**COMP4142 E-PAYMENT AND CRYPTOCURRENCY**

**Group Report**

**1. Implementation Details -**

**1.1 Blockchain Prototype**

Detailed description of the implemented blockchain structure based on the provided specifications.

**1. Block class**

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

For the sake of storing information in a block, we decided to create a ‘Block’ class. The following table will illustrate the meaning of each variable.

|  |  |  |  |
| --- | --- | --- | --- |
| Variable | Type | Explanation | |
| index | integer | The position of the current block in the blockchain |
| timestamp | string | The time when the block is created |
| proof | integer | The proof value is created by the proof of work algorithm |
| previous\_hash | string | The hash value of the previous block, created by a hash method |
| current\_hash | string | The hash value of the current block, created by a hash method |
| difficulty | integer | The dynamic difficulty is created by a method from Blockchain class |
| nonce | integer | A random number |
| merkle\_root | string | The root of the merkle tree of the current block |
| data | list of dictionary | Every transaction will be represented as a dictionary, and all transactions of the block will be stored in a list |

**2. Blockchain class**

The ‘Blockchain’ class is the main class in the entire program, it is responsible for storing all ‘Block’ and linking them together. The following section will discuss the main methods we developed for the purpose of constructing a blockchain.

2.1 Constructor

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

* self.transactions à create an empty list for storing transactions later
* self.chain à the blockchain (a list of ‘Block’ class)
* self.nodes à the node of the blockchain
* self.node\_id à the node id
* self.create\_block à a method to create a new block, one of the main methods. In this case, since the blockchain is just initialized, it will create the genesis block, setting proof to 1 and hash value of the previous block to “genesis” (no previous block)

2.2 Create a new block and more methods

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

The create\_block method create a new block and then append the block to the blockchain. One thing special is that transactions now are set to empty, and we will manipulate different methods to handle it afterward. Regarding all the methods applied in creating a block, the following section will explain them one by one:

* **index = len(self.chain)**

Simply equal to the length of the blockchain

* **timestamp = str(datetime.now())**

Equal to the current time

* **transactions = self.transactions**

The transaction list is set to empty now, further implementation will be explained in 4.3

* **proof = proof or self.proof\_of\_work(self.last\_block)**

\*The proof\_of\_work method will be explained in 4.2.2

* **previous\_hash = prev\_hash or self.hash(self.last\_block)**

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

The hash function is for creating a SHA-256 hash of a block. The input parameter is a block object, while returning a string as an output. In this case, the last block of the blockchain will be the input.

* **current\_hash = self.calculate\_current\_hash(proof,prev\_hash)**

**一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

自動產生的描述**

The calculate\_current\_hash takes proof and the hash value of the previous block as input, creates a preliminary block and stores these two value into it. After that, it use the abovementioned hash method to hash the block, eventually returning a hash value. (A preliminary block is needed since the hash method only take a block object as input)

* **difficulty = self.calculate\_difficulty()**

The calculate\_difficulty method will be explained in 4.2.1

* **nonce = self.calculate\_nonce(proof, prev\_hash)**

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

自動產生的描述

The calculate\_nonce method takes the proof and the hash of the last block as input, then applying validate\_proof method to validate the proof. It is done by hashing 2 values mentioned plus the proof value of the current block and checking whether the number of leading zero of this hashed value is equal to the number of difficulty. That is, if the difficulty is 4, the hashed value should be something like 0000xxx……. If it is correct (validated), the algorithm will return the proof value. If not, we add the proof by 1 each time until it passes the validation. In this way, we are able to adjust the proof and generate a hash until it has a hash with a leading number of zeros.

* **merkle\_root = self.calculate\_merkle\_root()**

一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

自動產生的描述

The calculate\_merkle\_root method is done by first hashing all the transactions in the block, then creating hash pairs of all the hashes. At the end, it returns hashes[0], which is the root of the merkle tree.

* **data = self.calculate\_data()**

**一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述**

The calculate\_data method is simply the transactions of the block.

After all the attributes are created, the Block will be appended to the Blockchain list.

