INDEX

Sr No.	Practical Name	Date	Signature
1	**Understand the structure of a blockchain and create a simple block chain.** **● Implement a basic blockchain in Python or JavaScript.** **● Define the structure of a block (block number, timestamp, data, previous hash).** **● Implement proof-of-work to add blocks to the chain.** **● Tools: Python, JavaScript.**	**11/01/25**	
2	**Create a peer-to-peer (P2P) network and simulate transactions.** **● Implement a basic peer-to-peer system where nodes communicate directly.** **● Simulate decentralized transaction broadcasting between nodes.** **● Compare the behavior of centralized vs. decentralized systems.** **● Tools: Python (socket), JavaScript (WebSocket).**	**13/01/25**	
3	**Learn the fundamentals of asymmetric cryptography by implementing the RSA algorithm.** **● Write a program to generate RSA public and private keys.** **● Implement encryption and decryption of messages using RSA.** **● Tools: Python (PyCryptodome), JavaScript.**	**20/01/25**	
4	**Implement cryptographic primitives used in blockchain.** **● Build a hash chain where each hash depends on the previous hash.** **● Implement a Merkle Tree and use it to verify the integrity of Transactions.** **● Tools: Python (hashlib), JavaScript.**	**07/02/25**	
5	**Understand how digital signatures are used in blockchain for authentication.** **● Implement a digital signature scheme using**	**08/02/25**	

	public/private keys. **● Use the digital signature to sign and verify blockchain transactions.** **● Tools: Python (cryptography), JavaScript (node-rsa).**		
6	**Simulate the basic structure of Bitcoin blocks and the mining process.** **● Create a Bitcoin-like block structure.** **● Implement proof-of-work to simulate the mining process.** **● Write a script to calculate the hash of each block and validate the blockchain.** **● Tools: Python, JavaScript.**	**10/02/25**	
7	**Compare Proof of Work (PoW) and Proof of Stake (PoS) consensus mechanisms.** **● Implement a basic Proof of Work algorithm for blockchain mining.** **● Implement Proof of Stake to simulate how staking works in a blockchain.** **● Analyze the performance and energy efficiency of PoW and PoS.** **● Tools: Python, JavaScript.**	**15/02/25**	
8	**Write and deploy a basic smart contract using Ethereum and Solidity.** **● Write a smart contract in Solidity (e.g., a basic voting or token contract).** **● Compile and deploy the contract using Remix IDE.** **● Tools: Remix IDE, Solidity, Web3.js, MetaMask.**	**24/02/25**	
9	**Set up a Hyperledger Fabric network and simulate transaction flow.** **● Set up a local Hyperledger Fabric network.** **● Tools: Hyperledger Fabric, Docker, Node.js.**	**07/03/25**	
10	**Understand the concept of gas in Ethereum and optimize gas usage.** **● Write a smart contract with multiple functions.** **● Estimate gas usage for different transactions using Remix IDE.** **● Tools: Remix IDE, Solidity, MetaMask.**	**17/03/25**	

Practical No:-1

Aim:- Understand the structure of a blockchain and create a simple block chain.

- Implement a basic blockchain in Python or JavaScript.
- Define the structure of a block (block number, timestamp, data, previous hash).
- Implement proof-of-work to add blocks to the chain.
- Tools: Python, JavaScript.

● Implement a basic blockchain in Python or JavaScript.

Code:-

```
import hashlib, time
class Block:
    def __init__(self, index, prev_hash, timestamp, data, proof):
        self.index, self.previous_hash, self.timestamp, self.data, self.proof = index, prev_hash, timestamp, data, proof
                                                                    self.hash = hashlib.sha256(f"{index}{prev_hash}{timestamp}{data}{proof}".encode()).hexdigest()

class Blockchain:
    def __init__(self): self.chain = [Block(0, "0", time.time(), "Genesis Block", 0)]
    def add_block(self, data):
        prev_block, proof = self.chain[-1], self.proof_of_work(self.chain[-1].hash)
        self.chain.append(Block(prev_block.index + 1, prev_block.hash, time.time(), data, proof))
    def proof_of_work(self, last_hash):
        proof = 0
        while hashlib.sha256(f"{last_hash}{proof}".encode()).hexdigest()[:4] != "0000":
            proof += 1
        return proof

blockchain = Blockchain(); blockchain.add_block("Block 1 Data"); blockchain.add_block("Block 2 Data")
for block in blockchain.chain: print(f"Index: {block.index}, Hash: {block.hash}, Data: {block.data}")
```

Output:-

```
Index: 0, Hash: d95bc119bf83e33fd38f8693c6c17ae6b23d7f47075ed183ed82b0a123cd890e, Data: Genesis Block
Index: 1, Hash: e6a857c957650226febbcd8d4268fd7fa95f80d41028945a301829b57255bb77, Data: Block 1 Data
Index: 2, Hash: a0f5a7ac38b5c31e2467c9e55d6a6fb76c4677d0c31e14c15a648fdc6a0b403e, Data: Block 2 Data
```

● Define the structure of a block (block number, timestamp, data, previous hash).

Code:-

```
import hashlib
from datetime import datetime

class Block:
    def __init__(self, block_number, data, previous_hash):
        self.block_number, self.data, self.previous_hash = block_number, data, previous_hash
        self.timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
                                                                    self.hash                    =
hashlib.sha256(f"{block_number}{self.timestamp}{data}{previous_hash}".encode()).hexdig
est()

block = Block(1, "Genesis Block", "0")
print(f"Block    Number:    {block.block_number}\nTimestamp:    {block.timestamp}\nData:
{block.data}\nPrevious Hash: {block.previous_hash}\nHash: {block.hash}")
```

Output:-

```
Block Number: 1
Timestamp: 2025-03-19 09:18:47
Data: Genesis Block
Previous Hash: 0
Hash: a3fc74537bbae59aa7b0306060ed2c71cd2f36ced75f86468f11ea0923f4126e
```

● Implement proof-of-work to add blocks to the chain.

Code:-

```
import hashlib, time
def proof_of_work(data, previous_hash, difficulty):
    nonce = 0
    while True:
        hash_attempt = hashlib.sha256(f"{data}{previous_hash}{nonce}".encode()).hexdigest()
        if hash_attempt[:difficulty] == '0' * difficulty: return nonce, hash_attempt
        nonce += 1
difficulty = 4
previous_hash = "0" * 64  # Initial hash for the genesis block
data = "First Block"
nonce, hash = proof_of_work(data, previous_hash, difficulty)
print(f"Block 1: Nonce={nonce}, Hash={hash}")
previous_hash = hash  # Update previous_hash to current block's hash
data = "Second Block"
nonce, hash = proof_of_work(data, previous_hash, difficulty)
print(f"Block 2: Nonce={nonce}, Hash={hash}")
```

Output:-

```
Block 1: Nonce=84494, Hash=0000ec7e9bf2f6e7c641d55d953c8f611bc498a31f0bacdefaada28ed6f7fba0
Block 2: Nonce=132225, Hash=0000ed984902e9bdeb7b4820c50aab7097a3da59e8805ee1978cfc13797a1b76
```

Practical No:-2

Aim:- Create a peer-to-peer (P2P) network and simulate transactions.

