PUT blockchain implementation report

Tom PICAUD

June 5, 2023

1 Introduction

The goal of this report is to explain the implementation by providing the design decisions and details about the technical aspects. We will dive into the architecture of the project and how it works, as well as the functionalities implemented and what can be done with the blockchain. We will also explore the verification process for both transactions and blocks.

2 Functionalities

2.1 Blockchain functionalities

- Create and verify blocks via Proof of Work
- Verify and store transactions
- Receive transactions in a mempool
- Handle a few HTTP requests (POST transaction, GET balance)
- Provide an interface to view its state
- Address security aspects such as double spending and DDoS resistance

2.2 Wallet functionalities

- Create a new key pair
- Get the balance of an address
- Send coins to another address

3 Architecture and Technical Decisions

3.1 Language

We chose Python as the development language for this project. Python is a well-established multi-paradigm language with good documentation and libraries. Additionally, its simple syntax was attractive in order to primarily focus on the blockchain aspect without spending too much time learning a new language.

3.2 Architecture

The project is divided into two applications: the blockchain and a wallet. The goal is to separate the responsibilities of storing transactions and sending transactions via a key pair. The wallet is responsible for interacting with the blockchain, while the blockchain is only responsible for verifying and storing transactions. Here is a brief diagram:

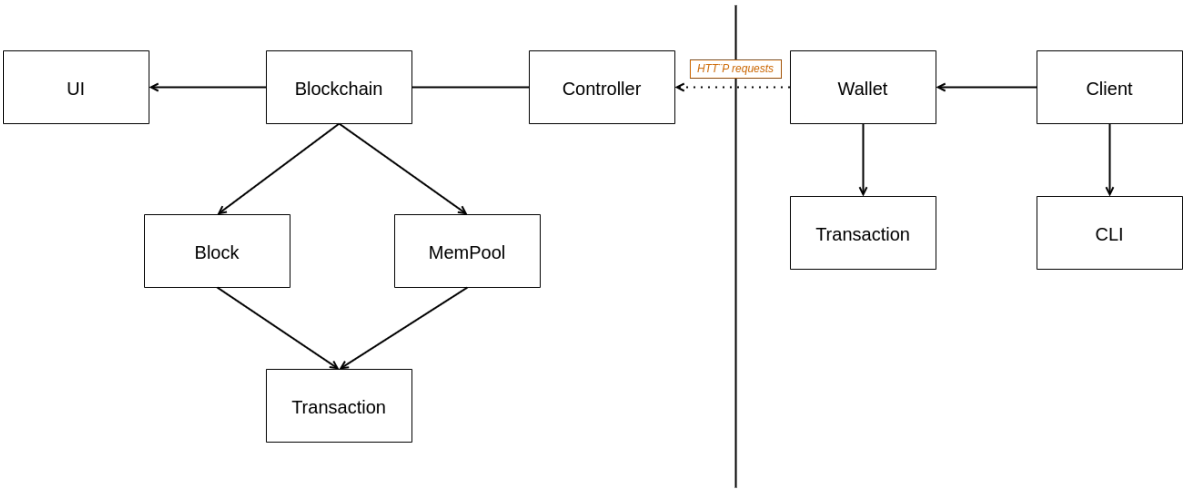

Figure 1: Blockchain and wallet diagram

3.3 Entities

There are two main entities in the application:

- **Block**: Composed of an index, a timestamp, a transaction list, a previous block hash, and the hash of the current block. It is the entity that stores transactions in the blockchain. The hash is calculated by mining it with a given difficulty (represented by the number of leading zeros in the hash). A nonce is used to find the solution.
- **Transaction**: Composed of a sender's public key, a recipient's public key, an amount, and a signature. It allows storing transaction data and facilitates the verification process, as the signature and data are in the same entity.

4 Process

4.1 Blockchain Process

When the blockchain is run, it starts a cycle:

1. Get transactions from the mempool.
2. Verify transactions.
3. Create a new block with valid transactions.
4. Mine the block and add it to the blockchain.
5. Remove transactions from the mempool.

This cycle is repeated until the blockchain is stopped, ensuring the integrity and liveliness of the data.

4.2 Signing and Verification Process

A transaction is composed of the sender's public key, the recipient's public key, the amount, and the signature. Transactions are signed with the Ed25519 algorithm. Note that public keys and the signature are converted to hexadecimal for better readability.

When a transaction is received by the blockchain, it is first parsed to create an instance of a Transaction. Then, it is added to the mempool, which collects pending transactions. Next, every transaction in the mempool is verified by matching the signature with the data inside, including the sender's public key. If valid, the transactions are added to the next block, which is then mined. Once the block is mined, the transactions are added to the blockchain and validated, and they are removed from the mempool, including any invalid transactions to avoid congestion.

4.3 Communication between Wallet and Blockchain

The blockchain provides endpoints to handle HTTP requests. It can receive transactions via a POST method and return the current balance of a given address using a GET method. The wallet is designed to ease the communication process and create transactions. The blockchain only accepts transactions in the following format in the request body: (sender, recipient, amount, signature). Otherwise, the transaction will be rejected.

5 Conclusion

This simple blockchain implementation allows for lightweight coin transfers between addresses. Data integrity and correctness are ensured by cryptographic signatures and hash functions. For a more comprehensive implementation, it would be useful to add decentralized features such as synchronization and mining between multiple nodes.