**1.2 Dynamic-difficulty Proof-of-Work Algorithm**

1.2.1 Dynamic difficulty

一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

自動產生的描述

(This method is within the Blockchain class)

The dynamic difficulty is achieved by a method called calculate\_difficulty. We first can whether the size of the blockchain is too small, if it has only one block, a default difficulty 4 will be initialized. Then, we calculate the time taken to generate the previous block by subtracting the timestamp of the second last block to that of the previous block (time difference between their timestamp). If the time taken is larger than target time 10 (noted that target time is set to 10 at the beginning of the initialization of a Blockchain class), we minus the difficulty by 1, otherwise add 1. After all, return the difficulty as a integer.

1.2.2 Proof-of-Work algorithm

一張含有 文字, 螢幕擷取畫面, 軟體, 字型 的圖片

自動產生的描述

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

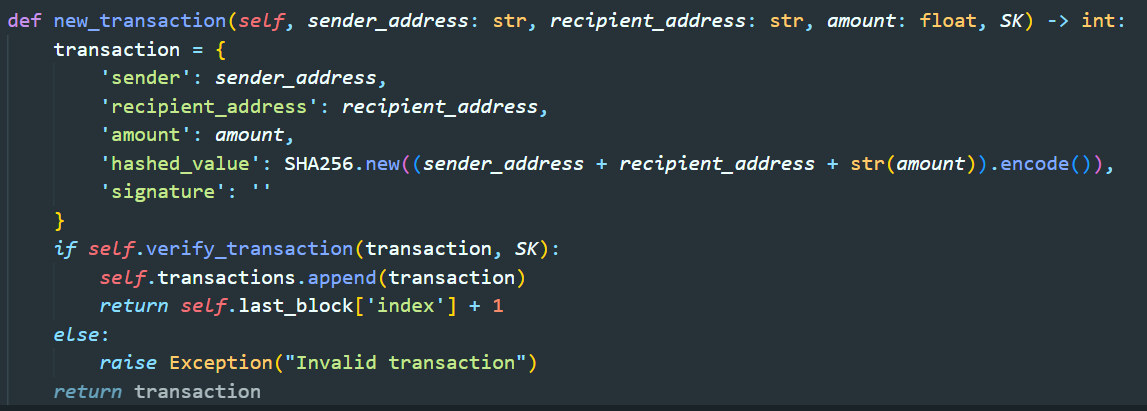
自動產生的描述

(These two methods are within the Blockchain class)

This is the proof of work algorithm. We first set the target proof to 0, then get the hash of the last block’s proof and the difficulty value. Next, we validate the proof. It is done by hashing all the 3 values mentioned and checking whether the number of leading zero of this hashed value is equal to the number of difficulty. That is, if the difficulty is 4, the hashed value should be something like 0000xxx……. If it is correct (validated), the algorithm will return the proof value. If not, we add the proof by 1 each time until it passes the validation. In this way, we are able to adjust the proof and generate a hash until it has a hash with a leading number of zeros.

**1.3 Transaction (car)**

1.3.1 P2PKH Transactions

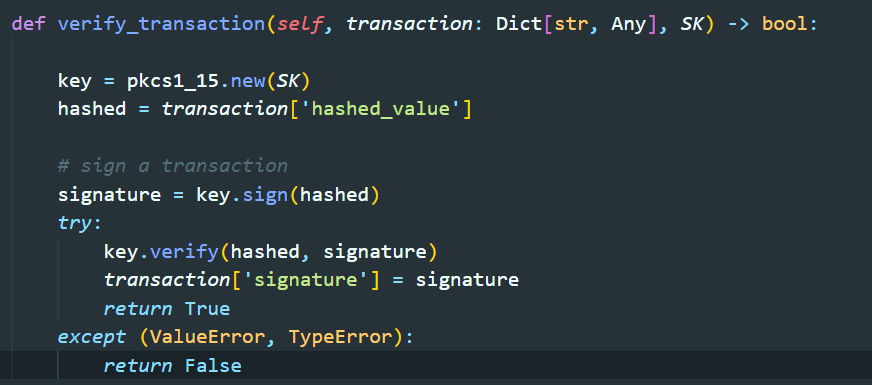


Regarding the transaction creation part, we have designed a method called ‘new\_transaction’ to accepts sender, recipient addresses, an amount and the sender’s private key ‘SK’. We create a transaction dictionary containing sender, recipient, amount and a hashed value derived from amount and the address of sender and recipient. Besides we adopt another method ‘verify\_transaction’ to verify the transaction that we created.

