MITIGATION OF DOUBLE SPENDING ATTACK IN CRYPTOCURRENCY BLOCKCHAIN

by

Laksha S (20BAI1186)

Rohan Alroy (20BAI1245)

Shangirne (20BAI1154)

A project report submitted to

Dr. Anusha K

Associate Professor, School of Computer Science and Engineering

in partial fulfilment of the requirements for the course of

CSE3501 – INFORMATION SECURITY ANALYSIS AND AUDIT

in

B. TECH., COMPUTER SCIENCE AND ENGINEERING

Vellore Institute of Technology, Chennai
Vandalur – Kelambakkam Road
Chennai – 600 127

November 2022

Chennai

BONAFIDE CERTIFICATE

This is to certify that the Project work titled **"Mitigation of double spending attack in cryptocurrency blockchain"** that is being submitted by 20BAI1186 Laksha S, 20BAI1245 Rohan Alroy B, 20BAI1154 Shangirne, is in partial fulfillment of the requirements for the award of **Bachelor of Technology in Computer Science and Engineering**, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

Dr. Anusha K

Guide

ABSTRACT

The basic idea behind our service is to alert senders and recipients of fraudulent transactions. Messages are relayed from rogue nodes to sender and receiver nodes in peer alerting systems. The risk of double spending is that a "digital currency" such as Bitcoin or Dogecoin will be used twice for him. This is a natural barrier to digital currencies because digital information can be easily replicated by knowledgeable people who understand the blockchain network and the computational power required to run it. A copy of the digital token will most likely be created and sent to the merchant or other party, while the original token will be kept. Bitcoin was the first major digital currency to address the issue of duplicate spending. This was accomplished by implementing this validation methodology and maintaining a common global ledger system. In this way, the Bitcoin blockchain has tracked time-stamped transactions since the cryptocurrency's inception in 2009.

The blockchain, which underpins digital currencies such as Bitcoin, is incapable of preventing double spending. All recent transactions are recorded in blocks, similar to a stock exchange's trading book. Every few minutes, information from blocks in the ledger is updated, and every node in the network keeps a copy of the blockchain ledger. In every transaction, high-level encryption protects the identities of buyers and sellers, and the ledger itself cannot be altered by outside sources. When the blockchain ledger is updated, all Bitcoin wallets are updated. Rather, each cryptocurrency's transactions are recorded on the blockchain, where they are individually verified and protected by a confirmation process. Bitcoin and many other cryptocurrencies are irreversible when confirmed in this manner. They have been published and will be kept forever.

Keywords : Block chain, cryptocurrency, double spending attack, network observer, peer alert system.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Objective and goal of the project

Our basic idea is to alert senders and recipients of fraudulent transactions. Peer alerting systems are designed to relay messages from rogue nodes to sender and receiver nodes. Double spending implies the possibility that a "digital currency" such as Bitcoin or Dogecoin will be used twice for him. This is an inherent barrier to digital currencies because digital information can be easily replicated by knowledgeable people who understand blockchain networks and the computational power required to run them. With digital currency, the owner is more likely to make a duplicate of his digital token and send it to a merchant or other party while keeping the original token.

1.2 Problem Statement

Our goal is to design a network observer and peer alert system. A network observer is placed between each pair of nodes. If a network observer notices that one of the transaction steps between a pair of nodes is anomalous, the surrounding nodes are alerted to the bad transaction. This allows you to make transactions in a different and safer way. Generating hashes for a series of Bitcoin transactions is cheap on modern computers, making the process particularly difficult to make work on the Bitcoin network. This option changes to mine a new block approximately every 10 minutes. In other words, it is added to the blockchain by creating a valid hash. A hash's complexity is determined by its purpose. The lower the target, the narrower the set of valid hashes and the harder it is to create hashes. In practice this means a hash starting with a long string of zeros. For example, the block contains slightly more than 1000 total Bitcoin transactions from 2012 and the header of the previous block. If the user changes the transaction amount by his 0.0001 bitcoin, the resulting hash becomes unrecognizable. Bitcoin and other decentralized digital currencies use a consensus mechanism to ensure transactions are confirmed.

A working confirmation mechanism is another name for a consensus mechanism. In effect, this procedure ensures that each transaction participant confirms the transaction. As a result, Bitcoin contains a historical public ledger backed by the blockchain, allowing empirical proof of ownership and transmission. Illegally double-spending a transaction requires market participants to expend significant computing power to remove previous blocks in the chain and effectively double-spend the transaction. Additionally, the number of block confirmations increases exponentially over time, further ensuring transaction integrity.

Our project is unique in terms of security, transaction, signature, network observer and peer alert system. This indicates that a double-spending attack on prompt payments is likely to succeed and can be performed at minimal cost if no additional detection algorithms were added to the Bitcoin implementation. Provides a new lightweight countermeasure that can detect double-spending attacks on high-speed transactions. Accountability becomes important when faced with such misconduct. Explain how accountability and privacy go hand in hand in the case of Bitcoin. To explain this contradiction, we define Bitcoin accountability and privacy and analyze Bitcoin's privacy and accountability mechanisms analytically and empirically. We believe this will reduce our reliance on transaction confirmations when it comes to expedited payment scenarios. I checked. Here is the first implementation of a double-spending attack against fast Bitcoin payments by a small number of hosts around the world. We used a systematic approach to summarize and evaluate on-chain mitigation strategies against double-spending attacks. We also considered implications for future research to mitigate double-spending attacks.

1.3 Motivation

The goal of this project was to use blockchain technology to address a current issue. It turns out that double-spend attacks are a big issue and that hackers can easily take advantage of them. As a result, we have created a method that not only identifies double-spending attacks but also notifies the system as a whole of rogue nodes. This method

makes use of network observers and an alert system. We seek out and put forth a solution to an existing problem.

1.4 Challenges

Since blockchain is a relatively new idea, nobody was familiar with it. So, we first had to read up and conduct research on blockchain. The next step was to examine double spending assaults, including how they happen and how to prevent them. We had to come up with a remedy for We had to conduct extensive research because there aren't many resources available in this field. One of the main difficulties we encountered was this.

