Blockchain technologies and quantum computing

Adam Gabała and Konrad Bankiewicz

June 14, 2023

Abstract

The title of the project was: "Simple Blockchain Implementation". In this project we implemented a basic blockchain structure, in which you can add blocks and transactions, store them, and verify data integrity and safety.

1 Introduction

Blockchain is a distributed ledger with growing lists of records which we call blocks, that are securely linked together via cryptographic hashes. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data. Because of that, the blocks form a chain, with each additional block linking to the ones before it. The primary use of blockchains is as a distributed ledger for cryptocurrencies such as bitcoin or Ethereum.

2 Objectives

The main objective of this project was to implement a basic blockchain structure that can be used to store and verify transactions. The blockchain should:

- be implemented in a programming language of our choosing,
- be able to add new blocks and contain transaction data,
- ensure the integrity and security of the data, by using cryptographic mechanisms,
- include mechanisms such as proof-of-work to ensure validity,
- include a user interface with adding and viewing transactions ,as well as observing the current status of the blockchain,
- include basic security measures to prevent unauthorized access,
- be tested for it's performance and security.

3 Implementation

We decided to implement the project using Python programming language. We choose this language, because Python has a clean and easily readable syntax, which makes it relatively simple to write and understand the code. This characteristic makes it easier to implement the complex logic and data structures required for a blockchain. In production environments languages like C++ or Go are often preferred for their efficiency and speed. However, Python is good for prototyping, learning, and smaller-scale blockchain projects.

Figure 1 presents the definition of creating a new block in our program. In the block are located such things like the index of the block, timestamp of when it was added, transactions, proof-of-work, and a hash of a previous block.

```python
def new_block(self, proof, previous_hash):
    block = {
        'index': len(self.chain) + 1,
        'timestamp': time(),
        'transactions': self.current_transactions,
        'proof': proof,
        'previous_hash': previous_hash or hashlib.sha256(json.dumps(self.chain[-1], sort_keys=True).encode()).hexdigest(),
    }
    self.current_transactions = []
    self.chain.append(block)
    return block
```

Figure 1: Definition of a new block

Figure 2 shows the definition of adding a new transaction. Inside we have a sender, reciever and the amount recieved.

```python
def new_transaction(self, sender, recipient, amount):
    self.current_transactions.append({
        'sender': sender,
        'recipient': recipient,
        'amount': amount,
    })
    return self.chain[-1]['index'] + 1
```

Figure 2: Definition of a new transaction

Figure 3 presents a definition of validity check of the blockchain. The function is checking if the hash of the current block is correct and if the proof-of-work is valid.

```python
def is_valid(self):
    for i in range(1, len(self.chain)):
        current_block = self.chain[i]
        previous_block = hashlib.sha256(json.dumps(self.chain[i - 1], sort_keys=True).encode()).hexdigest()

        # Check if the hash of the current block is correct
        if current_block['previous_hash'] == previous_block:
            return True
        # Check if the proof of work is valid
        if hashlib.sha256(str(current_block['proof']).encode()).hexdigest()[:4] == '0000':
            return True
    return False
```

Figure 3: Definition of block validity check

Figure 4 presents the definition of loading the blockchain and saving it to a file.

```python
def import_data_from_file(self, filename):
    with open(filename, 'r') as file:
        data = json.load(file)
        self.current_transactions = data['current_transactions']
        self.chain = data['chain']

def export_to_file(self, filename):
    data = {
        'current_transactions': self.current_transactions,
        'chain': self.chain
    }
    with open(filename, 'w') as file:
        json.dump(data, file)
```

Figure 4: Definition of importing and exporting data

Figure 5 presents the initiation of the program, as well as parsing the logins and passwords of the accounts. The accounts are stored in a loginData.txt file and the contents of the file are visible on Figure 6. the passwords are encrypted via sha256 hash algorithm to ensure security of the accounts.

```
# Program start
blockchain = Blockchain()

# variables and data structures
loginsAndPasswords = {}

# parse txt file with logins and passwords
with open("loginData.txt", 'r') as credentials:
    oneLine = credentials.readlines()
for i in oneLine:
    if('\n') in i:
        i = i[:-1]
    j = i.split('-')
    loginsAndPasswords[j[0]] = j[1]
```

Figure 5: Initiation of the program and parsing the accounts credentials

```
konrad-60a5d3e4100fe8afa5ee0103739a45711d50d7f3ba7280d8a95b51f5d04aa4b8
adam-e1314da69f589c36ec5a7cac8ae270780c9bc5d57965765ff8ccd81f86473add
admin-d864f8d1540c43f7283a7a6836064103d13e3fc9a8b3e41d585b0fb26c7ab759
test-b0ab895ecc083f8d4ef68cb97911b9b7d31379251c0f4fc92c5843a4a22d8354
```