This can achieve the P2PKH transaction structure by incorporating sender and recipient addresses, ensuring security by using cryptographic hashing to create a unique hashed value for the transaction

A P2PKH transaction could provide us with enhanced security by using public –private key pairs and digital signatures. It ensures that only the owner of the private key can authorize and validate their authenticity without revealing any sensitive information.

1.3.2 Digital signatures and transaction verification with asymmetric cryptography



In terms of verifying transaction, the method ‘verify\_transaction’ that we created takes the transaction dictionary and the sender’s private key (‘SK’) as the parameter.

We would use the ‘hashed value’ of the transaction and sign it with the sender’s private key to create a signature. Then, the signature will be used to verify against the hashed value using the sender’s public key.

In order to create digital signatures using asymmetric cryptography, we have leverage the module called ‘Crypto.Signature’ to generate and verify digital signatures. The ‘verify\_transaction’ method uses the sender's private key to sign the hashed value of the transaction and then verifies this signature with the public key. If successful, the transaction will be considered as a valid transaction. By adopting this method in the process of verification, it can ensure the integrity of each transaction.

**1.4 Network**

1.4.1 API for broadcast

A screen shot of a computer program

Description automatically generatedA computer screen shot of text

Description automatically generated

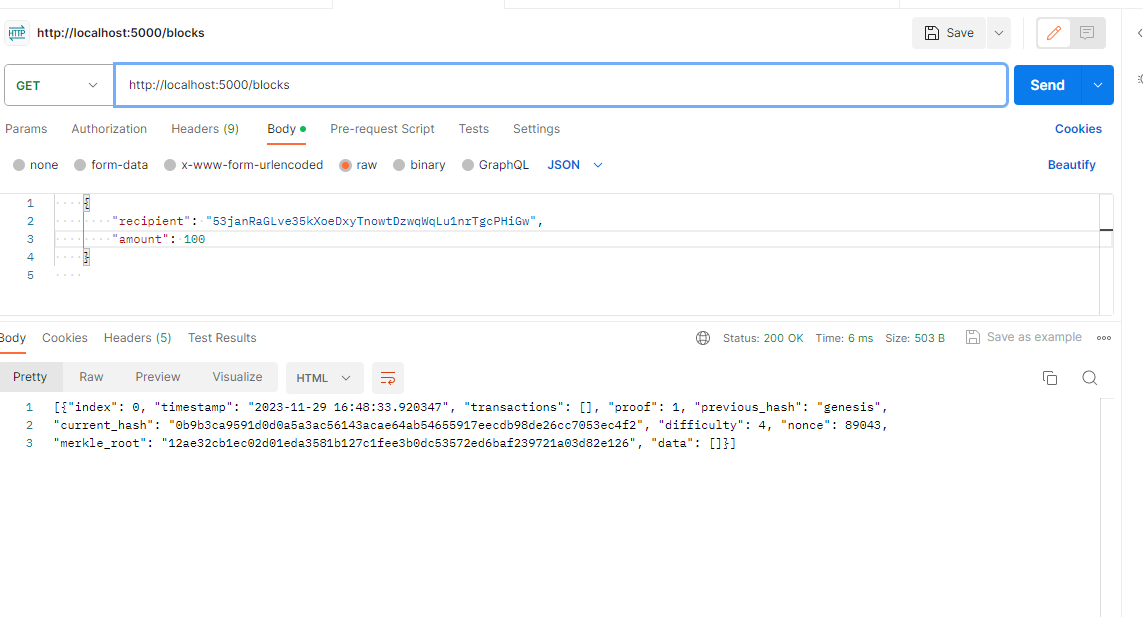
TO implement the API that enable the broadcasting and retrieval of new blocks from other nodes. The API should allows user to interact with the blockchain through HTTP requests, socket or different ports. In below, we will demonstrate the API.

