

# EE4017 Internet Finance

## Laboratory 5 – Blockchain

### 1. Overview

The purpose of this laboratory assignment is to familiarize with PyCharm as an integrated development environment (IDE) to develop python applications. The assignment utilizes PyCharm to build a simple python Blockchain application.

### 2. Objective

The students are expected to be familiar with PyCharm and Python as a development tool. Also, they are expected to be familiar with the properties of Blockchain

There are two checkpoints:

- Craft a transaction
- Generate new block with Proof-of-work

### 3. Environment and tools

This assignment is implemented using Python3 and 'hashlib' module.

All required packages listed below have been pre-installed:

- Python 3.6.8
- PyCharm IDE 2018.3.7 (Community Edition)

\*hashlib module is integrated with Python3

### 4. Import python modules

Import the following modules at the beginning of your python application:

```
from hashlib import sha256
import datetime
import Crypto
import Crypto.Random
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
from Crypto.Hash import SHA
import binascii
import json
```

## 5. JSON file format

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. In this tutorial, we will use JSON as a data exchange file format because we need to communicate with other peers when the whole blockchain network is established.

## 6. Creating Block Class

To begin with the previous tutorial on Transaction, a basic block consists of 4 parameters including index, transactions, timestamp and the hash of the last block. Hash value and nonce also exists but we need to calculate it after adding the transactions.

```
class Block:
    def __init__(self, index, transactions, timestamp, previous_hash):
        self.index = index
        self.transactions = transactions
        self.timestamp = timestamp
        self.previous_hash = previous_hash
        self.hash = block_hash
        self.nonce = nonce
```

Then, we need a method to calculate the hash value of a block. Note that we don't need the empty hash value as the input for the hash function. To resolve this, a method that dumps selected variables is needed. Also, 'to\_json' method is needed for transferring blocks to other peers using json format.

```
def to_dict(self):
    return {
        'index': self.index,
        'transactions': self.transactions,
        'timestamp': self.timestamp,
        'previous_hash': self.previous_hash,
        'nonce': self.nonce
    }
def to_json(self):
    return json.dumps(self.to_dict())
def compute_hash(self):
    return sha256(str(self.to_dict()).encode()).hexdigest()
```

## 7. Creating Blockchain Class

```
class Blockchain:
    difficulty = 2

    def __init__(self):
        self.unconfirmed_transactions = []
        self.chain = []
        self.create_genesis_block()

    def create_genesis_block(self):
        block_reward = Transaction("Block_Reward",
                                    myWallet.identity, "5.0").to_json()
        genesis_block = Block(0, block_reward, datetime.datetime.now().
                               strftime("%m/%d/%Y, %H:%M:%S"), "0")
        genesis_block.hash = genesis_block.compute_hash()
        self.chain.append(genesis_block.to_json())
```

The blockchain class has 2 important elements: unconfirmed transactions and the blockchain itself. The first block in the blockchain is the Genesis block. The above code creates and puts the genesis block into the blockchain. The next thing to do is creating a method to handle new transactions. By using the code in the last tutorial, we can create this method easily. Note that we need to check the signature before adding it to the list of unconfirmed transactions pool.

```
def add_new_transaction(self, transaction: Transaction):
    if transaction.verify_transaction_signature():
        self.unconfirmed_transactions.append(transaction.to_json())
        return True
    else:
        return False

def add_block(self, block, proof):
    previous_hash = self.last_block['hash']
    if previous_hash != block.previous_hash:
        return False
    if not self.is_valid_proof(block, proof):
        return False
    block.hash = proof
    self.chain.append(block.to_json())
    return True
```

The next thing we need is implementing Proof-of-work in our blockchain system. Since we might receive blocks from another node, a validation method is also needed.

```
def is_valid_proof(self, block, block_hash):
    return (block_hash.startswith('0' * Blockchain.difficulty) and
            block_hash == block.compute_hash())

def proof_of_work(self, block):
    block.nonce = 0
    computed_hash = block.compute_hash()
    while not computed_hash.startswith('0' * Blockchain.difficulty):
        block.nonce += 1
        computed_hash = block.compute_hash()
    return computed_hash
```

Then, we need a mining method to generate new blocks and claim the block reward. This method writes unconfirmed transactions into blocks by using the proof-of-work method. Note that we use JSON to store the transaction in the blockchain because JSON format is easy to read for computer programs.

```
def mine(self, myWallet):
    block_reward = Transaction("Block_Reward",
                               myWallet.identity, "5.0").to_json()
    self.unconfirmed_transactions.insert(0, block_reward)
    if not self.unconfirmed_transactions:
        return False

    new_block = Block(index=self.last_block['index'] + 1,
                      transactions=self.unconfirmed_transactions,
                      timestamp=datetime.datetime.now().
                        strftime("%m/%d/%Y, %H:%M:%S"),
                      previous_hash=self.last_block['hash'])

    proof = self.proof_of_work(new_block)
    if self.add_block(new_block, proof):
        self.unconfirmed_transactions = []
        return new_block
    else:
        return False
```

The last block could be frequently asked by another node. We can add a property to handle such query.

```
@property
def last_block(self):
    return json.loads(self.chain[-1])
```

Finally, the blockchain system is ready. We can test our system by sending a fake transaction.

```
blockchain = Blockchain()
Alice = Wallet()
Bob = Wallet()

t = Transaction(Alice.identity, Bob.identity, 5.0)
t.add_signature(Alice.sign_transaction(t))
blockchain.add_new_transaction(t)

blockchain.mine(Alice)
print(blockchain.last_block._dict_)
blockchain.mine(Alice)
print(blockchain.last_block._dict_)
```

```
-----
{'index': 1, 'transactions': [{'sender': 'Block_Reward', 'recipient': ...
{'index': 2, 'transactions': [{'sender': 'Block_Reward', 'recipient': ...
```

### **Checkpoint:**

- Craft a transaction
- Generate new block with Proof-of-work. A sample program of "proof of work" was upload under Week 6 Lab session

### **What to submit to CANVAS?**

- 1) To apply "proof of work" from lab 3 to generate a new block. A sample program of "Proof of Work" was upload under Week 6 Lab session. You may use it if you have problem of your own.
- 2) A sample transaction signed and exported in JSON format
- 3) The screen shots of your work

**Please save your project for the next tutorial and the mini project!!!**

## References

- 1 [https://www.tutorialspoint.com/python\\_blockchain/index.htm](https://www.tutorialspoint.com/python_blockchain/index.htm)
- 2 <http://adilmoujahid.com/posts/2018/03/intro-blockchain-bitcoin-python/>
- 3 <https://www.ibm.com/developerworks/cn/cloud/library/cl-develop-blockchain-app-in-python/index.html>
- 4 [https://en.bitcoin.it/wiki/Main\\_Page](https://en.bitcoin.it/wiki/Main_Page)
- 5 <https://bitcoin.org/bitcoin.pdf>
- 6 <https://www.json.org/>