INDEX

PRACTICAL 01

PRACTICAL NO 1(A)

AIM: Develop a secure messaging application where users can exchange messages securely using RSA encryption. Implement a mechanism for generating RSA key pairs and encrypting/decrypting messages.

CODE:

```
# Install required library
pip install pycryptodome


# Import libraries
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import base64


# Function to generate RSA key pair
def generate_keys():
    key = RSA.generate(2048)
    private_key = key.export_key()
    public_key = key.publickey().export_key()
    return private_key, public_key


# Function to encrypt a message using public key
def encrypt_message(message, public_key):
    rsa_key = RSA.import_key(public_key)
    cipher = PKCS1_OAEP.new(rsa_key)
    encrypted_msg = cipher.encrypt(message.encode())
    return base64.b64encode(encrypted_msg).decode()


# Function to decrypt a message using private key
def decrypt_message(encrypted_message, private_key):
```

```
    rsa_key = RSA.import_key(private_key)
    cipher = PKCS1_OAEP.new(rsa_key)
    decoded_encrypted_msg = base64.b64decode(encrypted_message.encode())
    decrypted_msg = cipher.decrypt(decoded_encrypted_msg)
    return decrypted_msg.decode()

# Generate RSA keys for User A and User B
private_key_A, public_key_A = generate_keys()
private_key_B, public_key_B = generate_keys()
print("RSA keys generated for both users.\n")

# User A sends message to B
message_from_A = "Hello B, this is a secure message from A!"
encrypted_for_B = encrypt_message(message_from_A, public_key_B)
print("Encrypted message sent from A to B:")
print(encrypted_for_B)

# User B decrypts message from A
decrypted_by_B = decrypt_message(encrypted_for_B, private_key_B)
print("\nDecrypted message by B:")
print(decrypted_by_B)

# User B sends message to A
message_from_B = "Hi A, I received your secure message!"
encrypted_for_A = encrypt_message(message_from_B, public_key_A)
print("\nEncrypted message sent from B to A:")
print(encrypted_for_A)

# User A decrypts message from B
decrypted_by_A = decrypt_message(encrypted_for_A, private_key_A)
```

```
print("\nDecrypted message by A:")
print(decrypted_by_A)
```

OUTPUT:

```
In [1]: runfile('D:/Jyotika/Sem 4/BC Practicals/1a.py', wdir='D:/Jyotika/Sem 4/BC Practicals')
RSA keys generated for both users.

Encrypted message sent from A to B:
WhwjyMZC+xz44U/hocogn3+vp/KzjkZR6RYRjcggtOBocJ/
cjYXuqxT5j7HNrKWc+E8jwFQyPNIDTRkKmLnwNhqA7a3qzK97gggcSZGtwV4qHrrg7czAuSvxx8FmbLxKjE0uWBe6zGow8jRgvqN4F7k85rxzGpNHfgncyVC26xHiZY+83zIH0b
iIxX50oEouN0RsbXvQEextP86Msd2wNjJ6YWCrluM2gdJ6bDV+rZM//q2uw3buLTp98sMpy5vU4lklyt2u4Que/
tv749+SW+vnYXdCyZz5xU+Xn9gbYiSq78gFlMrTxUSQzYfxWweyxbPl/m4U9p6iTFUdhK+PtQ==

Decrypted message by B:
Hello B, this is a secure message from A!

Encrypted message sent from B to A:
SCLwxyt+d8t0f2U/FJCg/p308Zla80pJWmChelqVNdl6C1/mhdBCVU+zWTqWyhoJP3W+/fa1AdLzqFngUEjaJ66gNY++IW41Gz/
kCJBeGihAkSnTfr4G+QL+AxHAjKhbxverTuy5tysDZFKUrk+s/0t51lnzA4eBRXwhED9bdAn9L5YL7NlG5JCcwaVSjsqMSZbWTkGECnIMJFdrfb3Ec+U6h1ISLMi/
lAa+hQ86RUm83cUSh9LHbgBh+5iJnrLN4dGMPzxp0O2cATGdIWVxjmQLQSAczKIe+/JvA6trQrBk3/ofz04UfUJuycJkfsnefzIlBuUNf7zYoSZgY0C3eQ==

Decrypted message by A:
Hi A, I received your secure message!
```

