Literature Surveys

"Misbehavior in Bitcoin: A study of double-spending and accountability"

Ghassan O. Karame, Elli Androulaki, Marc Roeschlin, Arthur Gervais, Srdjan Čapkun

Bitcoin is a decentralized payment system that relies on Proof-of-Work (PoW) to resist double-spending through a distributed timestamping service. To ensure the operation and security of Bitcoin, it is essential that all transactions and their order of execution are available to all Bitcoin users. Unavoidably, in such a setting, the security of transactions comes at odds with transaction privacy. Motivated by the fact that transaction confirmation in Bitcoin requires tens of minutes, we analyze the conditions for performing successful double-spending attacks against fast payments in Bitcoin, where the time between the exchange of currency and goods is short (in the order of a minute).

We show that unless new detection techniques are integrated in the Bitcoin implementation, double-spending attacks on fast payments succeed with considerable probability and can be mounted at low cost. We propose a new and lightweight countermeasure that enables the detection of double-spending attacks in fast transactions. In light of such misbehavior, accountability becomes crucial. We show that in the specific case of Bitcoin, accountability complements privacy. To illustrate this tension, we provide accountability and privacy definition for Bitcoin, and we investigate analytically and empirically the privacy and accountability provisions in Bitcoin.

"Countering Double-Spend Attacks on Bitcoin Fast-Pay Transactions"

John P. Podolanko, Jiang Ming (2019)

Bitcoin does not rely on a trusted authority. As its popularity grows, more businesses are starting to accept it as payment, including services like fast food restaurants and vending machines that must deliver goods soon upon payment. Unfortunately, these fast pay scenarios are vulnerable to fraudulent double-spending of Bitcoins. To protect consumers from having to cover the costs of these attacks, there is a growing need to find countermeasures to such attacks that are scalable and realistic to deploy. Several possible countermeasures have been proposed in prior work but not evaluated

carefully.

In this paper, we analyze these countermeasures and simulate their effects in a scaled Bitcoin P2P network using the Shadow framework. We find that integrating several of these countermeasures into an enhanced observer can be effective in detecting and alerting a vendor to an incoming double-spend attack in less than 28 seconds on average in our model.

"Blockchain Attacks, Analysis and a Model to Solve Double Spending Attack"

A. Begum, A. H. Tareq, M. Sultana, M. K. Sohel, T. Rahman, and A. H. Sarwar (2020)

Blockchain is such a technology that helps us to use a shared ledger. Although the ledger is in a shared manner, the total system is quite secure. Bitcoin is a crypto currency which uses blockchain technology. Value of blockchain is much higher than dollars or some other expensive currency. In this paper, we want to show the attacks on blockchain, their targeted area, reason and their possible proposed solution as review. Besides this, Double spending attack is a major attack on blockchain which has occurred twice till now and caused a huge loss of crypto currency. In this paper,

We also want to represent the reasons for these attacks and propose one solution that can prevent Double Spending Attacks. Our findings will provide some future direction for new researchers and also help the crypto business analysts to predict about present security in the aspects of blockchain networks.

"A novel double spending attack countermeasure in blockchain"

Kervins Nicolas, Yi Wang (2019)

A blockchain database containing files regarding transactions of cryptocurrency is sometimes vulnerable to double spending attacks. This type of attack pertains to a coin being spent more in more than one transaction in the network. This paper is motivated by a goal to create a blockchain that can withstand double spending attacks. This way, honest miners will be able to safely and securely exchange cryptocurrency. There currently lack valuable prevention methods in the network

therefore we designed a novel countermeasure to combat double spending attacks on the blockchain system.

We proposed the MSP (Multistage Secure Pool) framework in order to address the vulnerabilities on the blockchain. This was designed to handle both discrete and general issues that affect the overall security of the blockchain. Our evaluation using this application shows that there was a decrease in the amount of attacks propagating through the system based on our system's robustness and capabilities. We also present machine learning capabilities of the system in our study in order to enable a progressive aspect to the design.

Providing our application with the ability to analyze data in order to recognize and classify distinct actions will enable for greater comprehension. An application that learns, updates and configures to meet specified defensive standards present key design features which enables for greater understanding and future analysis of the overall blockchain network.

Distributed Hybrid Double-Spending Attack Prevention Mechanism for Proof-of-Work and Proof-of-Stake Blockchain Consensuses

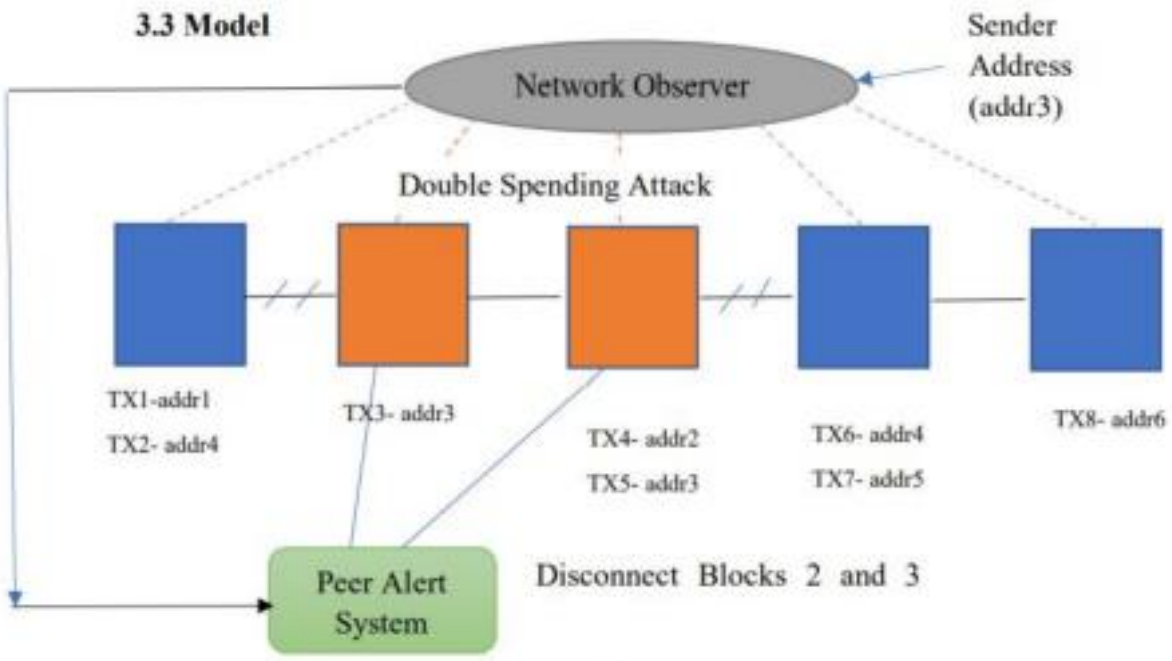
Nur Arifin Akbar, Amgad Muneer, Narmine ElHakim and Suliman Mohamed Fati (2021)

Blockchain technology is a sustainable technology that offers a high level of security for many industrial applications. Blockchain has numerous benefits, such as decentralization, immutability and tamper proofing. Blockchain is composed of two processes, namely, mining (the process of adding a new block or transaction to the global public ledger created by the previous block) and validation (the process of validating the new block added). Several consensus protocols have been introduced to validate blockchain transactions, Proof-of-Work (PoW) and Proof-of-Stake (PoS), which are crucial to cryptocurrencies, such as Bitcoin. However, these consensus protocols are vulnerable to double-spending attacks. Amongst these attacks, the 51% attack is the most prominent because it involves forking a blockchain to conduct double spending. Many attempts have been made to solve this issue, and examples include delayed proof-of-work (PoW) and several Byzantine fault tolerance mechanisms. These attempts, however, suffer from delay issues and unsorted block sequences. This study proposes a hybrid algorithm that combines PoS and PoW mechanisms to provide a fair mining reward to the miner/validator by conducting forking to combine PoW and PoS consensuses. As demonstrated by the experimental results, the proposed algorithm can reduce the possibility of intruders performing double mining because it requires achieving 100% dominance in the network, which

is impossible.

5. Architecture

MODEL OVERVIEW



Network observers: Network observers can monitor transactions and verify their authenticity. Peer alert systems are designed to quickly notify other nodes about any suspicious activity.

