A Simple Blockchain Implementation in Python

Sebastian Rodrigo M[1] and Cristian Lozano J[2]

***Abstract*— This project has the purpose of creating and implementing our own Blockchain, using the language program Python, as Blockchain has become a technology that can be used in a variety of fields, and has a dominant trend in software development, in secure information, and in transactions that do not require a third party involved, just the client, the customer, an the Blockchain needed.**

I. INTRODUCTION

In the technology that has evolved in our modern era the Blockchain has marked a new way of making transactions and has even given us a special variety of currency that can be used, where the Blockchain is used as the base for building the Bitcoin, which is the mos popular currency in the world.

The Bitcoin is also the first virtual currency, created by Satoshi Nakamoto, which created a tendency of creation of virtual currencies.

The purpose of this project is to launch our own virtual currency, and for that we need to create a Blockchain, responsible of maintaining a record of all the transactions that use our virtual coin, we called it Cybercoin. With it we can make transactions in any level, from an internal purchase of a product, to making a big purchase in an international service provider, if the provider agrees to join the network, and ultimately, using the virtual coin as currency for giving their services.

II. COMPONENTS IN THE DEVELOPMENT OF A BLOCKCHAIN

First lets talk about the major components of our project:

[1]srmedinac@unal.edu.co, cod. 2879665, Student of the Department of Software and Computing Engineering, National University of Colombia

[2]cclozanoj@unal.edu.co, cod. 2879666, Student of the Department of Software and Computing Engineering, National University of Colombia

A. Client

The client is the user of the Cybercoin, and he can supply or receive coins, depending of the transaction made, clients can sell services or supplies and boy another client services or supplies.

B. Miner

When a transaction is started by two Clients, this transaction is put in a pool, and the job of the Miner is to take the transaction, and assemble it as a block. This work has to be accompanied with a valid proof, or proof-of-work, so that the transaction can be finalized, and the Miner gets a reward for the work done, giving him a defined sum of the currency in place. With his rewards the Miner can act as a Client of the network.

C. Blockchain

The Blockchain is, as previously mentioned, the record of the transactions, which ends up being a chain of block mined, ordered in a chronological manner. The Blockchain can't be changed because of the use of it, and results in a well proved transaction record, secure in an inmutable way.

III. DEVELOPMENT

A. Client Development

We start the project with the definition of a Client class in our code, in which he will be able to have a wallet, and send, or receive, virtual coins to another Client. The way in which the money is spent is in the creation of a transaction, where he specifies the name of the sender, and the amount meant to be spent, and for receiving money the Client publish his identity to the paying Client, or the sender of the virtual coin.

The balance amount of the Clients wallet is not stored, because in the process of the transaction, we compute the real balance, ensuring the Client has a positive balance, enough to make the transaction.

So, for the development, we import the next libraries from Python:

```
import hashlib
import random
import string
import json
import binascii
import numpy as np
import pandas as pd
import pylab as pl
import logging
import datetime
import collections
# Cryptography libraries
import Crypto
import Crypto.Random
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
```

Those libraries define from the basic functionality of a Client, like using a random, or math operations, to a more complex set of options, like sign a transaction, or generate a hash based on an object, etc.

Next we define how the Client is going to execute the transactions, so we start defining a pair of keys, one public, and one private, using the RSA algorithm of Python. As priority, the private key should be saved in an alternative way, and the public key should be made public, so the network has a way of making transactions with the Client, we call this public key the identity of the Client, defined as follows:

```
class Client:
    def __init__(self):
        random = Crypto.Random.new().read
        self._private_key = RSA.generate(1024, random)
        self._public_key = self._private_key.publickey()
        self._signer  = PKCS1_v1_5.new(self._private_key)

    @property
    def identity(self):
        return binascii.hexlify(self._public_key.exportKey(format='DER')).decode('ascii')
```

B. Transaction Development

The next step for the development is to create a way in which a Client is going to perform a transaction, so he can send, or receive, money from another Client.

The transaction then needs three parameters to be used: first the public key of the sender, second the public key of the receiver, and finally the amount of money to deal in the transaction. Those parameters, including a time stamp for the transaction, are used in a way of dictionary so it can be accessible in all the transaction class.

So, for the beginning of a transaction, the first block in a Blockchain is called a Genesis block, which contains the first transaction of the Client, and thus the identity of this first Client can be a secret, so, for the project, we make the name of the first Client as 'Genesis' if he is the one who begins the transaction, or the identity of the external Client who stars it.

The last step in the transaction is the sign of the data provided as parameters. For this we use a built-on PKI with an algorithm SHA, and then we decode the signature for an ASCII representation for the storage in the Blockchain.

```
class Transaction:
    def __init__(self, sender, recipient, value):
        self.sender = sender
        self.recipient = recipient
        self.value = value
        self.time = datetime.datetime.now()

    def to_dict(self):
        if self.sender == "Genesis":
            identity  = "Genesis"
        else:
            identity = self.sender.identity

        return collections.OrderedDict({
            'sender': identity,
            'recipient': self.recipient,
            'value': self.value,
            'time': self.time
        })

    def sign_transaction(self):
        private_key = self.sender._private_key
        signer = PKCS1_v1_5.new(private_key)
        h = SHA.new(str(self.to_dict()).encode('utf-8'))
        return binascii.hexlify(signer.sign(h)).decode('ascii')
```