CHAPTER II

RELATED AND EXISTING WORKS

2.1 Blockchain System Defensive Overview for Double-Spend and Selfish Mining Attacks: A Systematic Approach

Objective: Blockchain is a technology that creates a decentralised, distributed database of records to safeguard data security. Blockchain-based solutions have been used to secure data in the fields of the Internet of Things, software engineering, healthcare systems, financial services, and smart power grids. The security of the blockchain technology is still a significant issue, though. In order to successfully safeguard the blockchain system, they have taken the initiative to conduct a thorough analysis of what defensive strategies are used. Additionally, they have focused on blockchain data security in particular with the intention of minimising the two data consistency attacks—selfish mining and double-spend.

Observation: Attackers have found novel and practical ways to compromise blockchain's integrity as a result of the security features and technological breakthroughs now present in the blockchain network. In order to enhance blockchain and make it a more resilient network that serves the blockchain community, this study aims to better comprehend 9 and discover the many advantages and disadvantages of alternative countermeasures.

Limitations: However, there are a number of false negatives and false positives produced by honest forks when evaluating the detection accuracy. Future efforts to improve this will make use of machine learning to boost the effectiveness of this monitoring strategy.

2.2 Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin

Objective: Here, they have looked at the security of utilising Bitcoin for speedy payments, where the time between exchanging money for goods is only a matter of

seconds. They primarily concentrate on double spending attacks on instant payments and demonstrate how easily these attacks can be executed on current Bitcoin iterations. Finally, they present a straightforward defence against rapid double-spending attacks.

Observation: They have shown that the Bitcoin network can withstand such bad behaviour so long as there are more good peers than bad peers who are working together. Additionally, they have produced a list of the steps peers do when receiving a Bitcoin payment. They also examined two strategies for preventing double spending that are promoted by Bitcoin developers: using a listening period and adding observers. They've also proposed a successful countermeasure based on alert message propagation, and we demonstrate that it doesn't call for significant changes to current clients. Overall, based on their research, they have proposed a simple mechanism that would enable the secure and imperfect verification of Bitcoin transactions.

Limitations: It is impossible to detect double-spending without state-of-the-art technology because we know that Bitcoin peer connectivity varies greatly with network churn. As a result, "V" must constantly check its connection count to ensure that it does not fall below a threshold that allows it to do so.

2.3 Recipient-Oriented Transaction for Preventing Double Spending Attacks in Private Blockchain

Objective: In this study, a novel method for preventing double-spending assaults is proposed: recipient-oriented notions in private blockchain networks. Transactional privacy is ensured along with the ideas of a stealth address and a master node at the recipient sides, and transaction recipients become active which gets transaction propagation unilaterally. The recipient-oriented mitigation strategy for double-spending attacks based on this technology is given in this work. The proposed solution can also prevent double spending attacks by leveraging the recipient's verification time and the transaction's blocking time.

Observation: The transaction is only carried out if the recipient decides it is legitimate within a predetermined window of time. With the exception of the receiver being involved in transaction execution and verification, this approach is exactly like the conventional blockchain. To stop fraudulent recipients from holding up transactions, a set waiting period is established. The transaction is then automatically repaid to the sender. In order to prevent the issue of double spending, this paper suggests recipient-oriented transactions, which operate as a check on transaction propagation. Instead of standard blockchains that accept transaction propagation unilaterally, stealth addresses and masternodes protect transactional privacy and turn transaction recipients into active participants. The recipient has access to the transaction and confirms it. After that, depending on the outcome, the recipient changes the transaction's status to approved and locked.

Limitations: The transaction's state is then known to the sender, but he or she is not aware of the broadcast time that is utilised to spread the transaction over the network. Therefore, double 12 spending attacks can be stopped by utilising the recipient's verification time and the transaction's blocking time. This paper does not evaluate performance using a software prototype that is used in the real world.

2.4 Misbehavior in Bitcoin: A Study of Double-Spending and Accountability

Objective: They have demonstrated that unless additional detection algorithms are added to the Bitcoin implementation, double-spending attacks on quick transfers are likely to succeed and may be carried out cheaply. In order to combat the risk of double-spending in rapid transactions, they have developed a novel, portable defence. Accountability becomes crucial in the face of such misbehaviour. They provided an example of how privacy and responsibility may coexist in the Bitcoin environment. They gave a privacy and accountability definition for bitcoin to show this contradiction, and we look at the privacy and accountability rules in bitcoin analytically and empirically.

Observation: They tested and graded Bitcoin's resistance to double spending in instantaneous payments in their report. They demonstrated that, in contrast to popular opinion, these assaults do not come at a great cost to the attacker and also have a high possibility of success. They have also offered a simple defence against double-spending attempts in quick transactions. They conducted an analytical and empirical analysis of Bitcoin's privacy and accountability measures after conducting their investigation. Their analysis reveals that the public Bitcoin transaction record contains a wealth of user personal data. This information can be used to link different Bitcoin addresses associated with Bitcoin users in order to implement accountability measures within the Bitcoin system (for instance, "black-listing" linked addresses from the Bitcoin network). As a result, they hope that their findings inspire more research in this area.

Limitations: It is possible to hide the amounts and times of transactions, for example, by using protocols and randomising the time of sending transactions over the network. Such methods have a high processing cost and call for changes to the Bitcoin protocol.

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

Objective: In these fast-pay scenarios, there is a risk of fraudulent double-spending using Bitcoins. Finding scalable and workable solutions to such assaults is becoming more and more important in order to protect consumers from having to pay for these attacks' costs. A number of potential defences have been put forth in the past, but they haven't been fully analysed. The Shadow 13 framework was used by the authors of this study to simulate the effects of various countermeasures in a sizable Bitcoin P2P network. For increased effectiveness against double-spending attacks, they suggested combining a hybrid variant of the observer countermeasure with a closer examination of transaction

specifics.They found that introducing new, special nodes called enhanced observers (ENHOBS) into the network makes up 2% of the total number of online nodes so that they can alert vendors to a double-spending attack in 22 seconds on average.

Observation: They discovered that by merging some of these defences into an improved observer, a vendor can be detected and informed of an impending double-spend attack in less than 28 seconds. When observer functionality is used, all clients are detected 100% of the time, either through direct detection or through receiving a double-spend alert message. They found that the average number of CPU cycles increased by 63.1% as a result of adding the deep transaction inspection capabilities required for all Bitcoin clients to behave as observers, increasing the average CPU utilisation from 31.1% to 50.6%.