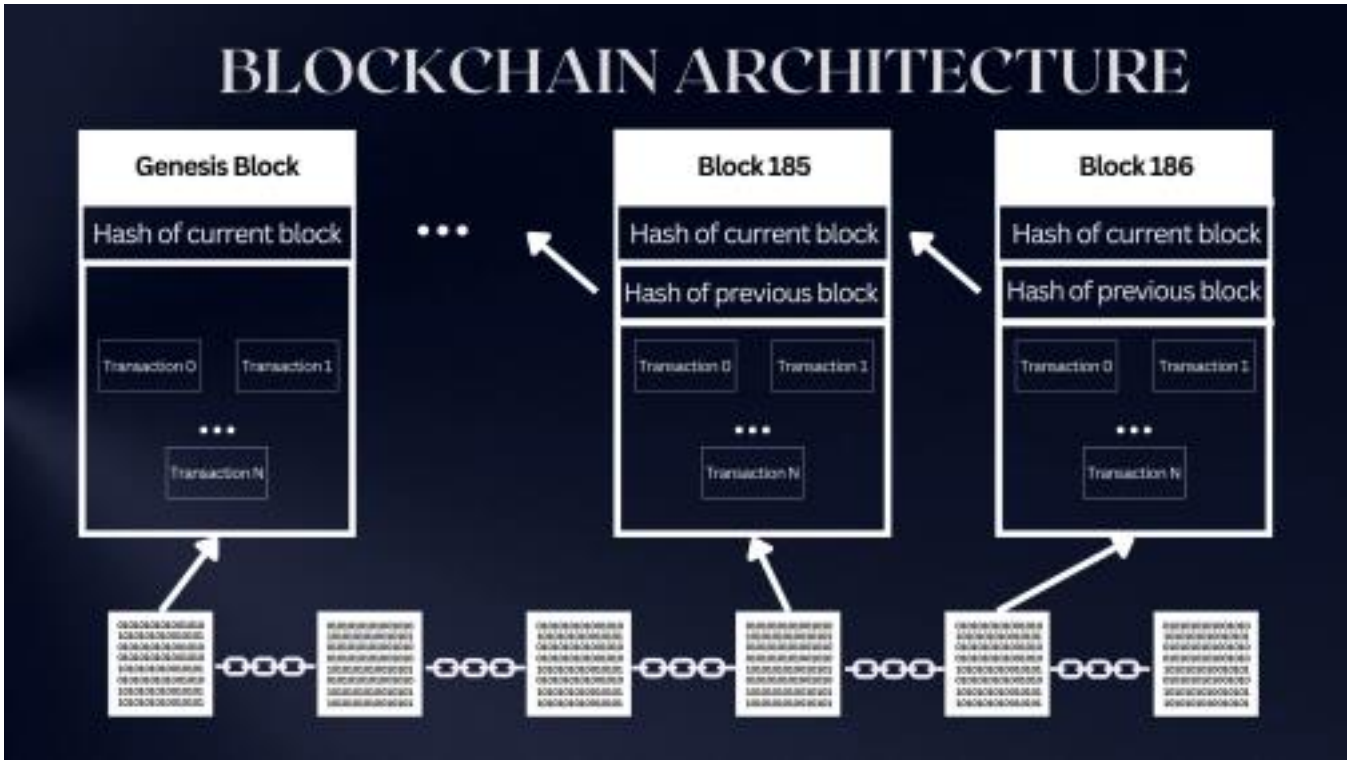
5.1 Network Observers

The network observers are placed between each pair of nodes. A sender node that contains the transaction amount, transaction ID and sender details will transfer this information to the neighboring node. The neighboring node will then transfer this to the corresponding forward node, after performing the transaction step. This continues till the receiver node receives the amount. The observers within each pair of nodes, will keep a record of the user ids, the number of transactions and amounts issued by them. With this record, an observer will check the frequency of node communications in its node pair based on the number of transactions issued by the user. If the frequency is higher than what is required, the anomaly will be logged, and the transaction will be aborted with an acknowledgement sent back to the sender node.

5.2 Peer Alert Systems

When a network observer observes an anomaly at one of the transaction steps between a node-pair, the surrounding nodes are alerted about the fraudulent transaction. This will help the neighboring nodes terminate their connections with the fraudulent node pair, such that their transactions can be performed along other safe routes.

5.3 Blockchain Architecture



CODES

CRYPTOBLOCKCHAIN.js

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

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 transaction for other wallets.'); }

 const hashTx= this.calculateHash();
 const sig= signingKey.sign(hashTx,'base64');
 this.signature= sig.toDER('hex');
 }

 isValid()

 {
 if(this.fromAddress === null)
 {
 console.log("1");
 return true;
 }

 if(this.signature.length === 0)
 {
 throw new Error('No signature in this transaction.');
 }

 const publicKey= ec.keyFromPublic(this.fromAddress, 'hex');
console.log("\nNew Transaction");
 console.log("\nThe signature is: "+this.signature);
 return publicKey.verify(this.calculateHash(), this.signature);

 }
 }

class CryptoBlock{
 constructor(timestamp, transactions, precedingHash=" "){
this.timestamp = timestamp;
 this.transactions = transactions;
 this.precedingHash = precedingHash;
 this.hash = this.computeHash();
 this.nonce = 0;
 this.ne=null;
```

```
 }

 //Function to compute the current hash based on the preceding hash,
timestamp, transactions and random nonce
 computeHash() {
 return SHA256(
 this.precedingHash +
 this.timestamp +
 JSON.stringify(this.transactions) +

 this.nonce
 ).toString();
 }



 //To increase difficulty level while mining blocks by appending extra zeros to the
hash
 mineBlock(difficulty) {
 while (
 this.hash.substring(0, difficulty) !== Array(difficulty + 1).join("0") ) {
 this.nonce++;
 this.hash = this.computeHash();
 }
 console.log("Block Mined: "+ this.hash);
 console.log("Timestamp: "+ this.timestamp);
 //this.ne= new NetworkObserver(this.hash);
 }
}


class CryptoBlockchain{
 constructor(){
 this.currenthash = "0";
 this.blockchain = [this.startGenesisBlock()];
 this.difficulty = 4;
 this.pendingTransactions = ["0"];
 this.miningReward = 10;
 }



 //Function to create initial block in the crytocurrency blockchain
startGenesisBlock(){
 let cb= new CryptoBlock(0,Date.now(),"0");
 this.currenthash = cb.hash;
 return cb;
 }



 //Function to receive the latest block in the cryptocurrency blockchain
obtainLatestBlock(){
 return this.blockchain[this.blockchain.length - 1];
```

```
 }



//Function to add an additional block to the blockchain and create an empty
transaction for this block
 minePendingTransactions(miningRewardAddress)
 {
 let block= new CryptoBlock(Date.now(), this.pendingTransactions,
this.currenthash);
 block.mineBlock(this.difficulty);

 console.log('Block successfully mined!');
 this.blockchain.push(block);
 //Pop the completed transaction
 //this.pendingTransactions.push(new Transaction(null,
miningRewardAddress, this.miningReward));

 this.currenthash= block.hash;

 }



 //Add new transaction to log array
 addTransaction(transaction)
 {

 if(!transaction.fromAddress || !transaction.toAddress)
 {
 console.log(transaction.fromAddress);
 console.log(transaction.toAddress);
 throw new Error('Transaction must include from and to address'); }

 if(!transaction.isValid())
 {
 throw new Error('Cannot add invalid transaction to blockchain');

 }

 this.pendingTransactions.push(transaction);
 }



 //Function to perform the transaction between the intended sender and
recipient blocks given their addresses
 getBalanceOfAddress(address)
 {
 var balance= 10;
 console.log("\nInitial balance was: "+balance);

 for(const block of this.blockchain)
```

```
 {

 if(block.transactions.length>0)
 {
 for(const trans of block.transactions)
 {
 if(trans.fromAddress === address)
 {
 balance-= trans.amount;

//this.pendingTransactions.splice(this.pendingTransactions.indexOf(trans),1); }

 if(trans.toAddress === address)
 {
 balance+= trans.amount;
 }

 }
 }
 }

 return balance;
 }



 //Function to authenticate every pair of nodes/blocks in the cryptocurrency
blockchain
 checkChainValidity() {
 for (let i = 1; i < this.blockchain.length; i++) {
 const currentBlock = this.blockchain[i];
 const precedingBlock = this.blockchain[i - 1];

 if(!currentBlock.hasValidTransactions())
 {
 return false;
 }

 if (currentBlock.hash !== currentBlock.computeHash()) {
return false;
 }

 if (currentBlock.precedingHash !== precedingBlock.hash) return false; }
 return true;
 }
 }

module.exports.CryptoBlockchain= CryptoBlockchain;
module.exports.CryptoBlock= CryptoBlock;
module.exports.Transaction= Transaction;
```

NETWORKOBSERVER.js