1. /blocks(POST): This endpoint is used to receive a new block from other nodes in the network. The block data is extracted from the request's JSON payload and stored in the blockvariable. If the block is valid, it is appended to the blockchain's chain blockchain.chain.append(block), and a success message is returned. Otherwise, an error message indicating an invalid block is returned.
2. /blocks(GET): This endpoint is used to retrieve the entire chain of blocks from the blockchain network. It returns the chain as a JSON response using json.dumps(blockchain.chain).

A screenshot of a computer

Description automatically generated

1. /balance(GET): This endpoint retrieves the balance of the wallet associated with the current user. The get\_balance function from the wallet object is called. It returns JSON response of balance: XXXXXX.
2. /wallet/address(GET):This endpoint retrieves the public address of the wallet associated with the current user. The public\_key\_to\_address function from the wallet object is called to convert the public key to an address. It returns as a JSON response of address address:XXXXXX.



1. /create\_transaction(POST): This endpoint is used to create a new transaction on the blockchain. The recipient and amount of the transaction are extracted from the request's JSON payload (request.get\_json()) and stored in recipient and amount variables, respectively. The create\_transaction function from the wallet object is then called to generate the transaction. The resulting transaction is returned as a JSON response using jsonify(transaction).

1.4.2 function to check if new blocks receive from others are valid or not.

In the /blocks (POST) endpoint, after receiving the block data from the request payload, the code calls the validate\_block function from the blockchain object to check the validity of the block. This function likely performs the hash comparison to validate the block. A screen shot of a computer code

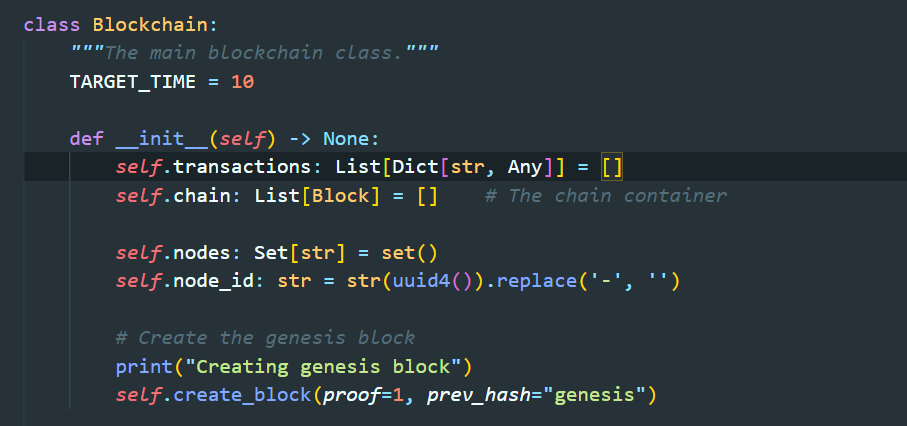
Description automatically generated

The validate\_block function compare the recomputed hash of the block with the given hash. If the hashes match, the block is considered valid and is appended to the blockchain's chain. An response “Block added to the chain” showed. Otherwise, if the hashes do not match, the block is deemed invalid, and an error response “Invalid block” is returned.

**1.5 Storage (car)**

Regarding storage capability, our code able us to store the latest state of blockchain in memory and store the transaction (UTXO) in a transaction pool respectively. I will explain in below.