The way the transaction works is through a queue system, in which the Miners will take a transaction, and add it to the block. So, for the data of a transaction to be visible, we created a function that can show us that information:

```
def display_transaction(transaction):
    dict = transaction.to_dict()
    print("Sender: "+ dict[('sender')])
    print("Recipient: "+ dict[('recipient')])
    print("Value: "+ str(dict[('value')]))
    print("Time: "+ str(dict[('time')]))
    print('----------------------')
```

Together with a display_all' function that allow us to see all the transactions in the queue:

```
def display_all(transaction_q):
    for transaction in transaction_q:
        display_transaction(transaction)
        print('----------------------')
```

C. Block Development

The block used in a Blockchain is a number of transactions that vary in time, and for the project

purpose we assume that our block consist on a fixed number, three, of transactions, in which only the verified transactions are going to be added. The block also has the hash value of its previous block, for the blockchain to become inmutable.

For the project, we generate a class Block that contains the verified transactions, the hash of the previous block, and the Nonce made y the miner, like this:

```
class Block:
    def __init__(self):
        self.verified_transactions = []
        self.previous_block_hash = ""
        self.Nonce = ""
```

With that we can start using the Blockchain, with a defined user and an initial transaction, initializing the user, the money to be transfered, and the previous hash and Nonce of the Blockchain, appending the transaction to the transactions verified of the Blockchain in mention:

```
god = Client()
t0 = Transaction("Genesis",god.identity,1000.0)
block0 = Block()
block0.previous_block_hash = None
Nonce = None
block0.verified_transactions.append(t0)
```

D. Cryptocurrency Development

The virtual coin that it's going to be used has to be initialized, or in this case, it has to be filled with the blocks chained of the transactions. So, for the project, we create a list that it is going to contain them, and a way of visualization of all the Blockchain through time on the network.

The visualization of the Blockchain is achieved through a function where the length of the blocks are mentioned, and the transactions in each of the blocks, as follows:

```
def show_blockchain(self):
    print("Numer of blocks in the chain: " + str(len(self)))
    for i in range(len(Cybercoin)):
        block_temp = Cybercoin[i]
        print("Block # " + str(i))
        for transaction in block_temp.verified_transactions:
            display_transaction(transaction)
            print('--------------------------')
        print("==============================")
```

And finally we can append the blocks to the Blockchain, starting with the Genesis block, like this:

```
Cybercoin.append(block0)
show_blockchain(Cybercoin)
```

E. Miners Development

One of the last step in our project is the development of a functionality that allows mining of transactions, generating a result on a string, the transactions, and a proof-of-work.

For the result of the transaction we need to create a function in which the message is treated, in our project we used a sha256 function, that encodes the message to ASCII, and generates an hexadecimal, returning it as the digest of the message, ass follows:

```
def sha256(msg):
    return hashlib.sha256(msg.encode('ascii')).hexdigest()
```

And lastly, we create a mining function that allows us to generate a hash of the transaction, with a defined difficulty, and a prefix variable that allows us to verify the existence of the prefix in a matter of brutal force until a result can be given, as seen next:

```
def mine(msg, difficulty):
    assert difficulty >= 1
    found = False
    prefix = '1' * difficulty
    i = 0
    while found == False:
        i += 1
        digest = sha256(str(hash(msg)) + str(i))
        if digest.startswith(prefix):
            found = True
            print("Found Nonce: " + digest + ", after " + str(i) + " iterations.")
            return digest
```

F. Adding First Block

The last step of the development of the project is the aggregation of a block in the Blockchain, and that process can be achieved trough a series of code lines in which we first track the number of transactions mined, then create a Block instance, and start to mine the last three transactions of the queue.

One a miner finish its work, the transaction must be verified and, in case of a correct verification, the transactions mined are incremented, setting the last block hash to the one that has just been mined, and adding that block to the Blockchain, validating the full transaction between the Clients.

The code that allows us to deal with the block adding goes like this:

```
last_transaction_index = 0
block = Block()
for i in range(3):
    temp_transaction = transaction_queue[last_transaction_index]
    block.verified_transactions.append(temp_transaction)
    last_transaction_index += 1

block.previous_block_hash = last_block_hash
block.Nonce = mine(block,2)
digest = hash(block)
Cybercoin.append(block)
last_block_hash = digest
```

IV. Conclusion

This project shows a basic functional use of Blockchain, but there can be more improvements, like a way of managing the queue of transactions, or a way or organize the Transactions in a way that Miners can choose more paid works, or more fast ones.

We can conclude that tools like Ethereum, seen in class, have a really easy way of explaining Blockchains, and an interesting form of virtual currency, but, by making our own Blockchain, we can have more control over the transaction and validity of the software.

Lastly, we see the need to create a graphic interface for the Clients, in which they can make Transactions in a more easy and secure manner, or to make the mining process more accessible to Miners.

References

[1] Blockchain, https://www.blockchain.com/

[2] Bitcoin: A Peer-to-Peer Electronic Cash System, Nakamoto, Satoshi, https://bitcoin.org/bitcoin.pdf

[3] Bitcoin Core Integration/Staging Tree, Wladimir J. van der Laan, https://github.com/bitcoin/bitcoin

[4] PKI: Public Key Infrastructure System, DigiCert, https://www.digicert.com/pki/