```
const {PeerAlert}= require('./PeerAlert');

class NetworkObserver
 {
 constructor(pending_transactions,cbc)
 {
 this.pending_transactions= pending_transactions;
 this.cbc= cbc;
 }
 //Function to get transaction records of all sender addresses in a blockchain
getRecords(sendAddr)
 {


 for(const address of sendAddr)
 {
          var initial_balance=
global.smashingCoin.getBalanceOfAddress(address);
          //Number of Transactions for the blockchain
          var num=0;
          //Total amount to pay based on number of transactions var
          total_payment_amt= 0;
          //Array to store transaction amounts from this address
          var amts= new Array();

          for(const trans of this.pending_transactions){

          if(trans.fromAddress=== address)
          {
          num=num+1;
          total_payment_amt= total_payment_amt + trans.amount;
          amts.push(trans.amount);
          }
          }
          this.check_DoubleSpendingAttack(total_payment_amt,
initial_balance, num, address, amts);
          }
 }



 //Function to check for double spending attacks
 check_DoubleSpendingAttack(total, balance, num, address, amts) {
 console.log("\nNumber of pending transactions from address: ");
```

```
console.log(address+" is:"+ num);
 console.log("\nTotal payment amount is: ("+amts+")");
if(total>balance && num>1)
 {
 console.log("\nYour transaction has been aborted due to a suspected
double-spending attack.");
 console.log("\nPlease cancel your last transaction or try again later.");

 var indices=[];

 for(const trans of this.pending_transactions){
          if(trans.fromAddress=== address)
          {

          indices.push(this.pending_transactions.indexOf(trans));
          }
          }

          const peeralert= new PeerAlert(indices,this.cbc);
          this.cbc= peeralert.process();
 }
 }

}

module.exports.NetworkObserver= NetworkObserver;
```

PEERALERT.js

```
class PeerAlert
{
        constructor(indices, cbc)
 {
                this.indices= indices;
                this.cbc= cbc;
        }
        process(){
                for(const v of this.indices)
 {
                if(v>=1){
                        console.log("Disconnected block number ",v,"due to a
suspected double spending attack.");
                }
                }

                for(const v of this.indices){
                this.cbc.pendingTransactions.splice(v,1);
                if(v<=this.cbc.blockchain.length - 1)
                {
                        this.cbc.blockchain[v+1].precedingHash = null;
```

```
                    this.cbc.blockchain.splice(v,1);
                }
                //this.indices.splice(this.indices.indexOf(v),1);
                for(var i=0; i<this.indices.length; i++)
                {
                this.indices[i]-=1;
                }
                }
                return this.cbc;
        }
}
module.exports.PeerAlert= PeerAlert;
```

KEYGENERATOR.js

```
const EC= require('elliptic').ec;
const ec= new EC('secp256k1');

const key= ec.genKeyPair();
//generate a hexadecimal public key
const publicKey= key.getPublic('hex');
//generate a hexadecimal private key
const privateKey= key.getPrivate('hex');

console.log();
console.log('Private key:', privateKey);

console.log();
console.log('Public key:', publicKey);
```

MAIN.js

```
const {CryptoBlockchain, CryptoBlock , Transaction}=
require('./CryptoBlockchain');
const {NetworkObserver} = require('./NetworkObserver');
const EC= require('elliptic').ec;
const ec= new EC('secp256k1');

const myKey=
ec.keyFromPrivate('7fe38a7cd191b531376ed5db174624ff3a474b736d824d174
677e32aeb703c29');
const Key1=
ec.keyFromPrivate('1c2e962350cbe889eeb39ab45728fa7e59624ad4d0b8af4f25
8a9dd439db3f8f');
const Key2=
ec.keyFromPrivate('af68820d23d0857f56aefba0382a631f6e4d54056cbf2ad8d2f
64fcc7d34441d');
```

```
 const myWalletAddress = myKey.getPublic('hex');
 const address1= Key1.getPublic('hex');
 const address2= Key2.getPublic('hex');


function fetchAddress()
  {
 const key= ec.genKeyPair();
 //generate a hexadecimal public key
 const publicKey= key.getPublic('hex');
 //generate a hexadecimal private key
 const privateKey= key.getPrivate('hex');
 const recvKey= ec.keyFromPrivate(privateKey);
 const recvWalletAddress = recvKey.getPublic('hex');

 return recvWalletAddress;
  }



 //console.log("\n\nInitial balance is: "+ 10);
 const recv_address= fetchAddress();
 new_address= fetchAddress();

 global.smashingCoin = new CryptoBlockchain();

 const sendAddr= [address2, myWalletAddress, myWalletAddress, address1];
 const recvAddr= [fetchAddress(), recv_address, new_address, fetchAddress()];


 const tx3 = new Transaction(address2,fetchAddress(),3);
tx3.signTransaction(Key2);
 global.smashingCoin.addTransaction(tx3);


 const tx1 = new Transaction(myWalletAddress,recv_address,10);
tx1.signTransaction(myKey);
 global.smashingCoin.addTransaction(tx1);


 const dsa_tx = new Transaction(myWalletAddress,new_address,10);
 dsa_tx.signTransaction(myKey);
 global.smashingCoin.addTransaction(dsa_tx);

 const tx2 = new Transaction(address1,fetchAddress(),5);
tx2.signTransaction(Key1);
 global.smashingCoin.addTransaction(tx2);


 console.log("\nThe current blockchain is:");
 console.log(global.smashingCoin);
```

```
console.log("\nStarting the miner...");

for(const addr of sendAddr)

{
global.smashingCoin.minePendingTransactions(addr);
}


const ne= new NetworkObserver(global.smashingCoin.pendingTransactions,
global.smashingCoin);

ne.getRecords(sendAddr);
console.log("\nThe current blockchain is:");
console.log(global.smashingCoin);

for(var block in global.smashingCoin.blockchain)
{
        if(block.precedingHash== null)
        {

        global.smashingCoin.blockchain.splice(global.smashingCoin.blockchain.
indexOf(block),1);
                let cb= new CryptoBlock(0,Date.now(),"0");
                global.smashingCoin.blockchain.push(cb);
        }
}

console.log("Current blockchain is:",global.smashingCoin);

console.log("\nPerforming Transactions:");
console.log("Current balance of address ",address2,"is:
"+smashingCoin.getBalanceOfAddress(address2));
console.log("Current balance of address ",address1,"is:
"+smashingCoin.getBalanceOfAddress(address1));
```

6. Implementation

To start the initial blockchain, run the following command after entering the directory from the command line or terminal:

node CryptoBlockchain.js

6.1 Additional Security Aspects

To increase security, a random nonce generator is used, which will be used in the hash creation. Moreover, to increase the time taken to mine individual blocks, a difficulty index is used to append additional zeros to the generated hash, which increases the mining time complexity. Also to validate the cryptocurrency blockchain, for every pair consisting of the current block and its preceding block, a two-fold validation methodology is utilized. The hash of the current block is recomputed with the same nonce, and the original and recomputed hash values are compared. The preceding hash of the current block and the hash of the preceding block are compared.

If a mismatch is found in either of these two steps, the validation function returns an error.

Here, the red highlights denote the appended zeros based on the difficulty index and the green highlights are the random nonces generated to compute the hash for every new block added.

When we observe and log the output of the check chain validity function, before tampering with any one of the blocks, it returns the value true, implying that the blockchain is valid.

console.log("Is the blockchain valid?"+ smashingCoin.checkChainValidity()); If instead we try to change the transfer quantity of block 1 from 50 to 200 in the chain, the function returns a value of false, implying an authentication error.

```
smashingCoin.blockchain[1].data = {amount: 200};
```

6.2 Performing Transactions

To perform transactions between the intended sender and recipient blocks or nodes, their addresses are mined, such that the transaction amount can be deducted from the sender node and added to the receiver node. An array contains the records of pending transactions for every block starting with a null amount transaction for the genesis block, and subsequently stores every new transaction. An example for demonstration is as follows, the first block sends 100 cryptocurrency units to the second block. The second block in turn returns 50 cryptocurrency units back to the first block. Hence the transaction array will contain the details of the transactions with amounts null, 100 and 50 respectively. After every successful transaction for a particular genesis block, a mining reward of 100 points is added to its balance.

