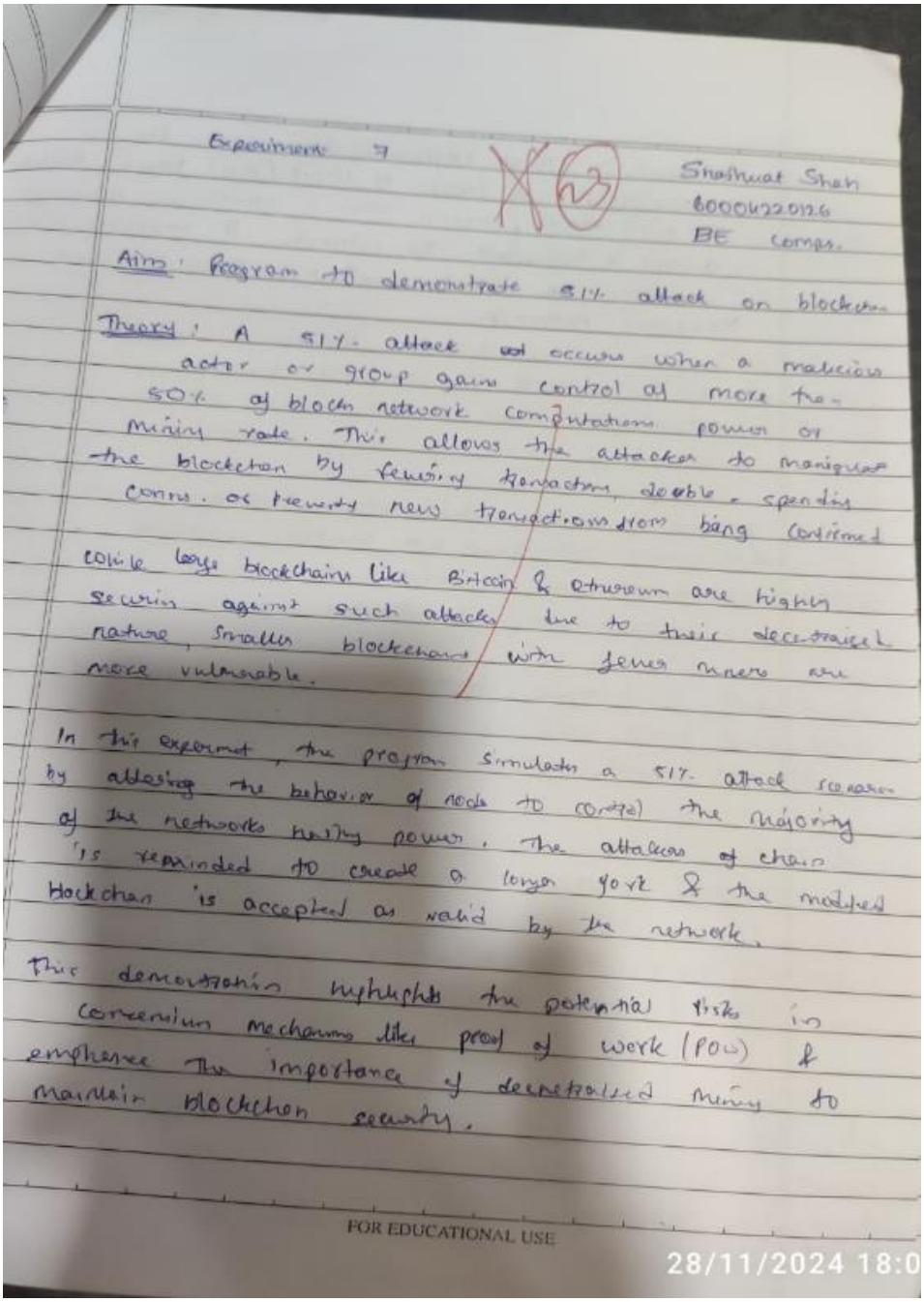
BLOCKCHAIN TECHNOLOGY

Name:-Preksha Ashok Patel
Sapid:-60004210126
Branch:-Computer Engineering
Div/Batch:-C2-1

EXPERIMENT NO.07


Experiment 7

Shashwat Shah
60004220126
BE comps.