Limitations: All clients suffered a negative effect, but those who don't conduct quick payment transactions particularly so. They still have to figure out how to strike a balance between the quantity of observer nodes, their performance, and the quantity of alert subscribers so that it is advantageous and affordable for all parties. Without significantly changing the source code's fundamental operations, it is only natural to not ask Bitcoin to permit observers to mine BTCs when they discover a double-spend. Everyone else already does this when they engage in hash farming. Additionally, a mining cap was built into Bitcoin so that after a certain period of time, no more BTCs can be mined.

CHAPTER III

REQUIREMENT SPECIFICATIONS

3.1 Hardware Requirements

Processor- 1.4 GHz 64 bit

Memory- 512 MB

Hard Disk Space- At Least 50 MB

3.2 Software Requirements

Environment- Visual Studio Code

Node.js Version- 14.15.0

Browser- IE/Firefox

Web Server- IIS/Apache

OS- Windows

CHAPTER IV
SIMULATION / IMPLEMENTATION RESULTS

4.1 System Design

1) We propose an algorithm where the Network observer is used to track unknown transactions performed without authorization. Further, 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.

2) Each pair of nodes has a network observer installed between them.

3) A sender node will send the transaction amount, transaction ID, and sender data to a neighboring node.

4) The addresses of the intended sender and recipient blocks or nodes are mined to complete transactions between them , so that the transaction amount can be deducted from the sender node and added to the receiver node.

5) The records of pending transactions for each block are stored in an array, which starts with a null amount transaction for the genesis block and then stores every new transaction.

6) Every transaction is signed with a signature that is generated using the private and public keys, further a hash is calculated for each block using the elliptic library, to ensure that it is unique.

7) After completing the transaction phase, the nearby node will pass this to the matching forward node.

8) This process continues until the amount is received by the recipient node.

9) Each observer keeps a track of the user ids, the number of transactions, and amounts issued by them.

10) Based on the number of transactions issued by the user, an observer can check the frequency of node communications in its node pair using this information.

11) The aberration will be logged and transaction aborted error will be displayed if the payment amount of two pending transactions from that address is the same, and the total amount is greater than the sender's initial balance.

12) The sender, who wants to commit a double-spending attack, must either retract any of the transactions within a specified timeout period, or he/she will be temporarily blocked from doing any new transactions.

13) Any future transactions authorized for this block will be diverted to alternate routes.

14) The surrounding nodes are warned about the fraudulent transaction when a network observer notices an irregularity at one of the transaction steps between a node pair.

15) The preceding hash is utilized as one of the parameters to generate the current hash of any block, the peer alert system will nullify the preceding hash of blocks that had fraudulent blocks preceding them.

16) This will assist neighboring nodes in terminating their connections with the fraudulent node pair, allowing them to conduct their transactions through other secure routes.

4.1.1 Architecture

4.1.1. (A) Additional Security

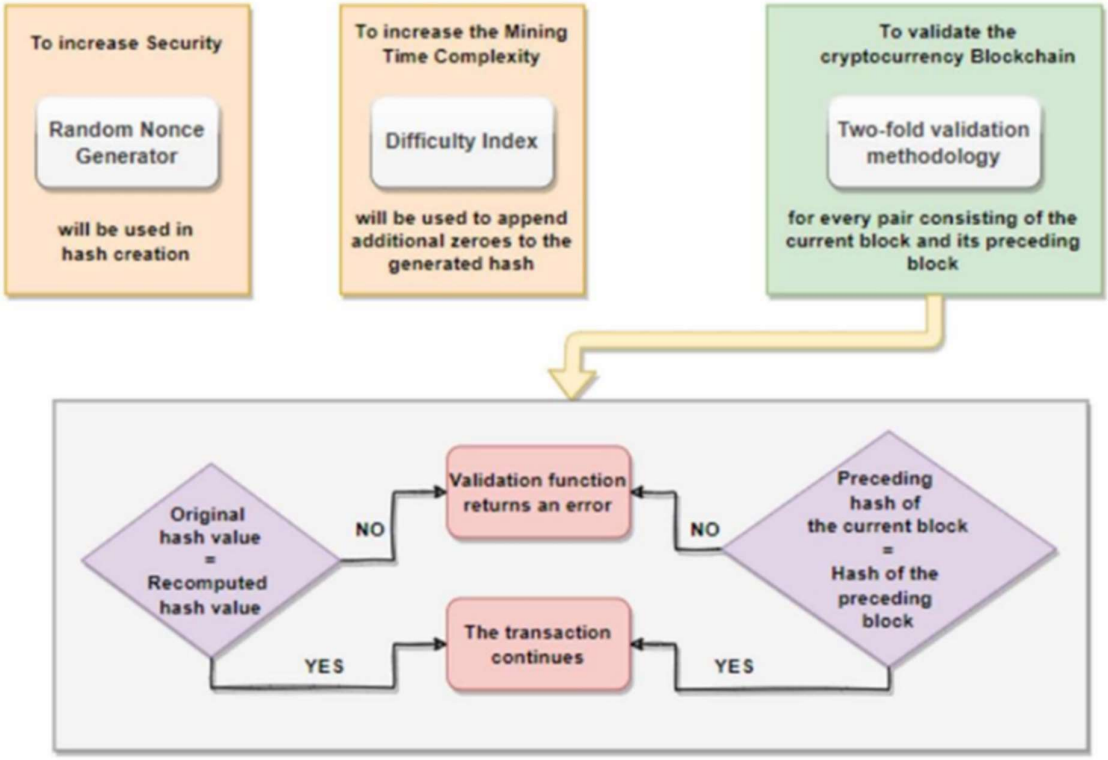

Fig. 1 - Architecture for maintaining Security

A random nonce generator will be used to create the hash in order to boost security. Additionally, a difficulty index is used to add extra zeros to the generated hash, which increases the mining time complexity and lengthens the time required to mine individual blocks. Additionally, a two-fold validation methodology is used to verify the cryptocurrency blockchain for every pair made up of the current block and its predecessor block.

1. The hash of the current block is recomputed with the same nonce, and the original and recomputed hash values are compared.

2. 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.