```
smashingCoin.createTransaction( new Transaction("addr1","addr2",100));

smashingCoin.createTransaction( new Transaction("addr2","addr1",50));

console.log("Starting the miner...");

smashingCoin.minePendingTransactions("myAddress");

console.log("Your balance is:

"+smashingCoin.getBalanceOfAddress('myAddress')); console.log("Starting the

miner again...");

smashingCoin.minePendingTransactions("myAddress");

console.log("Your balance is: "+smashingCoin.getBalanceOfAddress('myAddress'));
```

6.3 Signature

To ensure that every transaction is unique, it is signed with a signature which is generated using the private and public key and calculated hash for a block, using the elliptic library (based on elliptic curve cryptography). This is also used as a security measure to authenticate every transaction. The following image describes a transaction of 10 cryptocurrency units from the genesis block, and a reception of 100 cryptocurrency units by the genesis block.

```
const tx1 = new Transaction(myWalletAddress,'public_key',10);

tx1.signTransaction(myKey);



smashingCoin.addTransaction(tx1);

console.log("Starting the miner...");

smashingCoin.minePendingTransactions(myWalletAddress);

console.log("Your balance is:

"+smashingCoin.getBalanceOfAddress(myWalletAddress));

console.log("Starting the miner again...");

smashingCoin.minePendingTransactions(myWalletAddress);

console.log("Your balance is:

"+smashingCoin.getBalanceOfAddress(myWalletAddress));
```

6.4 Double Spending Attack

When a sender tries to perform the same transaction (with the same amount) twice, but with two different receivers, even though his initial balance is not sufficient to satisfy both the transactions, a double spending attack occurs. This can be illustrated in the below example. Consider a sender to have 10 cryptocurrency units, who plans to give these 10 units to 2 different receivers, concurrently, across separate blocks. For both these transactions to occur, a total of 20 units should be available with the sender, whereas he has just 10, implying that he plans to reuse the same money of the first transaction, in the second transaction, such that the 10 units is deducted at the same time (so only 10 units is deducted from his balance finally).

```
const recv_address= fetchAddress();

const tx1 = new Transaction(myWalletAddress,recv_address,10);

tx1.signTransaction(myKey);

global.smashingCoin.addTransaction(tx1);

console.log(global.smashingCoin.pendingTransactions)

new_address= fetchAddress();
```

```
const dsa_tx = new Transaction(myWalletAddress,new_address,10);

dsa_tx.signTransaction(myKey);


global.smashingCoin.addTransaction(dsa_tx);

console.log(global.smashingCoin.pendingTransactions)
```

As seen above, the same sender hash address (highlighted in yellow) is involved in two transactions of 10 cryptocurrency units each (underlined in blue), but with different receiver hash addresses. Thus, if these 2 transactions are performed simultaneously (i.e. at the same timestamp), only 10 units will be deducted from his balance at the same time, whereas his actual deduction amount should be 20. The sender can thus exploit this issue to his advantage, and perform many such transactions, but by paying only half the actual amount.

6.5 Network Observers

A network observer will check the number of pending transactions from a particular sender address in a blockchain. If the payment amount of two pending transactions from that address is the same, and the total amount is greater than the initial balance of the sender, a transaction aborted error will be displayed, and the user will be prompted to either cancel the last transaction with the same amount, or to try performing the transaction later.

```
if(total>balance && num>1)

 {

console.log("\nYour transaction has been aborted due to a suspected double spending

attack.");