PRACTICAL NO (1B)

AIM: Allow users to create multiple transactions and display them in an organised format

CODE:

```
# Install required library
pip install pycryptodome


# Import libraries
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import base64
import datetime


# RSA Key Generation
def generate_keys():
    key = RSA.generate(2048)
    private_key = key.export_key()
    public_key = key.publickey().export_key()
    return private_key, public_key


# Encrypt message with recipient's public key
def encrypt_message(message, public_key):
    rsa_key = RSA.import_key(public_key)
    cipher = PKCS1_OAEP.new(rsa_key)
    encrypted = cipher.encrypt(message.encode())
    return base64.b64encode(encrypted).decode()


# Decrypt message with recipient's private key
def decrypt_message(encrypted_msg, private_key):
    rsa_key = RSA.import_key(private_key)
```

```
    cipher = PKCS1_OAEP.new(rsa_key)
    decrypted = cipher.decrypt(base64.b64decode(encrypted_msg.encode()))
    return decrypted.decode()

# Generate RSA keys for User A and User B
private_key_A, public_key_A = generate_keys()
private_key_B, public_key_B = generate_keys()

# Store chat messages (transactions)
chat_log = []

# Function to simulate sending a message
def send_message(sender, receiver, msg, sender_priv, receiver_pub):
    encrypted = encrypt_message(msg, receiver_pub)

    # Select correct private key for receiver
    receiver_private_key = private_key_A if receiver == 'A' else private_key_B
    decrypted = decrypt_message(encrypted, receiver_private_key)

    chat_log.append({
        "timestamp": datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
        "sender": sender,
        "receiver": receiver,
        "original": msg,
        "encrypted": encrypted,
        "decrypted": decrypted
    })

# Simulate multiple secure transactions
send_message('A', 'B', "Hello B, this is A!", private_key_A, public_key_B)
```

```
send_message('B', 'A', "Hi A, message received!", private_key_B, public_key_A)
send_message('A', 'B', "Let's meet at 5 PM.", private_key_A, public_key_B)
send_message('B', 'A', "Sure! See you then.", private_key_B, public_key_A)

# Display all messages in an organized format
print("\nSecure Chat Log:\n")

for msg in chat_log:
    print(f"[{msg['timestamp']}] {msg['sender']} ➜ {msg['receiver']}")
    print(f"Original : {msg['original']}")
    print(f"Encrypted: {msg['encrypted'][:60]}...")
    print(f"Decrypted: {msg['decrypted']}")
    print("-" * 70)
```

OUTPUT:

```
In [1]: runfile('D:/Jyotika/Sem 4/BC Practicals/1b.py', wdir='D:/Jyotika/Sem 4/BC Practicals')

Secure Chat Log:

[2026-01-21 11:25:48] A ➜ B
Original : Hello B, this is A!
Encrypted: x0mRR+mxUggRU/t1SsF+qSJfDhMh7lUIaB0Mx44LDZRPg3iiLui7FvVjFi4C...
Decrypted: Hello B, this is A!
----------------------------------------------------------------------
[2026-01-21 11:25:48] B ➜ A
Original : Hi A, message received!
Encrypted: P1m4h3YBWAxZqzGAcYwhmk334sJFvn5BtIo2yh8cz3dOewSlNYoowBJy79jl...
Decrypted: Hi A, message received!
----------------------------------------------------------------------
[2026-01-21 11:25:48] A ➜ B
Original : Let's meet at 5 PM.
Encrypted: ut0/vpqoY075LhIKl0DVBQLO1Vs6TS4X1YPvD74RKUl4LhtVNRGpuaOV1xvZ...
Decrypted: Let's meet at 5 PM.
----------------------------------------------------------------------
[2026-01-21 11:25:48] B ➜ A
Original : Sure! See you then.
Encrypted: IMVfxpRJTouSY18+ER3u9fv+KnKAwbwViXotlRPmHhUOHDAKmuS8Yv5yqjT/...
Decrypted: Sure! See you then.
----------------------------------------------------------------------
```

PRACTICAL NO (1C)

AIM: Create a Python class named Transaction with attributes for sender, receiver, and amount. Implement a method within the class to transfer money from the sender's account to the receiver's account.