4.1.1. (B) Performing Transactions

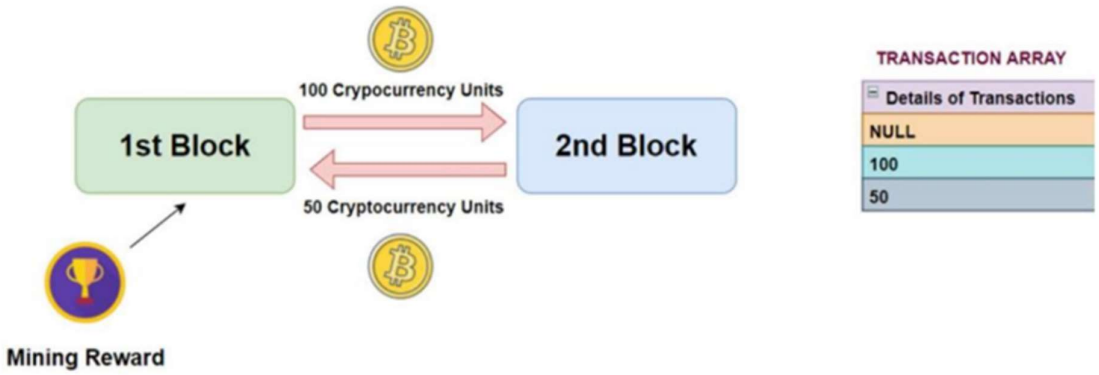

Fig 2: Performing Transactions

The addresses of the intended sender and destination blocks or nodes are mined in order to carry out transactions between them. This allows the transaction value to be added to the receiver node and subtracted from the sender node. Beginning with the genesis block's null amount transaction and continuing to store each subsequent transaction, an array holds the records of all pending transactions for each block. The first block transfers 100 bitcoin units to the second block as an illustration. The second block then gives the first block back 50 bitcoin units. Because of this, the transaction array will include information about the transactions with amounts of 0, 100, and 50, respectively. A 100 bitcoin mining reward follows each successful transaction for a specific genesis block.After every successful transaction for a particular genesis block, a mining reward of 100 points is added to its balance.

4.1.1 (C) Signature

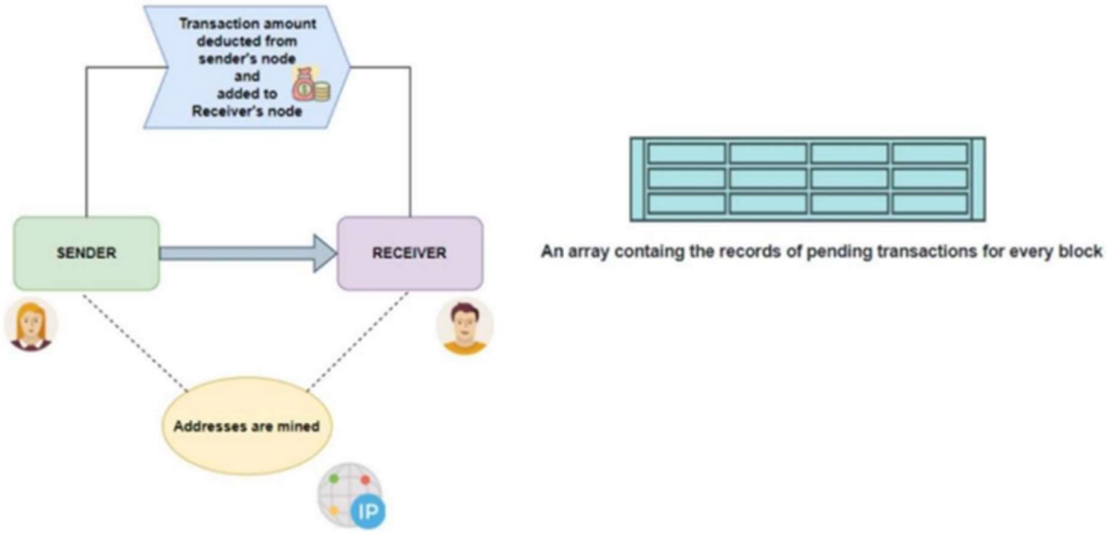

Fig 3. Signature

The addresses of the intended sender and destination blocks or nodes are mined in order to carry out transactions between them. This allows the transaction value to be added to the receiver node and subtracted from the sender node. Beginning with the genesis block's null amount transaction and continuing to store each subsequent transaction, an array holds the records of all pending transactions for each block.

4.1.1 (D) Double Spending Attack

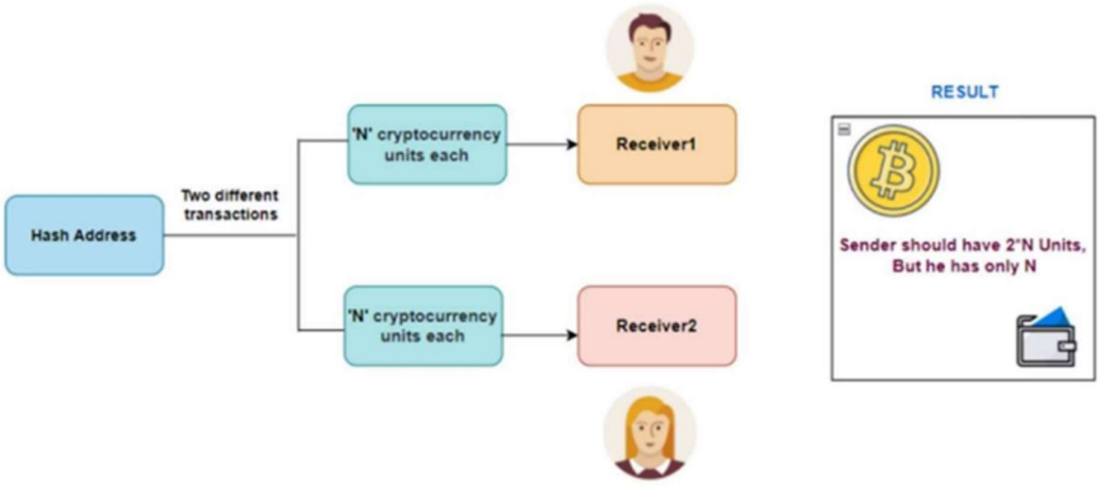

Fig 4, Double Spending Attack

A double spending attack happens when a sender tries to carry out the identical transaction (for the same amount) twice, but with two distinct receivers, even when his starting balance is insufficient to cover both transactions. The example below can be used to demonstrate this. Suppose a sender has 10 bitcoin units and wants to simultaneously send them over 2 distinct blocks to 2 different recipients. The sender needs to have a total of 20 units on hand for both transactions to take place, but only has 10. This suggests that he intends to utilise the identical funds from the first transaction in the second transaction so that the 10 units are subtracted simultaneously.(so only 10 units is deducted from his balance finally).