console.log("\nPlease cancel your last transaction or try again later."); }
```

6.6 Peer Alert Systems

In order to disconnect from the block where the fraudulent transaction or double spending attack has taken place, the neighbouring blocks are alerted. Any subsequent transactions that are scheduled for this block are redirected to other routes, and these neighbouring blocks terminate their connections with it. The sender (who wanted to conduct the double spending attack), has to either withdraw any one of the transactions within a given timeout period, or will be blocked temporarily before being able to perform any new transactions again.

7. Result and Discussion

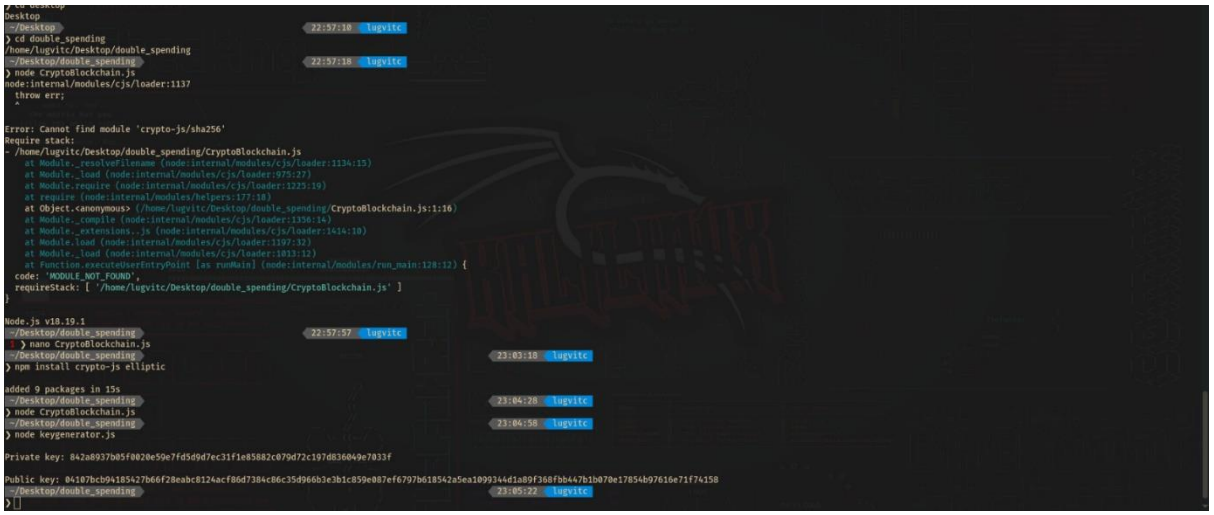
- Creating four new transactions in the blockchain
- Displaying the current blockchain with the genesis block
- Mining blocks to perform the transactions
- A double spending attack occurs in the blockchain.
- The network observer throws an error, asking the user to delete the last transaction, or be temporarily suspended.
- After this, the peer alert system will terminate connections of the other blocks to the fraudulent blocks (alerting neighboring blocks)

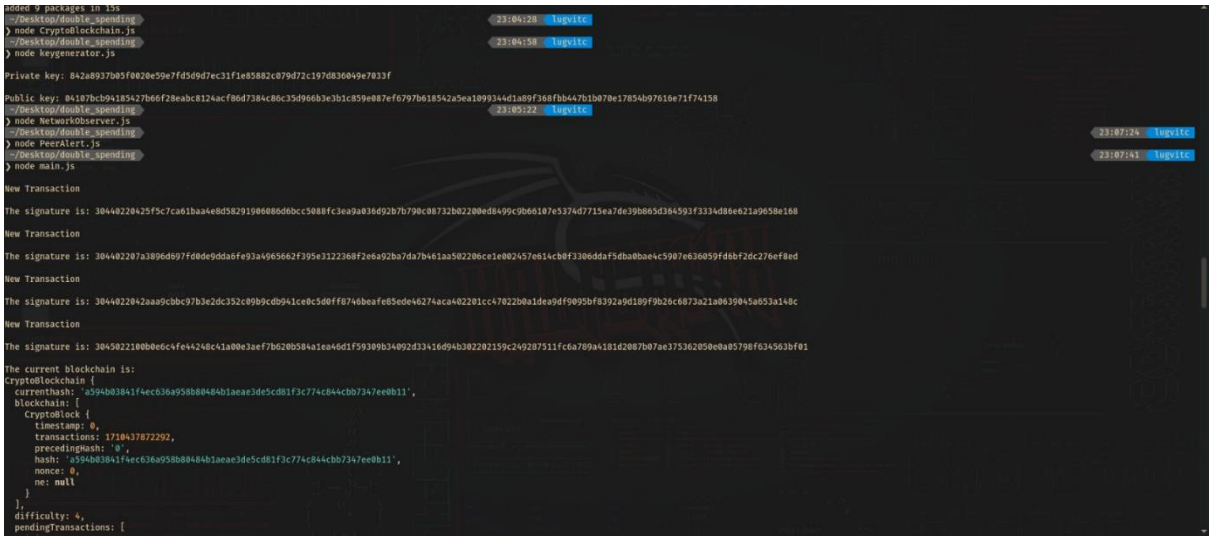
Therefore, due to the double spending attack, the number of transactions reduces from four to two (the ones involved in the attack are aborted). The peer alert system will nullify the preceding hash of blocks that had fraudulent blocks preceding them, due to the fact that the preceding hash is used as one of the parameters to calculate the current hash of any block.

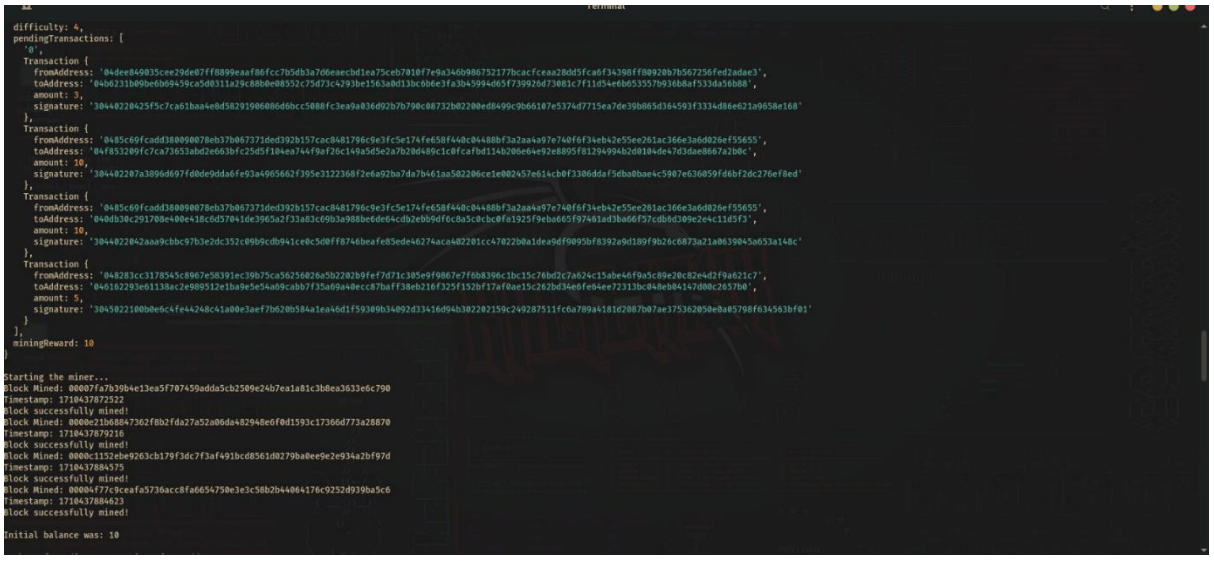
To overcome this issue of having a null preceding hash, this block is replaced with a new genesis block, with a preceding hash of '0'. This will also lead to a new current hash being generated, that can be used to perform the same transaction as before (Change of transaction route).

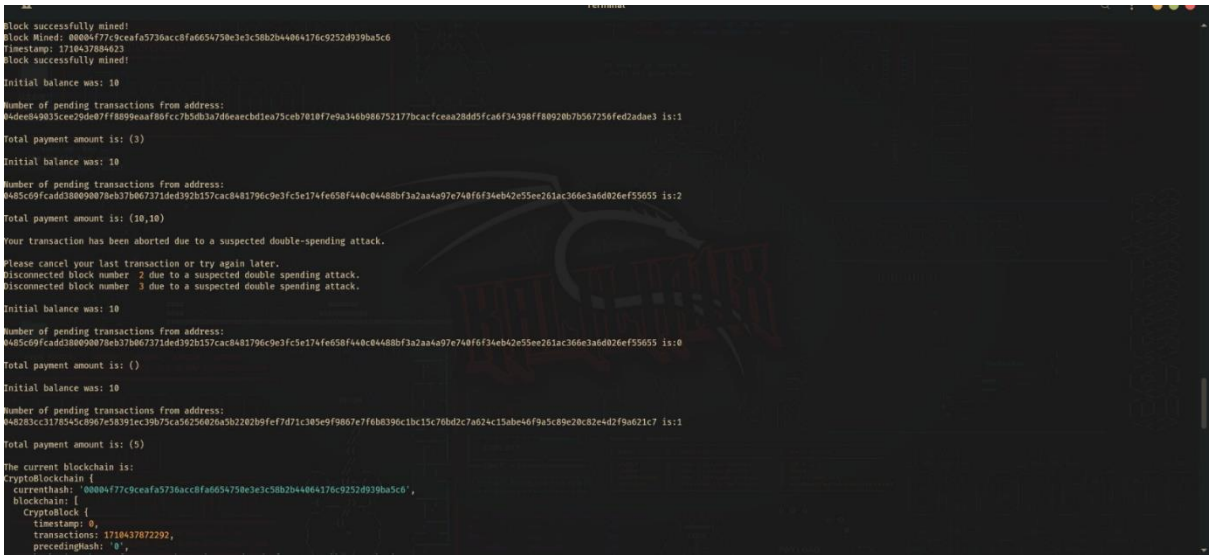
Finally, after only genuine transactions are contained in the blockchain, they are processed, and the balances of each of the addresses are updated accordingly.

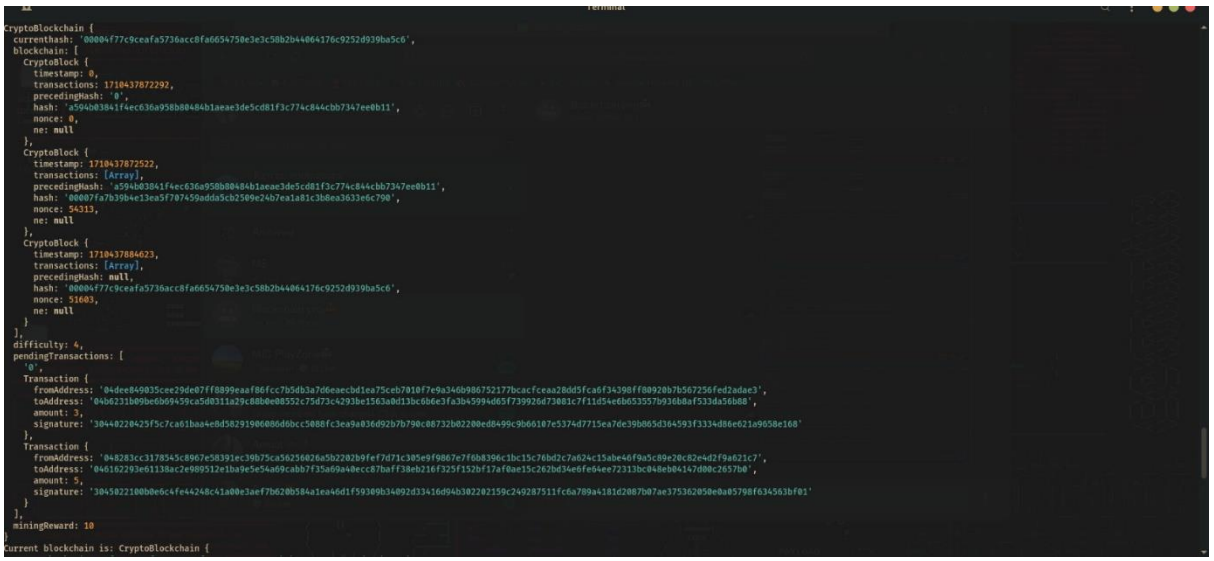
Appendix 1


Desktop
~/Desktop 22:57:10 lugvitc
❯ cd double_spending
/home/lugvitc/Desktop/double_spending
~/Desktop/double_spending 22:57:18 lugvitc
❯ node CryptoBlockchain.js
node:internal/modules/cjs/loader:1137
throw err;
^
Error: Cannot find module 'crypto-js/sha256'
Require stack:
- /home/lugvitc/Desktop/double_spending/CryptoBlockchain.js
at Module._resolveFilename (node:internal/modules/cjs/loader:1134:15)
at Module._load (node:internal/modules/cjs/loader:975:27)
at Module.require (node:internal/modules/cjs/loader:1225:19)
at require (node:internal/modules/helpers:177:18)
at Object.<anonymous> (/home/lugvitc/Desktop/double_spending/CryptoBlockchain.js:1:16)
at Module._compile (node:internal/modules/cjs/loader:1358:14)
at Module._extensions..js (node:internal/modules/cjs/loader:1414:10)
at Module.load (node:internal/modules/cjs/loader:1197:32)
at Module._load (node:internal/modules/cjs/loader:1013:12)
at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:128:12) {
code: 'MODULE_NOT_FOUND',
requireStack: ['/home/lugvitc/Desktop/double_spending/CryptoBlockchain.js']
}
Node.js v18.19.1
~/Desktop/double_spending 22:57:57 lugvitc
1 ❯ nano CryptoBlockchain.js
~/Desktop/double_spending 23:03:18 lugvitc
❯ npm install crypto-js elliptic
added 9 packages in 15s
~/Desktop/double_spending 23:04:28 lugvitc
❯ node CryptoBlockchain.js
~/Desktop/double_spending 23:04:58 lugvitc
❯ node keygenerator.js
Private key: 842a8937b05f0020e59e7fd5d9d7ec31f1e85882c079d72c197d836049e7033f
Public key: 04107bcb94185427b66f28eabc8124acf86d7384c86c35d966b3e3b1c859e087ef6797b618542a5ea1099344d1a89f368fbb447b1b070e17854b97616e71f74158
~/Desktop/double_spending 23:05:22 lugvitc
❯


added 9 packages in 15s
~/Desktop/double_spending 23:04:28 lugvitc
❯ node CryptoBlockchain.js
~/Desktop/double_spending 23:04:58 lugvitc
❯ node keygenerator.js
Private key: 842a8937b05f0020e59e7fd5d9d7ec31f1e85882c079d72c197d836049e7033f
Public key: 04107bcb94185427b66f28eabc8124acf86d7384c86c35d966b3e3b1c859e087ef6797b618542a5ea1099344d1a89f368fbb447b1b070e17854b97616e71f74158
~/Desktop/double_spending 23:05:22 lugvitc
❯ node NetworkObserver.js
~/Desktop/double_spending 23:07:24 lugvitc
❯ node PeerAlert.js
~/Desktop/double_spending 23:07:41 lugvitc
