COL 819 Assignment 3

Dipika Tanwar (2020MCS2456)
Hemant Dhankhar (2020MCS2458)

May 14, 2021

1 Introduction to bitcoin and block chain

For all the money transactions we currently have, there is a central agency which controls the flow and value of the money. In USA federal bank is the central agency and in India Reserve bank of India, are such authorities.The reason for this is simple - to have a centralised authority on which people can trust. Like mentioned above, this authority is usually some government institution of a country like RBI in India. Bit coin is the the first implementation of a distributed, decentralized crypto- currency. This way since there is no central authority to analyse the flow of transactions- it gives a sense of anonymity and privacy and also trust as there is no single point of failure in the system, which makes up for the exponential popularity of bitcoin which is still increasing. [3] [1] [2] [4] Bit coin is implemented upon the revolutionary technology of block chain. Block Chain has following features

- It consists a chain of block. (block has relevant data, transaction info in case of bit coin, previous block signature (hash), nonce)
- A block can have multiple transactions.
- block are concerted via signature (hash) of previous blocks, thus makes a chain

Along with pros of bitcoin that come with privacy and decentralisation, there are certain points which make bitcoin an easy way to use in unlawful activities like tax invasion, payment in places like dark web etc. as almost full anonymity is provided by the decentralised system of the bitcoin and other crypto- currencies. These points have been further discussed in the last section of the report.

2 Details of Implementation

For implementing bitcoin we have used Python 3 programming language. We have tested the code on a machine with Intel i5 processor with 4GB of RAM running on Linux(Ubuntu) OS. For easy debugging and comprehension of code we have divided the code into following files :

- Main.py (Start Point of Simulation)
- Node.py (Working on Individual Node)
- Config.py (Runtime configuration)
- Helper.py (Helper function related to cryptography)
- Blockchain.py (Main block chain class)
- Block.py (Block class)
- UTXO.py (Unspent Transaction output class)
- MarkelTree.py (Markel Tree functions)

Out of the above files Main.py files serves as a vessel for the final integration while Config.py contains various runtime constants (markel tree arity, block mining reward, number of nodes, confirmation block count, etc). Helper.py consites all the function related to digital signature assignment and verification, public and private key generation and other utilities. All features of Blockchain are implemetated inside BlockChain.py file. Node.py file has working of individual nodes that participates in currency transactions. Block.py class define structure of a bitcoin block chain block. Inside block chain transactions are handled according to unspent transaction output model, functions related to this are defined in UTXO.py file. Markel Tree is defined in MarkelTree.py file.

```
import threading
from Crypto.Hash import SHA256,SHA1,SHA224,SHA384,SHA512

```

```
TOTAL_NODES = 10
PROF_OF_WORK_ZEROS = 3
MINING_REWARD = 10
CONFIRMATION_BLOCK = 1
MARKEL_TREE_ARITY = 2
SIMULATION_TIME = 60
WAIT_BEFORE_NEW_TRX = 2
BLOCK_CREATION_TIME = 7
CRYPTO_METHOD = SHA1

nodeInfo ={}
nodePublicKeyToId = {}
commonDataLock = threading.Lock()


class MSG():
    MINE_BLOCK = 0
    PERFORM_TRX = 1
    RECEIVE_BLOCK = 2
    BROADCAST_PUBLIC_KEY = 3
    BROADCAST_GENESIS_BLOCK = 4
```

