TRANSACTION MANAGEMENT

using Blockchain

25-04-2022
CRYPTOGRAPHY

TEAM MEMBERS:

Tushar Tiwari	2019A8PS1332H
Samar Jaish	2019A3PS1309H
Aditya Saini	2019A3PS1292H

CONTENTS

Problem Statement

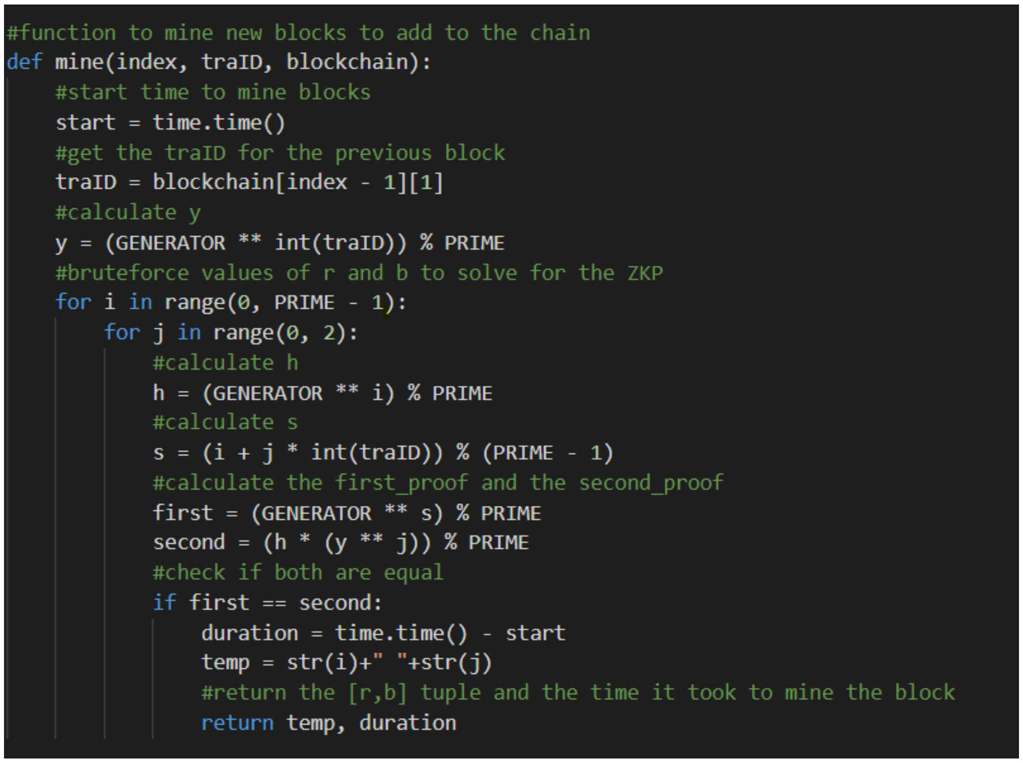
Data is the most crucial entity and its processing should be done with utmost seriousness. Cloud storage is the more general way of storing data. But due to being in high demand, it may be misused. In cloud storage, the data is centralized and generally unprotected due to not being encrypted during transactions. The centralized way of doing things makes it easy for hackers. Thus, we are in need of a decentralized way of doing transactions. This is where blockchain comes into the picture.
So, we are going to use blockchain for storing the transaction details.

Blockchain

Blockchain is a distributed ledger, which maintains data in a chained format. A distributed ledger is a database that is distributed and synced by a group of people across many sites, organizations, etc.
Decentralization guarantees that the authority to manipulate or change data is not concentrated in the hands of a single entity or organization, but rather is shared by all. This increases its trustworthiness because all modifications to the existing database can only be done when a consensus has been reache d. This improves dependability and adds an extra layer of trustworthiness.

Implementation of Zero Knowledge Proof

Zero-Knowledge Proofs (ZKP) refers to a proof construction that allows you to persuade someone that your results or conclusions are correct without revealing any sensitive or confidential information you know in the process.