- Implement a basic peer-to-peer system where nodes communicate directly.
- Simulate decentralized transaction broadcasting between nodes.
- Compare the behavior of centralized vs. decentralized systems.
- Tools: Python (socket), JavaScript (WebSocket).
- **Implement a basic peer-to-peer system where nodes communicate directly.**

Code:-

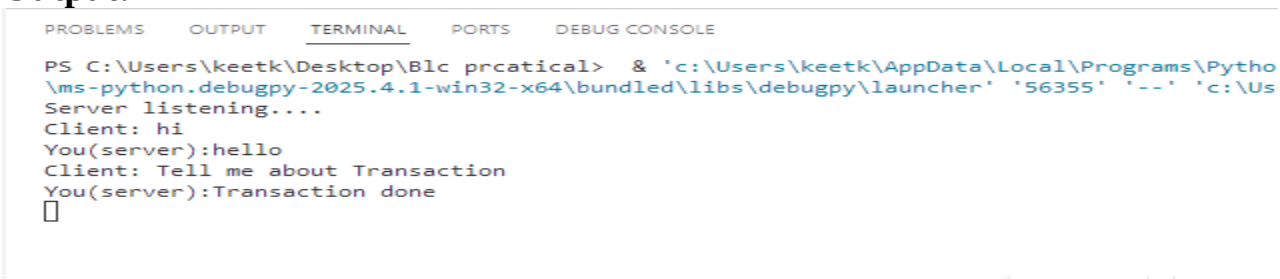
Srever.py

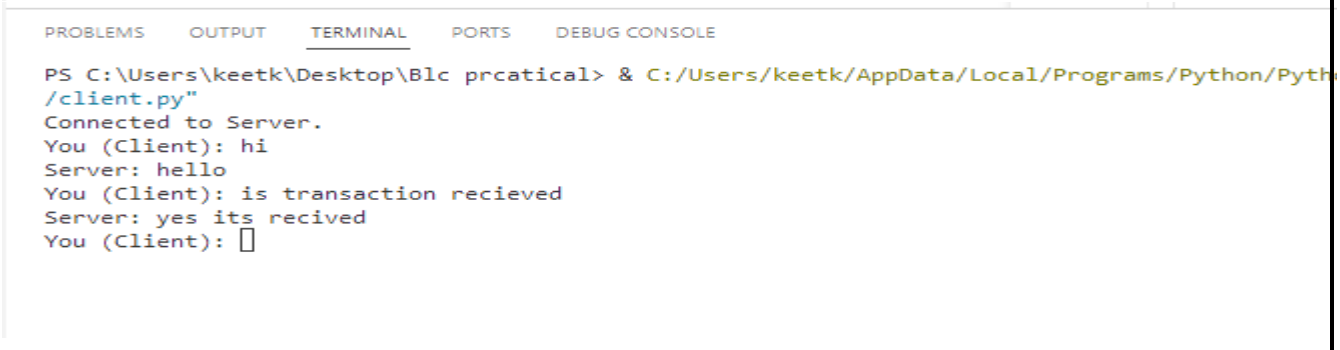
```
import socket
s = socket.socket(); s.bind(("localhost", 12345)); s.listen(1)
print("Server listening..."); conn, _ = s.accept()
while True:
    client_msg = conn.recv(1024).decode()
    if client_msg.lower() == "exit": break
    print("Client:", client_msg)
    conn.send(input("You (Server): ").encode())
conn.close()
```

Client.py

```
import socket
c = socket.socket(); c.connect(("localhost", 12345))
print("Connected to Server.")
while True:
    c.send(input("You (Client): ").encode())
    server_msg = c.recv(1024).decode()
    if server_msg.lower() == "exit": break
    print("Server:", server_msg)
c.close()
```

Output:-





● Simulate decentralized transaction broadcasting between nodes.

Code:-

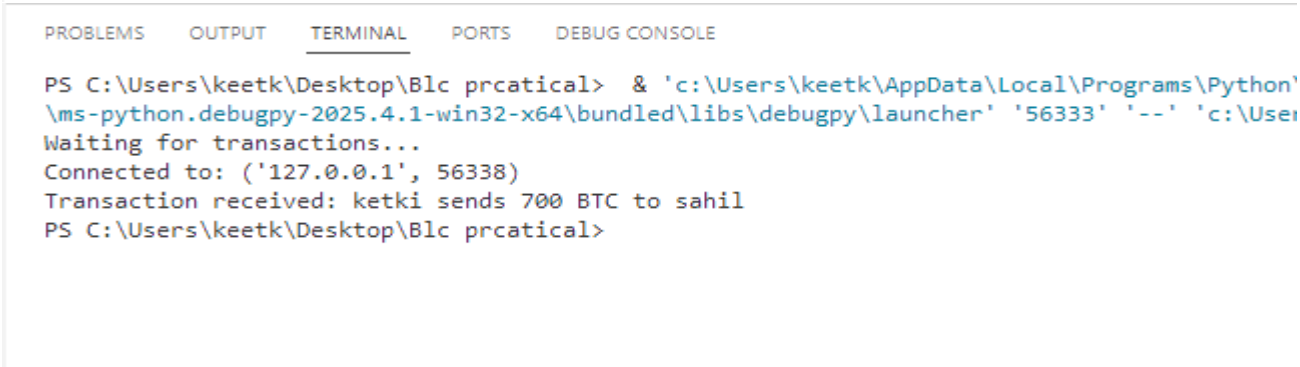
Node_reciver.py

```
import socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(("localhost", 5000))
server.listen(1)
print("Waiting for transactions...")
conn, addr = server.accept()
print(f"Connected to: {addr}")
transaction = conn.recv(1024).decode()
print("Transaction received:", transaction)
conn.close()
server.close()
```

Node_sender.py

```
import socket
sender = input("Enter sender name: ")
receiver = input("Enter receiver name: ")
amount = input("Enter the amount of BTC: ")
transaction = f"{sender} sends {amount} BTC to {receiver}"
try:
    node = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    node.connect(("localhost", 5000))
    node.send(transaction.encode())
    print("Transaction broadcasted:", transaction)
    node.close()
except ConnectionRefusedError:
    print("Connection failed: Is the receiver running?")
```

Output:-



● Compare the behavior of centralized vs. decentralized systems.

