Blockchain Exp - 2

AIM: Create a Blockchain using Python.

Theory:

1. What is a Blockchain?

A **Blockchain** is a decentralized, distributed, and immutable digital ledger used to record transactions across multiple computers in a secure and transparent manner. Instead of relying on a central authority, blockchain operates on a **peer-to-peer network**, where each participant (node) maintains a copy of the ledger.

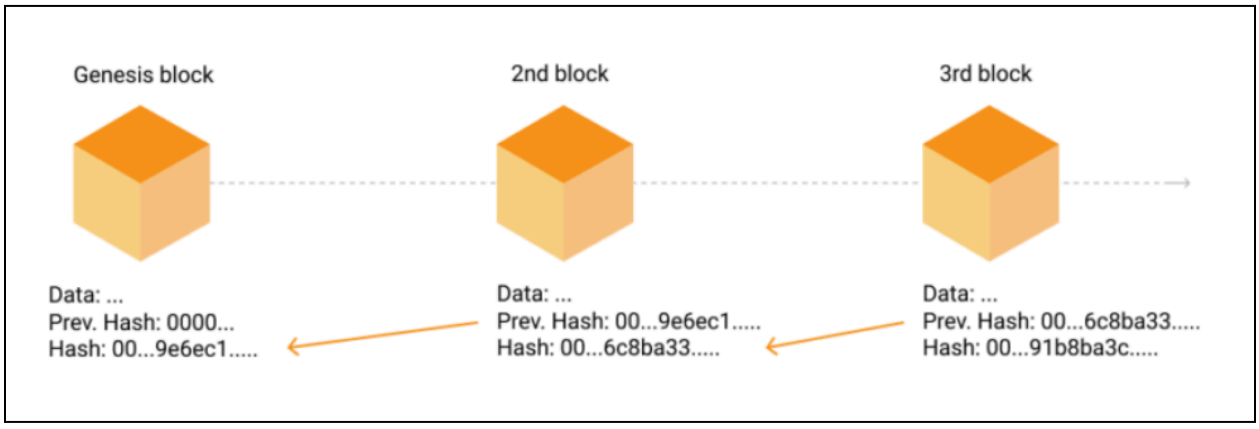
A blockchain consists of a sequence of **blocks**, where each block contains:

- A list of transactions or data
- A timestamp indicating when the block was created
- A cryptographic hash of the previous block
- A proof value generated using a consensus mechanism

Blocks are linked together using cryptographic hashes, forming a chain. Any modification to a block changes its hash, which breaks the chain, making blockchain **tamper-resistant** and **immutable**.

Key characteristics of blockchain include:

- **Decentralization**: No single entity controls the data
- **Immutability**: Once data is recorded, it cannot be altered
- **Transparency**: Transactions can be publicly verified
- **Security**: Cryptographic techniques ensure data integrity



2. Process of Mining

Mining is the process through which new blocks are added to a blockchain in a secure and decentralized manner. It is carried out using a consensus mechanism known as **Proof of Work (PoW)**, which ensures that all participating nodes agree on the state of the blockchain.

The mining process begins when a miner retrieves the **latest block** from the blockchain. This block contains a proof value and a hash of the previous block. Using this information, the miner attempts to compute a new proof value (nonce) that satisfies a predefined difficulty condition.