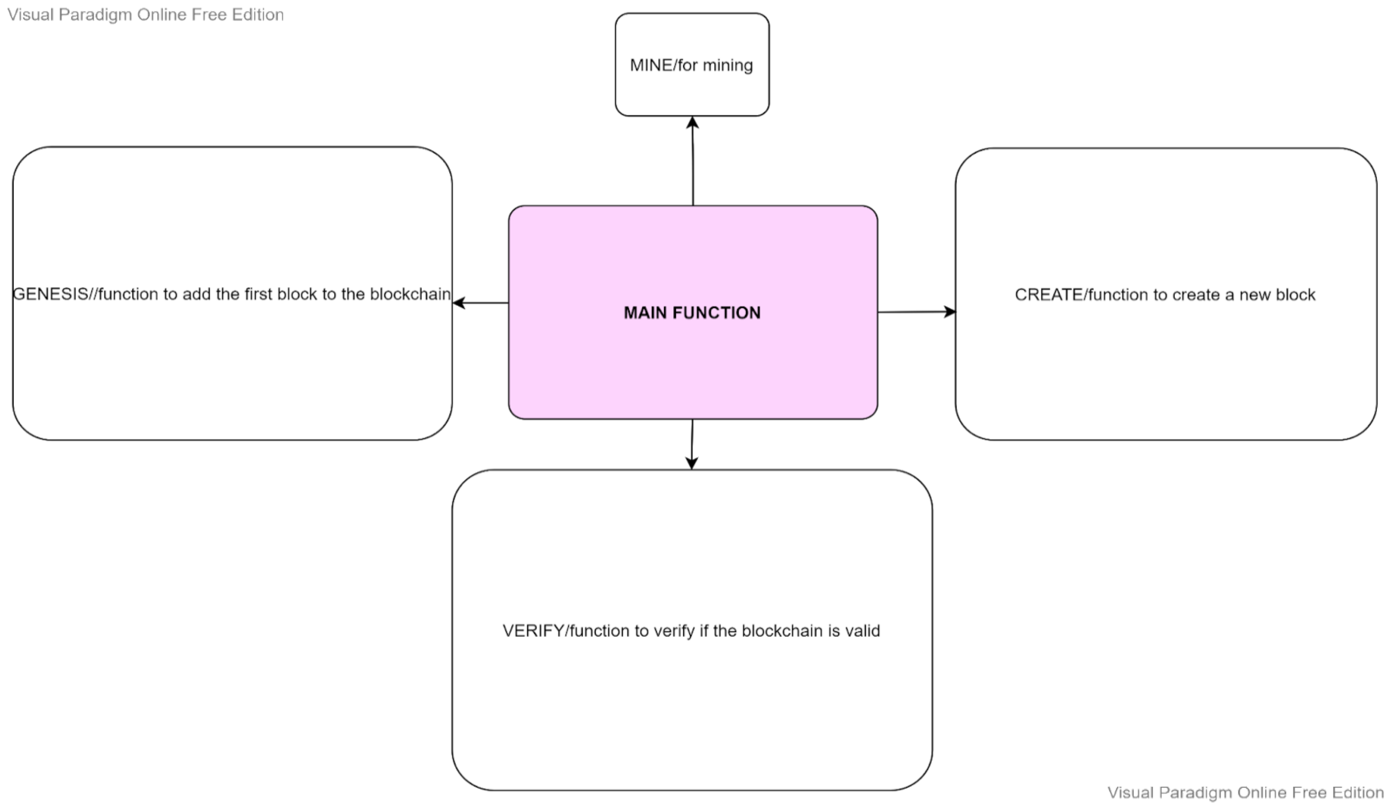


```python
#function to mine new blocks to add to the chain
def mine(index, traID, blockchain):
    #start time to mine blocks
    start = time.time()
    #get the traID for the previous block
    traID = blockchain[index - 1][1]
    #calculate y
    y = (GENERATOR ** int(traID)) % PRIME
    #bruteforce values of r and b to solve for the ZKP
    for i in range(0, PRIME - 1):
        for j in range(0, 2):
            #calculate h
            h = (GENERATOR ** i) % PRIME
            #calculate s
            s = (i + j * int(traID)) % (PRIME - 1)
            #calculate the first_proof and the second_proof
            first = (GENERATOR ** s) % PRIME
            second = (h * (y ** j)) % PRIME
            #check if both are equal
            if first == second:
                duration = time.time() - start
                temp = str(i)+" "+str(j)
                #return the [r,b] tuple and the time it took to mine the block
                return temp, duration
```

We have used the mine function to produce proof of work.
Here, we are taking the transaction ID of the previous block as confidential information.
First, we calculate y by using $y=g^x \bmod(p)$ where g(generator) = 4, p(prime) = 1101 and x = transaction ID.
Then we are running a loop for variable i from 0 to p-1; variable j(the random bit with value 0/1) and calculating h by $h=g^i \bmod(p)$.
Also calculating $s=(i+jx) \bmod(p-1)$.
After finding these variables they are sent for verification, we calculate the first proof and second proof:
First proof: $g^s \bmod(p)$
Second proof: $hy^j \bmod(p)$