❯ node main.js
New Transaction
The signature is: 3044022042 5f5c7ca61baa4e8d5829190608 6d6bcc5088fc3ea9a036d92b7b790c08732b02200ed8499c9b66107e5374d7715ea7de39b865d364593f3334d86e621a9658e168
New Transaction
The signature is: 304402207a3896d697fd0de9dda6fe93a4965662f395e3122368f2e6a92ba7da7b461aa502206ce1e002457e614cb0f3306ddaf5dba0bae4c5907e636059fd6bf2dc276ef8ed
New Transaction
The signature is: 304402204 2aaa9cbbc97b3e2dc352c09b9cdb941ce0c5d0ff8746beafe85ede46274aca402201cc47022b0a1dea9df9095bf8392a9d189f9b26c6873a21a0639045a653a148c
New Transaction
The signature is: 3045022100b0e6c4fe44248c41a00e3aef7b620b584a1ea46d1f59309b34092d33416d94b302202159c249287511fc6a789a4181d2087b07ae37536205 0e0a05798f634563bf01
The current blockchain is:
CryptoBlockchain {
currenthash: 'a594b03841f4ec636a958b88484b1aeae3de5cd81f3c774c844cbb7347ee0b11',
blockchain: [
CryptoBlock {
timestamp: 0,
transactions: 1710437872292,
precedingHash: '0',
hash: 'a594b03841f4ec636a958b88484b1aeae3de5cd81f3c774c844cbb7347ee0b11',
nonce: 0,
ne: null
}
],
difficulty: 4,
pendingTransactions: [


difficulty: 4,
pendingTransactions: [
'0',
Transaction {
fromAddress: '04dee849035cee29de07ff8899eaaf86fcc7b5db3a7d6eaecbd1ea75ceb7010f7e9a346b986752177bcacfceaa28dd5fca6f34398ff88920b7b567256fed2adae3',
toAddress: '04b6231b09be6b69459ca5d0311a29c88b0e08552c75d73c4293be1563a0d13bc6b6e3fa3b45994d65f739926d73081c7f11d54e6b653557b936b8af533da56b88',
amount: 3,
signature: '3044022042 5f5c7ca61baa4e8d5829190608 6d6bcc5088fc3ea9a036d92b7b790c08732b02200ed8499c9b66107e5374d7715ea7de39b865d364593f3334d86e621a9658e168'
},
Transaction {
fromAddress: '0485c69fcadd380090078eb37b067371ded392b157cac8481796c9e3fc5e174fe658f440c04488bf3a2aa4a97e740f6f34eb42e55ee261ac366e3a6d026ef55655',
toAddress: '04f853209fc7ca73653abd2e663bfc25d5f104ea744f9af26c149a5d5e2a7b20d489c1c0fcafbd114b206e64e92e8895f81294994b2d0104de47d3dae8667a2b0c',
amount: 10,
signature: '304402207a3896d697fd0de9dda6fe93a4965662f395e3122368f2e6a92ba7da7b461aa502206ce1e002457e614cb0f3306ddaf5dba0bae4c5907e636059fd6bf2dc276ef8ed'
},
Transaction {
fromAddress: '0485c69fcadd380090078eb37b067371ded392b157cac8481796c9e3fc5e174fe658f440c04488bf3a2aa4a97e740f6f34eb42e55ee261ac366e3a6d026ef55655',
toAddress: '040db30c291708e408e418c6d57041de3965a2f33a83c69b3a988be6de64cdb2eb89df6c8a5c0cbc0fa1925f9eba665f97461ad3ba66f57cdb0d309e2e4c11d5f3',
amount: 10,
signature: '304402204 2aaa9cbbc97b3e2dc352c09b9cdb941ce0c5d0ff8746beafe85ede46274aca402201cc47022b0a1dea9df9095bf8392a9d189f9b26c6873a21a0639045a653a148c'
},
Transaction {
fromAddress: '048283cc3178545c8967e58391ec39b75ca56256026a5b2202b9fef7d71c305e9f9867e7f6b8306c3bc15c76bd2c7a624c15abe46f9a5c89e20c82e4d2f9a621c7',
toAddress: '0461022993e61138ac2e909512e1ba8e5e54a69cabb7f35a69a40ecc87baff38eb216f325f152bf17af0ae15c262bd34e6fe64ee72313bc04beb04147d00c2657b0',
amount: 5,
signature: '3045022100b0e6c4fe44248c41a00e3aef7b620b584a1ea46d1f59309b34092d33416d94b302202159c249287511fc6a789a4181d2087b07ae37536205 0e0a05798f634563bf01'
}
],
miningReward: 10
}
Starting the miner...
Block Mined: 00007fa7b39b4e13ea5f707459adda5cb2509e24b7ea1a81c3bBea3633e6c790
Timestamp: 1710437872522
Block successfully mined!
Block Mined: 0000e21b68847362f8b2fda27a52a06da48294 8e6f0d1593c17366d773a28870
Timestamp: 1710437879216
Block successfully mined!
Block Mined: 0000c1152ebe9263cb179f3dc7f3af491bcd8561d0279ba0ee9e2e934a2bf97d
Timestamp: 1710437884575
Block successfully mined!
Block Mined: 00004f77c9ceafa5736acc8fa6654750e3e3c58b2b44064176c9252d939ba5c6
Timestamp: 1710437884623
Block successfully mined!
Initial balance was: 10


Block successfully mined!
Block Mined: 00004f77c9ceafa5736acc8fa6654750e3e3c58b2b44064176c9252d939ba5c6
Timestamp: 1710437884623
Block successfully mined!
Initial balance was: 10
Number of pending transactions from address:
04dee849035cee29de07ff8899eaaf86fcc7b5db3a7d6eaecbd1ea75ceb7010f7e9a346b986752177bcacfceaa28dd5fca6f34398ff80920b7b567256fed2adae3 is:1
Total payment amount is: (3)
Initial balance was: 10
Number of pending transactions from address:
0485c69fcadd38090078eb37b067371ded392b157cac8481796c9e3fc5e174fe658f440c04488bf3a2aa4a97e740f6f34eb42e55ee261ac366e3a6d026ef55655 is:2
Total payment amount is: (10,10)
Your transaction has been aborted due to a suspected double-spending attack.
Please cancel your last transaction or try again later.