Figure 6: Login and password storage

Figure 7 shows the implementation of logging on the accounts. If the password or login don't match those in the credentials file the message "Wrong data" is displayed.

```
# infinite loop of program
while(True):
    login = attemptL = input("Login: ")
    attemptP = pwinput.pwinput("Password: ")

    # login attepmts
    try:
        loginsAndPasswords[attemptL]
    except:
        print("Wrong data")
        continue
    if (hashlib.sha256(attemptP.encode("utf-8")).hexdigest() != loginsAndPasswords[attemptL]):
        print("Wrong data")
    else:
        filename = "chain.json"
        blockchain.import_data_from_file(filename)
```

Figure 7: Login implementation

On Figure 8 we can see the implementation of Menu of the program. If the user inputs the wrong number or symbol the message "Wrong data" is displayed.

```
# instructions
while(True):
    print("\n")
    print("1 - add a new block")
    print("2 - add a new transaction")
    print("3 - check validity of blockchain")
    print("4 - see blockchain")
    print("5 - save blockchain")
    print("6 - logout")

    try:
        userInteraction = int(input())
    except:
        print("Wrong data")
        continue
```

Figure 8: Menu implementation

Figure 9 presents the implementation of adding a new block to the chain. First we generate a proof-of-work and then we hash the data of the last block of the blockchain.

```
# new block
if(userInteraction == 1):
    proof = random.randint(10000, 100000)
    last_block_out = hashlib.sha256(json.dumps(blockchain.chain[-1], sort_keys=True).encode()).hexdigest()
    previous_hash = last_block_out
    blockchain.new_block(proof, previous_hash)
    print("\n")
    print("New block added!")
```

Figure 9: Implementation of adding a new block

Figure 10 presents the implementation of adding a new transaction to the blockchain. the function consists of the information about the sender and data to input: reciever and amount of the transaction.

```
# new transaction
elif(userInteraction == 2):
    print("Sender: {}".format(login))
    receiver = input("Type receiver: ")
    amount = input("How much transfer: ")
    blockchain.new_transaction(login, receiver, amount)
    print("\n")
    print("Completed!")
```

Figure 10: Implementation of adding a new transaction

Figure 11 shows the implementation of checking the validity the blockchain. The fuction "is-valid" is called from the definition seen in Figure 3.

```
# validity
elif(userInteraction == 3):
    print("\n")
    print("Validity", blockchain.is_valid())
```

Figure 11: Implementation of checking the validity of the blockchain

Figure 12 presents the implementation of a function to view the status of the blockchain. The data is divided to chunks that are easy to read in contrast to the original file.

```
# show in terminal
elif(userInteraction == 4):
    for block in blockchain.chain:
        print(json.dumps(block, indent=2))
        print('-' * 50)
```

Figure 12: Implementation of viewing the current status of blockchain

Figure 13 shows the implementation of saving the blockchain to a .json file. The contents of this file is seen on Figure 14

```
# save to file
elif(userInteraction == 5):
    filename = "chain.json"
    blockchain.export_to_file(filename)

    print("Blockchain exported to file:", filename)
```