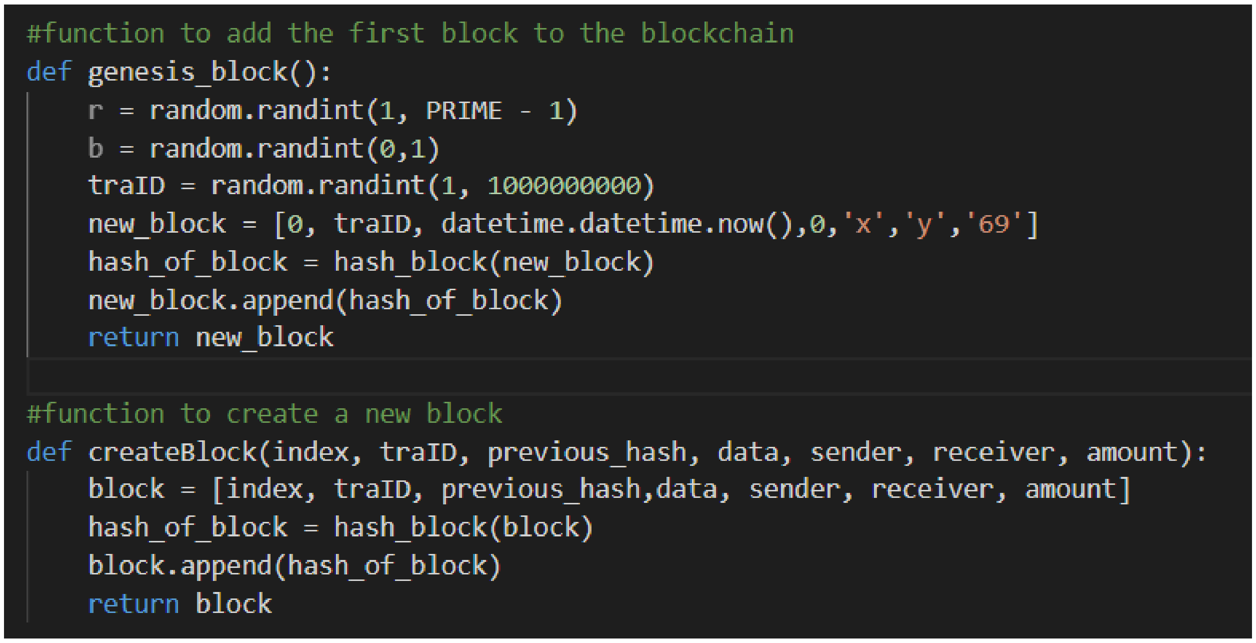
If both the proofs are found to be equal then we have completed the mining process and we can return data (formed by i and j) and duration. In the whole process, we haven't explicitly disclosed transaction ID thus maintaining zero-knowledge proof.