As can be seen in the example above, two transactions with a total of ten bitcoin units each utilise the same sender hash address (highlighted in yellow), but separate receiver hash addresses. Thus, even though his true deduction amount should be 20, if these 2 transactions are executed simultaneously (i.e. at the same date), only 10 units will be taken from his balance. The sender can take advantage of this situation and complete numerous transactions while just paying half the total price.

4.1.1 (D) Network Observers

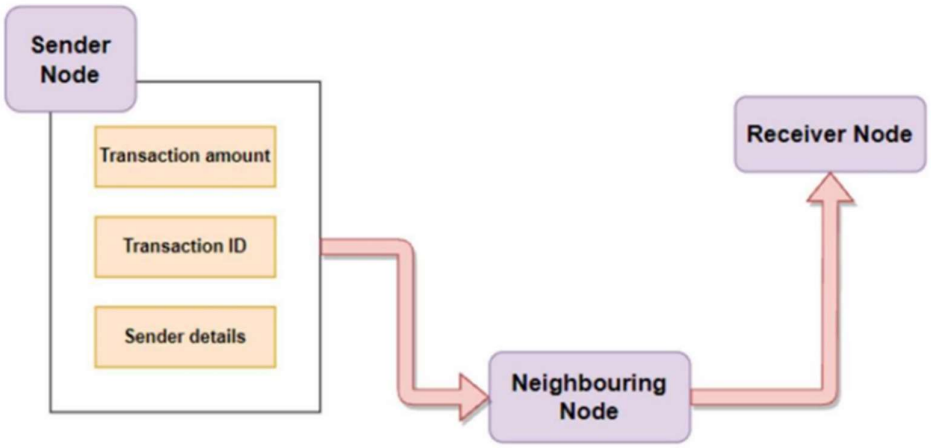

Fig 5. Network Observers:

In a blockchain, a network observer will look for the total number of pending transactions from a specific sender address. A transaction aborted error will be displayed and the user will be given the option to either cancel the most recent transaction with the same amount or try performing the transaction again later if the payment amount of two pending transactions from that address is the same and the total amount is higher than the sender's starting balance.

4.1.1 (E) Peer Alert Systems

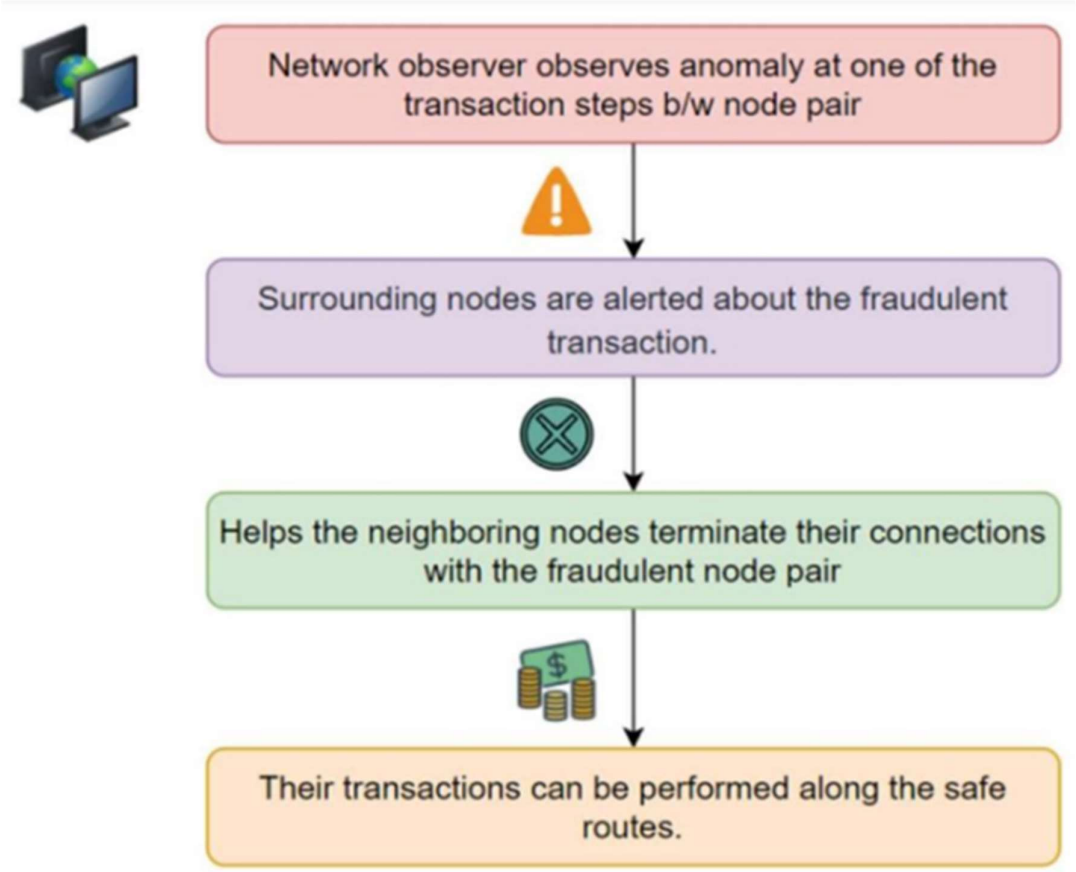

Fig 6. Peer Alert Systems

The surrounding blocks are informed in order to disassociate from the block where the fraudulent transaction or double spending assault occurred. Following transactions scheduled for this block are diverted to alternate paths, and its adjacent blocks cut off communication with it. The sender (who intended to carry out the double spending attack) will either be temporarily blocked before being able to perform any new transactions again, or they must withdraw one of the transactions within a specified timeout period.

4.2. Implementation of System

4.2.1 Mathematical Model

We'll need the message to sign and the private key (da) to sign it with in order to make a signature. This is how the "simplified" signing procedure looks:

1. Make a hash (e) of the message you want to sign.
2. For k, generate a safe random value.
3. Calculate point (x1, y1) on the elliptic curve by multiplying k by the elliptic curve's G constant.
4. Determine r = x1 mod n. Return to step 2 if r equals zero.
5. Determine s = k1(e + rda) mod n. Return to step 2 if s equals zero.
6. Keccak256("x19Ethereum Signed Message: n32" + Keccak256(message)) is commonly used to calculate the hash in Ethereum. This prevents the signature from being used for anything other than Ethereum.