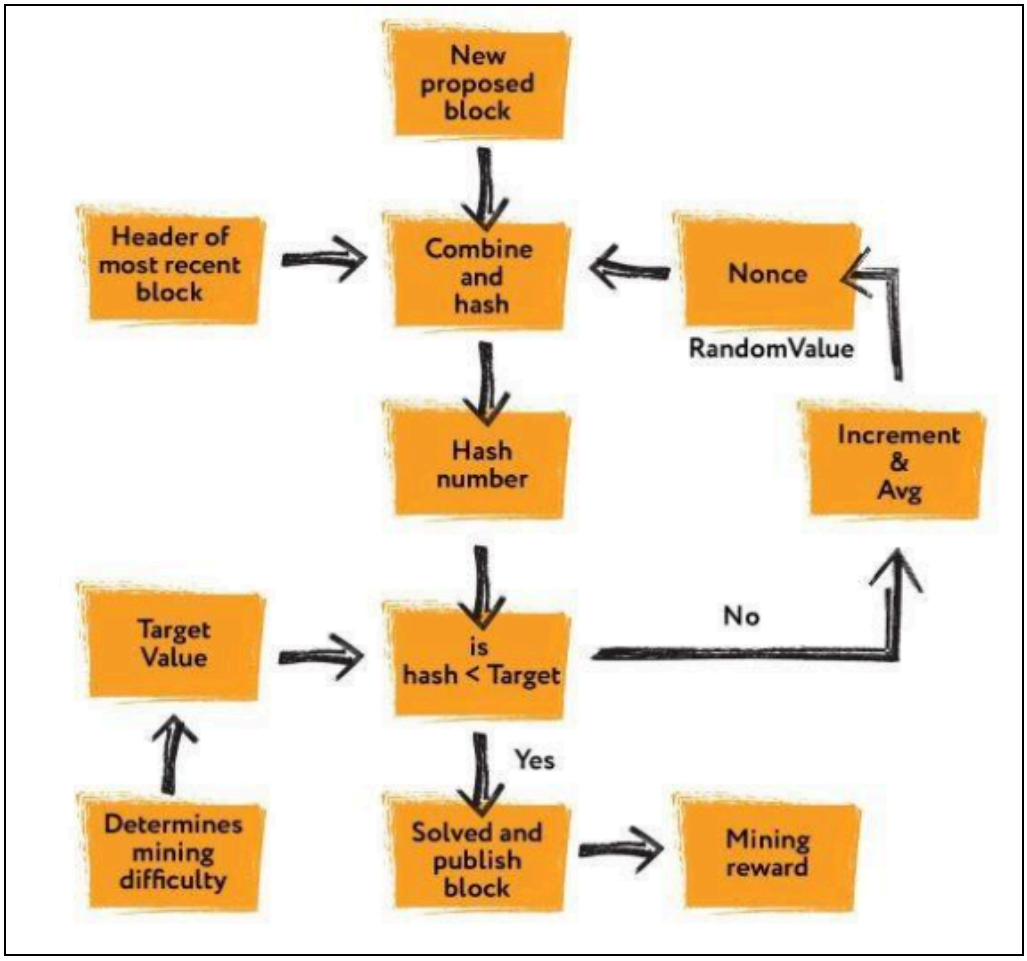
The miner repeatedly performs the following steps:

1. A candidate proof value (nonce) is selected.
2. A mathematical expression involving the new proof and the previous proof is computed.
3. The result is passed through a cryptographic hash function such as **SHA-256**.
4. The generated hash is checked against the difficulty constraint (for example, the hash must start with a certain number of leading zeros).
5. If the condition is not satisfied, the nonce is incremented and the process is repeated.

This iterative process is computationally intensive and requires significant processing power, making it difficult for malicious users to manipulate the blockchain. Once a valid proof is found, the miner creates a new block containing:

- Block index
- Timestamp
- Proof value
- Hash of the previous block

The newly mined block is then added to the blockchain and shared with other nodes for verification. Mining ensures **network security**, **data integrity**, and **consensus**, while also preventing attacks such as double spending.



3. How to Check the Validity of Blocks in a Blockchain

Blockchain validation is the process of verifying that all blocks in the chain are authentic and that the data has not been altered. Validation is crucial for maintaining trust in a decentralized system.

The validity of a blockchain is checked using the following mechanisms:

1. Previous Hash Verification

Each block stores the cryptographic hash of the previous block. During validation, the hash of the previous block is recalculated and compared with the stored `previous_hash` value. If both values match, the linkage between blocks is confirmed. Any mismatch indicates data tampering.

2. Proof of Work Verification

For every block, the proof value is verified by recomputing the hash operation used during mining. The blockchain is considered valid only if the recomputed hash satisfies the difficulty condition (such as leading zeros). This ensures that each block was mined legitimately.

3. Sequential Integrity Check

Blocks must appear in the correct order, starting from the genesis block. Each block must reference only the immediately preceding block, ensuring continuity of the chain.

4. Genesis Block Validation

The first block of the blockchain, known as the genesis block, is checked separately. It must have a fixed previous hash (usually “0”) and a predefined proof value.

If all these conditions are satisfied for every block in the chain, the blockchain is considered **valid and trustworthy**. If any single check fails, the blockchain is declared invalid, indicating possible corruption or unauthorized modification.

Program & Output:

1. **Installing flask**

```
prajj@_pep MINGW64 ~/Desktop/College/sem-8/Blockchain Lab (main)
$ pip install flask==2.2.5
Collecting flask==2.2.5
  Downloading Flask-2.2.5-py3-none-any.whl.metadata (3.9 kB)
Collecting Werkzeug>=2.2.2 (from flask==2.2.5)
  Downloading werkzeug-3.1.5-py3-none-any.whl.metadata (4.0 kB)
Requirement already satisfied: Jinja2>=3.0 in c:\python313\lib\site-packages (from flask==2.2.5) (3.1.6)
Collecting itsdangerous>=2.0 (from flask==2.2.5)
  Using cached itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Collecting click>=8.0 (from flask==2.2.5)
  Downloading click-8.3.1-py3-none-any.whl.metadata (2.6 kB)
Requirement already satisfied: colorama in c:\python313\lib\site-packages (from click>=8.0->flask==2.2.5) (0.4.6)
Requirement already satisfied: MarkupSafe>=2.0 in c:\python313\lib\site-packages (from Jinja2>=3.0->flask==2.2.5) (3.0.3)
Downloading Flask-2.2.5-py3-none-any.whl (101 kB)
Downloading click-8.3.1-py3-none-any.whl (108 kB)
Using cached itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Downloading werkzeug-3.1.5-py3-none-any.whl (225 kB)
Installing collected packages: Werkzeug, itsdangerous, click, flask
Successfully installed Werkzeug-3.1.5 click-8.3.1 flask-2.2.5 itsdangerous-2.2.0
```