CODE:

```
# Install required library
pip install pycryptodome


# Import libraries
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import base64


# Function to generate RSA key pair
def generate_keys():
    key = RSA.generate(2048)
    private_key = key.export_key()
    public_key = key.publickey().export_key()
    return private_key, public_key


# Function to encrypt a message using public key
def encrypt_message(message, public_key):
    rsa_key = RSA.import_key(public_key)
    cipher = PKCS1_OAEP.new(rsa_key)
    encrypted_msg = cipher.encrypt(message.encode())
    return base64.b64encode(encrypted_msg).decode()


# Function to decrypt a message using private key
def decrypt_message(encrypted_message, private_key):
    rsa_key = RSA.import_key(private_key)
```

```
    cipher = PKCS1_OAEP.new(rsa_key)
    decoded_encrypted_msg = base64.b64decode(encrypted_message.encode())
    decrypted_msg = cipher.decrypt(decoded_encrypted_msg)
    return decrypted_msg.decode()

# Generate RSA keys for User A and User B
private_key_A, public_key_A = generate_keys()
private_key_B, public_key_B = generate_keys()

print("RSA keys generated for both users.\n")

# User A sends message to B
message_from_A = "Hello B, this is a secure message from A!"
encrypted_for_B = encrypt_message(message_from_A, public_key_B)
print("Encrypted message sent from A to B:")
print(encrypted_for_B)

# User B decrypts message from A
decrypted_by_B = decrypt_message(encrypted_for_B, private_key_B)
print("\nDecrypted message by B:")
print(decrypted_by_B)

# User B sends message to A
message_from_B = "Hi A, I received your secure message!"
encrypted_for_A = encrypt_message(message_from_B, public_key_A)
print("\nEncrypted message sent from B to A:")
print(encrypted_for_A)

# User A decrypts message from B
decrypted_by_A = decrypt_message(encrypted_for_A, private_key_A)
```

```
print("\nDecrypted message by A:")
print(decrypted_by_A)
```

OUTPUT:

```
In [1]: runfile('D:/Jyotika/Sem 4/BC Practicals/1c.py', wdir='D:/Jyotika/Sem 4/BC Practicals')
RSA keys generated for both users.

Encrypted message sent from A to B:
hmVTwcu96z24pZOtNkC0V0cZ8Yy7/o6On4gKAa6fsVwUY0qAl5DE9D9OD57jxpCjJxjP4MdBSfuZwrL/
ZmPAnVezdbQYSJkgtZqr8cxPvs3aJelx1jEHCwLwgkHzDrgIGUzDJc726Ps1uI6xFEnLRjGIH8P93cTtE48aYRl2jlvHRGK3LbVbg8sJYWbV/
SGoS8eOEjOWWYikQ0sABaCWeS1WRrnLkDyLaVrlzkg4HdgmovxdX6peEhoOk4dGmhog5oSQMtgZCmH9GJe3XdSpuRM9sZ3L733m1DRbM81EFMNieHE/
zeCmsuwuJJ+lF6fQ5jEEjyA8h/TZFVWDgunV9w==

Decrypted message by B:
Hello B, this is a secure message from A!

Encrypted message sent from B to A:
D05UEq6KDbGt5oCEI55D7Z1XRsGamxC6OO3LvMwH0kX7SfFj94Xrm4FhQCNDzY0TY0iVQYusbIWD0FOulTycSLNLWiAvYn4sw7N6r96Ja+3wyRVMLE7KW8giB7N3GMa/
CzFUYfzCia54Q3Px0Fo+A392fEFxtdGYc9y/
oe7lUgFSa+gX9kLdSrQOXAfEFkttTReu+mM+sBEJtdPs1iZAEgq6zd727Zx1prg9oP117ZnMKEoloZ4xMUPvMxb0qimzZDclRzMp2zdldZJYoUEVQZpPxWzlSm5MwHEUcrboSrG
MWEMLneU2xNqnMtgZNR7j5hKM7it/vozjCxM39sIDzw==

Decrypted message by A:
Hi A, I received your secure message!
```