Since we choose a random number for k, the signature we receive varies from time to time. The private key can be determined by adding two signatures when k isn't sufficiently random or the value isn't kept secret ("fault attack"). But how can this be secure if a message signed with MyCrypto always produces the same output? These deterministic signatures make use of the RFC 6979 standard, which describes how to generate a secure value for k based on the private key and message.

4.2.2 Recovering the Public Key

Steps for recovering public key

1. To recover the message, calculate the hash(e).
2. Calculate the elliptic curve point R = (x1, y1), where x1 equals r for v = 27, or r + n for v = 28.
3. Calculate u1 = -zr1 mod n and u2 = sr1 mod n for u1 and u2 respectively.
4. iv. Qa = (xa, ya) = u1 G + u2 R = u1 G + u2 R = u1 G + u2 R = u1 G + u2 R = u1 G + u2

R = u1

5. The point of the public key for the private key used to sign the address is Qa. We can get an address from this and compare it to the address provided. The signature is legitimate if it does.

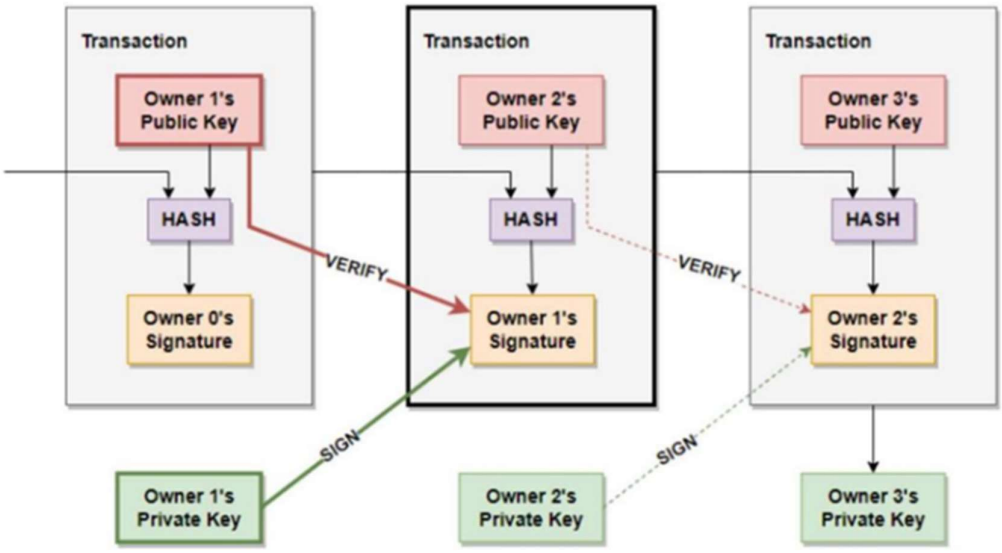

Fig 7. Recovering the Public Key

The first transaction contains Owner 1's public key, which is confirmed by the signature of the next owner before being further converted into a hash value and

the signature of the current owner. Additionally, the signature of the subsequent owner is on the private key of the first owner.

Similar to the first transaction, the private key is signed by the third owner's signature in the second transaction, and the private key is further converted into a hash value and a signature. This cycle is repeated until all transactions are finished.

4.3 Results and Discussions

Additional Security:

To increase security, a random nonce generator is used, and it will be used in the hash construction. The computed hash is additionally extended with more zeros using a difficulty index, which increases the complexity of the mining process. The cryptocurrency blockchain is also validated twice, once for the current block and once for its predecessor, for each pair.

- The current block's hash is recalculated using the same nonce, then the original and recalculated hash values are compared.
- The previous hash of the present block and the previous block's hash is compared.

The validation method produces an error if there is a discrepancy in either of these two steps.After incorporating these extra functions, the resulting cryptocurrency blockchain looks like this:

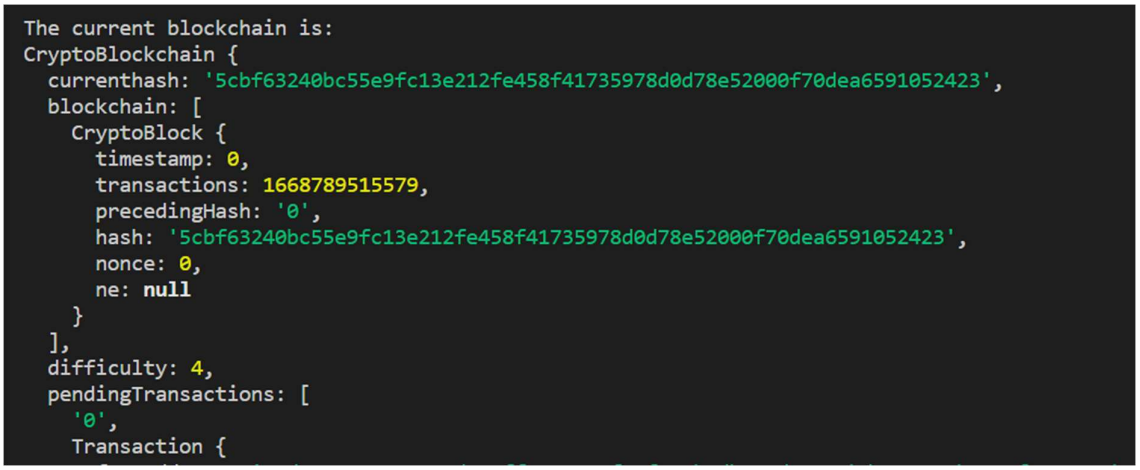

Fig 8 Current Blockchain

Before interfering with any of the blocks, we monitor and log the result of the verify chain validity function, which returns the value true, signalling that the blockchain is genuine.

```
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);

}
```

Fig 9. Validation

If we try to increase the transfer quantity of block 1 in the chain from 5 to 200, the function returns false, signalling an authentication problem.