Centralized System (Client-Server)

Code:-

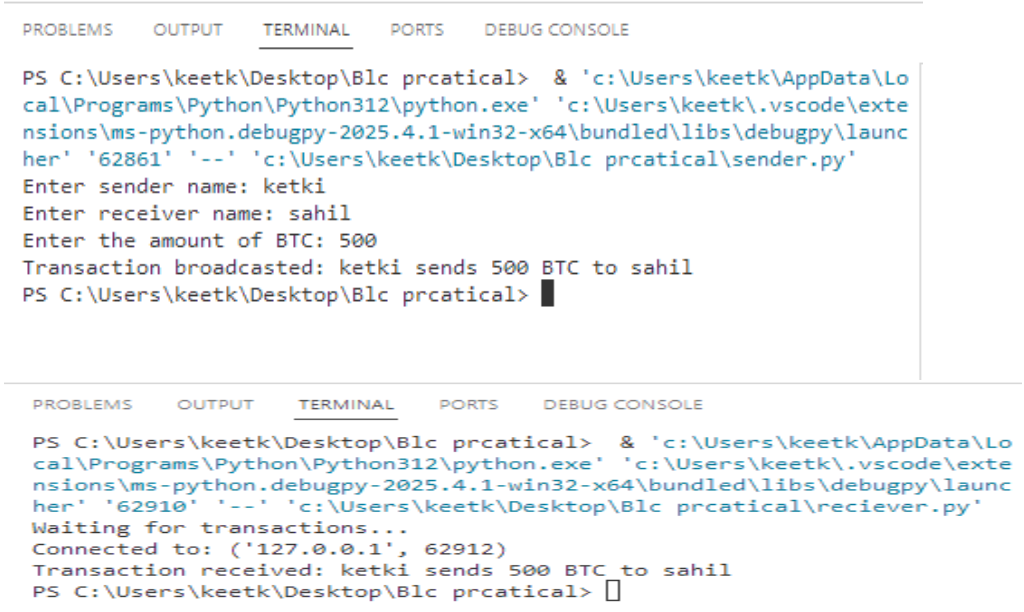
Sender.py

```
import socket
sender = input("Enter sender name: ")
receiver = input("Enter receiver name: ")
amount = input("Enter the amount of BTC: ")
transaction = f"{sender} sends {amount} BTC to {receiver}"
try:
    node = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    node.connect(("localhost", 8080))
    node.send(transaction.encode())
    print("Transaction broadcasted:", transaction)
    node.close()
except ConnectionRefusedError:
    print("Connection failed: Is the receiver running?")
```

reciver.py

```
import socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(("localhost", 8080))
server.listen(1)
print("Waiting for transactions...")
conn, addr = server.accept()
print(f"Connected to: {addr}")
transaction = conn.recv(1024).decode()
print("Transaction received:", transaction)
conn.close()
server.close()
```

Output:-



Decentralized System Output (Peer-to-Peer)

Code:-

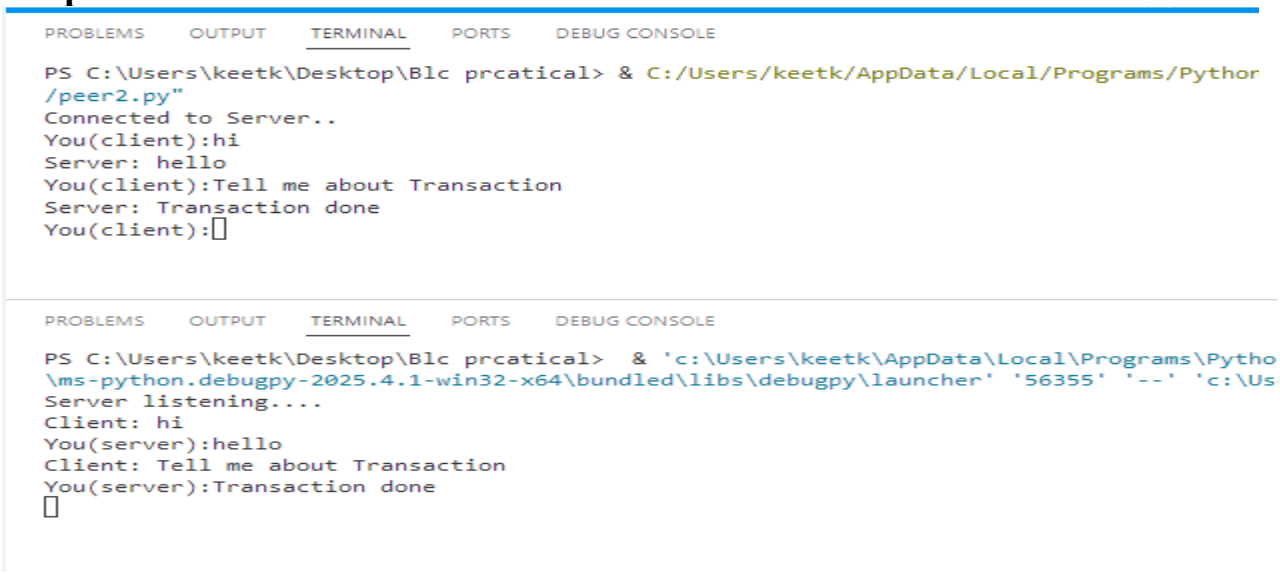
Peer1.py

```
import socket
s= socket.socket();s.bind(("localhost",12345)); s.listen(1)
print("Server listening...."); conn, _  = s.accept()
while True:
    client_msg=conn.recv(1024).decode()
    if client_msg.lower()=="exit":break
    print("Client:",client_msg)
    conn.send(input("You(server):").encode())
conn.close()
```

Peer2.py

```
import socket
c= socket.socket(); c.connect(("localhost",12345))
print("Connected to Server..")
while True:
    c.send(input("You(client):").encode())
    server_msg=c.recv(1024).decode()
    if server_msg.lower()=="exit":break
    print("Server:",server_msg)
c.close()
```

Output:-



Practical No:-3

Aim:- Learn the fundamentals of asymmetric cryptography by implementing the RSA algorithm.

- Write a program to generate RSA public and private keys.
- Implement encryption and decryption of messages using RSA.
- Tools: Python (PyCryptodome), JavaScript.

