NAME: Kunal Punjabi
Roll No : 43
Batch: C
Subject: Blockchain & DLT
Experiment : 1

Title

Understanding SHA-256 Cryptography, Proof of Work, and Merkle Tree in Blockchain using Python

Aim

The aim of this experiment is to study and practically implement important cryptographic concepts used in blockchain technology. This includes generating SHA-256 hashes, understanding the role of nonce and Proof of Work in the mining process, and constructing a Merkle Tree to generate the Merkle Root for ensuring data integrity and secure transaction verification using Python.

Theory

Cryptographic Hash Function in Blockchain

A cryptographic hash function is a mathematical algorithm that converts input data of any size into a fixed-length output called a hash. In blockchain technology, cryptographic hash functions play a critical role in ensuring data security and integrity. SHA-256 is the most commonly used hash function in blockchain systems.

Cryptographic hash functions have the following important properties:

- They are deterministic, meaning the same input always produces the same output.
- They generate a fixed-length hash regardless of the size of the input data.
- They are irreversible, so the original data cannot be retrieved from the hash.
- They show the avalanche effect, where a small change in input causes a large change in the output hash.

In blockchain, hash functions are used to secure transaction data, link blocks together, perform mining operations, and detect any tampering with stored data.

Merkle Tree

A Merkle Tree is a cryptographic data structure used to efficiently store and verify large sets of transactions in a blockchain. It is also known as a hash tree. In a Merkle Tree, each leaf node represents the hash of a transaction, while parent nodes store the hash of their child nodes.

Merkle Trees help blockchain systems verify transaction integrity without needing to process all transaction data, making the system efficient and scalable.

Structure of Merkle Tree

The structure of a Merkle Tree is hierarchical and resembles an inverted tree. It consists of three main levels:

1. Leaf Level: This level contains the hash values of individual transactions.
2. Intermediate Level: This level contains hashes generated by combining two child hashes.
3. Root Level: This top level contains a single hash called the Merkle Root.

If the number of transactions is odd, the last transaction hash is duplicated to maintain a complete binary tree structure.

Merkle Root

The Merkle Root is the final hash obtained at the top of the Merkle Tree. It uniquely represents all the transactions included in a block. Any modification in even a single transaction results in a completely different Merkle Root.

In blockchain, the Merkle Root is stored in the block header and is used to quickly verify the integrity of all transactions in the block.

Working of Merkle Tree

The working of a Merkle Tree follows a systematic process:

1. Each transaction is first converted into a hash using a cryptographic hash function.
2. These transaction hashes are paired and concatenated.
3. The concatenated pairs are hashed again to form parent nodes.
4. This process continues level by level until only one hash remains.
5. The final hash produced is called the Merkle Root.

This method allows fast and secure verification of transaction data.

Benefits of Merkle Tree

Merkle Trees provide several benefits in blockchain systems:

- Efficient verification of large numbers of transactions
- Easy detection of data tampering
- Reduced storage requirements
- Improved scalability of the blockchain network
- Strong security through cryptographic hashing

Use of Merkle Tree in Blockchain

In blockchain technology, Merkle Trees are used to organize transactions within a block. They enable quick verification of transactions, help lightweight nodes verify data without downloading the entire blockchain, and ensure the integrity of transaction data stored in blocks.

Merkle Trees also support Simplified Payment Verification (SPV), which allows users to verify transactions efficiently.

Use Cases of Merkle Tree

Merkle Trees are widely used in various real-world applications:

1. Blockchain and Cryptocurrencies: Used in Bitcoin and Ethereum for transaction verification.
2. Distributed Systems: Used to compare and synchronize data efficiently.
3. Version Control Systems: Git uses Merkle Trees to track file changes.
4. Database Systems: Used for data validation and integrity checks.
5. Peer-to-Peer Networks: Helps ensure secure data sharing.

Task 1: Hash Generation using SHA-256

Objective

To generate a SHA-256 hash for a given input string.

Code

```
import hashlib

data = input("Enter a string: ")

hash_object = hashlib.sha256(data.encode())

hash_value = hash_object.hexdigest()

print("SHA-256 Hash:", hash_value)
```

Input

```
Enter a string: a
```

Output

```
SHA-256 Hash: ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afe48bb
```

Task 2: Target Hash Generation with Nonce

Objective

To generate a hash by combining user input data with a nonce value.

Code

```
import hashlib

data = input("Enter data: ")

nonce = input("Enter nonce: ")

combined = data + nonce

hash_value = hashlib.sha256(combined.encode()).hexdigest()

print("Generated Hash:", hash_value)
```

Input

```
Enter data: Kunal

Enter nonce: 2
```

Output

```
Generated Hash: 757cbecdbeeb4d3a388853826a3bb538bc34f35908e67d6dc6b87be2a86a29
```

Task 3: Proof-of-Work Puzzle (Mining Simulation)

Objective

To simulate the mining process by finding a nonce that produces a hash with a specified number of leading zeros.