Listing 1: "Configuration for block chain simulation"

```
class Node():
    def __init__(self, nodeId, totalNodes):
        self.nodeId = nodeId
        self.totalNodes = totalNodes
        self.rsaKey, self.private_key, self.public_key = Helper.rasKey()
        self.startTime = time()
        self.receiveQueue = deque()
        self.receiveQueueLock = threading.Lock()
        self.publicKeyList = {}
        self.lastTrasactionVarified = True

    def sendFunds(self, recepient, amount):
        if self.getBalance() < amount: return None
        inputUTXO = []
        total = 0
        for key,value in self.blockChain.UTXO.items():
            if value.recepient == self.public_key:
                total += value.amount
                inputUTXO.append(TrxInput(key, value))
            if total > amount:break

        data = {
```

```
            'senderAddr': self.public_key,
            'receiverAddr': recepient,
            'amount': amount
        }

        trx = Transaction(data, inputUTXO)
        trx.sign(self.private_key)

        return trx

    def work(self):
        while working:
            ret, msgType,trx = self.receive(poleTime)
            if ret:
                trx = copy.deepcopy(trx)
                timeWaited = 0
                if msgType == MSG.PERFORM_TRX:
                    self.blockChain.performTransaction(trx)
                elif msgType == MSG.RECEIVE_BLOCK:
                    self.blockChain.appendBlock(trx)
                    lastBlockCreatedTime = time()
                else:
                    print('[INVALID MSG RECEIVED-{}]'.format(str(self.nodeId)))
            else:
                timeWaited += poleTime
                if timeWaited >= SIMULATION_TIME:
                    working = False
                else:
                    if time() - lastTrasactionTime > WAIT_BEFORE_NEW_TRX and self.lastTrasactionVarified:
                        if Helper.isMyTurn(self.nodeId, self.totalNodes):
                            receiver = Helper.pickReceiver(self.nodeId, self.totalNodes)
                            balance = self.getBalance()
                            if balance > 0:
                                lastTrasactionTime = time()
                                transfer_amount = random.randint(1, balance)
                                trx = self.sendFunds(nodeInfo[receiver].public_key, transfer_amount)
                                if trx:
                                    for i in range(self.totalNodes):
                                        if i!=self.nodeId:
                                            self.sendMessage(MSG.
```

```
    PERFORM_TRX, nodeInfo[i], trx)

                                    self.blockChain.performTransaction(trx
    )
                                    timeWaited = 0

                    if (time() - lastBlockCreatedTime) >
    BLOCK_CREATION_TIME:
                        if Helper.isMyTurn(self.nodeId, self.totalNodes):
                            for t in self.blockChain.pendingTransaction: t
    .data['amount']
                            block = self.blockChain.createBlock(self.
    public_key)
                            if block:
                                temp = []
                                addMyBlock = True
                                while True:
                                    ret, msgType,data = self.receive(
    poleTime)
                                    if not ret:
                                        break
                                    else:
                                        if msgType == MSG.RECEIVE_BLOCK:
                                            self.blockChain.appendBlock(
    data)
                                            addMyBlock = False
                                            break
                                        else:
                                            temp.append((msgType, data))
                                for msgType, data in temp:self.sendMessage
    (msgType, self, data)
                                if addMyBlock:
                                    for i in range(self.totalNodes):
                                        if i!=self.nodeId:
                                            self.sendMessage(MSG.
    RECEIVE_BLOCK, nodeInfo[i], block)
                                    self.blockChain.appendBlock(block)
```

Listing 2: "Node.py File"

```

class Block():
    def __init__(self, timestamp, transactions, preHash):
        self.timestamp = timestamp
        self.transactions = copy.deepcopy(transactions)
        self.nonce = 0
```

```
        self.prevHash = preHash
        self.prof_of_work_zeros = PROF_OF_WORK_ZEROS
        self.markelTree = MarkelTree(self.transactions)
        self.markelRoot = self.markelTree.hashTree[-1][0]
        self.hash = self.calculateHash()

    def calculateHash(self):
        s = str(self.timestamp) + str(self.prevHash) + str(self.nonce) +
    self.markelTree.hashTree[-1][0]
        return Helper.getHash(s)

    def mineBlock(self):
        while True:
            if(self.hash[:self.prof_of_work_zeros] == "0"*self.
    prof_of_work_zeros):break
            self.nonce += 1
            self.hash = self.calculateHash()

    def hasAllTransactionValid(self):
        for t in self.transactions:
            if not t.isValid():return False
        if not self.markelTree.varifyTransactions(self.transactions):
    return False
        return True

    def verifyPOW(self):
        return self.hash[:self.prof_of_work_zeros] == "0"*self.
    prof_of_work_zeros
```

Listing 3: "Block.py File"

3 Experiments

Figure 1 Shows time required to mine a block vs Proof of work zeros. As Proof of work zeros increase time to mine a block also increase. We have taken a log scale to show time increment on Y axis. Function is exponential.

Figure 2 Shows Markel tree arity vs block size. This characteristics shows that as markel tree arity increases size of the block chain block decrease
Figure 3 Shows block size vs different type of cryptographic hash functions.