Aim: Program to demonstrate 51% attack on blockchain

Theory: A 51% attack occurs when a malicious actor or group gain control of more than 50% of blockchain network computations power or mining rate. This allows the attacker to manipulate the blockchain by reversing transactions, double-spending coins, or preventy new transactions from being confirmed

While large blockchains like Bitcoin & Ethereum are highly secure against such attacks due to their decentralized nature, smaller blockchains with fewer miners are more vulnerable.

In this experiment, the program simulates a 51% attack scenario by altering the behavior of nodes to control the majority of the network's mining power. The attacker's chain is rewarded to create a longer fork & the modified blockchain is accepted as valid by the network.

This demonstration highlights the potential risks in consensus mechanisms like proof of work (POW) & emphasizes the importance of decentralized mining to maintain blockchain security.

CODE & OUTPUT :-

```
from hashlib import sha256
from time import time
import random

class Block:
    def _init_(self, index, transactions, timestamp, previous_hash):
        self.index = index
        self.transactions = transactions
        self.timestamp = timestamp
        self.previous_hash = previous_hash
        self.nonce = 0
        self.hash = self.calculate_hash()

    def calculate_hash(self):
        block_string =
f"{self.index}{self.transactions}{self.timestamp}{self.previous_hash}{self.non
ce}"
        return sha256(block_string.encode()).hexdigest()

class Blockchain:
    def _init_(self, difficulty=4):
        self.chain = [self.create_genesis_block()]
        self.difficulty = difficulty
        self.pending_transactions = []

    def create_genesis_block(self):
        return Block(0, [], time(), "0")

    def get_latest_block(self):
        return self.chain[-1]

    def mine_block(self, miner_address):
        block = Block(
            len(self.chain),
            self.pending_transactions,
            time(),
            self.get_latest_block().hash
        )

        # Proof of Work
        while block.hash[:self.difficulty] != "0" * self.difficulty:
            block.nonce += 1
            block.hash = block.calculate_hash()

        self.chain.append(block)
        self.pending_transactions = []
```

```
        return block

    def is_chain_valid(self):
        for i in range(1, len(self.chain)):
            current_block = self.chain[i]
            previous_block = self.chain[i-1]

            # Verify hash
            if current_block.hash != current_block.calculate_hash():
                return False

            # Verify chain linkage
            if current_block.previous_hash != previous_block.hash:
                return False

        return True

def simulate_51_percent_attack(honest_chain, attacker_chain, num_blocks=5):
    """Simulate a 51% attack by creating a parallel chain with more mining
power"""
    print("\nSimulating 51% attack...")

    # Attacker mines blocks faster (simulating >51% hash power)
    attacker_difficulty = honest_chain.difficulty - 1

    # Create divergent chain
    for _ in range(num_blocks):
        # Honest network mines one block
        honest_chain.pending_transactions = [f"honest_tx_{_}"]
        honest_block = honest_chain.mine_block("honest_miner")
        print(f"Honest chain mined block {honest_block.hash[:10]}...")

        # Attacker mines two blocks (faster due to more hash power)
        attacker_chain.pending_transactions = [f"attacker_tx_{_}"]
        attacker_block = attacker_chain.mine_block("attacker")
        print(f"Attacker chain mined block {attacker_block.hash[:10]}...")

    print("\nChain lengths:")
    print(f"Honest chain length: {len(honest_chain.chain)}")
    print(f"Attacker chain length: {len(attacker_chain.chain)}")

    # Check if attack was successful (attacker chain longer)
    if len(attacker_chain.chain) > len(honest_chain.chain):
        print("\nATTACK DETECTED: Attacker chain is longer than honest
chain!")
        print("This indicates a potential 51% attack as the attacker was able
to create a longer valid chain.")
    else:
```

```
        print("\nNo 51% attack detected. Honest chain remains longest.")

def main():
    # Initialize blockchain
    honest_chain = Blockchain(difficulty=4)

    # Create attacker's chain (fork of honest chain)
    attacker_chain = Blockchain(difficulty=4)
    attacker_chain.chain = honest_chain.chain.copy()

    # Simulate attack
    simulate_51_percent_attack(honest_chain, attacker_chain)

if _name_ == "_main_":
    main()
```