Code

```
import hashlib

data = input("Enter data: ")

difficulty = int(input("Enter difficulty (number of leading zeros): "))

prefix = '0' * difficulty

nonce = 0

while True:

    text = data + str(nonce)

    hash_value = hashlib.sha256(text.encode()).hexdigest()

    if hash_value.startswith(prefix):

        print("Nonce found:", nonce)

        print("Hash:", hash_value)

        break

    nonce += 1
```

Input

```
Enter data: Block1

Enter difficulty (number of leading zeros): 4
```

Output

```
Nonce found: 51924

Hash: 0000ae9814da50ea2a4a63a965d512dff475d5339472a838420f52fa7561
```

Task 4: Merkle Tree Construction and Merkle Root Generation

Objective

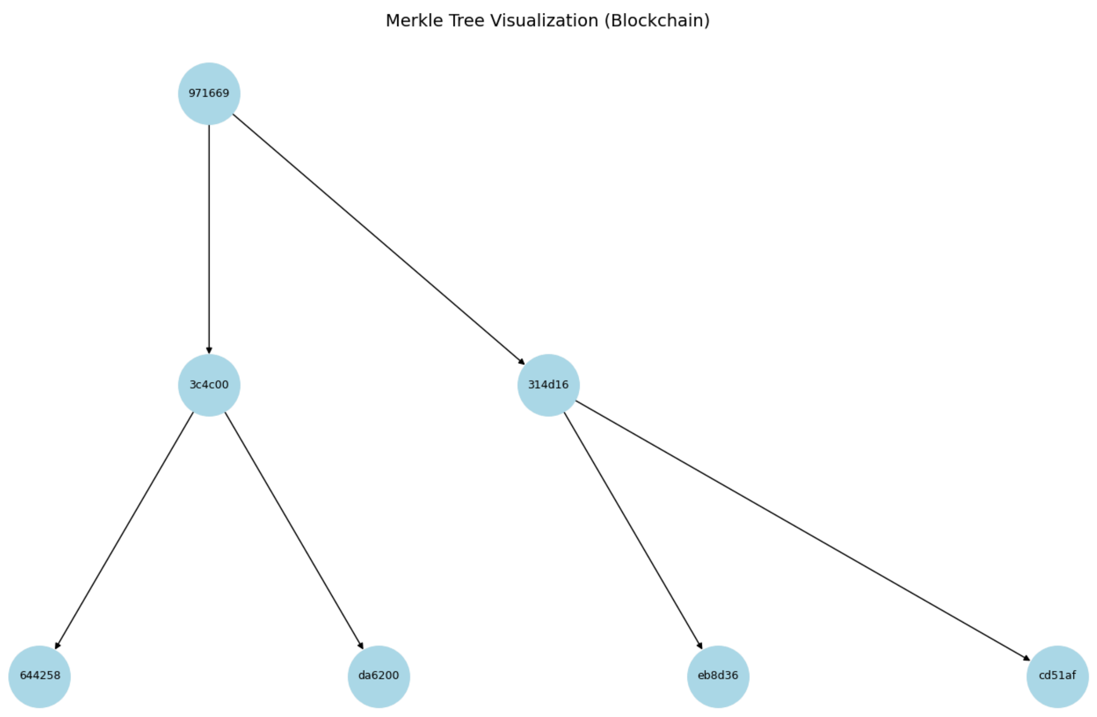
To construct a Merkle Tree from a list of transactions and generate the Merkle Root.

Code

```
import hashlib

def sha256(data):

    return hashlib.sha256(data.encode()).hexdigest()

def merkle_root(transactions):

    hashes = [sha256(tx) for tx in transactions]

    while len(hashes) > 1:

        if len(hashes) % 2 != 0:

            hashes.append(hashes[-1])

        new_level = []

        for i in range(0, len(hashes), 2):

            combined = hashes[i] + hashes[i+1]

            new_level.append(sha256(combined))

        hashes = new_level

    return hashes[0]

transactions = [

    "Alice pays Bob 10 BTC",

    "Bob pays Charlie 5 BTC",

    "Charlie pays Dave 2 BTC",

    "Dave pays Eve 1 BTC"

]

root = merkle_root(transactions)

print("Merkle Root:", root)
```

Output

```
Merkle Root: 9715669f88ef595580b4602f751cda34f82a21d87fb8547b3c31508b
```



Result

Thus, the SHA-256 hashing technique, nonce-based hash generation, Proof-of-Work mining simulation, and Merkle Tree construction were successfully implemented using Python, demonstrating core cryptographic concepts used in blockchain technology.

Conclusion

This experiment provides a practical understanding of how cryptography secures blockchain systems. SHA-256 ensures data integrity, Proof of Work provides consensus and security, and Merkle Trees enable efficient transaction verification.