PRACTICAL NO (1D)

AIM: Implement a function to add new blocks to the miner and dump the blockchain.

CODE:

```
class Account:
    def __init__(self, name, balance):
        self.name = name
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
            return True
        else:
            return False

    def __str__(self):
        return f"{self.name} - Balance: ₹{self.balance}"


class Transaction:
    def __init__(self, sender: Account, receiver: Account, amount: float):
        self.sender = sender
        self.receiver = receiver
        self.amount = amount

    def process(self):
```

```
    print(f"Processing transaction of ₹{self.amount} from {self.sender.name} to
{self.receiver.name}")

    if self.sender.withdraw(self.amount):
        self.receiver.deposit(self.amount)
        print("Transaction successful!")
    else:
        print("Transaction failed: Insufficient balance.")



# Example usage

# Create two accounts
user_A = Account("Priya", 1000)
user_B = Account("Rahul", 500)

# Display initial balances
print("\nInitial Balances:")
print(user_A)
print(user_B)

# Create and process a transaction
txn = Transaction(sender=user_A, receiver=user_B, amount=300)
txn.process()

# Display updated balances
print("\nUpdated Balances:")
print(user_A)
print(user_B)
```

OUTPUT:

```
In [2]: runfile('D:/Jyotika/Sem 4/BC Practicals/1d.py', wdir='D:/Jyotika/Sem 4/BC Practicals')

Initial Balances:
Priya - Balance: ₹1000
Rahul - Balance: ₹500
Processing transaction of ₹300 from Priya to Rahul
Transaction successful!

Updated Balances:
Priya - Balance: ₹700
Rahul - Balance: ₹800
```

PRACTICAL NO 02

PRACTICAL NO (2A)

AIM: Write a python program to demonstrate mining.

CODE:

```
import hashlib
import time
import json



class Block:
    def __init__(self, index, data, previous_hash, difficulty=4):
        self.index = index
        self.timestamp = time.time()
        self.data = data
        self.previous_hash = previous_hash
        self.nonce = 0
        self.difficulty = difficulty
        self.hash = self.mine_block()

    def compute_hash(self):
        block_content = json.dumps({
            'index': self.index,
            'timestamp': self.timestamp,
            'data': self.data,
            'previous_hash': self.previous_hash,
            'nonce': self.nonce
        }, sort_keys=True).encode()

        return hashlib.sha256(block_content).hexdigest()
```

```
    def mine_block(self):
        print(f"Mining block {self.index}...")
        prefix = '0' * self.difficulty

        while True:
            hash_val = self.compute_hash()
            if hash_val.startswith(prefix):
                print(f"Block {self.index} mined with nonce {self.nonce}")
                return hash_val
            self.nonce += 1

    def __repr__(self):
        return json.dumps({
            'index': self.index,
            'timestamp': self.timestamp,
            'data': self.data,
            'previous_hash': self.previous_hash,
            'nonce': self.nonce,
            'hash': self.hash
        }, indent=4)


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

    def create_genesis_block(self):
```

```
        genesis = Block(0, "Genesis Block", "0", self.difficulty)
        self.chain.append(genesis)

    def add_block(self, data):
        last_block = self.chain[-1]
        new_block = Block(len(self.chain), data, last_block.hash, self.difficulty)
        self.chain.append(new_block)

    def print_chain(self):
        print("\nBlockchain:")
        for block in self.chain:
            print(block)
            print("-" * 60)


# Example usage
if __name__ == "__main__":
    blockchain = Blockchain(difficulty=4)  # Proof-of-work difficulty

    blockchain.add_block("Priya sent ₹200 to Rahul")
    blockchain.add_block("Rahul sent ₹150 to Sneha")
    blockchain.add_block("Sneha sent ₹100 to Ravi")

    blockchain.print_chain()
```