2. Running the script

```
prajj@_pep MINGW64 ~/Desktop/College/sem-8/Blockchain Lab (main)
$ python blockchain.py
 * Serving Flask app 'blockchain'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.31.144:5000
Press CTRL+C to quit
127.0.0.1 - - [30/Jan/2026 22:07:50] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [30/Jan/2026 22:08:04] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [30/Jan/2026 22:08:41] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [30/Jan/2026 22:10:39] "GET /mine_chain HTTP/1.1" 404 -
127.0.0.1 - - [30/Jan/2026 22:10:50] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [30/Jan/2026 22:10:55] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [30/Jan/2026 22:11:06] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [30/Jan/2026 22:12:30] "GET /is_valid HTTP/1.1" 200 -
```

3. The python script

```
# ------------------------------------------------------------

# Required installation:(run cmd as administrator)
# pip install flask==2.2.5

# Importing libraries
import datetime      # To generate timestamps for blocks
import hashlib       # To generate SHA256 hashes
import json          # To convert block data into JSON
from flask import Flask, jsonify    # To create API endpoints

# ------------------------------------------------------------
# Part 1 - Building the Blockchain
# ------------------------------------------------------------

class Blockchain:

    def __init__(self):
        # This list will store the entire chain of blocks
        self.chain = []

        # Create the genesis block (first block)
        # proof = 1 → arbitrary
        # previous_hash = '0' → since no previous block exists
        self.create_block(proof=1, previous_hash='0')
```

```
    def create_block(self, proof, previous_hash):
        """
        Creates a new block and adds it to the chain.
        Each block contains:
        - index
        - timestamp
        - proof (PoW output)
        - previous block hash
        """
        block = {
            'index': len(self.chain) + 1,              # Position of block in chain
            'timestamp': str(datetime.datetime.now()), # Current timestamp
            'proof': proof,                            # PoW result
            'previous_hash': previous_hash             # Hash of the previous block
        }

        # Add block to the chain
        self.chain.append(block)
        return block

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

    def proof_of_work(self, previous_proof):
        """
        Simple Proof of Work (PoW) algorithm:
        - Find a number (new_proof)
        - Such that the SHA256 hash of (new_proof^2 - previous_proof^2)
          starts with '0000'

        This is a computational puzzle to secure the network.
        """
        new_proof = 1
        check_proof = False

        while check_proof is False:
            # Cryptographic puzzle: hash difference of squares
            hash_operation = hashlib.sha256(
                str(new_proof**2 - previous_proof**2).encode()
            ).hexdigest()
```

```
            # Check if hash begins with 4 leading zeros
            if hash_operation[:4] == '0000':
                check_proof = True
            else:
                new_proof += 1

        return new_proof

    def hash(self, block):
        """
        Creates a SHA256 hash of a block.
        - Sort keys to ensure consistent hashing
        """
        encoded_block = json.dumps(block, sort_keys=True).encode()
        return hashlib.sha256(encoded_block).hexdigest()

    def is_chain_valid(self, chain):
        """
        Validates the blockchain by checking:
        1. The previous_hash field matches the actual hash of the previous block.
        2. The PoW condition (hash starting with '0000') is satisfied for each block.
        """
        previous_block = chain[0]   # Genesis block
        block_index = 1             # Start checking from block 2

        while block_index < len(chain):
            block = chain[block_index]

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

            # Validate Proof of Work
            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

            # Move to next block
            previous_block = block
```

```
            block_index += 1

        return True


# ---------------------------------------------------------
# Part 2 - Mining our Blockchain (Flask API)
# ---------------------------------------------------------

# Initialize Flask web application
app = Flask(__name__)

# Create an instance of the Blockchain
blockchain = Blockchain()

# ---------------------------------------------------------
# Home Route
# ---------------------------------------------------------
@app.route('/', methods=['GET'])
def home():
    response = {
        'message': 'Welcome to the Blockchain API',
        'endpoints': {
            '/mine_block': 'Mine a new block',
            '/get_chain': 'Get full blockchain',
            '/is_valid': 'Check blockchain validity'
        }
    }
    return jsonify(response), 200


# ---------------------------------------------------------
# Favicon handler (optional, avoids 404 log spam)
# ---------------------------------------------------------
@app.route('/favicon.ico')
def favicon():
    return '', 204


# ---------------------------------------------------------
# API Endpoint: Mine a new block
# ---------------------------------------------------------
@app.route('/mine_block', methods=['GET'])
def mine_block():
```

