Omkar Gholap | D20A | Roll No. 9

Experiment No. 2

Aim : To design and implement a Blockchain using Python.

Theory

1. What is Blockchain

Blockchain is an innovative technology that works as a shared and unchangeable digital ledger. The term *blockchain* is derived from its design, where data is stored in blocks, and each new block is connected to the previous one, forming a continuous chain.

Each block contains important information such as transaction details, a timestamp, and a unique cryptographic hash. This hash is created using the contents of the block along with the hash of the previous block, which ensures a strong connection between blocks.

- The chained structure of blockchain makes any data modification easily detectable, as changing one block disrupts the entire chain.
- It operates as a distributed ledger, where transaction records are stored across multiple systems.
- Transactions are validated by a majority of participants, ensuring authenticity.
- Decentralization eliminates the possibility of control or manipulation by a single authority.

Blockchain technology is both decentralized and distributed, meaning no central entity governs it. Instead, several computers known as nodes maintain identical copies of the blockchain. This synchronization ensures that once data such as a transaction is verified and recorded, it becomes almost impossible to modify or remove.

2. What is a Block

A block in a blockchain functions like a link in a chain. In cryptocurrency systems, blocks act as records that store transaction data, similar to entries in a ledger, and these records are encrypted using hash functions. Since a massive number of transactions occur daily worldwide, users rely on the block structure to track and verify these transactions efficiently. The overall block structure of a blockchain is illustrated in the initial diagram of this article.

3. Components of a Block

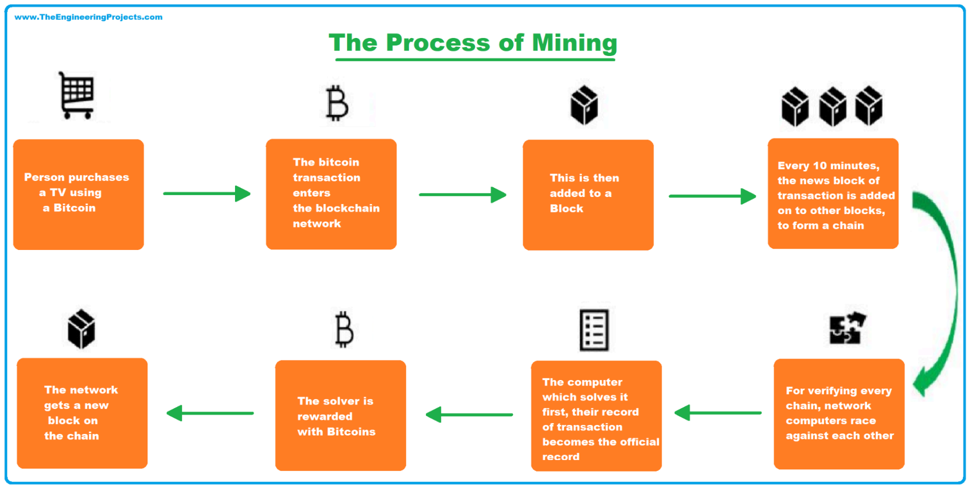
1. **Header**
   The block header uniquely identifies a block within the blockchain. It manages

block-related information and is repeatedly hashed by miners during the mining process by altering the nonce value. The block header contains three sets of metadata.

2. **Previous Block Address / Hash**
   This component links the current block to the previous block by storing the hash of the parent block, ensuring continuity within the blockchain.
3. **Timestamp**
   A timestamp records the exact date and time when the block is created. It uniquely identifies a transaction or event and confirms when the data was added to the blockchain.
4. **Nonce**
   A nonce is a number that is used only once. It plays a crucial role in the Proof-of-Work mechanism. Miners continuously test different nonce values until a valid one is found that satisfies the target condition.
5. **Merkle Root**
   The Merkle Root is a data structure that represents all transactions within a block. A Merkle Tree generates a single digital fingerprint of all transactions, allowing users to efficiently verify whether a specific transaction belongs to a block.

```
Block {
    Index: 2
    Timestamp: 2026-01-22 10:05:00
    Data: {"Sender":"Alice", "Receiver":"Bob", "Amount":50}
    Nonce: 102345
    Previous Hash: "0000a1b2c3d4e5..."
    Hash: "0000f3b7d6c1e8..."
}
```

4. Process of Mining


www.TheEngineeringProjects.com
The Process of Mining
Person purchases a TV using a Bitcoin
The bitcoin transaction enters the blockchain network
This is then added to a Block
Every 10 minutes, the news block of transaction is added on to other blocks, to form a chain
For verifying every chain, network computers race against each other
The computer which solves it first, their record of transaction becomes the official record
The solver is rewarded with Bitcoins
The network gets a new block on the chain

Mining is the method of adding a new block to the blockchain while maintaining its security and integrity through the Proof-of-Work (PoW) mechanism.

