Name: **Anjali punsi**
Roll No.: **57**
Div: **D20B**
Lab: **Blockchain**

Lab - 1: **Experiment No : 1** **Dated : 02/02/2024**

AIM : **Cryptography in Blockchain, Merkle root Tree Hash**

Lab Objectives : To realize the basic techniques to build intelligent systems

Lab Outcomes (LO) **:** Creating Cryptographic Hash using Merkle Tree (LO1)

Task to be performed : **Write a program in any language to create a Merkle Tree as a Binary Tree Data Structure using SHA-256**

Hint : Consider SHA256 to hash the transactions.

Input : Given the following transactions

T1 : Alice gave Rs. 200 to Bob

T2 : Bob gave Rs. 500 to Dave

T3 : Dave gave Rs. 100 to Eve

T4 : Eve gave Rs. 300 to Alice

T5 : Roo gave Rs. 50 to Bob

Output : Display the Hash at each step of merging till the Merkle Root

Tools & Libraries used :

- Python Libraries : **hashlib**

Code:

```
import hashlib
import json
from time import time

def merkle_tree(transactions):
    if not transactions:
        return "No transactions", []
```

```
    merkle_tree, merkle_tree_steps = build_merkle_tree(transactions)
    merkle_root = merkle_tree[0]
    return merkle_root, merkle_tree_steps

def build_merkle_tree(transactions):
    if len(transactions) == 1:
        return [hashlib.sha256(str(transactions[0]).encode()).hexdigest()], []

    hashes = [hashlib.sha256(str(tx).encode()).hexdigest() for tx in transactions]
    merkle_tree_steps = [hashes.copy()]

    while len(hashes) > 1:
        merged_hashes = []
        merged_steps = []
        for i in range(0, len(hashes), 2):
            if i + 1 == len(hashes):
                combined_hashes = hashes[i] + hashes[i]
                merged_steps.append(((i, i), (hashes[i], hashes[i])))
            else:
                combined_hashes = hashes[i] + hashes[i + 1]
                merged_steps.append(((i, i + 1), (hashes[i], hashes[i + 1])))
            merged_hashes.append(hashlib.sha256(combined_hashes.encode()).hexdigest())

        hashes = merged_hashes
        merkle_tree_steps.append(merged_steps)

    return hashes, merkle_tree_steps

class Block:
    def __init__(self, index, previous_hash, timestamp, transactions, merkle_tree_root):
        self.index = index
        self.previous_hash = previous_hash
        self.timestamp = timestamp
        self.transactions = transactions
        self.merkle_tree_root = merkle_tree_root
        self.hash = self.calculate_hash()
```

```
    def calculate_hash(self):
        block_info = {
            "index": self.index,
            "previous_hash": self.previous_hash,
            "timestamp": self.timestamp,
            "transactions": self.transactions,
            "merkle_tree_root": self.merkle_tree_root,
        }
        block_string = json.dumps(block_info, sort_keys=True)
        return hashlib.sha256(block_string.encode()).hexdigest()

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

    def create_genesis_block(self):

        genesis_block = Block(0, "0", time(), [], "genesis_merkle_root")
        self.chain.append(genesis_block)

        print(
            "Index: {}\nPrevious Hash: {}\nTimestamp: {}\nMerkle Root: {}\nTransactions:
{}\nHash: {}\n{}".format(
                genesis_block.index,
                genesis_block.previous_hash,
                genesis_block.timestamp,
                genesis_block.merkle_tree_root,
                genesis_block.transactions,
                genesis_block.hash,
                "=" * 50,
            )
        )

    def add_block(self, transactions):
        previous_block = self.chain[-1]
```

```
        merkle_tree_root, merkle_tree_steps = merkle_tree(transactions)
        new_block = Block(
            len(self.chain),
            previous_block.hash,
            time(),
            transactions,
            merkle_tree_root,
        )
        self.chain.append(new_block)

        print(
            "Index: {}\nPrevious Hash: {}\nTimestamp: {}\nMerkle Root: {}\nTransactions:
{}\nHash: {}\n{}".format(
                new_block.index,
                new_block.previous_hash,
                new_block.timestamp,
                new_block.merkle_tree_root,
                new_block.transactions,
                new_block.hash,
                "=" * 50,
            )
        )
        print("\nMerkle Tree Steps for Block #{}:\n".format(new_block.index))
        for i, steps in enumerate(merkle_tree_steps[1:]):
            print(f"Step {i + 1}: Merging {steps}")
        print("\nMerkle Tree Root: Final Hash of Merged Transactions =", merkle_tree_root)

    def calculate_merkle_tree(self, transactions):
        if len(transactions) == 0:
            return "No transactions", []
        return build_merkle_tree(transactions)

if __name__ == "__main__":
    blockchain = Blockchain()
    transactions1 = ["Alice gave Rs. 200 to Bob", "Bob gave Rs. 500 to Dave", "Dave gave Rs.
100 to Eve", "Eve gave Rs. 300 to Alice", "Roo gave Rs. 50 to Bob"]
    blockchain.add_block(transactions1)
```