OUTPUT:

```
In [1]: runfile('D:/Jyotika/Sem 4/BC Practicals/2a.py', wdir='D:/Jyotika/Sem 4/BC Practicals')
Mining block 0...
Block 0 mined with nonce 121583
Mining block 1...
Block 1 mined with nonce 38716
Mining block 2...
Block 2 mined with nonce 25450
Mining block 3...
Block 3 mined with nonce 179035

Blockchain:
{
    "index": 0,
    "timestamp": 1768975816.5869992,
    "data": "Genesis Block",
    "previous_hash": "0",
    "nonce": 121583,
    "hash": "000095bf54dd0b00a73566a420c2b8018cb90865b3234586b55a6ddea5b26bb0"
}
------------------------------------------------------------
{
    "index": 1,
    "timestamp": 1768975817.148089,
    "data": "Priya sent \u20b9200 to Rahul",
    "previous_hash": "000095bf54dd0b00a73566a420c2b8018cb90865b3234586b55a6ddea5b26bb0",
    "nonce": 38716,
    "hash": "00008f2fa0bffc731df4e66c1a4ad69c32a3a11748f09c7b9b1ac0986dd8990e"
}
------------------------------------------------------------
{
    "index": 2,
    "timestamp": 1768975817.3336394,
    "data": "Rahul sent \u20b9150 to Sneha",
    "previous_hash": "00008f2fa0bffc731df4e66c1a4ad69c32a3a11748f09c7b9b1ac0986dd8990e",
    "nonce": 25450,
    "hash": "0000d44499b55bb3196652fb3621df8ab580f567756faec2ea7798d5bd6dd715"
}
------------------------------------------------------------
{
    "index": 3,
    "timestamp": 1768975817.452557,
    "data": "Sneha sent \u20b9100 to Ravi",
    "previous_hash": "0000d44499b55bb3196652fb3621df8ab580f567756faec2ea7798d5bd6dd715",
    "nonce": 179035,
    "hash": "00009fae1a640f3ab2d4a1abaca4b9e2d9023ecdc492ae324226436b3806ad70"
}
------------------------------------------------------------
```

PRACTICAL NO (2B)

AIM: Demonstrate the use of the Bitcoin Core API to interact with a Bitcoin Core node.

