Blockchain Lab

Exp 1

Aim :- To Write a Python program to understand SHA and Cryptography in Blockchain, Merkle root tree hash.

Theory :-

1. Cryptographic Hash Functions in Blockchain

Cryptographic hash functions are algorithms that take input data of any size and generate a fixed-length, irreversible hash. In blockchain, SHA-256 is commonly used to secure transactions and blocks. These functions are deterministic, collision-resistant, and produce a completely different output even with a small change in input. Hash functions ensure data integrity, link blocks together securely, and prevent tampering, forming the backbone of blockchain security.

2. Merkle Tree

A Merkle Tree is a binary tree structure that efficiently summarizes and verifies large amounts of data using cryptographic hashes. In a Merkle Tree, the leaf nodes represent the hashes of individual transactions, while the parent nodes are hashes of their child nodes combined. The topmost node, called the Merkle Root, uniquely represents all the transactions in the block. This structure is widely used in blockchain to secure transaction integrity.

3. Structure of Merkle Tree

- Leaf Nodes: Hash of individual transactions
- Intermediate Nodes: Hash of two child nodes combined
- Merkle Root: Final single hash at the top

If the number of nodes is odd, the last hash is duplicated.

4. Merkle Rule

The Merkle rule states that each parent node in the tree is computed as the hash of its two child nodes. If there is an odd number of nodes at any level, the last node is duplicated before hashing. This process continues recursively until a single hash, called the Merkle Root, is obtained. This rule ensures that the root hash uniquely represents all underlying transactions and any modification in a transaction can be quickly detected.

5. Working of Merkle Tree

The working of a Merkle Tree involves first hashing all individual transactions to create leaf nodes. Then, pairs of hashes are combined and hashed repeatedly to form intermediate nodes. This process continues until only the Merkle Root remains at the top of the tree. The Merkle Root serves as a single, verifiable summary of all transactions, allowing efficient verification without needing to examine every transaction individually.

6. Benefits of Merkle Tree

- Efficient data verification
- Reduced storage requirements
- Fast integrity checking
- Tamper detection
- Scalable for large datasets

7. Use of Merkle Tree in Blockchain

In blockchain, Merkle Trees are used to securely store transactions in each block and generate the Merkle Root. This allows nodes to verify transactions efficiently without downloading the entire block, which is especially useful for lightweight clients. Merkle Trees also ensure data integrity, prevent tampering, and maintain a verifiable link between all transactions within a block

8. Use Cases of Merkle Tree

- Blockchain transaction verification
- Distributed systems
- File integrity verification
- Version control systems (Git)
- Peer-to-peer networks
- Database consistency checking

Code :-

```
# Experiment 1

import hashlib

# SHA-256 Hash Generation

def sha256_hash(data):
```

```
    """
    Generates SHA-256 hash of the given input string
    """
    return hashlib.sha256(data.encode()).hexdigest()

# Hash Generation with Nonce
def hash_with_nonce(data, nonce):
    """
    Generates SHA-256 hash by combining data with nonce
    """
    combined_data = data + str(nonce)
    return sha256_hash(combined_data)

# Proof-of-Work (Mining Simulation)

def proof_of_work(data, difficulty):
    """
    Finds a nonce such that the hash has leading zeros
    based on the difficulty level
    """
    nonce = 0
    target = "0" * difficulty

    while True:
        hash_result = hash_with_nonce(data, nonce)
        if hash_result.startswith(target):
            return nonce, hash_result
        nonce += 1

# Merkle Tree Construction

def merkle_root(transactions):
    """
    Generates Merkle Root from a list of transactions
    """
    # Step 1: Hash all transactions
    current_level = [sha256_hash(tx) for tx in transactions]

    # Step 2: Build tree until one hash remains
    while len(current_level) > 1:
```

```
        next_level = []

        # If odd number of hashes, duplicate last hash
        if len(current_level) % 2 != 0:
            current_level.append(current_level[-1])

        # Hash pairs of nodes
        for i in range(0, len(current_level), 2):
            combined_hash = current_level[i] + current_level[i + 1]
            parent_hash = sha256_hash(combined_hash)
            next_level.append(parent_hash)

        current_level = next_level

    return current_level[0]


# Main program


print("========== TASK 1: SHA-256 HASH GENERATION ==========")
input_data = input("Enter a string to hash: ")
hash_result = sha256_hash(input_data)
print("SHA-256 Hash:", hash_result)

print("\n========== TASK 2: HASH WITH NONCE ==========")
nonce_value = int(input("Enter a nonce value: "))
hash_nonce = hash_with_nonce(input_data, nonce_value)
print("Hash with Nonce:", hash_nonce)

print("\n========== TASK 3: PROOF-OF-WORK ==========")
difficulty = int(input("Enter difficulty level (e.g., 2, 3, 4): "))
nonce, pow_hash = proof_of_work(input_data, difficulty)
print("Nonce found:", nonce)
print("Valid Hash:", pow_hash)

print("\n========== TASK 4: MERKLE TREE ==========")
num_tx = int(input("Enter number of transactions: "))
transactions = []
```

Blockchain Lab

```
for i in range(num_tx):
    tx = input(f"Enter transaction {i + 1}: ")
    transactions.append(tx)

root = merkle_root(transactions)
print("Merkle Root:", root)

print("\n========== EXPERIMENT COMPLETED ==========")
```

Output :-

```
========== TASK 1: SHA-256 HASH GENERATION ==========
Enter a string to hash: qwertykeyboard
SHA-256 Hash: 08222af35f7c79715c018c2ec3101a725f417fd2c7223eff2546fc031a89ff9e

========== TASK 2: HASH WITH NONCE ==========
Enter a nonce value: 20012026
Hash with Nonce: 76f499975122f50eb25fd10a9dbfd8da46d9298c6a9c55988b1749327bf95533

========== TASK 3: PROOF-OF-WORK ==========
Enter difficulty level (e.g., 2, 3, 4): 5
Nonce found: 1058567
Valid Hash: 0000006f4e33c7ab1c5b1e85e7ba169a80324333ba24f8ff1779a3b09466439a

========== TASK 4: MERKLE TREE ==========
Enter number of transactions: 3
Enter transaction 1: 12
Enter transaction 2: 24
Enter transaction 3: 36
Merkle Root: 466cf447ec27f73e45fd90c092b97c2db6ce8f6bf8a10a0c23e8598726d75cac

========== EXPERIMENT COMPLETED ==========
```

Conclusion :-

This experiment was performed on Google Colab. The concept of Nonce value, its purpose, how it contributes to the mining process, and the concept of Proof-of-Work was understood.