Output:

```
PS C:\Users\Anjali\Downloads\pyt> python pp.py
Index: 0
Previous Hash: 0
Timestamp: 1707049284.1940467
Transactions: []
Hash: 85773e0225f6dfc49f76a76910dc6154aefd0ecc0995cb2191ef0c9c8c67ee60
================================================
Index: 1
Previous Hash: 85773e0225f6dfc49f76a76910dc6154aefd0ecc0995cb2191ef0c9c8c67ee60
Timestamp: 1707049284.198026
Merkle Root: 4b57b8dfd128471d9308edcfdc71fda46eecaa3d13358a493cacfe1850799023
Transactions: ['Alice gave Rs. 200 to Bob', 'Bob gave Rs. 500 to Dave', 'Dave gave Rs. 100 to Eve', 'Eve gave Rs. 300 to Alice', 'Roo gave Rs. 50 to Bob']
Hash: 59e38d48c333d8fe5431c1b153bc8fa831f5d975e9f7cb1ef0e351cb542b6fc5
================================================
```

Hash generation steps:

```
------------------------------------------------

Merkle Tree Steps for Block #1:

Step 1: Merging [((0, 1), ('a12ffcfbf500ee9e089bfe6996c324d513931bf8980315c1bb0d409218158fc0', '35e47f1ecec97e51624c1277debb379603001c17c9b201ac72195af9dd52bf7e')), ((2, 3), ('d9dc48e2d8d6fffd54e0407a8
577f14bd8dfd82026f1934f7ad9605e9ed3e323', 'c3389c1ae52386e2352cee571212303c49e6c9aaee51ec3d6841dfa5ad9b9e71')), ((4, 4), ('9d02109ff442ce155d149151486cb58f77d373551a388ca71c8b20a7f12b21d0', '9d02109ff4
42ce155d149151486cb58f77d373551a388ca71c8b20a7f12b21d0'))]
Step 2: Merging [((0, 1), ('71113eb6fe8b095cdef81d23b68d46af587ab273034cd60bed7f2642894f590e', '15ea2c5547f4b51199af0f95ea6c730eb3a603807fbf6c9cbbc21469c0973207')), ((2, 2), ('decfcecbc69cacb08f674f751
77eae8d1912aef34728c4f8070c2cfa770bebba', 'decfcecbc69cacb08f674f75177eae8d1912aef34728c4f8070c2cfa770bebba'))]
Step 3: Merging [((0, 1), ('3416d86cde9891208d25e89fe74a50681f3fa1f615bdb08303250914ad17ba7f', '488ebefa506edeb70186b46219b5046fa854dd47eee9c79ea53575fed3c14850'))]

Merkle Tree Root: Final Hash of Merged Transactions = 4b57b8dfd128471d9308edcfdc71fda46eecaa3d13358a493cacfe1850799023
PS C:\Users\Anjali\Downloads\pyt>
```

Conclusion:- In this experiment, we learned how to perform Cryptography in Blockchain, Merkle root Tree Hash