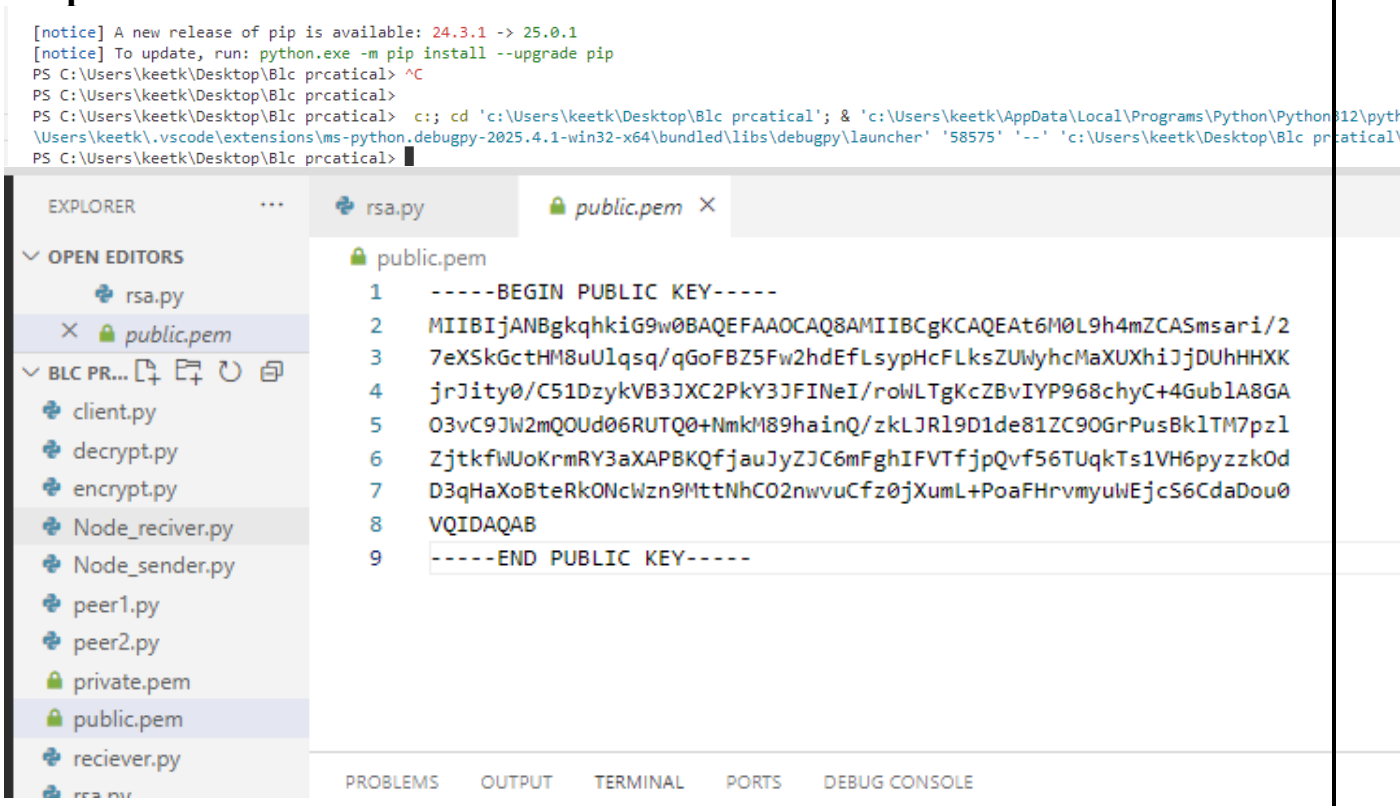
● Write a program to generate RSA public and private keys.

Code:-

rsa.py

```
!pip install pycryptodome
from Crypto.PublicKey import RSA
key = RSA.generate(2048)
private_key = key.export_key()
public_key = key.publickey().export_key()
with open("private.pem", "wb") as f:
    f.write(private_key)
with open("public.pem", "wb") as f:
    f.write(public_key)
```

Output:-



● Implement encryption and decryption of messages using RSA.

Code:-

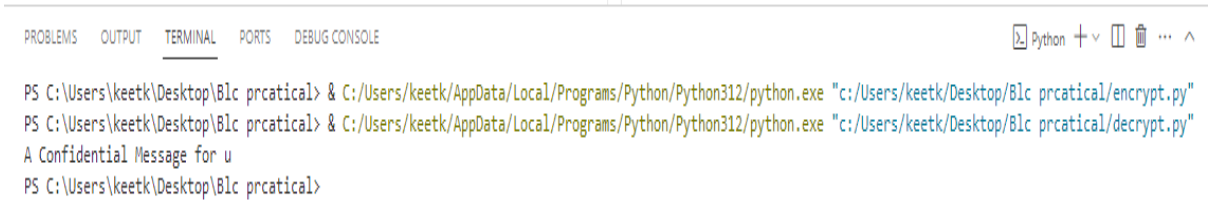
encrypt.py

```
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA
message = b"A Confidential Message for u"
public_key = RSA.import_key(open("public.pem").read())
cipher = PKCS1_OAEP.new(public_key)
encrypted = cipher.encrypt(message)
open("encrypted.bin", "wb").write(encrypted)
```

decrypt.py

```
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA
private_key = RSA.import_key(open("private.pem").read())
cipher = PKCS1_OAEP.new(private_key)
encrypted = open("encrypted.bin", "rb").read()
decrypted = cipher.decrypt(encrypted)
print(decrypted.decode())
```

Output:-



Practical No:-4

Aim:- Implement cryptographic primitives used in blockchain.

- Build a hash chain where each hash depends on the previous hash.
- Implement a Merkle Tree and use it to verify the integrity of transactions.
- Tools: Python (hashlib), JavaScript.

● Build a hash chain where each hash depends on the previous hash.

Code:-

```
import hashlib
chain = ["Genesis"]
for i in range(1, 6):
    prev_hash = hashlib.sha256(chain[-1].encode()).hexdigest()
    print(f"Block {i}:")
    print(f"Previous Hash: {chain[-1]}")
    print(f"New Block Hash: {prev_hash}\n")
    chain.append(prev_hash)
```

Output:-

```
Block 1:
Previous Hash: Genesis
New Block Hash: 81ddc8d248b2dccdd3fdd5e84f0cad62b08f2d10b57f9a831c13451e5c5c80a5

Block 2:
Previous Hash: 81ddc8d248b2dccdd3fdd5e84f0cad62b08f2d10b57f9a831c13451e5c5c80a5
New Block Hash: 854b6903a2723b374db0b6281df6f5cdaeb96557d3503407cd5b74256599c8d7

Block 3:
Previous Hash: 854b6903a2723b374db0b6281df6f5cdaeb96557d3503407cd5b74256599c8d7
New Block Hash: 80345e482d2413b0acb9c373fa4efa345e4ee8399b8661989ac4e34950e962d0

Block 4:
Previous Hash: 80345e482d2413b0acb9c373fa4efa345e4ee8399b8661989ac4e34950e962d0
New Block Hash: b998e6aad5bb5a0ee24fd5e80459d7782137ae282f9475314b6d91ce81720bfe

Block 5:
Previous Hash: b998e6aad5bb5a0ee24fd5e80459d7782137ae282f9475314b6d91ce81720bfe
New Block Hash: 24bb4a7a24d91250082ef2241de46db52f1276adc179dab54b69694474b4d485
```

● Implement a Merkle Tree and use it to verify the integrity of transactions.

