Bhartiya Vidya Bhavan's
Sardar Patel Institute of Technology, Mumbai-400058
Department of Computer Science and Engineering
OEIT1:Blockchain Technology and Applications

Lab3A: Blockchain Programming-I

Name : Adwait Purao
UID: 2021300101
Batch: B
Division: Comps

Objective: Develop a simple Blockchain

Outcomes: After successful completion of lab students should be able to
Develop and demonstrate Blockchain fundamentals
Write a code in python to build a Blockchain
Demonstrate the Blockchain basic working
Solve cryptographic puzzles for the desired prefix of the hash value of a blockchain.

System Requirements:
PC (C2D, 4GB RAM, 100GB HDD space and NIC)
Ubuntu Linux 14.04/20.04
Internet connectivity
Python Cryptography and Pycrypto

Part-3A: Implementing Simple Blockchain using Python

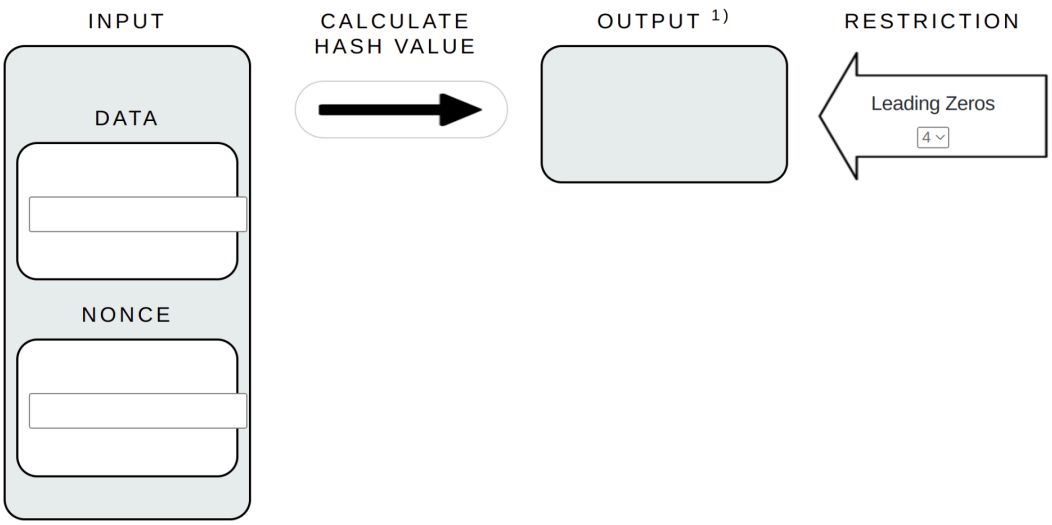


Figure-1: Hash puzzle

Write a code for a simple blockchain concept.
Compute the hash.
Solve the cryptographic puzzle (nonce) with prefix four zeros to hash value of a block.

