MIT World Peace University

Blockchain Technology
Fourth Year B. Tech, Semester 8

Implementing Blockchain in Python

Lab Assignment 8

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 15

April 3, 2025

Contents

1 Objective

The objective of this assignment is to understand the fundamentals of blockchain technology by implementing a simple blockchain using Python. This includes:

- Understanding the structure of a blockchain.
- Implementing blocks that store transactions.
- Using hashing and Proof of Work (PoW) to secure the blockchain.
- Validating the blockchain to ensure data integrity.
- Simulating basic transactions between users.

2 Theory

2.1 What is a Blockchain?

A blockchain is a decentralized digital ledger that records transactions in a series of blocks. Each block contains transaction data, a timestamp, a proof of work, and a reference to the previous block via a cryptographic hash. This structure ensures the immutability and security of data.

2.2 Key Components of a Blockchain

- **Block:** A unit of data storage that holds transactions.
- **Hash:** A unique identifier generated using cryptographic algorithms to secure the block.
- **Proof of Work (PoW):** A computational puzzle that miners solve to validate and add new blocks to the chain.
- **Transactions:** Records of value transfers between users.
- **Chain:** A linked sequence of blocks, where each block references the previous one.

2.3 Working of a Blockchain

1. A user initiates a transaction.
2. The transaction is broadcasted to a network of computers (nodes).
3. Miners validate the transaction using Proof of Work.
4. A new block is created and added to the blockchain.
5. The transaction is now recorded permanently.

2.4 Implementation in Python

In this assignment, we implement a simple blockchain in Python with the following features:

- A class-based structure for the blockchain.
- Functions for creating new blocks and validating the chain.
- A Proof of Work algorithm to secure new blocks.
- Transaction handling for simulating simple value transfers.

2.5 Blockchain Security

Blockchain ensures data integrity and security through:

- **Hashing:** Prevents data tampering by linking blocks using cryptographic hashes.
- **Decentralization:** No single point of failure since copies exist on multiple nodes.
- **Consensus Mechanism:** Proof of Work ensures only valid blocks are added.

3 Code

```
import hashlib
import json
import time

class Blockchain:
    def __init__(self):
        self.chain = []
        self.pending_transactions = []

        # Create the genesis block
        self.create_block(proof=1, previous_hash='0')

    def create_block(self, proof, previous_hash):
        """Creates a new block and adds it to the blockchain."""
        block = {
            'index': len(self.chain) + 1,
            'timestamp': time.time(),
            'transactions': self.pending_transactions,
            'proof': proof,
            'previous_hash': previous_hash
        }
        self.pending_transactions = []  # Reset the list of pending transactions
        self.chain.append(block)
        return block

    def get_previous_block(self):
        """Returns the last block in the blockchain."""
        return self.chain[-1]

    def proof_of_work(self, previous_proof):
        """Implements a simple Proof of Work algorithm."""
        new_proof = 1
```

```python
        check_proof = False
        while not check_proof:
            hash_operation = hashlib.sha256(str(new_proof**2 - previous_proof**2).encode()).hexdigest()
            if hash_operation[:4] == '0000':  # Condition for valid proof
                check_proof = True
            else:
                new_proof += 1
        return new_proof

    def hash(self, block):
        """Creates a SHA-256 hash of a block."""
        encoded_block = json.dumps(block, sort_keys=True).encode()
        return hashlib.sha256(encoded_block).hexdigest()

    def add_transaction(self, sender, receiver, amount):
        """Adds a new transaction to the list of pending transactions."""
        self.pending_transactions.append({
            'sender': sender,
            'receiver': receiver,
            'amount': amount
        })
        return self.get_previous_block()['index'] + 1

    def is_chain_valid(self, chain):
        """Checks if the blockchain is valid."""
        previous_block = chain[0]
        index = 1
        while index < len(chain):
            block = chain[index]

            # Check if previous_hash matches actual hash of previous block
            if block['previous_hash'] != self.hash(previous_block):
                return False

            # Check if Proof of Work is valid
            previous_proof = previous_block['proof']
            proof = block['proof']
            hash_operation = hashlib.sha256(str(proof**2 - previous_proof**2).encode()).hexdigest()
            if hash_operation[:4] != '0000':
                return False

            previous_block = block
            index += 1
        return True

# Test the blockchain
if __name__ == "__main__":
    blockchain = Blockchain()

    # Add transactions and mine a new block
    blockchain.add_transaction(sender="Alice", receiver="Bob", amount=10)
    previous_block = blockchain.get_previous_block()
    previous_proof = previous_block['proof']
    proof = blockchain.proof_of_work(previous_proof)
    previous_hash = blockchain.hash(previous_block)
    blockchain.create_block(proof, previous_hash)

```

```python
        # Print the blockchain
        print(json.dumps(blockchain.chain, indent=4))
```

4 Output


```
[
    {
        "index": 1,
        "timestamp": 1712168400.123456,
        "transactions": [],
        "proof": 1,
        "previous_hash": "0"
    },
    {
        "index": 2,
        "timestamp": 1712168425.678901,
        "transactions": [
            {
                "sender": "Alice",
                "receiver": "Bob",
                "amount": 10
            }
        ],
        "proof": 53992,
        "previous_hash": "5d6c5e1e6e2..."
    }
]
```


5 FAQs

1. **What is a blockchain?** A blockchain is a decentralized, distributed ledger that records transactions across multiple computers in a way that ensures security, transparency, and immutability.

2. **What is the purpose of the genesis block?** The genesis block is the first block in a blockchain. It serves as the foundation for all subsequent blocks and does not have a previous hash.

3. **How does Proof of Work (PoW) function in this blockchain?** Proof of Work (PoW) requires miners to find a valid proof by solving a computational puzzle. In this implementation, a valid proof is a number that, when used in a hash operation, produces a hash with a specific number of leading zeros.

4. **How are transactions added to a block?** Transactions are first stored in a pending transactions list. When a new block is mined, these transactions are included in the block, and the list is cleared.

5. **How is the blockchain verified?** The blockchain is verified by checking that each block's previous hash matches the hash of the preceding block and that the Proof of Work conditions are met.

6 Glossary

- **Blockchain:** A chain of blocks containing transaction data that ensures security and transparency.
- **Genesis Block:** The first block in the blockchain.
- **Hash:** A cryptographic function that generates a unique fixed-length output for input data.
- **Proof of Work (PoW):** A consensus algorithm that requires solving a computational puzzle to add new blocks.
- **Mining:** The process of finding a valid Proof of Work to add a new block to the blockchain.
- **Transaction:** A record of value transfer between two parties.
- **Ledger:** A record-keeping system that maintains all transactions in the blockchain.

References

[1] Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. Retrieved from https://bitcoin.org/bitcoin.pdf

[2] Solidity Documentation. Available at: https://soliditylang.org/docs/

[3] Ethereum Developer Documentation. Available at: https://ethereum.org/en/developers/

[4] Menezes, A. J., Vanstone, S. A., and Oorschot, P. C. (1996). Handbook of Applied Cryptography. CRC Press.

[5] Rivest, R. L. (1992). The MD5 Message-Digest Algorithm. RFC 1321. Retrieved from https://tools.ietf.org/html/rfc1321