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

Fig 10. amount change

Performing Transactions:

The records of pending transactions for each block are stored in an array, which starts with a null amount transaction for the genesis block and then stores every new transaction.

```
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);
```

```
Starting the miner...
Block Mined: 00006a71ef46837126ca5ef9a6f55d07dabf89bf7110de2ace996fd8cdb2faee
Timestamp: 1668789515843
Block successfully mined!
Block Mined: 0000d2697e7a0275e2466e03ce66394a4383e7a4f46f8408a7e612d6e6d57b68
Timestamp: 1668789534665
Block successfully mined!
Block Mined: 0000fabad4c691a8aeed73bc7ef9c1e4b2ebc527d597b9b405bd9836e932323c
Timestamp: 1668789537393
Block successfully mined!
Block Mined: 00005299fb0ace36416196952fff3b13388cbf16c1f3544df202c33f05dbda8e
Timestamp: 1668789591874
Block successfully mined!
```

Fig 11a,11b Performing Transactions code output

Signature:

Every transaction has a signature that is created using the private and public keys, and to ensure that it is unique, an elliptic library hash is calculated for each block.

```
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);
```

Fig 12a. Signature making

```
PS C:\ISAA Project> node .\main.js

New Transaction

The signature is: 3045022100d0cc6217495b353b7607bc8210387b91a16b28d144fe5ad2fecca2b4af877f1b02206f16a44dd8f1831e80c14634b7fab6
0354587b525caebfe75877732d687dc777

New Transaction

The signature is: 304402205314164b2a4361fcb31b6bcf91e0cf208fa635ec81cffd8d242e3b88e92d26fa022003073d0c196d716a1765b82a48391f90
d34f742d272786cebfee3f21cf3d8f95

New Transaction

The signature is: 3046022100a5ca636c7996d86b259063b19b239174784f92669298b70e07f14c65f264bc52022100a0f41d16ddd7802ae2112c203608
a16c7ffa46e3aa4c4a398c5e7ec9aff7b521

New Transaction

The signature is: 3046022100b5bc59bb8b3bb9dffc7d54d27921d6c6bc73fd24e7439590e7c11122bbc6f7a5022100fe6e6b25178c8b42aa7e9f2840b4
df6303b732294620b0e7238ed7caf07909f0
```

Fig 12b. Signature Display

Network Observers:

A network observer in a blockchain will look at the quantity of pending transactions coming from a certain sender address. If the total of two pending transactions from that address with identical payment amounts exceeds the sender's beginning balance, a transaction aborted error will be shown. In order to proceed with the transaction, the user will be given the option to either cancel the previous transaction with the same amount or try it again 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.");

    var indices=[];

    for(const trans of this.pending_transactions){
        if(trans.fromAddress=== address)
        {
            indices.push(this.pending_transactions.indexOf(trans));
        }
    }
```

Fig 13. Network Observer's Checking and Flagging code

Double Spending:

When a sender tries to complete the same transaction (with the same amount) twice, but with two different receivers, even though his initial balance is insufficient to cover both transactions, this is known as a double spending attack. The example below exemplifies this. Think about a sender who has 10 bitcoin units and wants to simultaneously send those ten units in separate blocks to two different recipients. The sender is required to provide a total of 20 units for both transactions, but he only has 10, indicating that he plans to use the same funds from the first transaction in the second transaction while simultaneously deducting the 10 units (so only 10 units is deducted from his balance finally).

```
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);
```

Fig 14a. Double Spending Transaction

```
    Transaction {
      fromAddress: '048283cc3178545c8967e58391ec39b75ca56256026a5b2202b9fef7d71c305e9f9867e7f6b8396c1bc15c76bd2c7a624c15abe46f9a5c89e
20c82e4d2f9a621c7',
      toAddress: '0493d39574503a5996d1d818ccc2049f8ebc8c904a98916ba0d475b775b461eb7b296d2bd2f2d6f0330ec03b4d76a433516ccc6ccaaef9680b8
0e221cfc08ee745',
      amount: 5,
      signature: '3046022100b5bc59bb8b3bb9dffc7d54d27921d6c6bc73fd24e7439590e7c11122bbc6f7a5022100fe6e6b25178c8b42aa7e9f2840b4df6303b
732294620b0e7238ed7caf07909f0'
    }
  ],
  miningReward: 10
}

Performing Transactions:

Initial balance was: 10
Current balance of address  04dee849035cee29de07ff8899eaaf86fcc7b5db3a7d6eaecbd1ea75ceb7010f7e9a346b986752177bcacfceaa28dd5fca6f34398
ff80920b7b567256fed2adae3 is: 7

Initial balance was: 10
Current balance of address  048283cc3178545c8967e58391ec39b75ca56256026a5b2202b9fef7d71c305e9f9867e7f6b8396c1bc15c76bd2c7a624c15abe46
f9a5c89e20c82e4d2f9a621c7 is: 5
```

Fig 14b. Performing transaction setup

