ECE 8930: Blockchain Technology and Web 3.0

Project 1

Saurabh Sharma
10/13/2023

Q How to run the code -
Make sure python is installed on the computer.
Then run the following command -
python crypto.py

Q About the code and it's output
The following code creates a transaction object which has sender, recipient, amount and previous transaction's hash value.

```
def create_transaction(sender_name, recipient_name, amount):
    # Create a transaction object
    transaction = {
        'sender': sender_name,
        'recipient': recipient_name,
        'amount': amount,
        'prev' : globalprevtranshash
    }
    return transaction
```

Below code signs and verifies the transaction we have just created.

```
def sign_transaction(transaction, private_key):
    # Convert the transaction to a string for hashing
    transaction_string = str(transaction)
    transaction_hash = hashlib.sha256(transaction_string.encode()).hexdigest()

    # Sign the transaction hash with the private key
    signature = rsa.sign(transaction_hash.encode(), private_key, 'SHA-256')
    return signature


#We verify the transaction using the public key. i.e. implemented digital signat
def verify_transaction(transaction, signature, public_key):
    # Convert the transaction to a string for hashing
    transaction_string = str(transaction)
    transaction_hash = hashlib.sha256(transaction_string.encode()).hexdigest()

    # Verify the signature using the public key
    try:
        rsa.verify(transaction_hash.encode(), signature, public_key)
        return True
    except:
        return False
```

Defines the structure of a block. It has previous block's hash, it's hash and the merkle root of all it's transactions.

```
class Block:
    def __init__(self, previous_hash, transactions):
        self.previous_hash = previous_hash
        self.transactions = transactions
        self.merkle_root = self.calculate_merkle_root()
        self.curhash = str(hash(self.previous_hash + str(self.merkle_root)))
```

Calculates the merkle tree and returns the merkle root of all the transactions to be put in a block.

```
def calculate_merkle_root(self):
    def merkle_tree(tx_list):
        if len(tx_list) == 1:
            return tx_list[0]

        new_tx_list = []
        for i in range(0, len(tx_list)-1, 2):
            combined = tx_list[i] + tx_list[i+1]
            new_tx_list.append(hashlib.sha256(combined.encode()).hexdigest())

        if len(tx_list) % 2 == 1:
            new_tx_list.append(hashlib.sha256(tx_list[-1].encode()).hexdigest())

        return merkle_tree(new_tx_list)

    tx_hashes = [hashlib.sha256(tx.encode()).hexdigest() for tx in self.transactions]
    return merkle_tree(tx_hashes)
```

Defines the blockchain itself and has functions to add a block object to it (it calls merkle tree function) and displays the complete blockchain which we'll see in the outputs section.

Saurabh Sharma

```python
class Blockchain:
    def __init__(self):
        self.chain = [self.genesis_block()]

    def genesis_block(self):
        return Block(previous_hash="0", transactions=["Genesis Transaction"])

    def add_block(self, transactions):
        previous_block = self.chain[-1]
        new_block = Block(previous_hash=previous_block.curhash, transactions=transactions)
        self.chain.append(new_block)

    def display_chain(self):
        for i, block in enumerate(self.chain):
            print(f"Block {i}")
            print(f"Previous Hash: {block.previous_hash}")
            print(f"Merkle Root: {block.merkle_root}")
            print(f"Current Hash: {block.curhash}\n")
```

All 12 transactions between 6 Users are defined in below code.

```python
for i,[x,y,z] in enumerate([["1", "2", "80"], ["1","1","20"] , ["2", "3", "75"], ["2","2","5"] ,["3", "4", "70"], ["3","3","5"] ,["4", "5", "65"],["4","4","5"] , ["5", "6", "60"], ["5","5","5'
    sender_name = x
    sender = public_key
    recipient_name = y
    recipient, _ = rsa.newkeys(512)  # Generate a new recipient's key pair for each transaction
    amount = z
```

Creates transactions and verifies it after creating a unique public key and private key pair for the receiver.

```python
transaction = create_transaction(sender_name, recipient_name, amount)
signature = sign_transaction(transaction, private_key)
is_valid = verify_transaction(transaction, signature, sender)
```

Every 4 transactions we are mining a new block and adding their merkle root to it. It is done in the below code

```python
if len(globaltransactionlist)==4:
    blockchain.add_block(globaltransactionlist)
    globaltransactionlist = []
    blockchain.display_chain()
```

—--- OUTPUT —--

Saurabh Sharma

Running the code -

```
C:\Users\Necro>cd Desktop

C:\Users\Necro\Desktop>python crypto.py
```