Code:-

```
import hashlib
def hash_pair(a, b): return hashlib.sha256((a + b).encode()).hexdigest()
def merkle_tree(leaves):
    tree = [[hashlib.sha256(leaf.encode()).hexdigest() for leaf in leaves]]
    while len(tree[-1]) > 1:
        level = tree[-1] + [tree[-1][-1]] if len(tree[-1]) % 2 else tree[-1]
        tree.append([hash_pair(level[i], level[i+1]) for i in range(0, len(level), 2)])
    return tree
transactions = ["A", "B", "C", "D"]
tree = merkle_tree(transactions)
for i, level in enumerate(tree):
    print(f"Transaction Level {i}: {level}")
print(f"Merkle Root Hash: {tree[-1][0]}")
```

Output:-

```
Transaction Level 0: ['559aead08264d5795d3909718cdd05abd49572e84fe55590eef31a88a08fdffd', 'df7e70e5021544f4834bbee64a9e3789febc4be81470df629cad6ddb03320a5c', '6b23c
Transaction Level 1: ['b30ab174f7459cdd40a3acdf15d0c9444fec2adcfb9d579aa154c084885edd0a', '26b5aabe804fe5d533c663dea833e8078188376ce5ca2b5c3371d09ef6b0657b']
Transaction Level 2: ['50a504831bd50fee3581d287168a85a8dcdd6aa777ffd0fe35e37290268a0153']
Merkle Root Hash: 50a504831bd50fee3581d287168a85a8dcdd6aa777ffd0fe35e37290268a0153
```

Practical No:-5

Aim:- Understand how digital signatures are used in blockchain for authentication.

- Implement a digital signature scheme using public/private keys.
- Use the digital signature to sign and verify blockchain transactions.
- Tools: Python (cryptography), JavaScript (node-rsa).

● Implement a digital signature scheme using public/private keys.

Code:-

```
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes
private_key = rsa.generate_private_key(public_exponent=65537, key_size=2048)
public_key = private_key.public_key()
message = b"Blockchain Authentication"
signature = private_key.sign(message, padding.PKCS1v15(), hashes.SHA256())
try:
    public_key.verify(signature, message, padding.PKCS1v15(), hashes.SHA256())
    print("Signature Verified!")
except:
    print("Verification Failed!")
```

Output:-

```
Signature Verified!
```

● Use the digital signature to sign and verify blockchain transactions.

Code:-

```
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes
private_key = rsa.generate_private_key(public_exponent=65537, key_size=2048)
public_key = private_key.public_key()
transaction = b"User1 sends 10 BTC to User2"
signature = private_key.sign(transaction, padding.PKCS1v15(), hashes.SHA256())
try:
    public_key.verify(signature, transaction, padding.PKCS1v15(), hashes.SHA256())
    print("Transaction Verified!")
except:
    print("Verification Failed!")
```

Output:-

```
Transaction Verified!
```

Practical No:-6

Aim:- Simulate the basic structure of Bitcoin blocks and the mining process.

● Create a Bitcoin-like block structure.
● Implement proof-of-work to simulate the mining process.
● Write a script to calculate the hash of each block and validate the blockchain.
● Tools: Python, JavaScript.

● Create a Bitcoin-like block structure.

Code:-

```
import hashlib, time
class Block:
    def __init__(self, index, prev_hash, data):
        self.index = index
        self.prev_hash = prev_hash
        self.data = data
        self.timestamp = time.time()
        self.nonce = 0
        self.hash = self.calculate_hash()
    def calculate_hash(self):
        block_content = f"{self.index}{self.prev_hash}{self.data}{self.timestamp}{self.nonce}"
        return hashlib.sha256(block_content.encode()).hexdigest()
    def __str__(self):
        return (f"\n{'='*40}\n"
                f"Block Index  : {self.index}\n"
                f"Previous Hash: {self.prev_hash}\n"
                f"Data         : {self.data}\n"
                f"Timestamp    : {self.timestamp}\n"
                f"Nonce        : {self.nonce}\n"
                f"Current Hash : {self.hash}\n"
                f"{'='*40}")
genesis_block = Block(0, "0", "First Block")
block_1 = Block(1, genesis_block.hash, "Second Block")
print(block_1)
```

Output:-

```
========================================
Block Index  : 1
Previous Hash: a9df0251dff3f2a13f6d1a63ba0be4de9f24328b326f48202bcb112e208b2ccc
Data         : Second Block
Timestamp    : 1742409022.4727678
Nonce        : 0
Current Hash : 95efdaccc8896be3fa7a8fbac9e1c60b94c86db762ab11a0b8495983f438d64b
========================================
```

● Implement proof-of-work to simulate the mining process.

Code:-

```
import hashlib
def mine_block(index, prev_hash, data, difficulty=2):
    nonce = 0
    while True:
        block_data = f"{index}{prev_hash}{data}{nonce}"
        hash_val = hashlib.sha256(block_data.encode()).hexdigest()
        if hash_val[:difficulty] == "0" * difficulty:
```

```
            print(f"Block mined! Nonce: {nonce}, Hash: {hash_val}")
            return hash_val
        nonce += 1
mine_block(1, "0000000000000000000000000000000000", "Transaction Data", difficulty=4)
```

Output:-

```
Block mined! Nonce: 106123, Hash: 0000d86e75501697f54f99fd2c63673af231fee1ccf4aa10a34569bfd208cf96
'0000d86e75501697f54f99fd2c63673af231fee1ccf4aa10a34569bfd208cf96'
```

● Write a script to calculate the hash of each block and validate the blockchain.

Code:-

```
import hashlib
import time
class Blockchain:
    def __init__(self):
        self.chain = [{'index': 0, 'previous_hash': '0', 'timestamp': time.time(), 'data': 'Genesis Block', 'hash': '0'}]
    def add_block(self, data):
        previous_block = self.chain[-1]
        index = previous_block['index'] + 1
        timestamp = time.time()
        block = {'index': index, 'previous_hash': previous_block['hash'], 'timestamp': timestamp, 'data': data, 'hash': ''}
        block['hash'] = hashlib.sha256(f"{block['index']}{block['previous_hash']}{block['timestamp']}{block['data']}".encode()).hexdigest()
        self.chain.append(block)

    def validate_chain(self):
        return all(block['previous_hash'] == self.chain[i-1]['hash'] and block['hash'] == hashlib.sha256(f"{block['index']}{block['previous_hash']}{block['timestamp']}{block['data']}".encode()).hexdigest() for i, block in enumerate(self.chain[1:], 1))

    def print_chain(self):
        for block in self.chain:
            print(f'{{"data": "{block["data"]}", "hash": "{block["hash"]}", "index": {block["index"]}, "previous_hash": "{block["previous_hash"]}", "timestamp": {block["timestamp"]}}}')
bc = Blockchain()
bc.add_block("Transaction 1")
bc.add_block("Transaction 2")
print("Blockchain valid:", bc.validate_chain())
bc.print_chain()
```

Output:-

```
Blockchain valid: True
{"data": "Genesis Block", "hash": "0", "index": 0, "previous_hash": "0", "timestamp": 1742409296.108251}
{"data": "Transaction 1", "hash": "f58925d6221e947f86a2ebaf4c3003aa536a5fabe2427638272d441d86413bde", "index": 1, "previous_hash": "0", "timestamp": 1742409296.1083
{"data": "Transaction 2", "hash": "910c76833d828aba4f2719107b65b748a10027618cffa99ead12df9045a13c6c", "index": 2, "previous_hash": "f58925d6221e947f86a2ebaf4c3003aa
```

Practical No:-7

Aim:- Compare Proof of Work (PoW) and Proof of Stake (PoS) consensus mechanisms.