Explanation of functions written in our code



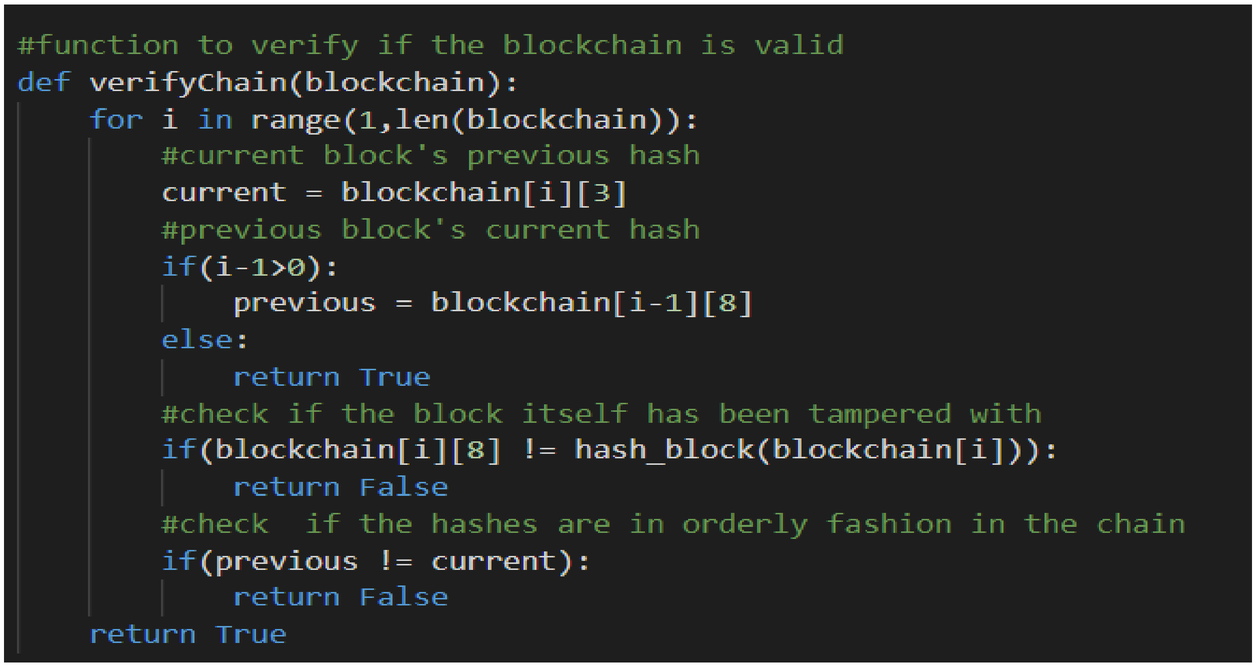
Visual Paradigm Online Free Edition
MINE/for mining
GENESIS//function to add the first block to the blockchain
MAIN FUNCTION
CREATE/function to create a new block
VERIFY/function to verify if the blockchain is valid
Visual Paradigm Online Free Edition

We have applied the theory of DES taught to us in the lecture sections, along with the concept of BlockChain and databases.
Our 'genesis_block' starts the process, where we take the id randomly and hash it. In the 'createBlock' we append the current value with the hashed value of the previous transaction and return the new value.



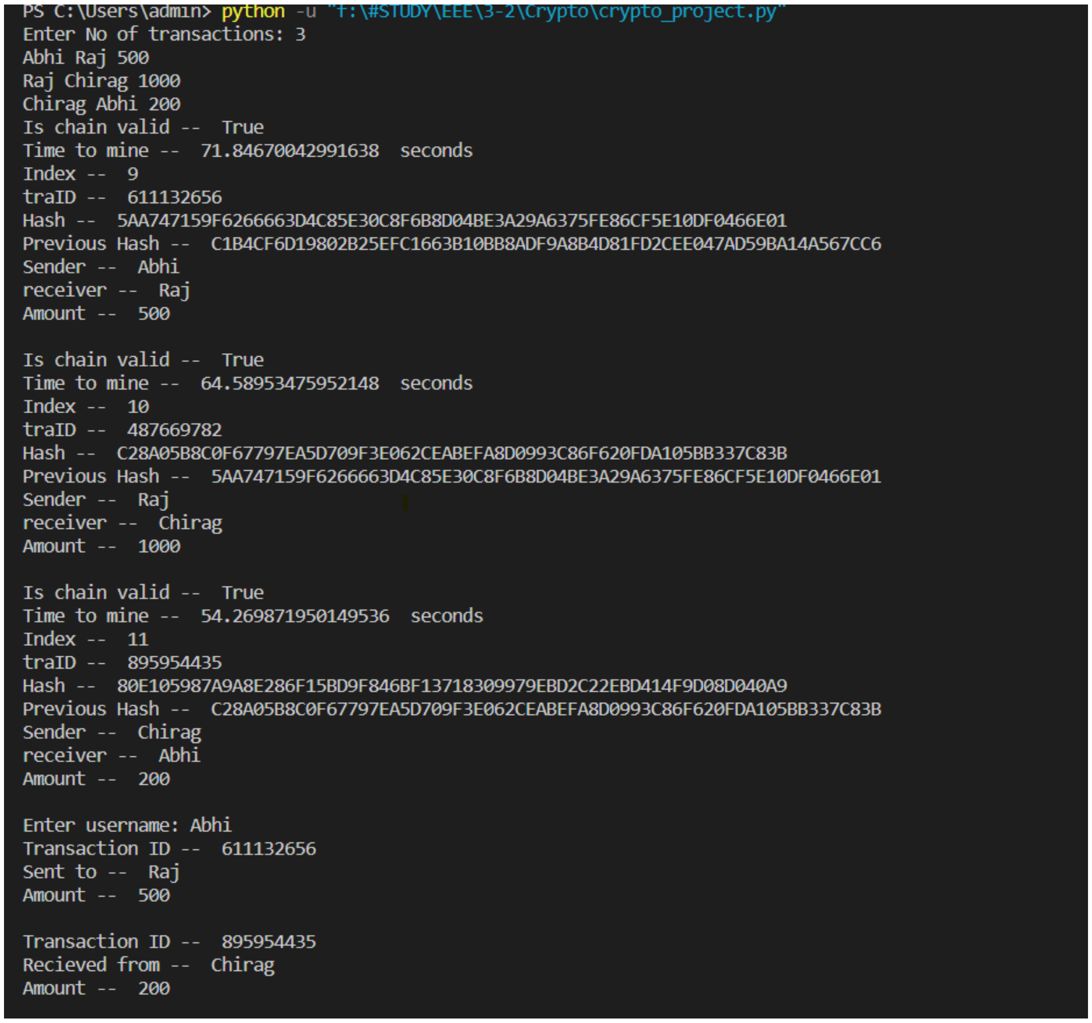
```
#function to add the first block to the blockchain
def genesis_block():
    r = random.randint(1, PRIME - 1)
    b = random.randint(0,1)
    traID = random.randint(1, 1000000000)
    new_block = [0, traID, datetime.datetime.now(),0,'x','y','69']
    hash_of_block = hash_block(new_block)
    new_block.append(hash_of_block)
    return new_block

#function to create a new block
def createBlock(index, traID, previous_hash, data, sender, receiver, amount):
    block = [index, traID, previous_hash,data, sender, receiver, amount]
    hash_of_block = hash_block(block)
    block.append(hash_of_block)
    return block
```