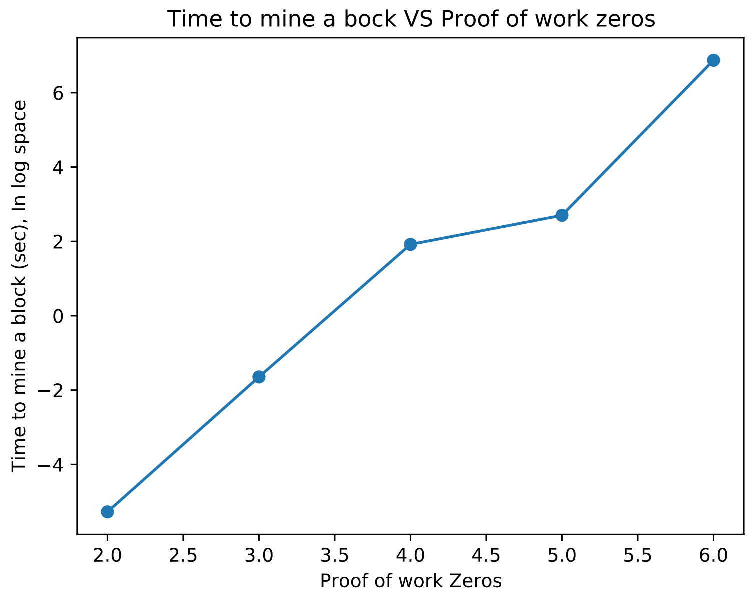

Figure 1: Time Vs Proof of Work Zeros. Here Time is given in log base e of seconds

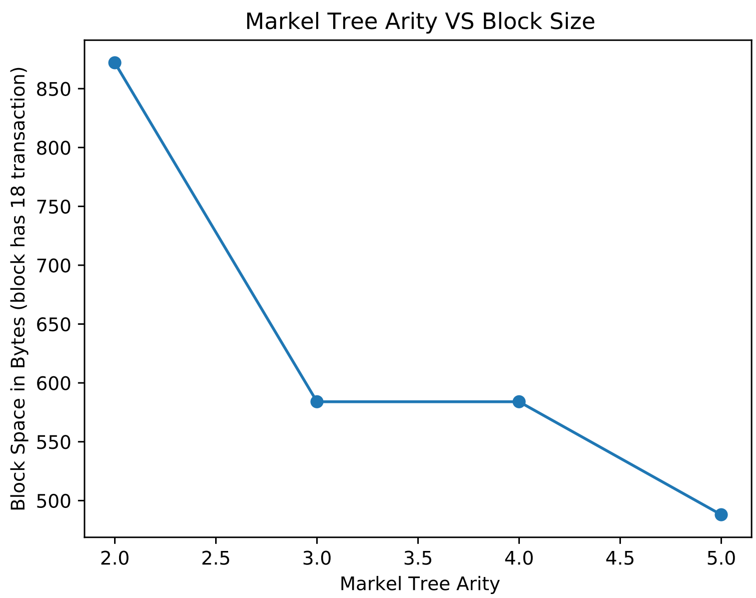

Figure 2: Markel Tree arity VS Block size (bytes)

4 How to Run

Following are the basic python package required to run the project.

```
matplotlib==3.3.4
numpy==1.19.5
Pillow==8.2.0
pycryptodome==3.10.1
```

Listing 4: "Basic Python Packages required"