- Implement a basic Proof of Work algorithm for blockchain mining.
- Implement Proof of Stake to simulate how staking works in a blockchain.
- Analyze the performance and energy efficiency of PoW and PoS.
- Tools: Python, JavaScript.

● Implement a basic Proof of Work algorithm for blockchain mining.

Code:-

POW.py

```
import hashlib
import time
def proof_of_work(last_proof):
    proof = 0
    while not valid_proof(last_proof, proof):
        proof += 1
    return proof
def valid_proof(last_proof, proof):
    guess = f'{last_proof}{proof}'.encode()
    guess_hash = hashlib.sha256(guess).hexdigest()
    return guess_hash[:5] == "00000"  # Increased difficulty level
if __name__ == "__main__":
    t1 = time.time()
    proof = proof_of_work(0)
    t2 = time.time()
    print(f"PoW Time Taken: {t2 - t1:.4f} seconds")
```

Output:-

```
PoW Time Taken: 0.7915 seconds
```

● Implement Proof of Stake to simulate how staking works in a blockchain.

Code:-

POS.py

```
import random
import time
def proof_of_stake(stakes):
    total_stake = sum(stakes.values())
    rnd = random.uniform(0, total_stake)
    cumulative = 0
    for validator, stake in stakes.items():
        cumulative += stake
        if cumulative >= rnd:
            return validator
if __name__ == "__main__":
    stakes = {"Alice": 70, "Bob": 30, "Charlie": 20}  # Example stakes
    t1 = time.time()
```

```
    time.sleep(0.1)  # Added small delay to simulate processing time
    winner = proof_of_stake(stakes)
    t2 = time.time()
    print(f"PoS Time Taken: {t2 - t1:.8f} seconds, Winner: {winner}")
```

Output:-

```
PoS Time Taken: 0.10014629 seconds, Winner: Alice
```

- **Analyze the performance and energy efficiency of PoW and PoS.**

Code:-

Analysis.py

```
import time
stakes = {"Alice": 50, "Bob": 30, "Charlie": 20}
t1 = time.time()
proof = proof_of_work(0)  # Call the function directly
t2 = time.time()
pow_time = t2 - t1
t1 = time.time()
time.sleep(0.1)  # Added delay to simulate processing
winner = proof_of_stake(stakes)  # Call the function directly
t2 = time.time()
pos_time = t2 - t1
print(f"PoW Time Taken: {pow_time:.4f} seconds")
print(f"PoS Time Taken: {pos_time:.8f} seconds, Winner: {winner}")
if pos_time > 0:
    print(f"PoS is ~{pow_time / pos_time:.2f} times more efficient than PoW")
else:
    print("PoS executed too quickly to compare efficiency reliably.")
```

Output:-

```
PoW Time Taken: 0.7767 seconds
PoS Time Taken: 0.10034132 seconds, Winner: Charlie
PoS is ~7.74 times more efficient than PoW
```

Practical No:-8

Aim:- Write and deploy a basic smart contract using Ethereum and Solidity.

- Write a smart contract in Solidity (e.g., a basic voting or token contract).
- Compile and deploy the contract using Remix IDE.
- Tools: Remix IDE, Solidity, Web3.js, MetaMask.

Code:-

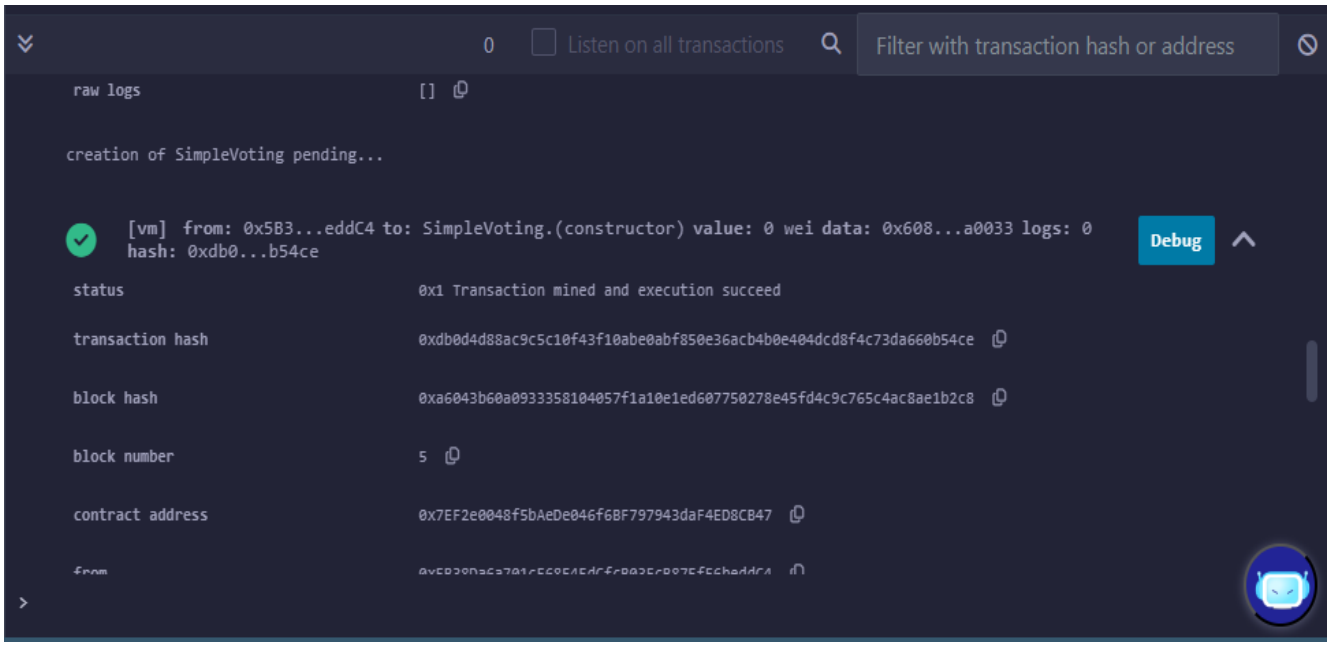
 # Solidity Code:

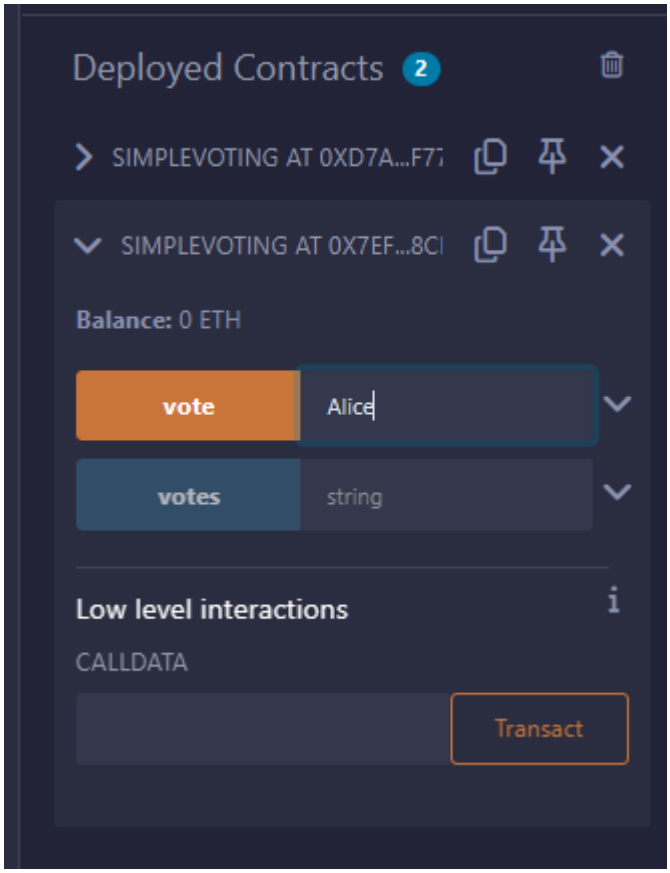
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract SimpleVoting {
    mapping(string => uint) public votes;

    function vote(string memory candidate) public {
        require(
            keccak256(abi.encodePacked(candidate)) == keccak256(abi.encodePacked("Alice")) ||
            keccak256(abi.encodePacked(candidate)) == keccak256(abi.encodePacked("Bob")),
            "Candidate must be Alice or Bob"
        );
        votes[candidate]++;
    }
}
```

Output:-




Deployed Contracts 2
SIMPLEVOTING AT 0XD7A...F77
SIMPLEVOTING AT 0X7EF...8CI
Balance: 0 ETH
vote
Alice
votes
string
Low level interactions
CALLDATA
Transact

Practical No:-9

Aim:- Set up a Hyperledger Fabric network and simulate transaction flow.

- Set up a local Hyperledger Fabric network.
- Tools: Hyperledger Fabric, Docker, Node.js.

Code:-

Set up a Hyperledger Fabric network and simulate transaction flow.

● Set up a local Hyperledger Fabric network.

Step 1: Open PowerShell and Enable WSL

1. Open **PowerShell** as Administrator.
2. Run the following command to enable WSL (Windows Subsystem for Linux):

Command:-wsl –install

```
PS C:\Users\keetk> wsl --install
Installing Windows optional component: VirtualMachinePlatform

Deployment Image Servicing and Management tool
Version: 10.0.22621.2792

Image Version: 10.0.22631.5039

Enabling feature(s)
[==========================100.0%==========================]
The operation completed successfully.
The requested operation is successful. Changes will not be effective until the system is rebooted.
PS C:\Users\keetk>
```

If WSL is already installed, update it:

Command:-wsl –update

```
PS C:\Users\keetk> wsl --update
Checking for updates.
The most recent version of Windows Subsystem for Linux is already installed.
PS C:\Users\keetk>


PS C:\Users\keetk> wsl --install
Downloading: Ubuntu
Installing: Ubuntu
A distribution with the supplied name already exists. Use --name to chose a different name.
Error code: Wsl/InstallDistro/Service/RegisterDistro/ERROR_ALREADY_EXISTS
PS C:\Users\keetk> Use --name
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\keetk> wsl --install
Downloading: Ubuntu
Installing: Ubuntu
Distribution successfully installed. It can be launched via 'wsl.exe -d Ubuntu'
PS C:\Users\keetk> wsl.exe -d Ubuntu
Provisioning the new WSL instance Ubuntu
This might take a while...
Create a default Unix user account: ketki_kumbhar
New password:
Retype new password:
passwd: password updated successfully
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 5.15.167.4-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Fri Mar 21 07:29:14 UTC 2025

  System load:  0.77               Processes:             31
  Usage of /:   0.1% of 1006.85GB  Users logged in:       0
  Memory usage: 6%                 IPv4 address for eth0: 172.20.42.10
  Swap usage:   0%

This message is shown once a day. To disable it please create the
/home/ketki_kumbhar/.hushlogin file.
ketki_kumbhar@LAPTOP-FSHPDUHB:/mnt/c/Users/keetk$
```

1. Restart your computer if prompted.

Step 2: Install Ubuntu on Windows

1. Open **Microsoft Store** and search for **Ubuntu**.
2. Click **Get** and install the latest version.
3. Once installed, open **Ubuntu** from the Start menu.
4. Set up a new UNIX username and password.

Step 3: Install Required Dependencies in Ubuntu

Open **Ubuntu Terminal** and install the following packages:

Command:-sudo apt update && sudo apt upgrade -y