In ‘verifyChain’, the hashed value of the currently stored value is compared to the hashed value of the previous value for verification, if it is correctly verified then the message,
‘**Is chain valid --True**’ is printed through the main function.



```
#function to verify if the blockchain is valid
def verifyChain(blockchain):
    for i in range(1,len(blockchain)):
        #current block's previous hash
        current = blockchain[i][3]
        #previous block's current hash
        if(i-1>0):
            previous = blockchain[i-1][8]
        else:
            return True
        #check if the block itself has been tampered with
        if(blockchain[i][8] != hash_block(blockchain[i])):
            return False
        #check  if the hashes are in orderly fashion in the chain
        if(previous != current):
            return False
    return True
```

Screenshots of working application



```
PS C:\Users\admin> python -u "f:\#STUDY\EEE\3-2\Crypto\crypto_project.py"
Enter No of transactions: 3
Abhi Raj 500
Raj Chirag 1000
Chirag Abhi 200
Is chain valid --  True
Time to mine --  71.84670042991638  seconds
Index --  9
traID --  611132656
Hash --  5AA747159F6266663D4C85E30C8F6B8D04BE3A29A6375FE86CF5E10DF0466E01
Previous Hash --  C1B4CF6D19802B25EFC1663B10BB8ADF9A8B4D81FD2CEE047AD59BA14A567CC6
Sender --  Abhi
receiver --  Raj
Amount --  500

Is chain valid --  True
Time to mine --  64.58953475952148  seconds
Index --  10
traID --  487669782
Hash --  C28A05B8C0F67797EA5D709F3E062CEABEFA8D0993C86F620FDA105BB337C83B
Previous Hash --  5AA747159F6266663D4C85E30C8F6B8D04BE3A29A6375FE86CF5E10DF0466E01
Sender --  Raj
receiver --  Chirag
Amount --  1000

Is chain valid --  True
Time to mine --  54.269871950149536  seconds
Index --  11
traID --  895954435
Hash --  80E105987A9A8E286F15BD9F846BF13718309979EBD2C22EBD414F9D08D040A9
Previous Hash --  C28A05B8C0F67797EA5D709F3E062CEABEFA8D0993C86F620FDA105BB337C83B
Sender --  Chirag
receiver --  Abhi
Amount --  200

Enter username: Abhi
Transaction ID --  611132656
Sent to --  Raj
Amount --  500

Transaction ID --  895954435
Recieved from --  Chirag
Amount --  200
```

On the output side, first, we need to enter the no. of transactions, followed by the sender-receiver and the amount of each transaction. Then the program computes whether the chain is valid before each transaction and the block is added to the list and stores the data in the database. After this, we are also printing each block (containing info about the index, transaction ID, hash, sender, receiver, amount, etc.) after the transaction is completed. We can also view transactions of a particular user by entering the username and the details are printed correspondingly.

This leads to a safe way of transactions and maintaining them using blockchain and databases.