Implement Blockchain using Python

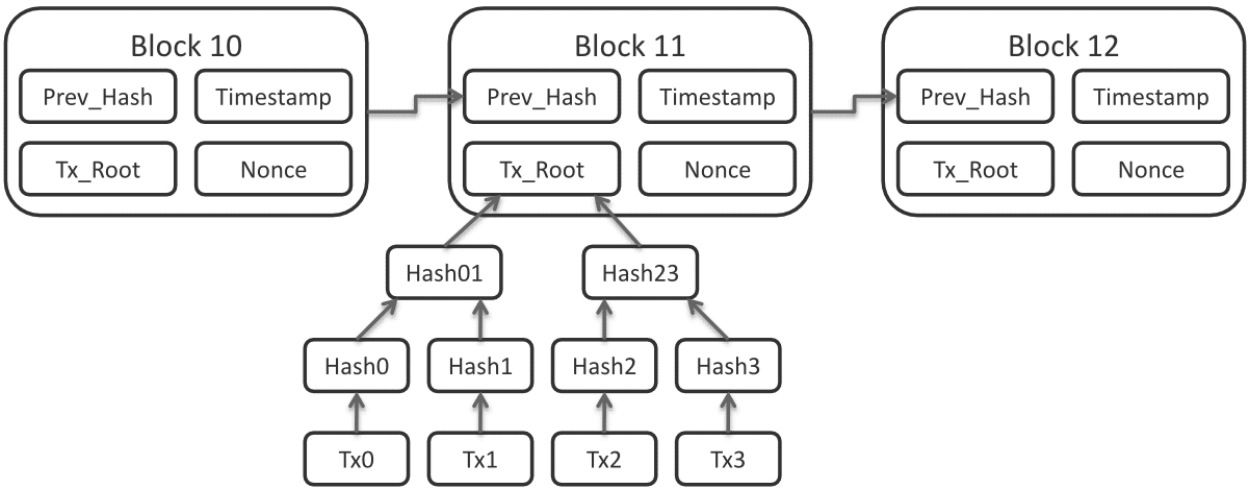


Figure-2: Blockchain Structure

Build a simple Blockchain with Block A, Block B, and Block C.

class Block:
id = None
history = None
parent_id = None

Sign data in Blockchain
Apply the Linked list to the blockchain
Perform a mining process (proof-of-work) for the hash output to have at least five zeros at the front.

```
import datetime

import hashlib

import time



class Block:

   def __init__(self, data, previous_hash=None):

       self.timestamp = datetime.datetime.now()

       self.data = data

       self.previous_hash = previous_hash

       self.hash = self.calculate_hash()



   def calculate_hash(self):

       sha = hashlib.sha256()

       sha.update(str(self.timestamp).encode('utf-8') +

                  str(self.data).encode('utf-8') +

                  str(self.previous_hash).encode('utf-8'))

       return sha.hexdigest()



class Blockchain:
```

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

    def create_genesis_block(self):
        return Block("Genesis Block", "0")

    def add_block(self, data, difficulty):
        start_time = time.time()
        previous_hash = self.chain[-1].hash
        new_block = Block(data, previous_hash)
        new_block = self.proof_of_work(new_block, difficulty)
        self.chain.append(new_block)
        end_time = time.time()
        time_taken = end_time - start_time
        print(f"Block mined in {time_taken:.4f} seconds with difficulty level {difficulty}")
        return time_taken

    def proof_of_work(self, block, difficulty):
        prefix_zeros = '0' * difficulty
        while True:
            if block.hash[:difficulty] == prefix_zeros:
```

```
            return block
            else:
            block.timestamp = datetime.datetime.now()
            block.hash = block.calculate_hash()

    def print_last_block(self):
        last_block = self.chain[-1]
        print("Last Block:")
        print("Timestamp:", last_block.timestamp)
        print("Data:", last_block.data)
        print("Previous Hash:", last_block.previous_hash)
        print("Hash:", last_block.hash)



blockchain = Blockchain()
data = "Block A Data"

difficulty_levels = range(8)
times = []

for difficulty in difficulty_levels:
    time_taken = blockchain.add_block(data, difficulty)
```

```
  times.append(time_taken)
```

```
Block mined in 0.0010 seconds with difficulty level 0
Block mined in 0.0001 seconds with difficulty level 1
Block mined in 0.0001 seconds with difficulty level 2
Block mined in 0.0716 seconds with difficulty level 3
Block mined in 0.8503 seconds with difficulty level 4
Block mined in 0.0565 seconds with difficulty level 5
Block mined in 7.1598 seconds with difficulty level 6
Block mined in 351.8290 seconds with difficulty level 7
```

```
import matplotlib.pyplot as plt

plt.plot(difficulty_levels, times, marker='o')

plt.title("Difficulty vs. Time")

plt.xlabel("Difficulty Level")

plt.ylabel("Time Taken (seconds)")

plt.xticks(difficulty_levels)

plt.grid(True)