```
ketki_kumbhar@LAPTOP-FSHPDUHB:/mnt/c/Users/keetk$ sudo apt update && sudo apt upgrade -y
[sudo] password for ketki_kumbhar:
Hit:1 http://archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:4 http://archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:5 http://archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
Get:6 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [671 kB]
Get:7 http://security.ubuntu.com/ubuntu noble-security/main Translation-en [130 kB]
Get:8 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [8960 B]
19% [5 Packages 2044 kB/15.0 MB 14%] [8 Components-amd64 5485 B/8960 B 61%]
```

Command:-sudo apt install curl git docker.io docker-compose -y

Command:-sudo apt install nodejs npm -y

Verify installations:

Command:-docker --version

Command:-docker-compose --version

Command:-node -v

Command:-npm -v

Step 4: Download Hyperledger Fabric Samples

1. Navigate to your home directory:

Command:-cd ~

2. Clone the Fabric Samples repository:

Command:-git clone https://github.com/hyperledger/fabric-samples.git

3. Move into the **fabric-samples** directory:

Command:-cd fabric-samples

Step 5: Install Hyperledger Fabric Binaries

Run the following script to download Fabric binaries and Docker images:

Command:-curl -sSL https://bit.ly/2ysbOFE | bash -s

Verify installation:

Command:-ls bin

If you see files like cryptogen, configtxgen, etc., the installation is successful.

Step 6: Set Up Environment Variables

Command:-echo 'export PATH=${PWD}/bin:$PATH' >> ~/.bashrc

Command:-echo 'export FABRIC_CFG_PATH=${PWD}/config' >> ~/.bashrc

Command:-source ~/.bashrc

Verify:

Command:-echo $PATH

Command:-echo $FABRIC_CFG_PATH

Step 7: Start the Test Network

Navigate to the **test-network** directory:

Command:-cd ~/fabric-samples/test-network

Start the network:

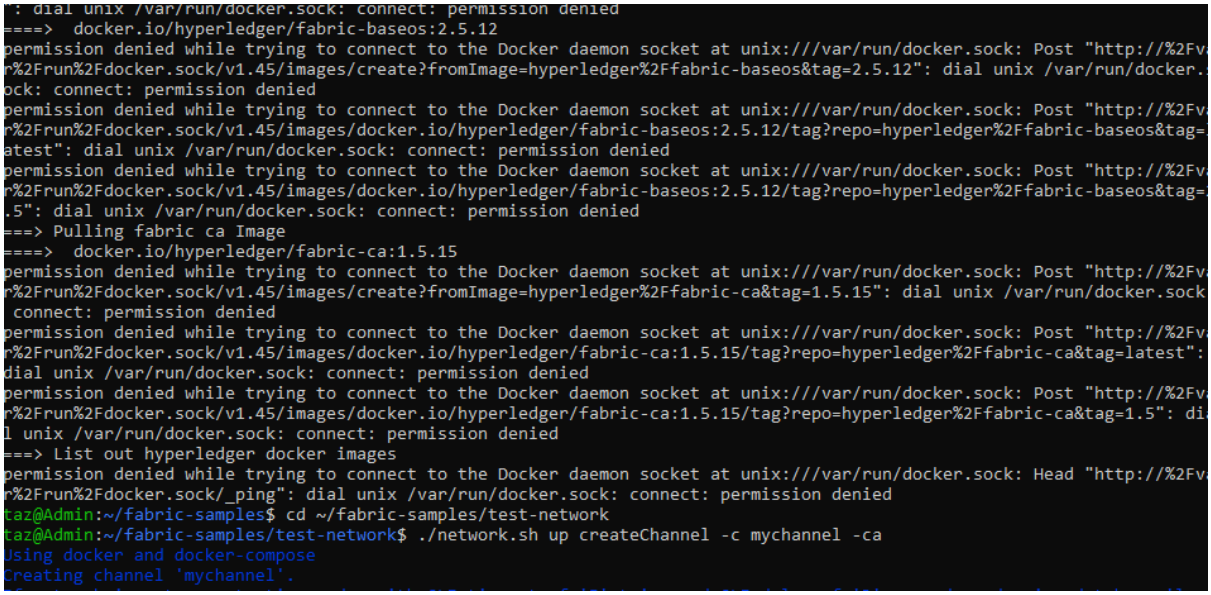
Command:-./network.sh up createChannel -c mychannel -ca

If you see an error about jq not found, install it:

Command:-sudo apt install jq -y

Then rerun:

Command:-./network.sh up createChannel -c mychannel -ca


': dial unix /var/run/docker.sock: connect: permission denied
====> docker.io/hyperledger/fabric-baseos:2.5.12
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fv
r%2Frun%2Fdocker.sock/v1.45/images/create?fromImage=hyperledger%2Ffabric-baseos&tag=2.5.12": dial unix /var/run/docker.
ock: connect: permission denied
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fv
r%2Frun%2Fdocker.sock/v1.45/images/docker.io/hyperledger/fabric-baseos:2.5.12/tag?repo=hyperledger%2Ffabric-baseos&tag=
atest": dial unix /var/run/docker.sock: connect: permission denied
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fv
r%2Frun%2Fdocker.sock/v1.45/images/docker.io/hyperledger/fabric-baseos:2.5.12/tag?repo=hyperledger%2Ffabric-baseos&tag=
.5": dial unix /var/run/docker.sock: connect: permission denied
===> Pulling fabric ca Image
====> docker.io/hyperledger/fabric-ca:1.5.15
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fv
r%2Frun%2Fdocker.sock/v1.45/images/create?fromImage=hyperledger%2Ffabric-ca&tag=1.5.15": dial unix /var/run/docker.sock
 connect: permission denied
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fv
r%2Frun%2Fdocker.sock/v1.45/images/docker.io/hyperledger/fabric-ca:1.5.15/tag?repo=hyperledger%2Ffabric-ca&tag=latest":
dial unix /var/run/docker.sock: connect: permission denied
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fv
r%2Frun%2Fdocker.sock/v1.45/images/docker.io/hyperledger/fabric-ca:1.5.15/tag?repo=hyperledger%2Ffabric-ca&tag=1.5": di
l unix /var/run/docker.sock: connect: permission denied
===> List out hyperledger docker images
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Head "http://%2Fv
r%2Frun%2Fdocker.sock/_ping": dial unix /var/run/docker.sock: connect: permission denied
taz@Admin:~/fabric-samples$ cd ~/fabric-samples/test-network
taz@Admin:~/fabric-samples/test-network$ ./network.sh up createChannel -c mychannel -ca
Using docker and docker-compose
Creating channel 'mychannel'.

Practical No:-10

Aim:- Understand the concept of gas in Ethereum and optimize gas usage.

- Write a smart contract with multiple functions.
- Estimate gas usage for different transactions using Remix IDE.
- Tools: Remix IDE, Solidity, MetaMask.

Code:-

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract MultiFunctionVoting {
    mapping(string => uint) public votes;
    string[] public candidates;
    address public owner;

    constructor() {
        owner = msg.sender; // Set contract deployer as owner
        candidates.push("Alice");
        candidates.push("Bob");
    }

    // Function to cast a vote
    function vote(string memory _candidate) public {
        require(isValidCandidate(_candidate), "Invalid candidate");
        votes[_candidate]++;
    }

    // Function to get total votes of a candidate
    function getVotes(string memory _candidate) public view returns (uint) {
        require(isValidCandidate(_candidate), "Invalid candidate");
        return votes[_candidate];
    }

    // Function to get the list of all candidates
    function getCandidates() public view returns (string[] memory) {
        return candidates;
    }

    // Function to reset all votes (only owner can call this)
    function resetVotes() public onlyOwner {
        for (uint i = 0; i < candidates.length; i++) {
            votes[candidates[i]] = 0;
        }
    }

    // Modifier to restrict access to the owner
    modifier onlyOwner() {
        require(msg.sender == owner, "Only owner can perform this action");
        _;
    }

    // Private function to check if a candidate is valid
    function isValidCandidate(string memory _candidate) private view returns (bool) {
```

```
        for (uint i = 0; i < candidates.length; i++) {
            if (keccak256(bytes(_candidate)) == keccak256(bytes(candidates[i]))) {
                return true;
            }
        }
        return false;
    }
}
```

Output:-