Disconnected block number 2 due to a suspected double spending attack.
Disconnected block number 3 due to a suspected double spending attack.
Initial balance was: 10
Number of pending transactions from address:
0485c69fcadd38090078eb37b067371ded392b157cac8481796c9e3fc5e174fe658f440c04488bf3a2aa4a97e740f6f34eb42e55ee261ac366e3a6d026ef55655 is:0
Total payment amount is: ()
Initial balance was: 10
Number of pending transactions from address:
048283cc3178545c8967e58391ec39b75ca56256026a5b2202b9fef7d71c305e9f9867e7f6b8396c1bc15c76bd2c7a624c15abe46f9a5c89e20c82e4d2f9a621c7 is:1
Total payment amount is: (5)
The current blockchain is:
CryptoBlockchain {
currenthash: '00004f77c9ceafa5736acc8fa6654750e3e3c58b2b44064176c9252d939ba5c6',
blockchain: [
CryptoBlock {
timestamp: 0,
transactions: 1710437872292,
precedingHash: '0',


CryptoBlockchain {
currenthash: '00004f77c9ceafa5736acc8fa6654750e3e3c58b2b44064176c9252d939ba5c6',
blockchain: [
CryptoBlock {
timestamp: 0,
transactions: 1710437872292,
precedingHash: '0',
hash: 'a594b03841f4ec636a958b80484b1aeae3de5cd81f3c774c844cbb7347ee0b11',
nonce: 0,
ne: null
},
CryptoBlock {
timestamp: 1710437872522,
transactions: [Array],
precedingHash: 'a594b03841f4ec636a958b80484b1aeae3de5cd81f3c774c844cbb7347ee0b11',
hash: '00007fa7b39b4e13ea5f707459adda5cb2509e24b7ea1a81c3b8ea3633e6c790',
nonce: 54313,
ne: null
},
CryptoBlock {
timestamp: 1710437884623,
transactions: [Array],
precedingHash: null,
hash: '00004f77c9ceafa5736acc8fa6654750e3e3c58b2b44064176c9252d939ba5c6',
nonce: 51603,
ne: null
}
],
difficulty: 4,
pendingTransactions: [
'0',
Transaction {
fromAddress: '04dee849035cee29de07ff8899eaaf86fcc7b5db3a7d6eaecbd1ea75ceb7010f7e9a346b986752177bcacfceaa28dd5fca6f34398ff80920b7b567256fed2adae3',
toAddress: '04b6231b09be6b69459ca5d0311a29c88b0e08552c75d73c4293be1563a0d13bc6b6e3fa3b45994d65f739926d73081c7f11d54e6b653557b936b8af533da56b88',
amount: 3,
signature: '3044022042 5f5c7ca61baa4e8d5829190 6086d6bcc5088fc3ea9a036d92b7b790c08732b02200ed8499c9b66107e5374d7715ea7de39b865d364593f3334d86e621a9658e168'
},
Transaction {
fromAddress: '048283cc3178545c8967e58391ec39b75ca56256026a5b2202b9fef7d71c305e9f9867e7f6b8396c1bc15c76bd2c7a624c15abe46f9a5c89e20c82e4d2f9a621c7',
toAddress: '046162293e61138ac2e980512e1ba9e5e54a69cabb7f35a69a40ecc87baff38eb216f325f152bf17af0ae15c262bd34e6fe64ee72313bc848eb04147d80c2657b0',
amount: 5,
signature: '3045022100b0e6c4fe44248c41a00e3aef7b620b584a1ea46d1f59309b34092d33416d94b302202159c249287511fc6a789a4181d2087b07ae375362050e0a05798f634563bf01'
}
],
miningReward: 10
}
Current blockchain is: CryptoBlockchain {


}
Current blockchain is: CryptoBlockchain {
currenthash: '00004f77c9ceafa5736acc8fa6654750e3e3c58b2b44064176c9252d939ba5c6',
blockchain: [
CryptoBlock {
timestamp: 0,
transactions: 1710437872292,
precedingHash: '0',
hash: 'a594b03841f4ec636a958b80484b1aeae3de5cd81f3c774c844cbb7347ee0b11',
nonce: 0,
ne: null
},
CryptoBlock {
timestamp: 1710437872522,
transactions: [Array],
precedingHash: 'a594b03841f4ec636a958b80484b1aeae3de5cd81f3c774c844cbb7347ee0b11',
hash: '00007fa7b39b4e13ea5f707459adda5cb2509e24b7ea1a81c3b8ea3633e6c790',
nonce: 54313,
ne: null
},
CryptoBlock {
timestamp: 0,
transactions: 1710437890716,
precedingHash: '0',
hash: '488104b667e29fd382026818cced09a2357a8d91e5e088f430c5f74ab914cdf7',
nonce: 0,
ne: null
}
],
difficulty: 4,
pendingTransactions: [
'0',
Transaction {
fromAddress: '04dee849035cee29de07ff8899eaaf86fcc7b5db3a7d6eaecbd1ea75ceb7010f7e9a346b986752177bcacfceaa28dd5fca6f34398ff80920b7b567256fed2adae3',
toAddress: '04b6231b09be6b69459ca5d0311a29c88b0e08552c75d73c4293be1563a0d13bc6b6e3fa3b45994d65f739926d73081c7f11d54e6b653557b936b8af533da56b88',
amount: 3,
signature: '3044022042 5f5c7ca61baa4e8d5829190 6086d6bcc5088fc3ea9a036d92b7b790c08732b02200ed8499c9b66107e5374d7715ea7de39b865d364593f3334d86e621a9658e168'
},
Transaction {
fromAddress: '048283cc3178545c8967e58391ec39b75ca56256026a5b2202b9fef7d71c305e9f9867e7f6b8396c1bc15c76bd2c7a624c15abe46f9a5c89e20c82e4d2f9a621c7',
toAddress: '046162293e61138ac2e980512e1ba9e5e54a69cabb7f35a69a40ecc87baff38eb216f325f152bf17af0ae15c262bd34e6fe64ee72313bc848eb04147d80c2657b0',
amount: 5,
signature: '3045022100b0e6c4fe44248c41a00e3aef7b620b584a1ea46d1f59309b34092d33416d94b302202159c249287511fc6a789a4181d2087b07ae375362050e0a05798f634563bf01'
}
],
miningReward: 10
}

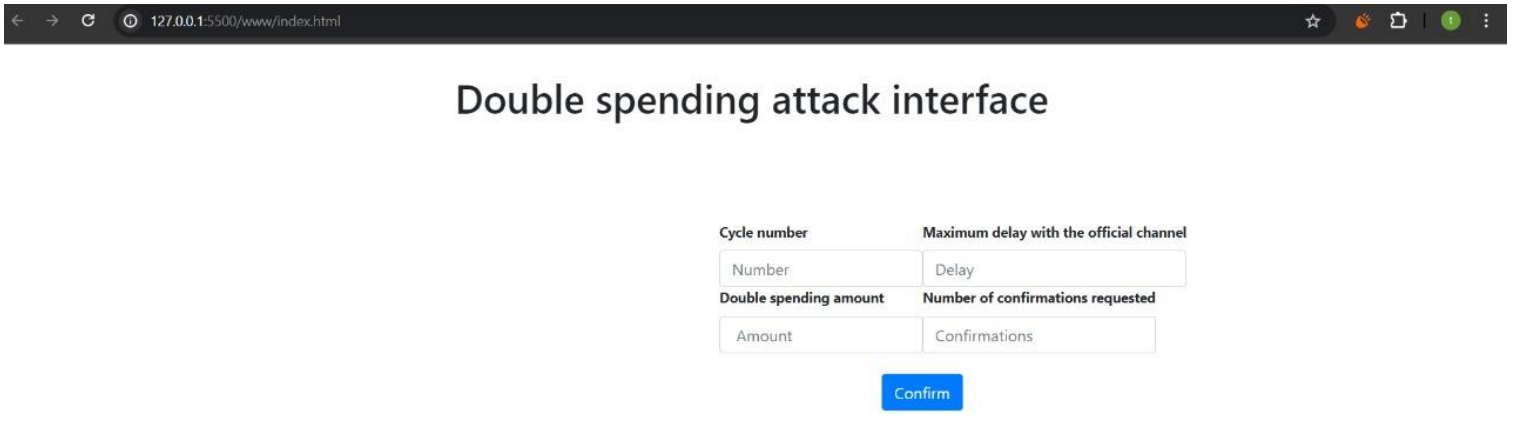