plt.show()
```

Conclusion:

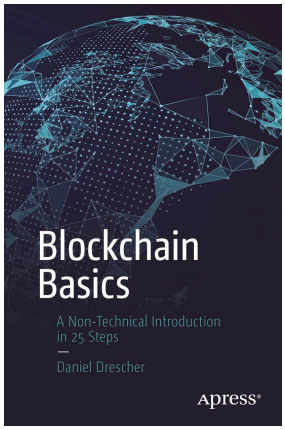
Through this experiment, I acquired a solid understanding of the core principles of blockchain technology and how to implement it using Python. The process of creating a basic blockchain allowed me to comprehend the linkage of blocks via cryptographic hashes and the use of proof-of-work in the mining of blocks. I also understood the importance of difficulty levels in proof-of-work and how they influence the duration of mining. This practical exercise offered a lucid comprehension of the basic elements of blockchain and the workings behind its functionality.

References:

[1] Blockchain Explained: What Is Blockchain & How Does It Work?

[2] What Is a Nonce? A No-Nonsense Dive into Proof of Work - CoinCentral

[3] Hash Puzzle



Refer to the book Blockchain Basics by Daniel

Lab3B: Blockchain Programming-II
Merkle-tree cryptographic library for generation and validation of Proofs

Note: *Do it yourself laboratory (DIY) and demonstrate.*

- *Read and understand the Merkle Trees and Blockchains*
- *Refer to the slides for your immediate reading*

Objective: Merkle Tree Implementation, generation and validation proofs

Outcomes: After successful completion of lab students should be able to
Implement Merkle Tree and demonstrate
Write a code in python to build a Merkle Tree
Demonstrate the Merkle Tree as a fundamental part of Blockchain.

System Requirements:
PC (C2D, 4GB RAM, 100GB HDD space and NIC)
Ubuntu Linux 14.04/20.04
Internet connectivity
Pymerkle
Python Cryptography and Pycrypto

Part-4A: Implementing Merkle Tree using Python

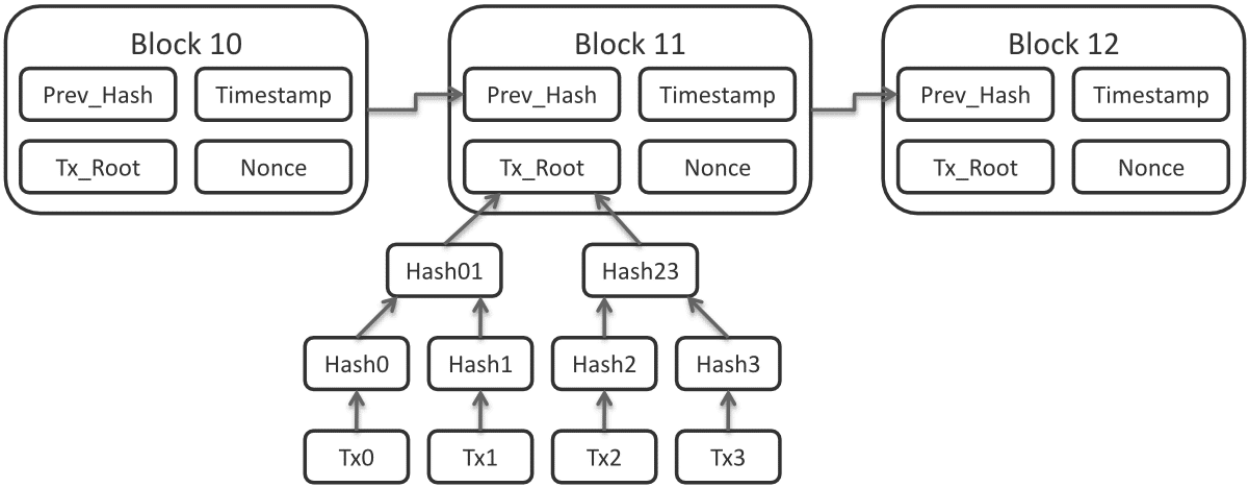


Figure-1: Merkle Tree

Procedure:
[1] Install merkel [1]:
pip install merkle

[2] Import merkle
from pymerkle import *

[3] Merkle tree object:
tree=MerkleTree()

[4] Explore configuration of tree and Attributes and properties
Refer to the official documentation of Merkle Tree implementation (pymerkle)

[5] Create 10 transactions record and build the Merkle tree and verify

Add screenshot with brief description.

[6] Save as a file:

```
with open('current_state', 'w') as f:
    f.write(tree.__repr__())
```

[7] Export and save as JSON

```
tree.export('backup.json')
```

[8] Recover the tree by means of the .loadFromFile classmethod:

```
loaded_tree = MerkleTree.loadFromFile('backup.json')
```

[9] Explore the Encryption modes

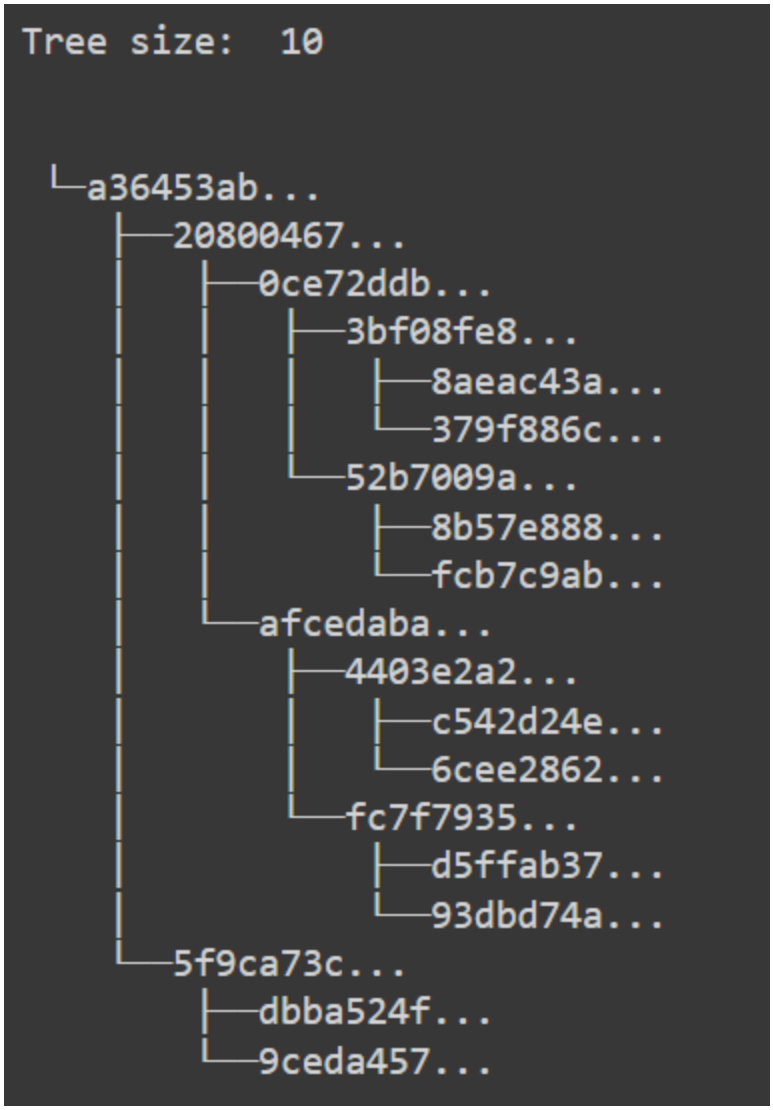
[10] Explore proof generation and validation

[11] Explore the Inclusion Tests

```
!pip install pymerkle
```

```
Requirement already satisfied: pymerkle in /usr/local/lib/python3.10/dist-packages (6.1.0)
Requirement already satisfied: cachetools==5.3.1 in /usr/local/lib/python3.10/dist-packages (from pymerkle) (5.3.1)
```

```
from pymerkle import InmemoryTree as MerkleTree

tree = MerkleTree(algorithm="sha256")

for i in range(1,11):

 tree.append_entry(bytes(("Transaction "+str(i)).encode()))

size = tree.get_size()

print("Tree size: ", size)

print(tree)
```



Conclusion:

This experiment provided me with a deep understanding of Merkle Trees and their crucial function in the realm of blockchain technology. By utilizing the pymerkle library for Python, I delved into the construction, validation, and alteration of Merkle Trees. The procedures of creating and verifying proofs, along with conducting inclusion tests, illuminated the efficiency and security features of Merkle Trees in a blockchain setting. This hands-on experience amplified my grasp of the importance of Merkle Trees in blockchain infrastructures, particularly in terms of maintaining data integrity and enhancing efficiency.

References:

[1] Merkle-tree cryptographic library for generation and validation of Proofs

https://pymerkle.readthedocs.io/en/latest/index.html?highlight=install#installation