Simple Blockchain Implementation

Jan Chlebek Szymon Krysztopolski

June 2023

The objective of this project is to implement a basic blockchain structure that can be used to store and verify transactions. The blockchain should include the necessary components such as blocks, transactions, and cryptographic mechanisms to ensure data integrity and security.

1 Selected development tools

Our blockchain was implemented using Python 3.10. As a base of cryptography in the project we decided to use cryptography and hashlib library combination

2 Overall algorithm implementation and adpoted simplifications

When planning the implementation, we decided not to proceed with any distributed form of computing. Our blockchain is stored as nodes on centralised single-process instance of server orchestrated by System user.

1. Blockchain is a one-way list. The entire run is controlled by the System (less problem with parallelism).

2. Users are known in advance and have their asymmetric key pair, login and password (stored as hash). There is no mechanism for adding them (I will focus on blockchain). They are hardcoded in the appropriate dictionary initialized at program startup.

3. The blocks are stored in text files and created by System user. The file has the following characteristics:

    - the file name is a hash (we assume unique)
    - each block / block file has a JSON with the following content:
        - subtitle TITLE encrypted with the author's private key
        - transactions added by the author of the block on behalf of the user who generated the transactions
        - hash of the previous block

- The first block is named INIT and is generated by the System user.
- The last block in the chain has no value indicating the previous block. When a block is added, this information is completed.
- When blockchain system is restarted, all existing blocks signatures are checked for validity to ensure the system has not been compromised.

4. Each user can add a new set of new transactions. Request accepts the System. It then sends a TASK to all users. The first one to execute it gets the opportunity to add a block.

5. We are working based on Proof-of-Work TASK which requires the generation of a hash starting with zero

3 Code snippets

3.1 Block file structure

filename: SHA256 hash

```
{
    "block_name": "SHA256 hash",
    "timestamp": "Encrypted message using RSA with OAEP padding",
    "transactions": [
        "List of transactions: USER: ------ Object",
        "AliceWonder: ------ Haste makes waste."
    ],
    "signer": "Username of signer",
    "signature": "RSA signature with PSS padding",
    "prev_block": "SHA256 hash of previous block"
}
```

3.2 Proof-of-work

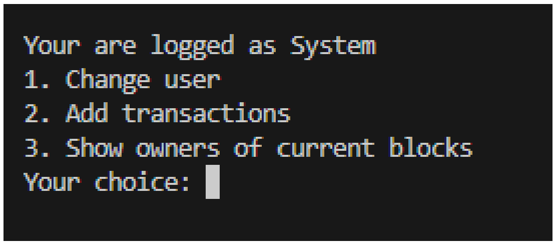
```
    def manager.is_winner(hash: str) -> bool:
        return hash.startswith("0")

    active_challange = True
    while active_challange:
        for i in challenge_order:
            user = User_list[i]
            if manager.is_winner(user.get_base64_hash()):
                manager.add_block_to_blockchain(user, new_transactions)
                active_challange = False
```

```
                         break
```

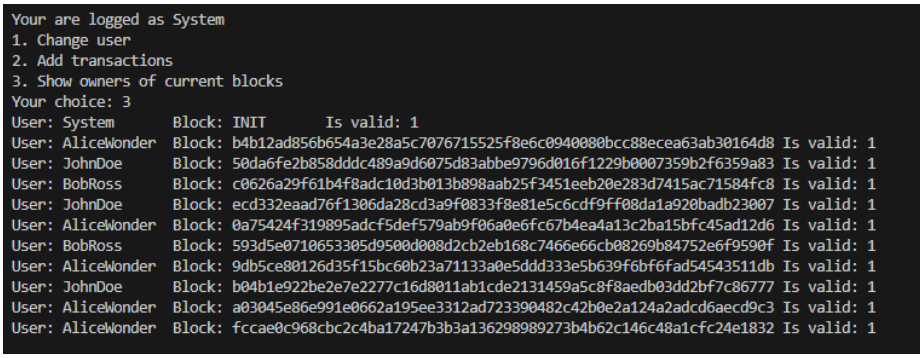
4 How to use the program

Our blockchain program logs into System user account when started. The startup panel has several options available.



```
Your are logged as System
1. Change user
2. Add transactions
3. Show owners of current blocks
Your choice:
```

- 1. User's account change - this option allows to change current account after provide proper credentials (e.g. JohnDoe, qwerty123).
- 2. Add transaction. A logged-in user can add new transactions. In current version of the program, this option allows you to specify how many predefined transactions are to be added to the blockchain. The maximum value is 5. Transactions are randomly selected from specific sample table.
- 3. Displays the current blocks with owners. In this option, the owner is the account that added block to the blockchain.



```
Your are logged as System
1. Change user
2. Add transactions
3. Show owners of current blocks
Your choice: 3
User: System       Block: INIT       Is valid: 1
User: AliceWonder  Block: b4b12ad856b654a3e28a5c7076715525f8e6c0940080bcc88ecea63ab30164d8 Is valid: 1
User: JohnDoe      Block: 50da6fe2b858dddc489a9d6075d83abbe9796d016f1229b0007359b2f6359a83 Is valid: 1
User: BobRoss      Block: c0626a29f61b4f8adc10d3b013b898aab25f3451eeb20e283d7415ac71584fc8 Is valid: 1
User: JohnDoe      Block: ecd332eaad76f1306da28cd3a9f0833f8e81e5c6cdf9ff08da1a920badb23007 Is valid: 1
User: AliceWonder  Block: 0a75424f319895adcf5def579ab9f06a0e6fc67b4ea4a13c2ba15bfc45ad12d6 Is valid: 1
User: BobRoss      Block: 593d5e0710653305d9500d008d2cb2eb168c7466e66cb08269b84752e6f9590f Is valid: 1
User: AliceWonder  Block: 9db5ce80126d35f15bc60b23a71133a0e5ddd333e5b639f6bf6fad54543511db Is valid: 1
User: JohnDoe      Block: b04b1e922be2e7e2277c16d8011ab1cde2131459a5c8f8aedb03dd2bf7c86777 Is valid: 1
User: AliceWonder  Block: a03045e86e991e0662a195ee3312ad723390482c42b0e2a124a2adcd6aecd9c3 Is valid: 1
User: AliceWonder  Block: fccae0c968cbc2c4ba17247b3b3a136298989273b4b62c146c48a1cfc24e1832 Is valid: 1
```

- e. Exits program.

If the program is run with **--init** as a parameter, the program will remove all blocks.

If the program is run with **--test xxx** as a parameter (where xxx is a number), the program will remove all blocks and create xxx number of 5 transaction blocks as a benchmark.