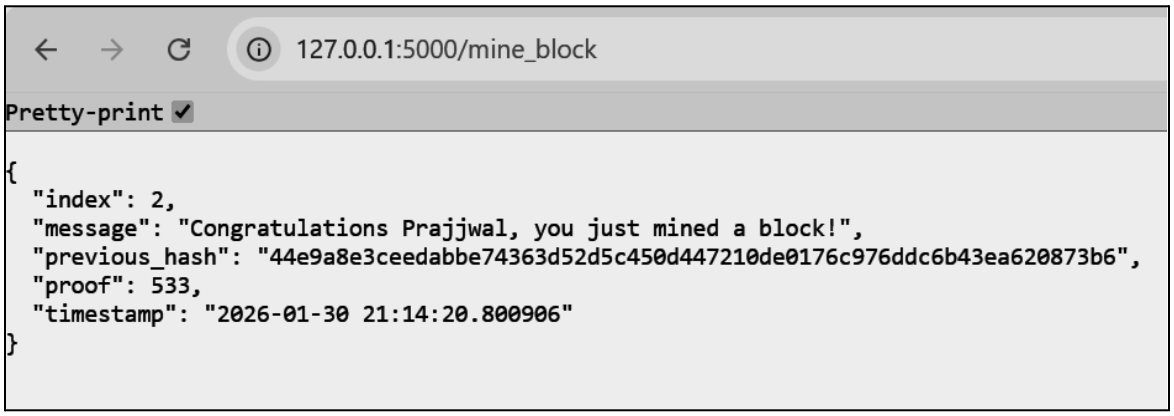
```
    # Get the previous block
    previous_block = blockchain.get_previous_block()

    # Extract previous proof
    previous_proof = previous_block['proof']

    # Run Proof of Work algorithm
    proof = blockchain.proof_of_work(previous_proof)

    # Get hash of previous block
    previous_hash = blockchain.hash(previous_block)

    # Create the new block
    block = blockchain.create_block(proof, previous_hash)

    # Prepare response
    response = {
        'message': 'Congratulations Prajjwal, you just mined a block!',
        'index': block['index'],
        'timestamp': block['timestamp'],
        'proof': block['proof'],
        'previous_hash': block['previous_hash']
    }
    return jsonify(response), 200


# ------------------------------------------------------------
# API Endpoint: Get the full blockchain
# ------------------------------------------------------------
@app.route('/get_chain', methods=['GET'])
def get_chain():
    response = {
        'chain': blockchain.chain,
        'length': len(blockchain.chain)
    }
    return jsonify(response), 200


# ------------------------------------------------------------
# API Endpoint: Check if blockchain is valid
# ------------------------------------------------------------
@app.route('/is_valid', methods=['GET'])
def is_valid():
    is_valid = blockchain.is_chain_valid(blockchain.chain)
```

```
    if is_valid:
        response = {'message': 'All good. The Blockchain is valid.'}
    else:
        response = {'message': 'Houston, we have a problem. The Blockchain is not valid.'}

    return jsonify(response), 200


# ---------------------------------------------------------
# Run the Flask app
# ---------------------------------------------------------
app.run(host='0.0.0.0', port=5000)
```

4. Output

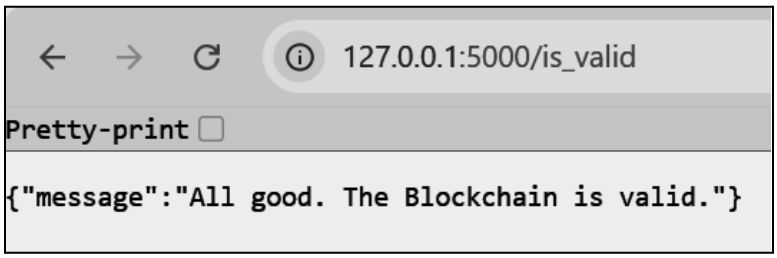
4.1 home url

4.2 /mine_block



4.3 /get_chain

4.4 /is_valid



Conclusion: In this experiment, a basic blockchain was successfully implemented using Python. The creation of blocks, mining using a Proof of Work algorithm, and validation of the blockchain demonstrated the core principles of blockchain technology. The experiment highlighted how cryptographic hashing, consensus mechanisms, and block linking work together to ensure data integrity, security, and immutability. This implementation provides a foundational understanding of how real-world blockchain systems operate.