4.1 Run Command

```

python Main.py
```

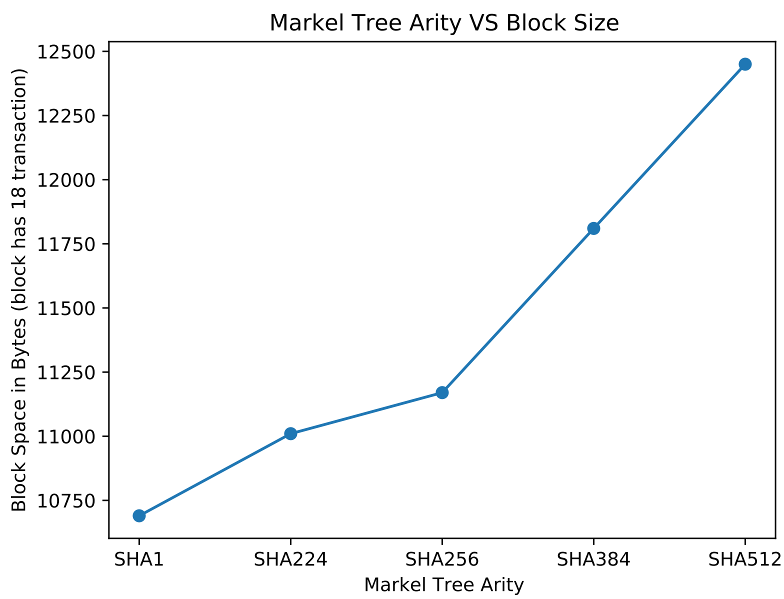

Figure 3: Hash Type VS Block size (bytes)

5 Simulation

For the simulation we have taken 10 identically but different nodes, working as python thread. Initially 1000 balance is given to nodeid 0, all other node has no balance.
The table 2 shows initial and final balance of participating nodes. Here the data is presented after 8 blocks are mined. Total Initial bit coins = 1000 Total Block rewards = 80

```
Balance of Node id  0  =  1000
Balance of Node id  1  =  0
Balance of Node id  2  =  0
Balance of Node id  3  =  0
Balance of Node id  4  =  0
Balance of Node id  5  =  0
Balance of Node id  6  =  0
Balance of Node id  7  =  0
Balance of Node id  8  =  0
Balance of Node id  9  =  0
Transaction between Node  0 ( balance  1000 ) and  4  of  488
```

NodeId	Initial Balance	Final Balance
0	1000	157
1	0	10
2	0	10
3	0	142
4	0	444
5	0	43
6	0	22
7	0	3
8	0	4
9	0	245

Table 1: Table to test captions and labels

```
MINER address  3  Reward given  10
Transaction between Node  3 ( balance  10 ) and  9  of  9
Transaction between Node  3 ( balance  1 ) and  8  of  1
Transaction between Node  7 ( balance  10 ) and  6  of  6
MINER address  7  Reward given  10
Transaction between Node  7 ( balance  14 ) and  3  of  3
Transaction between Node  7 ( balance  11 ) and  4  of  7
Transaction between Node  7 ( balance  4 ) and  6  of  4
Transaction between Node  8 ( balance  11 ) and  3  of  1
Transaction between Node  8 ( balance  10 ) and  5  of  8
Transaction between Node  3 ( balance  14 ) and  4  of  5
Transaction between Node  5 ( balance  28 ) and  2  of  14
Transaction between Node  3 ( balance  9 ) and  5  of  3
Transaction between Node  2 ( balance  24 ) and  5  of  2
Transaction between Node  6 ( balance  20 ) and  1  of  5
Transaction between Node  6 ( balance  15 ) and  4  of  15
MINER address  3  Reward given  10
Transaction between Node  3 ( balance  16 ) and  1  of  6
Transaction between Node  3 ( balance  10 ) and  8  of  3
Transaction between Node  1 ( balance  21 ) and  0  of  8
Transaction between Node  7 ( balance  10 ) and  0  of  4
MINER address  7  Reward given  10
Transaction between Node  7 ( balance  16 ) and  5  of  9
Transaction between Node  0 ( balance  544 ) and  3  of  114
Transaction between Node  7 ( balance  7 ) and  2  of  5
Transaction between Node  7 ( balance  2 ) and  2  of  1
Transaction between Node  7 ( balance  1 ) and  2  of  1
Transaction between Node  3 ( balance  131 ) and  9  of  100
```

```
Transaction between Node  2 ( balance  39 ) and  6  of  25
Transaction between Node  2 ( balance  14 ) and  1  of  6
MINER address  3  Reward given  10
Transaction between Node  9 ( balance  119 ) and  3  of  19
Transaction between Node  9 ( balance  100 ) and  4  of  19
Transaction between Node  8 ( balance  15 ) and  5  of  12
Transaction between Node  9 ( balance  81 ) and  3  of  4
Transaction between Node  9 ( balance  77 ) and  5  of  1
Transaction between Node  7 ( balance  10 ) and  2  of  2
Transaction between Node  3 ( balance  64 ) and  7  of  43
Transaction between Node  9 ( balance  76 ) and  5  of  50
Transaction between Node  4 ( balance  544 ) and  7  of  61
Transaction between Node  7 ( balance  112 ) and  0  of  96
MINER address  7  Reward given  10
Transaction between Node  6 ( balance  35 ) and  1  of  16
Transaction between Node  1 ( balance  45 ) and  4  of  35
Transaction between Node  7 ( balance  26 ) and  0  of  22
Transaction between Node  2 ( balance  20 ) and  4  of  10
MINER address  6  Reward given  10
Transaction between Node  7 ( balance  4 ) and  5  of  1
Transaction between Node  2 ( balance  10 ) and  6  of  7
Transaction between Node  2 ( balance  3 ) and  6  of  1
Transaction between Node  6 ( balance  36 ) and  0  of  14
Transaction between Node  2 ( balance  2 ) and  3  of  1
Transaction between Node  2 ( balance  1 ) and  8  of  1
Transaction between Node  5 ( balance  112 ) and  4  of  69
Transaction between Node  0 ( balance  572 ) and  3  of  415
Transaction between Node  3 ( balance  447 ) and  5  of  86
Transaction between Node  3 ( balance  361 ) and  9  of  219
MINER address  2  Reward given  10
```