Figure 13: Implementation of saving the blockchain

{"current_transactions": [{"sender": "konrad", "recipient": "dfshjdfsh", "amount": "100"}], "chain": [{"index": 1, "timestamp": 1685701876.173428, "transactions": [], "proof": 100, "previous_hash": "1"}, {"index": 2, "timestamp": 1685702755.24381, "transactions": [{"sender": "Konrad", "recipient": "Adam", "amount": "4"}, {"sender": "Adam", "recipient": "Konrad", "amount": "44"}], "proof": 32233, "previous_hash": "a810b443c1dc29ff42793f73fb72d66572d9c0d1dca37c8dab4dcd7fb8bb6b1a"}, {"index": 3, "timestamp": 1685702855.100139, "transactions": [{"sender": "loremimpsum", "recipient": "uuus", "amount": "6969"}], "proof": 61182, "previous_hash": "266ac25e6161f42b6a42578498070696f4e893142333fd8631061f7b4a66d065"}, {"index": 4, "timestamp": 1685703893.443834, "transactions": [{"sender": "tigerbonzo", "recipient": "hacker", "amount": "444"}], "proof": 33104, "previous_hash": "42d0e554f585a18a143954f47950e4b08e3f1bd8644397ecab438da475d2c5b2"}, {"index": 5, "timestamp": 1685979537.0732422, "transactions": [{"sender": "Adam", "recipient": "Konrad", "amount": "69999"}], "proof": 98314, "previous_hash": "0423d7af10211b764f5405f97766561ef204b4f334e33196512f3ff2b272543b"}, {"index": 6, "timestamp": 1685980013.9062731, "transactions": [{"sender": "Admin", "recipient": "test", "amount": "55"}], "proof": 77203, "previous_hash": "52903be8a7fca0ba3fbc9eac4e52a0788823ffd713a0890b0e3dd6ca2945b957"}, {"index": 7, "timestamp": 1685980071.2030132, "transactions": [{"sender": "admin", "recipient": "loremipsum", "amount": "33"}], "proof": 53837, "previous_hash": "e9278eab07edf584e8921e77aa1e0c374138768b3e0c6b231ae3070d94c5834e"}, {"index": 8, "timestamp": 1685983075.4555485, "transactions": [], "proof": 89162, "previous_hash": "a15370bfc6a59cad2480d9fe0614d372dea9ff5c98c417314e48baa6ad448cf8"}, {"index": 9, "timestamp": 1685983500.9649925, "transactions": [{"sender": "admin", "recipient": "dupa", "amount": "2000"}, {"sender": "admin", "recipient": "konrad", "amount": "dupa"}], "proof": 32787, "previous_hash": "c7d261069b08171dd8c8fcc38de472644f76699398ff287c5c70ca17b27224f3"}, {"index": 10, "timestamp": 1685992655.043156, "transactions": [], "proof": 61285, "previous_hash": "8dd4424a5eee7363454cbddf111ba05beab2f345b6d6c100ce709a39f470e31d"}, {"index": 11, "timestamp": 1685992664.853444, "transactions": [{"sender": "konrad", "recipient": "adam", "amount": "55"}], "proof": 70902, "previous_hash": "d08ce70d3434a2647534274b6a1bf8c7154d6a7e72a76bdc623fa2dbd11057a4"}, {"index": 12, "timestamp": 1685994751.6178172, "transactions": [], "proof": 76081, "previous_hash": "1ff2e29f16a53c7c7190e7adcf3cf2862fe987ded472a1914ed960c2eb44d89e"}, {"index": 13, "timestamp": 1685994922.9097128, "transactions": [{"sender": "adam", "recipient": "konrad", "amount": "30"}], "proof": 24688, "previous_hash": "d1f5f6a1fef443dd693b6561e46257a6eb348417e28a080af799638007334758"}, {"index": 14, "timestamp": 1685994953.720696, "transactions": [{"sender": "konrad", "recipient": "adam", "amount": "333"}], "proof": 72975, "previous_hash": "dd16b6de734cf4dad7ef7d771fedce0b61646919d4bba79fbaf401b674b0a068"}, {"index": 15, "timestamp": 1686739094.395823, "transactions": [], "proof": 77678, "previous_hash": "cddb1c07f7cae8825926cb02194646519a53f80e6d3dleee4f4d548a6bd7f9b1"}, {"index": 16, "timestamp": 1686748243.89307, "transactions": [{"sender": "konrad", "recipient": "dupa", "amount": "100"}], "proof": 34278, "previous_hash": "1b1ae703164798999fd5e25c77508a005f4baee45c2ac421a9ee3057949a28fd9"}, {"index": 17, "timestamp": 1686748554.798065, "transactions": [{"sender": "konrad", "recipient": "ja", "amount": "44"}], "proof": 87902, "previous_hash": "a070c744a8d876240ac41f53bfec6a95d7acaa439fb402f4cddcfba01ee0d3da"}, {"index": 18, "timestamp": 1686748621.238544, "transactions": [{"sender": "konrad", "recipient": "2", "amount": "3"}], "proof": 38136, "previous_hash": "fd8bfb0b433cf0545c33c52f18bc88614dd2a535b076dfd3b3c81cec5101009f"}, {"index": 19, "timestamp": 1686749474.1590421, "transactions": [{"sender": "konrad", "recipient": "2", "amount": "3"}], "proof": 48783, "previous_hash": "2d2f7dd155lb35ce324f8ea4656ae66a69752d98bd8b4d80b329469758889332"}, {"index": 20, "timestamp": 1686749636.495035, "transactions": [{"sender": "konrad", "recipient": "sa", "amount": "3"}], "proof": 37648, "previous_hash": "fe6fbbe64ddeldblcc61e89942b6ca34d432204fe9f86a04078ea6bc46352611"}, {"index": 21, "timestamp": 1686750689.042583, "transactions": [{"sender": "konrad", "recipient": "3", "amount": "4"}], "proof": 59666, "previous_hash": "a65c7c4a432069e2298bb05bc2a36606606c2e4e69127fdee86ab49089d02c67"}, {"index": 22, "timestamp": 1686753204.6929064, "transactions": [{"sender": "konrad", "recipient": "2", "amount": "3"}, {"sender": "konrad", "recipient": "4", "amount": "5"}, {"sender": "konrad", "recipient": "2", "amount": "2"}], "proof": 17696, "previous_hash": "f38d5b53eb0b56026e036c200f2fb9ddb6d7d8b339620d6463fabf97f374aafe"}, {"index": 23, "timestamp": 1686753243.5339515, "transactions": [{"sender": "konrad", "recipient": "uil", "amount": "100"}], "proof": 32276, "previous_hash": "d2e0f2e636b20f78b190b4b72d43a6c65eda863c26e505f2748fae2292c481d7"}, {"index": 24, "timestamp": 1686758514.1049058, "transactions": [{"sender": "konrad", "recipient": "daa", "amount": "12"}], "proof": 43393, "previous_hash": "fd247814eaa202dc69012fcab803f7e8bb3793cc2b68c5e1d92ab93df3208c5b"}]}