The output of 4 (and their 4 complimentary) transactions -

```
C:\Users\Necro\Desktop>python crypto.py
Transaction 1:
  Transaction: {'sender': '1', 'recipient': '2', 'amount': '80', 'prev': None}
  Signature: b"&\xe1\xdcl\x96\xd7Pf\x88\xcf\xb9\x98\x13\xfe'$\x89'\x9dr\x83\x800G\x17\x9b\xeb\xacgQ\xc8\xc6cA[\x1fl\xae\x03\x8e$(\x0e\xa5\x18Y\x176CI\x88\xcb\xa9\xce\x1a\tA:\
x10\xc4\x13^\x03e"
  Is Valid: True
==============================
Transaction 2:
  Transaction: {'sender': '1', 'recipient': '1', 'amount': '20', 'prev': '1668412945945737553'}
  Signature: b"\x11\xb1\x1cqN\x16/\x9cx\xac\xfe\xcf^\x9d{\xaf\x82'\xa4U\x15\xdf\xf12\xa5\x1f\x17\xe5\x1b+\xe0;\tq\xe9+XO\x80<\xefAN<\xb1^\xdd$\xe8M\xfd\x8af\xdd\xb8\xc8h\xab\
x9d\xc5Bd\x1b\xcf"
  Is Valid: True
==============================
Transaction 3:
  Transaction: {'sender': '2', 'recipient': '3', 'amount': '75', 'prev': '4270726284607269330'}
  Signature: b'\x87\xaci\x0c-\x10h\x06\x88\xa9\r)\xf2f\xaba\xa8\xe9\x85\xc6(\x04bvS\x15\x19\xda\xd1QB\x19Q)\xd3\xb1||r\x1b\xee!\xb1e/\xd4\xfdJ\xcf\x84O@&\xe4\xfed-\x84\xb6\x0
e\xe0\x91\x0cj'
  Is Valid: True
==============================
Transaction 4:
  Transaction: {'sender': '2', 'recipient': '2', 'amount': '5', 'prev': '-8385787380432526686'}
  Signature: b'\rj\xc73x\xf5\xf7\xf9\t\x04\xe5k\xad\xcf\x99\xeb4\x1eL:\x90\xa5\xbcz\xcf5\xf7\xae\x02\x01Qp\xf5\xd9\x15\xf1\x97=\x1d1\x98\xa0+G\xa5w\xf3\xe9\x00\xfd\x1d\xc4WL\
xd9\xa5+\x14,\xebh7Y\xfc'
  Is Valid: True
==============================
Transaction 5:
  Transaction: {'sender': '3', 'recipient': '4', 'amount': '70', 'prev': '1256332313568323739'}
  Signature: b'VG=z\x13\xb9\x1c\xead\xc8\xbf\x8d,\xad\x13n\x03\xcb\xc6;V\xc2C\x1c\xf8S\x90\xb5\x1d\xe6o\x1e\xb0\xaf\xaa\xf9\xf3!\xa4lh53C_\xf3cK\xef5\x08#\x0eo\xf8\xa7\xe6\xa
8H`\xa6\xecZ\n'
  Is Valid: True
==============================
Transaction 6:
  Transaction: {'sender': '3', 'recipient': '3', 'amount': '5', 'prev': '7256650913894794244'}
  Signature: b'#*\x80U\x10\xaa\xff\xf9\x07\xac\r\x15s"|.\xd8\xea\x1a!\x99f\x9e\xb40\x90\x95\xf7\xe9v\x8c/\xaf\xb2\xb9\x8e\x8e\x1b\xde\x05J\x9a\x8a\xcd\x8b4\x10\xfd\xdf\xba\x8
0\xc62\x02\xaa\'\x90\xcf\xfe\xedo\xb9\x88$'
  Is Valid: True
==============================
```

```
==============================
Transaction 6:
  Transaction: {'sender': '3', 'recipient': '3', 'amount': '5', 'prev': '7256650913894794244'}
  Signature: b'#*\x80U\x10\xaa\xff\xf9\x07\xac\r\x15s"|.\xd8\xea\x1a!\x99f\x9e\xb40\x90\x95\xf7\xe9v\x8c/\xaf\xb2\xb9\x8e\x8e\x1b\xde\x05J\x9a\x8a\xcd\x8b4\x10\xfd\xdf\xba\x8
0\xc62\x02\xaa\'\x90\xcf\xfe\xedo\xb9\x88$'
  Is Valid: True
==============================
Transaction 7:
  Transaction: {'sender': '4', 'recipient': '5', 'amount': '65', 'prev': '-955060125790466818'}
  Signature: b'0E\x12Z\x9d\xd6\x9a\x9d\x02:8-\x8f\x06\xa6\xe0\x8f\xbe\x8e\xe6\xbe&\xbeX\x91\x93SG\x00>\xe1)\xd0\x19i\x94\xc8+\xa1V\x83.\xf5\xbc\x9a\r\xcaI"uv\x87\xed\xfc\x0f\
xb4\x905\x1f\x8c\xd5G\xe2\x15'
  Is Valid: True
==============================
Transaction 8:
  Transaction: {'sender': '4', 'recipient': '4', 'amount': '5', 'prev': '-2001315893792004918'}
  Signature: b'\x82\x16\xe2p\x99}DO\xb7gi\xb9\xf9\xd3\xeb\x02o\xa6\x88\\\xc9d\xf9\x8dpq6\x0eF\xca\xcd\x94g\xc9\xc34"\x13d\xf5\xaa\x1b\xbd\x04\xbe3\x8fN\x11Zp\x9f\x92\xd6\xa38
\x13\xc4\xf0\r\x0c\xc9+\xa6'
  Is Valid: True
==============================
```

After every 4 transactions, a block is mined, and its output is shown below.

```
Block 0
Previous Hash: 0
Merkle Root: 78242c0d0e007f3f2ec67699880fb78d12cbbec434bd839d5b5a1f93aa1abb42
Current Hash: -6437589006464395513

Block 1
Previous Hash: -6437589006464395513
Merkle Root: b5adfb0569f65428869249e181a1e55d3306c6a12f3eb7b29b93d3a20731b1c8
Current Hash: -3607724073803336985
```