

# Blockchain Technology and Quantum Computations - Blockchain implementation report

Filip Kokosza 145210      Szymon Czajkowski 145384

June 14, 2023

# Contents

<b>1</b>	<b>Implementation</b>	<b>3</b>
1.1	Design decisions . . . . .	3
1.2	Classes . . . . .	4
1.2.1	User . . . . .	4
1.2.2	Transaction . . . . .	5
1.2.3	Block . . . . .	6
1.2.4	Blockchain . . . . .	7
1.2.5	NetworkNodes - Blockchain nodes . . . . .	8
<b>2</b>	<b>Application interface</b>	<b>9</b>
2.1	Login screen . . . . .	9
2.2	User menu . . . . .	10
2.3	Transaction list . . . . .	10
2.4	Blockchain . . . . .	11
<b>3</b>	<b>Test</b>	<b>12</b>
3.1	Testing platform . . . . .	12
3.2	Results . . . . .	12

# 1 Implementation

## 1.1 Design decisions

- User password hash is stored to ensure security
- Database is replaced by `.pkl` files which stores the current state of the blockchain system
- The verification process of the transaction and the mined block is handled indirectly by Users through the `NetworkNodes` class
- Clean UI is provided in a console rather than in a proper front-end application
- User has to log in to perform any operations on the blockchain system
- Proof-Of-Work mechanism was chosen as a consensus mechanism

## 1.2 Classes

### 1.2.1 User

User class has the following properties:

**name** Name of a user.

**token** Hash value calculated from users password.

**public\_key** Public key of a user.

**private\_key** Private key of a user.

User class defines the node in Blockchain system. When new User enters the Blockchain network it receives pair of *Public* and *Private Keys*. With which user can sign newly created transaction.

In real world blockchain system, a user is able to verify both new transaction and new block that was mined, but given the fact that this implementation purpose is to demonstrate processes that take place in blockchain system, the verification functions are handled by `NetworkNodes` class shown later in this paper.

```
class User:

    def __init__(self, name: str, pwd_hash):
        self.name = name
        self.token = pwd_hash
        self.public_key: rsa.PublicKey
        self.private_key: rsa.PrivateKey
        self.__generate_key_pair()

    def __str__(self):
        return f"{self.name}"

    def __generate_key_pair(self):
        self.public_key, self.private_key = rsa.newkeys(keyLength
        )

    def sign(self, transaction):
        transaction.signature = rsa.sign(
            str(transaction).encode('ascii'), self.private_key, '
            SHA-256')
```

Code Listing 1: User class

### 1.2.2 Transaction

Transaction class has the following properties:

- id** unique identifier of a transaction
- sender** user that sends initiate a transaction
- receiver** user that is recipient of a transaction
- amount** transaction amount
- signature** signature of transaction made by sender

Transaction class represents a single transaction that can be added by a User. After the transaction is created, the sender is obligated to sign it with his *PrivateKey*.

```
class Transaction:

def __init__(self, sender: User, receiver: User, amount:
float):

    self.id = uuid.uuid1()
    self.sender = sender
    self.receiver = receiver
    self.amount = amount
    self.signature = b'0'
    self.sign_transaction()

def __str__(self):
    return f"Id: {self.id}\n{self.sender} -> {self.receiver}:
{self.amount}"

def sign_transaction(self):
    self.sender.sign(self)
```

Code Listing 2: Transaction class

### 1.2.3 Block

Block class has the following properties:

```
prev  hash  hash of a previous block in blockchain
```

**transactions** list of a transactions in block

proof of work calculated proof of work

**hash** hash of current block

Block class is provided with function that enables to calculate its hash. After new transaction is added to the transaction list it automatically recalculates its hash.

The `calculate_proof_of_work` function is used in the mining process. It calculates the Proof-Of-Work that is needed in order to achieve current target hash.

```
class Block:

    def __init__(self, transactions, prev_hash):
        self.prev_hash = prev_hash
        self.transactions = transactions[:]
        self.proof_of_work = 0
        self.hash = self.calc_hash()

    def __str__(self):
        tmp = f"\nHash: {self.hash}\nTransactions ----- \n\n"
        for x in self.transactions:
            tmp += f"{str(x)}\n"
            tmp += f"Signature: {binascii.hexlify(x.signature).  
                        decode('ascii')}\n\n"
            "
        tmp += f"End Transactions ----- \nProof of work: {self.  
                        proof_of_work}\n"

        return tmp

    def calc_hash(self):
        temp = str(self.prev_hash)
        for x in self.transactions:
            temp += str(x)
        temp += str(self.proof_of_work)
        return hashlib.sha256(temp.encode()).hexdigest()

    def add_transaction(self, transaction: Transaction):
        self.transactions.append(transaction)
        self.hash = self.calc_hash()

    def calculate_proof_of_work(self):
```

```

while not self.hash.startswith("0" * numberOfZeros):
    self.proof_of_work += 1
    self.hash = self.calc_hash()

```

Code Listing 3: Block class

#### 1.2.4 Blockchain

Blockchain class has the following properties:

**chain** list of blocks currently added to the blockchain

Blockchain class `add_block` function when called adds new block to the blockchain. Before the addition the block hash is calculated to fit the target.

