Prajyot Shinde Batch : B , Roll No : 55

EXPERIMENT NO: 01

Aim

To study and implement cryptographic hashing using SHA-256, simulate blockchain mining using nonce and Proof-of-Work, and construct a Merkle Tree to generate the Merkle Root for transaction integrity.

Theory

1. Cryptography in Blockchain

Cryptography plays a fundamental role in securing blockchain networks. It ensures **data confidentiality, integrity, and authenticity**. One of the most important cryptographic tools used in blockchain is the **cryptographic hash function**.

A cryptographic hash function converts input data of any size into a **fixed-length unique output** known as a *hash*. This output acts as a digital fingerprint of the original data. Even a tiny modification in the input produces a completely different hash, making it extremely difficult to tamper with blockchain records.

Blockchain systems such as **Bitcoin and Ethereum** use the **SHA-256 hashing algorithm**, which provides strong security and reliability.

2. SHA-256 Hash Algorithm

SHA-256 (Secure Hash Algorithm – 256 bit) belongs to the SHA-2 family and generates a **256-bit hash value**.

Key Features of SHA-256:

- **Deterministic:** Same input always gives the same hash.
- **Fixed Output Size:** Output is always 256 bits.
- **Fast Execution:** Generates hash quickly.
- **One-Way Function:** Original input cannot be retrieved.
- **Collision Resistant:** Two different inputs rarely generate the same hash.
- **Avalanche Effect:** Small change in input creates a totally different output.

In blockchain, SHA-256 is used to:

- Link blocks together.
- Secure transaction records.
- Perform mining and verification.
- Maintain immutability.

3. Nonce and Proof-of-Work

A **nonce** is a numeric value added to block data to generate a valid hash that satisfies mining difficulty rules.

Proof-of-Work (PoW) is a mechanism where miners repeatedly try different nonce values until they find a hash that starts with a required number of **leading zeros**. This process ensures:

- Security
- Fair competition
- Network consensus

4. Merkle Tree and Merkle Root

A **Merkle Tree** is a hierarchical data structure used to efficiently verify transaction integrity.

Each transaction is hashed, and pairs of hashes are combined and hashed again until only **one hash remains**, known as the **Merkle Root**.

Advantages of Merkle Tree:

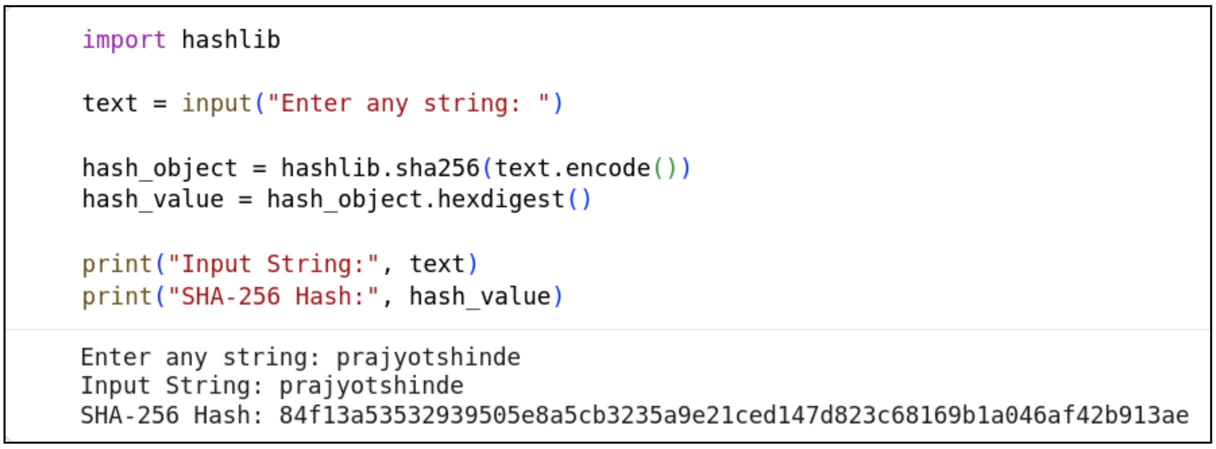
- Fast transaction verification
- Reduced storage
- High security
- Efficient data validation

If any transaction changes, the Merkle Root changes instantly, making tampering easy to detect.