```
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:
0485c69fcadd380090078eb37b067371ded392b157cac8481796c9e3fc5e174fe658f440c04488bf3a2aa4a97e740f6f34eb42e55ee261ac366e3a6d026ef55655 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
```

Fig 14c. Catching Double Spending Transaction

Final Output

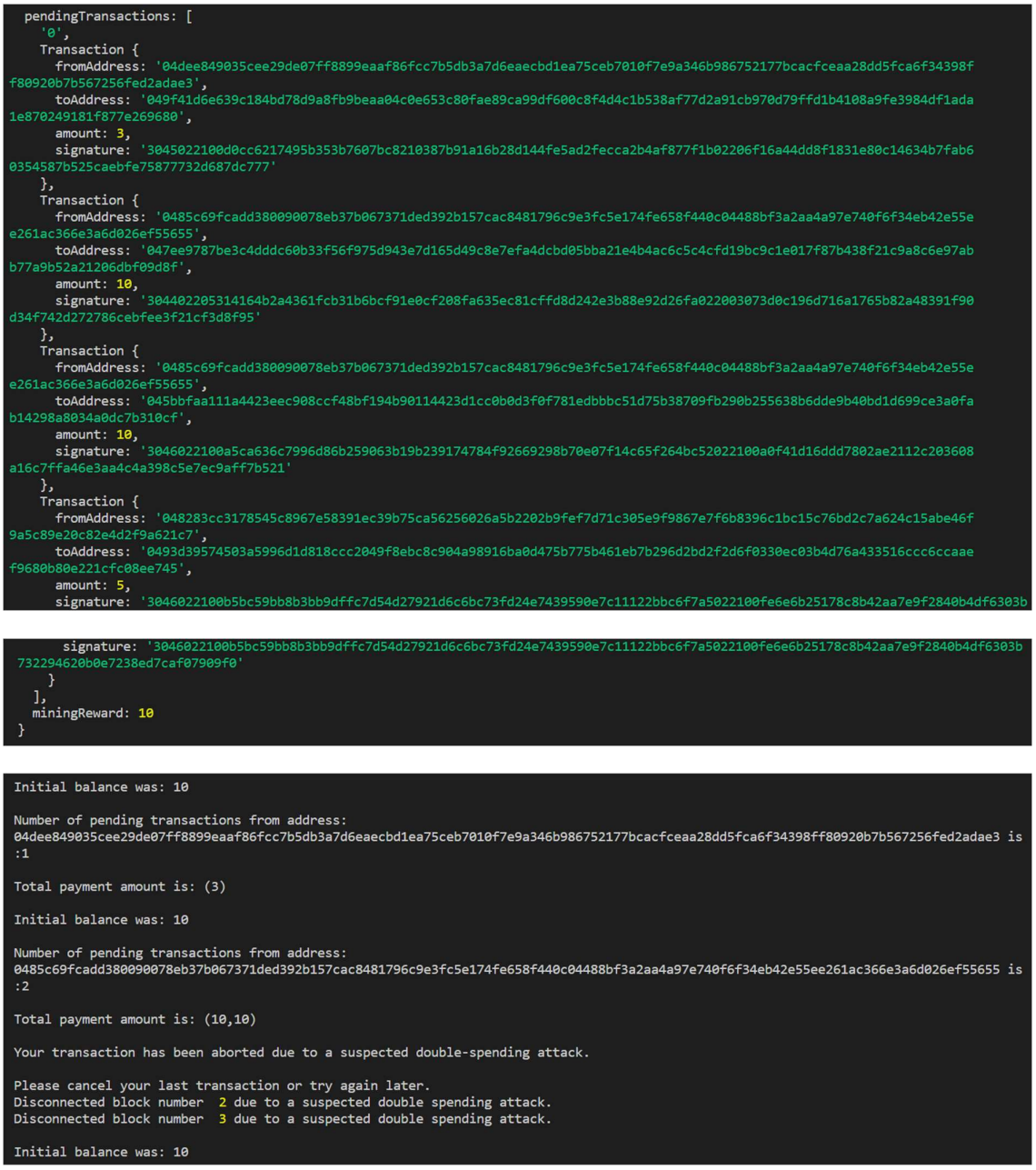

Fig 15. Final Output

CHAPTER V

CONCLUSION AND FUTURE WORKS

5.1 Conclusion and Future Works

Since Bitcoin is a digital currency and there is no central authority to check its spending records, double spending is a risk. According to our analysis, a market player would require a lot of computational effort to doublespend a transaction fraudulently by erasing the previous blocks in the chain. Further validating the integrity of the transaction is the block's exponentially increasing number of confirmations over time. When dealing with quick payment circumstances, we believe that this helps reduce the dependency on transaction confirmation. Research has been done into the factors that make double-spending attacks against Bitcoin quick payments effective. Our project is distinctive in terms of security, transactions, signatures, network observers, and peer alert systems. We demonstrate that, absent the inclusion of new detection algorithms to the Bitcoin implementation, double-spending attacks on quick payments are likely to be successful and may be executed at a low cost. We offer a fresh, simple defence against double-spending attempts in quick transactions. Accountability becomes essential in the face of such misconduct. In order to assist the user, this project aims to construct a Network Observer and a Peer-Alert System. In this design, the Network Observer is placed in-between each pair of nodes. When a network observer detects an anomaly at one of the transaction steps between a node pair, the Peer Alert system alerts the nearby nodes about the fraudulent transaction. As a result, the nearby nodes will be able to cut off their connections to the fraudulent node pair and carry out their transactions over other secure channels. A hash's complexity is governed by its "target"; the lower the target, the smaller the set of

acceptable hashes, and the more challenging it is to create one. To compile and assess the defences against double-spending threats in the blockchain, we used a methodical approach. Future research directions' implications for defences against double-spending attacks have also been taken into account.

CHAPTER VI

REFERENCES

[1] Nicolas, Kervins, Yi Wang, George C. Giakos, Bingyang Wei, and Hongda Shen. "Blockchain system defensive overview for double-spend and selfish mining attacks: A systematic approach." IEEE Access 9 (2020): 3838-3857.

[2] Karame, Ghassan O., Elli Androulaki, and Srdjan Capkun. "Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin." Cryptology EPrint Archive (2012).

[3] Lee, Hyunjae, Myungjae Shin, Kyeong Seon Kim, Yeongeun Kang, and Joongheon Kim. "Recipient-oriented transaction for preventing double spending attacks in private blockchain." In 2018 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), pp. 1-2. IEEE, 2018.

[4] Karame, Ghassan O., Elli Androulaki, Marc Roeschlin, Arthur Gervais, and Srdjan Čapkun. "Misbehavior in bitcoin: A study of double-spending and accountability." ACM Transactions on Information and System Security (TISSEC) 18, no. 1 (2015): 1-32.

[5] Podolanko, John P., Jiang Ming, and Matthew Wright. "Countering double-spend attacks on bitcoin fast-pay transactions." In Proc. Workshop Technol. Consum. Protection, pp. 1-3. 2017.

[6] Chohan, Usman W. "The double spending problem and cryptocurrencies." Available at SSRN 3090174 (2021).

[7] Poluyanenko, Nikolay, Nadia Pisarenko, Vladyslav Safonenko, Tymur Makushenko, Olha Pushko, Yevhena Zaburmekha, and Kateryna Kuznetsova. "Simulation of a Double Spending Attack on the Proof of Work Consensus Protocol." In CybHyg, pp. 32-59. 2019.

[8] Yang, Jin, Tao Yang, and Ping Lin. "Double-Spending Detection for Fast Bitcoin Payment Based on Artificial Immune." In Theoretical Computer Science: 35th National Conference, NCTCS 2017, Wuhan, China, October 14-15, 2017, Proceedings, vol. 768, p. 133. Springer, 2017.

CHAPTER VII

APPENDIX

CODE:

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('7fe38a7cd191b531376ed5db174624ff3a474b736d824d174677e32aeb703c29');
const Key1=
ec.keyFromPrivate('1c2e962350cbe889eeb39ab45728fa7e59624ad4d0b8af4f258a9dd439db3f8f');
const Key2=
ec.keyFromPrivate('af68820d23d0857f56aefba0382a631f6e4d54056cbf2ad8d2f64fcc7d34441d');
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.i
ndexOf(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));
```

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.pendingTransact
ions.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);
```