Function `get_tail_hash` is a helper function that returns hash of last block in the blockchain.

```

class Blockchain:

    def __init__(self):
        self.chain = [self.create_init_block()]

    def __str__(self):
        tmp = ''
        for x in self.chain:
            tmp += "\nBlock -----"
            tmp += str(x)
            tmp += "End block -----"
        return tmp

    def create_init_block(self):
        return Block([], 0)

    def add_block(self, new_block: Block):
        new_block.prev_hash = self.chain[-1].hash
        new_block.calculate_proof_of_work()
        self.chain.append(new_block)

    def get_tail_hash(self):
        return self.chain[-1].hash

```

Code Listing 4: Blockchain class

### 1.2.5 NetworkNodes - Blockchain nodes

NetworkNodes class has the following properties:

**blockchain** blockchain object

**users** list of users in the blockchain

The NetworkNodes class represents Blockchain nodes. Because this implementation doesn't include storage of the user list and blockchain in the database and doesn't provide simultaneous access to the application by multiple users, this class only imitates this behaviour acting as a central point where user list and blockchain copy is stored.

Also, the user list and blockchain objects are stored in .pkl files locally, which means that while using this application locally after exiting it, a user can re-open it and the state of blockchain and user list will be loaded into this application.

Verification of newly added transaction and block by Blockchain nodes is imitated by `verify_transaction` and `verify_block`. The `mine_block` initiated mining process of a block.

```
class NetworkNodes:

    def __init__(self, blockchain: Blockchain, users: list[User]):
        self.blockchain = blockchain
        self.users = users

    def get_userPubKey(self, sender: User) -> rsa.PublicKey:
        return next((user.public_key for user in self.users
                      if user.name == sender.name))

    def find_user(self, name):
        try:
            next((user.public_key for user in self.users
                  if user.name == name))
        except Exception:
            return False
        return True

    def verify_transaction(self, transaction: Transaction):
        try:
            return rsa.verify(
                str(transaction).encode('ascii'),
                transaction.signature,
                self.get_userPubKey(transaction.sender))
            == 'SHA-256'
        except Exception:
            return False

    def verify_block(self, block: Block):
```



```
    for tx in block.transactions:
        if not self.verify_transaction(tx):
            return False
    if block.calc_hash() == block.hash:
        return True

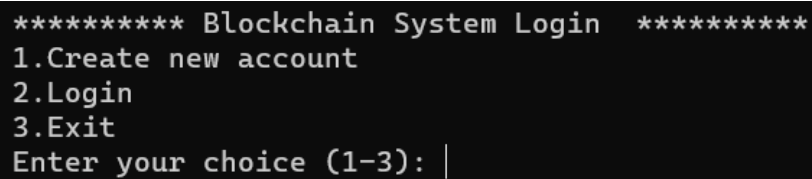
def mine_block(self, block: Block):
    self.blockchain.add_block(block)
    return self
```

Code Listing 5: NetworkNodes class

## 2 Application interface

### 2.1 Login screen

Here users can create a new account, log in to an existing one or exit the application.



```
***** Blockchain System Login *****
1.Create new account
2.Login
3.Exit
Enter your choice (1-3): |
```

Figure 1: Login menu

## 2.2 User menu

User menu allows the logged user to display the current state of the blockchain, add new transactions, show transactions that are not yet added to the blockchain and mine a block.

```
Currently logged as: Filip           Transactions: 4
----- Blockchain User Interface -----
1. Show current blockchain
2. Add new transaction
3. Show transactions not in blockchain
4. Mine block of current transactions
5. Logout
Enter your choice (1-5): |
```

Figure 2: User menu

## 2.3 Transaction list

Here we can see transactions that have not yet been added to a block.

```
----- List of non-mined transactions -----
Id: bdf71af5-0ae3-11ee-ae5a-c894025ed30c
Szymon -> Bartek: 30.0
Id: c1ccc667-0ae3-11ee-b146-c894025ed30c
Szymon -> Adam: 50.0
Id: ccc5550f-0ae3-11ee-87f7-c894025ed30c
Filip -> Adam: 3.0
Id: d11533f0-0ae3-11ee-a530-c894025ed30c
Filip -> Bartek: 70.0
-----
Return? (yes/no): |
```