Algorithms

Algorithm 1 – SHA-256 Hash Generation

1. Take input string.
2. Encode input into bytes.
3. Generate SHA-256 hash.
4. Display hexadecimal hash output.

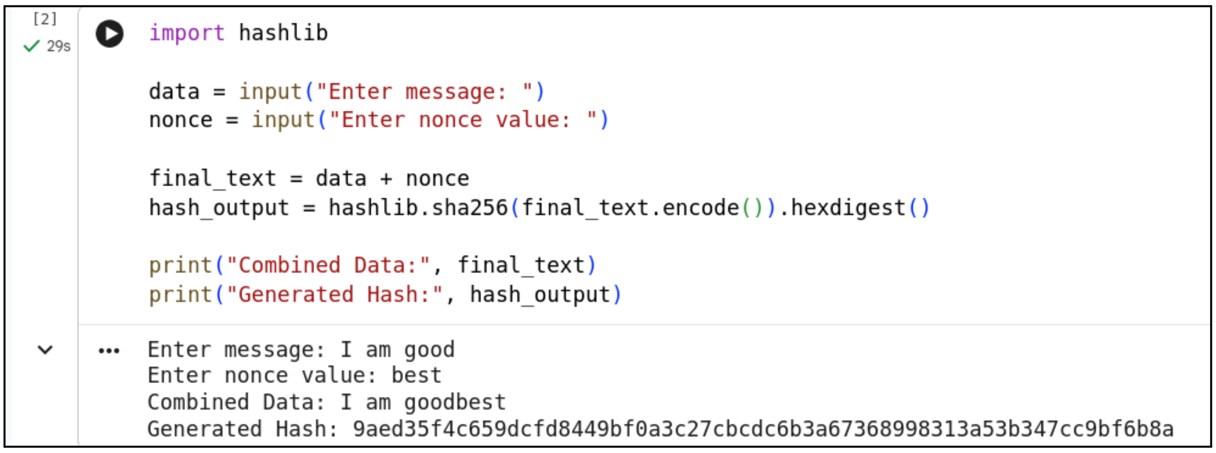


```python
import hashlib

text = input("Enter any string: ")

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

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

```
Enter any string: prajyotshinde
Input String: prajyotshinde
SHA-256 Hash: 84f13a53532939505e8a5cb3235a9e21ced147d823c68169b1a046af42b913ae
```

Algorithm 2 – Hash with Nonce

1. Accept data and nonce.
2. Concatenate them.
3. Apply SHA-256.
4. Display generated hash.

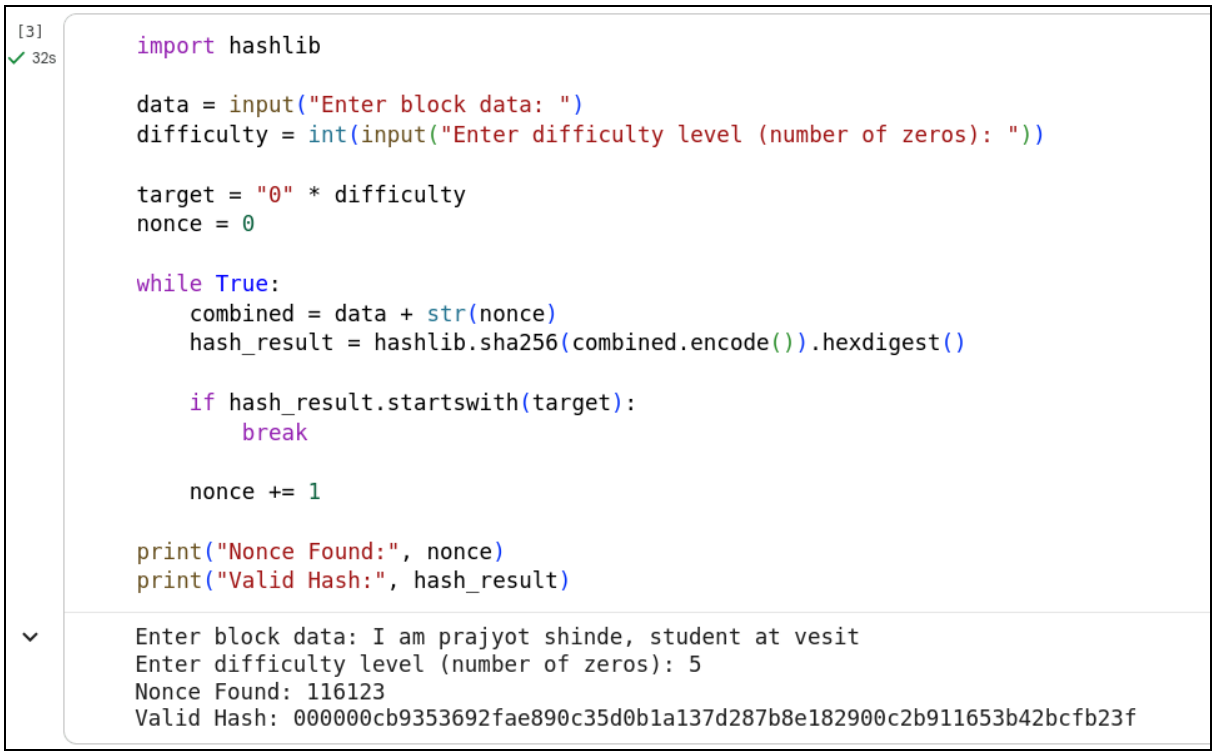