8 FRONTEND INTERFACE

The website serves as an interactive simulation interface for exploring and understanding double spend attacks within blockchain systems. Users can input parameters such as the number of simulation cycles, maximum delay with the official chain, amount of double spending, and required number of confirmations. Upon initiating the simulation, the website visualizes the results through a line chart, depicting metrics such as the number of successful double spend attempts, time taken for each attempt, and comparison of confirmations obtained. This tool facilitates learning, analysis, and experimentation with double spend attacks, enabling users to grasp their implications on blockchain security and explore strategies for mitigation.

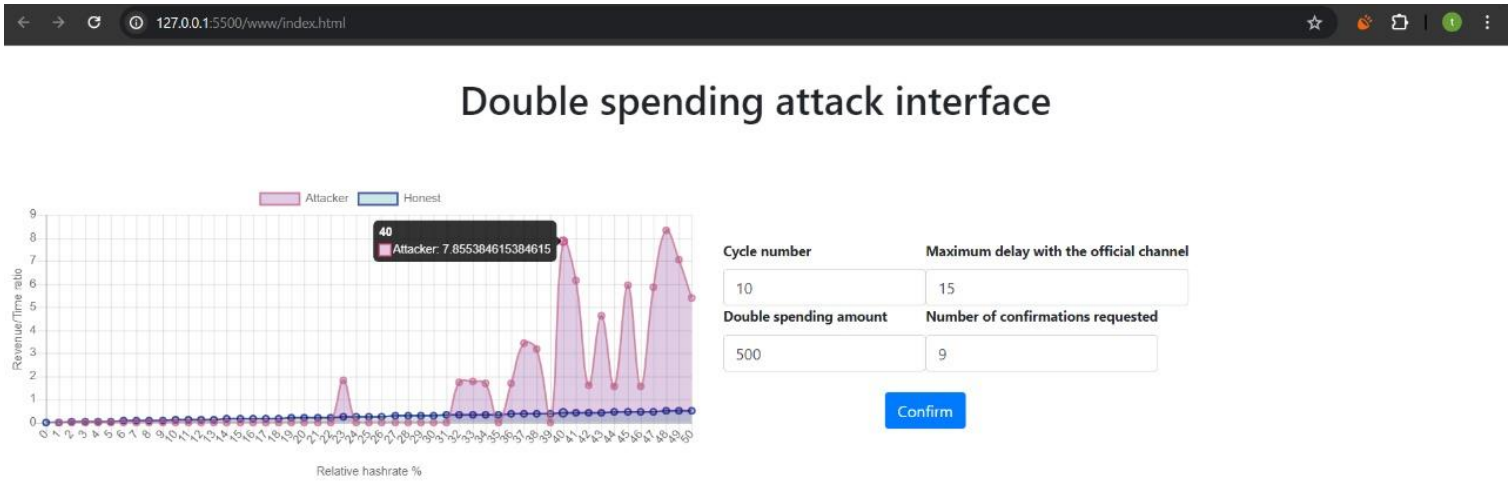
8.1 Functionality:

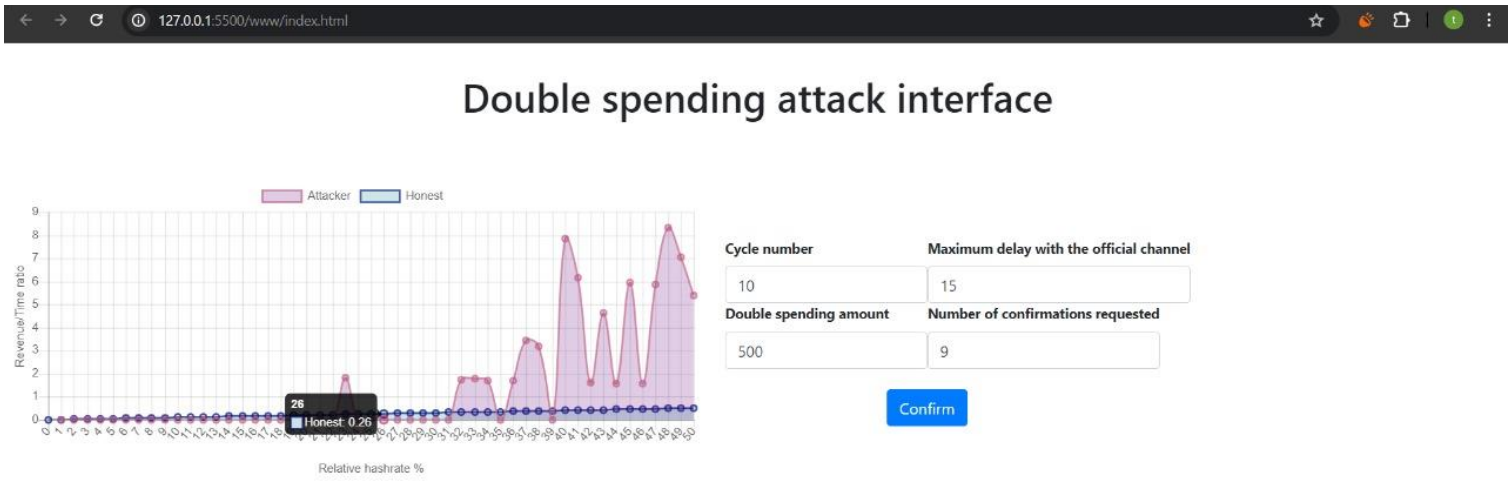
8.1.1. Simulation Interface : The website provides an interface for users to simulate double spend attacks in blockchain systems.
8.1.2. Parameter Input : Users can input parameters such as the number of simulation cycles, maximum delay with the official chain, amount of double spending, and required number of confirmations.
8.1.3. Simulation Process : Upon clicking the "Confirmer" button, the website triggers the simulation process using the provided parameters.
8.1.4. Chart Visualization : The results of the simulation are visualized through a line chart, displayed using Chart.js.
8.1.5. Chart Metrics : The line chart depicts metrics such as the number of successful double spend attempts, time taken for each attempt, and comparison of confirmations obtained.
8.1.6. Analysis and Understanding : Users can analyze the chart to understand the behavior and impact of double spend attacks on blockchain systems.

Appendix 2




127.0.0.1:5500/www/index.html
Double spending attack interface
Attacker
Honest
Revenue/Time ratio
Relative hashrate %
Cycle number
Maximum delay with the official channel
10
15
Double spending amount
Number of confirmations requested
500
9
Confirm


127.0.0.1:5500/www/index.html
Double spending attack interface
Attacker
Honest
40
Attacker: 7.855384615384615
Revenue/Time ratio
Relative hashrate %
Cycle number
Maximum delay with the official channel
10
15
Double spending amount
Number of confirmations requested
500
9
Confirm


127.0.0.1:5500/www/index.html
Double spending attack interface
Attacker
Honest
26
Honest: 0.26
Revenue/Time ratio
Relative hashrate %
Cycle number
Maximum delay with the official channel
10
15
Double spending amount
Number of confirmations requested
500
9
Confirm

8. References

1. Singh, S., Hosen, A. S., & Yoon, B. (2021). Blockchain security attacks, challenges, and solutions for the future distributed iot network. IEEE Access, 9, 13938-13959.
2. Begum, A., Tareq, A., Sultana, M., Sohel, M., Rahman, T., & Sarwar, A. (2020). Blockchain attacks analysis and a model to solve double spending attack. International Journal of Machine Learning and Computing, 10(2), 352-357.
3. Karame, G. O., Androulaki, E., Roeschlin, M., Gervais, A., & Čapkun, S. (2015). Misbehavior in bitcoin: A study of double-spending and accountability. ACM Transactions on Information and System Security (TISSEC), 18(1), 1-32.
4. Saad, M., Spaulding, J., Njilla, L., Kamhoua, C., Shetty, S., Nyang, D., & Mohaisen, D. (2020). Exploring the attack surface of blockchain: A comprehensive survey. IEEE Communications Surveys & Tutorials, 22(3), 1977-2008.
5. Chen, Y., Chen, H., Zhang, Y., Han, M., Siddula, M., & Cai, Z. (2022). A survey on blockchain systems: Attacks, defenses, and privacy preservation. High-Confidence Computing, 2(2), 100048.
6. Akbar, N. A., Muneer, A., ElHakim, N., & Fati, S. M. (2021). Distributed hybrid double-spending attack prevention mechanism for proof-of-work and proof-of-stake blockchain consensuses. Future Internet, 13(11), 285.
7. Nicolas, K., & Wang, Y. (2019, October). A novel double spending attack countermeasure in blockchain. In 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON) (pp. 0383-0388). IEEE.
8. Podolanko, J. P., Ming, J., & Wright, M. (2017, April). Countering double-spend attacks on bitcoin fast-pay transactions. In Proc. Workshop Technol. Consum. Protection (pp. 1-3).
9. https://discovery.ucl.ac.uk/id/eprint/10063353/1/iacr.pdf
10. https://ieeexplore.ieee.org/document/8992991
11. http://www.ijmlc.org/vol10/942-M114.pdf