Figure 3: Transaction list

## 2.4 Blockchain

List of all the blocks in the blockchain with the transactions that are a part of them.

```
Block -----
Hash: 000000f87c4d071fc26f4d6491fc193d4593aa4aea657fdcd65292987ef0ecd0
Transactions -----

Id: bdf71af5-0ae3-11ee-ae5a-c894025ed30c
Szymon -> Bartek: 30.0
Signature: 2647930d943b1577c5ecdcd241b5cf4627c431f036e0b591a9fac327a6f51ee019e1b31a14d7aba
3caa79b2ba125893e757c86f1193bd20958b5f797e665dc4f934d350b92d3012dbb4897a3cbd142bc9cf3483eb
ddf53373e9fa65c1e38c9d84194a7d6ca673ad9771dcbcb8e9a87d6e1073eef885b29b8db9c85568ace0fc6

Id: clccc667-0ae3-11ee-b146-c894025ed30c
Szymon -> Adam: 50.0
Signature: 9846ddc9c5692918474a716f07c0b4299a4ed42504d3c6390314d8ed4ad77de6f2001fa32b877eb
7c5708b77e7aa2b62717a4b875cc5cdc88756274095d68dbd0be365bad49834e39d77a2411bf631a59f54d484e
0fe17d446a9485cfb112047c04459c14f5079f094b27ce7e4f923f3213f4e79ae9d426f2dec68c78ac5f6a5

Id: ccc5550f-0ae3-11ee-87f7-c894025ed30c
Filip -> Adam: 3.0
Signature: 443261b0f1f510708418d6d391026f0902fab622cb329449bd7b6b1a19caa419b545b0c8e704de3
f25e0ca461886d4750781b38a4dc02740cd469c9c6ed8116884328f6ec2a8871ec5d97eccef7d004fbedalcd2a
f6abd4f58823f52c99d6f0f2716c396a485f6243caacb0eebd333fff070006d22a0e2baeddaefe462eac659

Id: dl1533f0-0ae3-11ee-a530-c894025ed30c
Filip -> Bartek: 70.0
Signature: 80d33109be6ae90b14f3d1fde4dd340abc0e57b560eef35603d14b178bce6b7cca15e68382f602b
0d83390ad65d1689c6e8203a65bf79d64faa67379cf718048fce3840c567d076c8e4d7de9437db179ae9c4b00f
69ebf1b654949e4175d299bdf0e08b4d25fc79368e432bf0c5a201bf2c58d84ff4dd3f6047bea37554f723d

End Transactions -----
Proof of work: 146687
End block -----
Block -----
Hash: 0000001d934e315771282335ef7188b13a1a92f1529cf6a0017db2e730f2b239
Transactions -----

Id: 24c08119-0ae4-11ee-ba2d-c894025ed30c
Filip -> Szymon: 10.0
Signature: 216364690476d02b02fde49d9941c6b019d97bb8135bb96f104e07be187914c2b515dbf56dfa208
751f137192ee84121bdd03fa05da94bcbde79d6470eb8ce1499ade9c847660272225bab8865b2284f662b51c1a
2a1bef3069443f54964e9632bd1d448a0c79d4c25a4daf843e122cc674703a178f1164b07e8d9166b264eed

Id: 2e836a91-0ae4-11ee-ba81-c894025ed30c
Szymon -> Filip: 10.0
Signature: 6eb0bff3f12776ea1880fdc24f868f2f735baeb7271f84d011bed72364ed7e7a2f3671aaec6b6a9
d5784b274beb3d53b378de2dc9960ea6ddeeb4e87053506583b3b5c6e3005d3e2ea4bbe3ce35cf4dfeffabb23c8
d9c4c85cdc15dada98f8e4041f5d19acae50cdec3f88dca5f0041e37e74f7dfb0bba1a8a3b64cf8d77989cb

End Transactions -----
Proof of work: 1491153
End block -----
```

Figure 4: Blockchain

## 3 Test

### 3.1 Testing platform

Processor – **Intel® Core™ i5-9300H 2.40GHz**

RAM memory – **16 GB**

OS – **Windows 10 Home 64 bit**

### 3.2 Results

When it comes to performance on our testing platform, during our tests we didn't find any bugs or errors in implementation, only calculations that required a longer amount of time was the calculation of the proof of work. For proof of work with a length of 6 zeros it took between 10s - 60s to finish the calculations.

When it comes to safety we tested it by applying small changes to the transaction that was detected during transaction verification and the transaction was discarded. Similarly we created a Block with valid transactions in it but with random proof of work and hash altered to look like one with proper proof of work, and again it was detected by our system.

Another important safety related matter is the way of storing passwords. In our application, namely instead of keeping passwords in plain-text in our database we only keep hashes calculated from user passwords which ensures security.