**1.5.1 Latest state of the blockchain in memory**

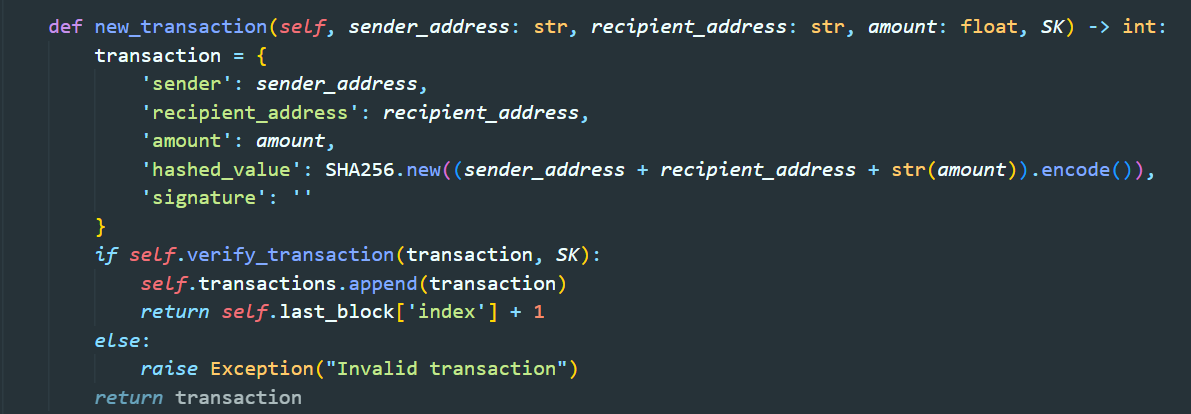


Regarding blockchain representation. We have defined the ‘self.chain’ as the attribute to store the entire blockchain in memory, which can maintain a record of all the blocks in our blockchain.

Besides, we have designed the code to trace the node by using two attributes, respectively the ‘self.nodes’ and ‘self.node\_id’.As for the ‘self.nodes’, it is a set that tracks the nodes involved in the blockchain network.On the other hand,’self.node\_id’ represents the unique identifier toward each node, which can help us identify the node within the blockchain network.

Various attributes (self.nodes, self.node\_id, etc.) within the Blockchain class manage the current state of the blockchain system. Regarding class ‘Blockchain’,these attributes remain in memory during program execution.

**1.5.2 Storing transaction (UTXO) in a Transaction Pool**



The self.transactions attribute serves as a transaction pool, holding pending transactions until they are included in blocks, effectively managing the UTXOs.All the pending transactions will be stored before included blocks. When a new transaction is created ‘new\_transaction’ , it will be added to self.transactions, holding the Unspent Transaction Outputs (UTXOs) until they are processed into blocks.

**1.6 Wallet (car)**

**Regarding implementing the functionalities of the crypto wallet, our code simply breakdown into several key functions, respectively create transaction, managing UTXO, calculating the balance and generating addresses.**

**4.6.1 Wallet Initialization**

**一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

自動產生的描述**

As for the first step, our group has defined a class named ‘Wallet’ which is initialized with a private key, public key, and an empty list ‘utxo’. Basically, a crypto wallet contains three main variables, respectively the private key, public key, and the list of unspent transactions output.

**4.6.2 Transaction Creation**

**一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述**

Regarding the process of a crypto wallet transaction, the transaction contains the public key of the sender, recipient’s address, and the amount. Besides, the transaction would be signed by the private key and being attached with a signature.

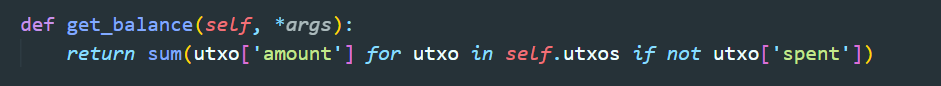
To achieve the functionalities of this process, we have designed a method in class ‘Wallet’ named ‘create\_transaction’. This method could help us construct a transaction object containing sender's public key, recipient's address, and the amount. After that, it could use the private key to sign the transaction.

**1.6.3 UTXO Management and Balance Calculation**

**一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述**

We have designed the method named ‘update\_utxos()’, which can updates the UTXOs list by removing spent outputs (inputs) and adding new unspent outputs (outputs) from a transaction.



In order to calculate the balance, we designed another method named ‘get\_balance()’, which could calculates the wallet's balance by summing the amounts of unspent transaction outputs (utxos).

**1.6.4 Address Generation**

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

Basically, the public key is equal to addressed but it is hashed. The method ‘public\_key\_to\_address()’ could help hash the public key and encode it into a base58 format, generating the readable wallet address.

**1.6.5 Integration with the blockchain**

**一張含有 文字, 螢幕擷取畫面, 軟體, 字型 的圖片

自動產生的描述**

The transaction should be integrated with the blockchain. To achieve this goal, we have designed a route for the method ‘create\_and\_add\_transaction()’ to receive transaction request (“POST”). Then, the method would create a transaction using the wallet's method ‘create\_transaction()’ , and finally adds it to the blockchain using blockchain.new\_transaction()

Reference:

<https://github.com/SentinelWarren/blockchain_proto>