```python
import hashlib

data = input("Enter message: ")
nonce = input("Enter nonce value: ")

final_text = data + nonce
hash_output = hashlib.sha256(final_text.encode()).hexdigest()

print("Combined Data:", final_text)
print("Generated Hash:", hash_output)
```

```
Enter message: I am good
Enter nonce value: best
Combined Data: I am goodbest
Generated Hash: 9aed35f4c659dcfd8449bf0a3c27cbcdc6b3a67368998313a53b347cc9bf6b8a
```

Algorithm 3 – Proof-of-Work Mining

1. Take input string.
2. Define difficulty.
3. Start nonce from zero.
4. Keep hashing until hash starts with required zeros.
5. Output nonce and final hash.

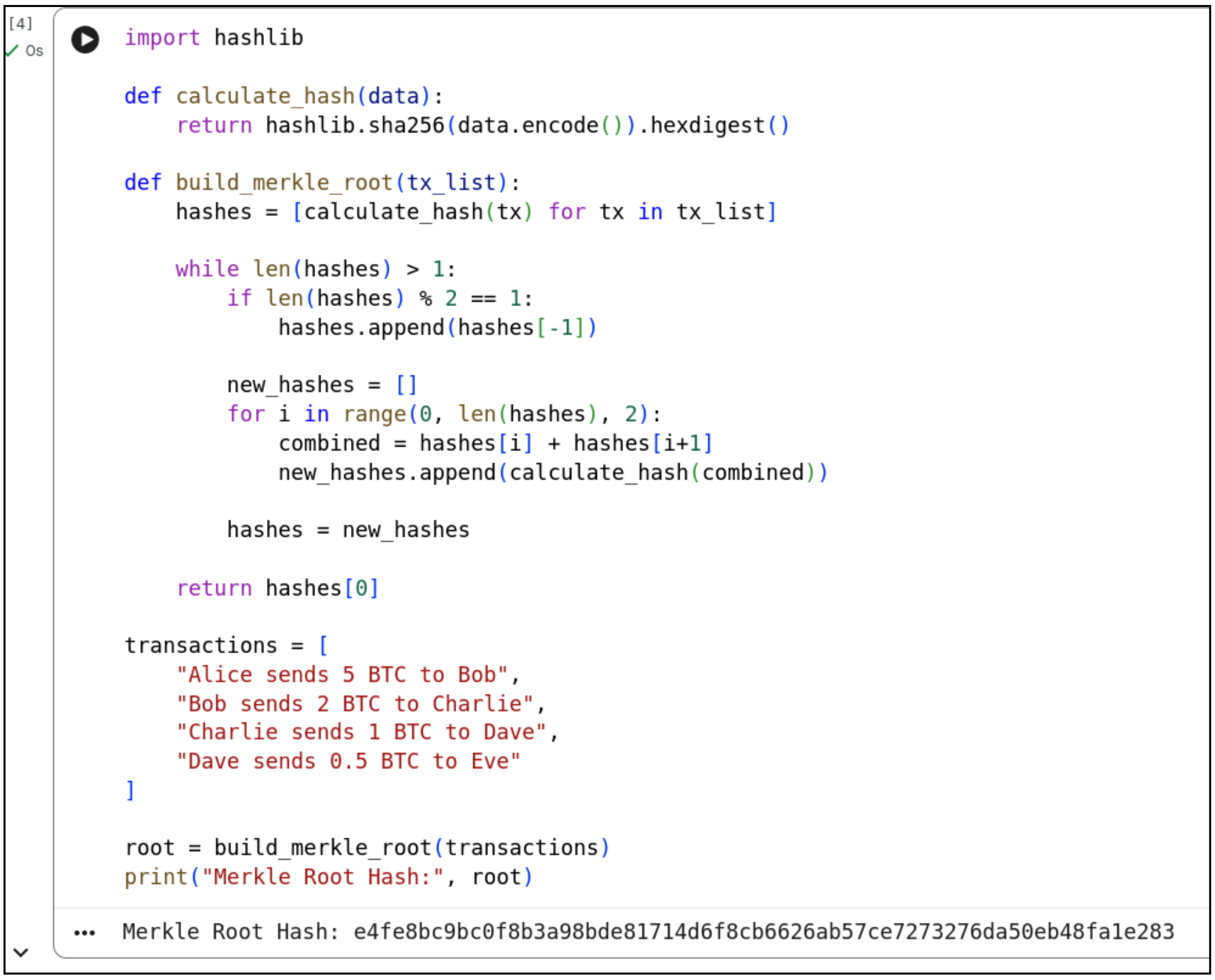