Reference:
https://www.geeksforgeeks.org/dsa/introduction-to-merkle-tree/

Step 1: Collect Transactions

- All unconfirmed transactions are gathered into a new block.
- Example: Alice transfers 50 coins to Bob.

Step 2: Create Block Header

The block header contains:

- Previous block hash
- Transaction data
- Timestamp
- Nonce (initially set to 0)

Step 3: Proof of Work (PoW)

- The miner repeatedly modifies the nonce value to generate a hash that meets the difficulty condition.
- Difficulty refers to the number of leading zeroes required in the hash (e.g., 4 leading zeroes).

```
SHA256(block_data + nonce) → hash starts with "0000"
```

- This process requires significant computational effort, which is why it is called Proof-of-Work.

Step 4: Validate and Broadcast

- Once a valid hash is discovered, the block is shared with all nodes in the network.
- Other nodes verify:
    1. Correctness of the hash
    2. Validity of the nonce
    3. Link to the previous block hash

Step 5: Add to Blockchain

- After successful verification, the block is added to the blockchain.
- The miner is rewarded, usually with cryptocurrency.

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

Step 1: Validate Previous Hash

- The `previous_hash` of the current block must match the hash of the preceding block.

```
if Block[n].previous_hash != Hash(Block[n-1]):
```

Step 2: Recalculate Current Hash

- Recompute the hash using the block's data and nonce.
- It must match the stored hash value.

```
if SHA256(Block[n].data + Block[n].nonce) != Block[n].hash:
```

Step 3: Check Proof-of-Work

- Ensure the hash meets the difficulty requirement.

```
if not Block[n].hash.startswith("0000"):
```

Step 4: Validate Entire Chain

- Repeat the above steps for all blocks from the Genesis block to the latest block.
- If any block fails validation, the blockchain is considered invalid.

Step 5: Validate Genesis Block

- The Genesis block is predefined and immutable.
- Its previous hash must be `"0"` and must remain unchanged.

Implementation

Installing Flask

blockchain.py

```
import datetime
import hashlib
import json
```

```
class Blockchain:
    def __init__(self):
        self.chain = []
        self.difficulty = 4  # 4 leading zeroes
        self.create_genesis_block()

    def create_genesis_block(self):
        genesis_block = {
            'index': 1,
            'timestamp': str(datetime.datetime.now()),
            'data': 'Genesis Block',
            'nonce': 0,
            'previous_hash': '0'
        }
        genesis_block['hash'] = self.calculate_hash(genesis_block)
        self.chain.append(genesis_block)

    def calculate_hash(self, block):
        block_copy = block.copy()
        block_copy.pop('hash', None)
        encoded = json.dumps(block_copy, sort_keys=True).encode()
        return hashlib.sha256(encoded).hexdigest()

    def mine_block(self, data):
        previous_block = self.chain[-1]
        nonce = 0
        while True:
            block = {
                'index': len(self.chain) + 1,
                'timestamp': str(datetime.datetime.now()),
                'data': data,
                'nonce': nonce,
                'previous_hash': previous_block['hash']
            }
            block_hash = self.calculate_hash(block)
            if block_hash.startswith('0' * self.difficulty):
                block['hash'] = block_hash
                self.chain.append(block)
                return block
            nonce += 1

    def is_chain_valid(self):
        for i in range(1, len(self.chain)):
            current = self.chain[i]
            previous = self.chain[i - 1]
            if current['previous_hash'] != previous['hash']:
                return False
            recalculated_hash = self.calculate_hash(current)
            if recalculated_hash != current['hash']:
                return False
            if not current['hash'].startswith('0' * self.difficulty):
                return False
        return True
```

app.py

```
from flask import Flask, jsonify
from blockchain import Blockchain

app = Flask(__name__)
blockchain = Blockchain()

@app.route('/mine_block', methods=['GET'])
def mine_block():
    block = blockchain.mine_block("Some transaction data")
    return jsonify({
        'message': '⛏️ Block mined successfully!',
        'block': block
    }), 200

@app.route('/get_chain', methods=['GET'])
def get_chain():
    return jsonify({
        'chain': blockchain.chain,
        'length': len(blockchain.chain)
    }), 200
```

```python
@app.route('/is_valid', methods=['GET'])
def is_valid():
    return jsonify({
        'is_valid': blockchain.is_chain_valid()
    }), 200

if __name__ == '__main__':
    print(" Mining blockchain with 4 leading zeroes...")
    app.run(debug=True, use_reloader=False)
```