Listing 5: "Simulation Logs"

6 Security: Dishonest nodes

A dishonest node in the current implementation will try to fork the transaction so as to temper the original transaction with malicious intent, while the protocol trying to inhibit such attempts. But if the number of dishonest nodes increase beyond a certain threshold, issues are created. Following are the results on that threshold.
We have performed experiments for finding fraud node. Experiment is performed with 50, 100 and 200 nodes. In all the cases, Block chain is affected

as soon as number of fraud nodes increase to more the 50% of total nodes.

Total Nodes	Number of Fraud Nodes
50	27
100	55
200	106

Table 2: Fraud is excepted after more then 50% nodes becomes fraudulent

7 Smart Contracts and multiple transaction support

In current implementation multiple input/output transaction are supported using unspent transaction output (UTXO) modal.

```
class TrxInput():
    def __init__(self,transactionOutputId, outputTx):
        self.transactionOutputId = transactionOutputId
        self.outputTx = outputTx

class TrxOutput():
    def __init__(self,recepient,amount, trxId):
        self.recepient = recepient
        self.amount = amount
        self.trxId = trxId
        self.id = Helper.getHash(str(self.recepient) + str(self.amount) +
    str(self.trxId))
```

Listing 6: "unspent transaction output (UTXO)"

We have also performed a smart contract between nodeId 4 and nodeId 5. In the contract if balance of node 4 is greater then 100 then 50 is transfered to node 5.

8 Open Limitations in Bitcoin

8.1 Security Issues in Asymmetric Keys

Since in any bitcoin transaction a pair of public private key is used, it is fundamentally necessary to secure that pair from leaking away as unlike traditional banking system where even if the password of a customers is stolen there are other layer of securities like 2 way authentication and bank liability in case of fraudulent transaction, there is no such security measures in bitcoin, therefore if the private key gets leaked the entire transaction can be forked as it is the only source to distinguish a transaction.

8.2 Extent of Privacy

In the current bitcoin architecture, it is very difficult to track someone's identity who is involved in the transaction as all the transactions happen with private-public key pairs and as no other identifying information is linked to this pair it is not possible to track the party doing the transaction, because of this users of bitcoin can anonymously do the transaction. While it is privacy preserving in a sense but it also encourages unlawful activities as people can use bitcoin to transact for illegal activities which if done using traditional banking methods could be tracked and people involved would be tried for the same. This creates debate over how much privacy do people really need, do people really need complete anonymity while paying for some good or not. Anonymity further creates problem like payment in dark web and use for transactions for tax invasions.

References

[1] Adam Back et al. "Hashcash-a denial of service counter-measure". In: (2002).

[2] Christian Cachin et al. "Architecture of the hyperledger blockchain fabric". In: *Workshop on distributed cryptocurrencies and consensus ledgers.* Vol. 310. 4. Chicago, IL. 2016.

[3] Satoshi Nakamoto. "A peer-to-peer electronic cash system". In: ().

[4] Martin Valenta and Philipp Sandner. "Comparison of ethereum, hyperledger fabric and corda". In: *Frankfurt School Blockchain Center* 8 (2017).