```
[3]
✓ 32s
import hashlib

data = input("Enter block data: ")
difficulty = int(input("Enter difficulty level (number of zeros): "))

target = "0" * difficulty
nonce = 0

while True:
    combined = data + str(nonce)
    hash_result = hashlib.sha256(combined.encode()).hexdigest()

    if hash_result.startswith(target):
        break

    nonce += 1

print("Nonce Found:", nonce)
print("Valid Hash:", hash_result)
```

```
Enter block data: I am prajyot shinde, student at vesit
Enter difficulty level (number of zeros): 5
Nonce Found: 116123
Valid Hash: 000000cb9353692fae890c35d0b1a137d287b8e182900c2b911653b42bcfb23f
```

Algorithm 4 – Merkle Root Generation

1. Hash each transaction.
2. Pair and hash adjacent values.
3. Duplicate last hash if count is odd.
4. Repeat until one hash remains.
5. Output Merkle Root.


[4]
✓ 0s

```python
import hashlib

def calculate_hash(data):
    return hashlib.sha256(data.encode()).hexdigest()

def build_merkle_root(tx_list):
    hashes = [calculate_hash(tx) for tx in tx_list]

    while len(hashes) > 1:
        if len(hashes) % 2 == 1:
            hashes.append(hashes[-1])

        new_hashes = []
        for i in range(0, len(hashes), 2):
            combined = hashes[i] + hashes[i+1]
            new_hashes.append(calculate_hash(combined))

        hashes = new_hashes

    return hashes[0]

transactions = [
    "Alice sends 5 BTC to Bob",
    "Bob sends 2 BTC to Charlie",
    "Charlie sends 1 BTC to Dave",
    "Dave sends 0.5 BTC to Eve"
]

root = build_merkle_root(transactions)
print("Merkle Root Hash:", root)
```

```
Merkle Root Hash: e4fe8bc9bc0f8b3a98bde81714d6f8cb6626ab57ce7273276da50eb48fa1e283
```

Result

Thus, the Python programs for **SHA-256 hashing, mining simulation using nonce, Proof-of-Work, and Merkle Tree construction** were successfully implemented and verified.

Conclusion

SHA-256 hashing ensures **data security, immutability, and integrity** in blockchain. Proof-of-Work introduces computational difficulty to maintain decentralization and trust. Merkle Trees provide **efficient transaction validation and storage optimization**. Together, these mechanisms form the backbone of modern blockchain security systems.