Figure 14: Blockchain status in file

Figure 15 shows the implementation of logging of the platform which is simply done by leaving the loop of the menu.

```
# logout
elif(userInteraction == 6):
    break
```

Figure 15: Logging out

4 Tests

To test the program we first created a couple of accounts and checked if the login process works correctly. Using the library pwinput we can make that when writing password in the console it shows us stars to conceal our input. When login is successfull, the menu interface shows up as seen on Figure 16.

Figure 16: View of logging and main menu

Then we made a bunch of blocks with transactions. By selecting the number 2 in the menu we are shown our account name in the sender category. We then must write the reciever and the amount of the transaction. We are then notified of the transaction coming through by a message seen on Figure 17.

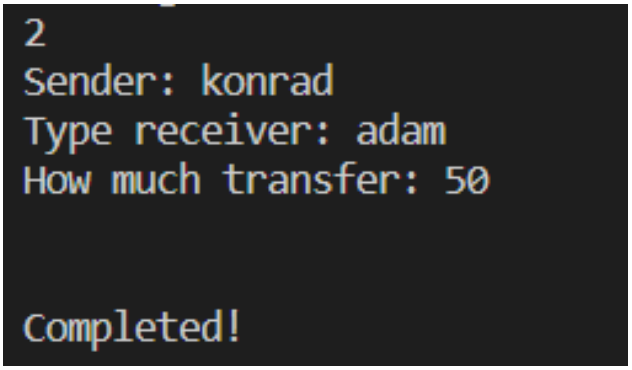

Figure 17: View of adding new transaction

After that we saved the blockchain to file as seen on Figure 18, and when we checked the validity of the blockchain it was correct.

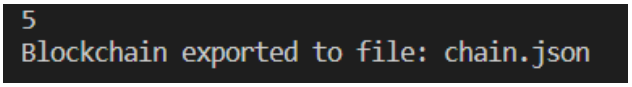

Figure 18: View of saving the blockchain

Lastly we checked the status of the whole blockchain. Part of it is presented on Figure 19 to show the format in which it is displayed.

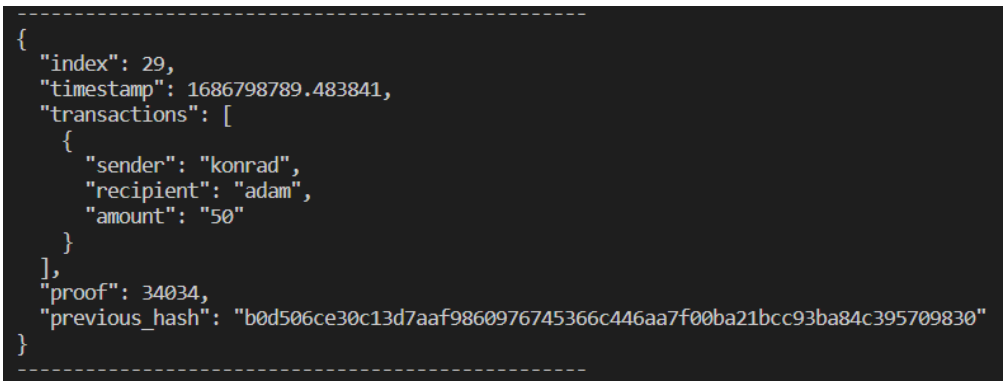

Figure 19: View of a part of blockchain status in program

5 Conclusions

In conclusion, we have successfully completed the project, achieving a functional blockchain structure. The project of creating a blockchain has been a challenge and provided a hands-on experience in blockchain implementation.