CODE:

```
# Install required library
pip install requests

# Import libraries
import json
import requests
from requests.auth import HTTPBasicAuth

# --- CONFIGURE THIS ---
RPC_USER = "ashish"
RPC_PASSWORD = "strongpassword123"
RPC_PORT = 8332
RPC_HOST = "127.0.0.1"   # Or your node IP

RPC_URL = f"http://{RPC_HOST}:{RPC_PORT}/"
HEADERS = {'content-type': 'application/json'}

# --- JSON-RPC FUNCTION ---
def call_rpc(method, params=None):
    if params is None:
        params = []

    payload = json.dumps({
        "method": method,
        "params": params,
        "jsonrpc": "2.0",
        "id": 1,
```

```
    })

    try:
        response = requests.post(
            RPC_URL,
            headers=HEADERS,
            data=payload,
            auth=HTTPBasicAuth(RPC_USER, RPC_PASSWORD)
        )
        response.raise_for_status()

        result = response.json()

        if result.get("error"):
            print("RPC Error:", result["error"])
            return None

        return result["result"]

    except requests.exceptions.ConnectionError:
        print("ERROR: Could not connect to Bitcoin Core. Is it running?")
    except requests.exceptions.HTTPError as e:
        print(f"HTTP Error: {e}")
    except Exception as e:
        print(f"Unexpected Error: {e}")

    return None


# --- MAIN CODE ---
```

```
if __name__ == "__main__":
    print("Connecting to Bitcoin Core node...\n")

    # 1. Blockchain info
    info = call_rpc("getblockchaininfo")
    if info:
        print("Blockchain Info:")
        print(json.dumps(info, indent=2))

    # 2. Latest block height
    height = call_rpc("getblockcount")
    if height is not None:
        print(f"\nBlock Height: {height}")

    # 3. Latest block hash
    latest_hash = call_rpc("getblockhash", [height]) if height is not None else None
    if latest_hash:
        print(f"\nLatest Block Hash: {latest_hash}")

    # 4. Block info
    block = call_rpc("getblock", [latest_hash]) if latest_hash else None
    if block:
        print("\nLatest Block Data:")
        print(json.dumps(block, indent=2))

    # 5. New address
    address = call_rpc("getnewaddress", ["ashish_wallet"])
    if address:
        print(f"\nNew BTC Address: {address}")
```

```
# 6. Wallet balance
balance = call_rpc("getbalance")
if balance is not None:
    print(f"\nWallet Balance: {balance} BTC")
```

OUTPUT:

```
In [1]: runfile('D:/Jyotika/Sem 4/BC Practicals/2b.py', wdir='D:/Jyotika/Sem 4/BC Practicals')
Connecting to Bitcoin Core node...

ERROR: Could not connect to Bitcoin Core. Is it running?
ERROR: Could not connect to Bitcoin Core. Is it running?
ERROR: Could not connect to Bitcoin Core. Is it running?
ERROR: Could not connect to Bitcoin Core. Is it running?
```

PRACTICAL NO (2C)

AIM: Demonstrating the process of running a blockchain node on your local machine.

CODE:

```
# Install required library
pip install flask requests


# Import libraries
import hashlib
import json
import time
from flask import Flask, jsonify, request
import requests
from urllib.parse import urlparse
import uuid
import sys



# ------------------------------
# Blockchain Class
# ------------------------------
class Blockchain:
    def __init__(self):
        self.chain = []
        self.current_transactions = []
        self.nodes = set()
        self.new_block(previous_hash='1', proof=100)  # Genesis block

    def register_node(self, address):
        parsed_url = urlparse(address)
        if parsed_url.netloc:
```

```
            self.nodes.add(parsed_url.netloc)
        elif parsed_url.path:
            self.nodes.add(parsed_url.path)

    def valid_chain(self, chain):
        last_block = chain[0]
        current_index = 1

        while current_index < len(chain):
            block = chain[current_index]

            if block['previous_hash'] != self.hash(last_block):
                return False

            if not self.valid_proof(last_block['proof'], block['proof']):
                return False

            last_block = block
            current_index += 1

        return True

    def resolve_conflicts(self):
        neighbours = self.nodes
        new_chain = None
        max_length = len(self.chain)

        for node in neighbours:
            try:
                response = requests.get(f'http://{node}/chain')
```

```
            if response.status_code == 200:
                length = response.json()['length']
                chain = response.json()['chain']

                if length > max_length and self.valid_chain(chain):
                    max_length = length
                    new_chain = chain
            except:
                continue

        if new_chain:
            self.chain = new_chain
            return True

        return False

    def new_block(self, proof, previous_hash=None):
        block = {
            'index': len(self.chain) + 1,
            'timestamp': time.time(),
            'transactions': self.current_transactions,
            'proof': proof,
            'previous_hash': previous_hash or self.hash(self.chain[-1]),
        }

        self.current_transactions = []
        self.chain.append(block)
        return block

    def new_transaction(self, sender, recipient, amount):
```

```
        self.current_transactions.append({
            'sender': sender,
            'recipient': recipient,
            'amount': amount,
        })

        return self.last_block['index'] + 1

    @property
    def last_block(self):
        return self.chain[-1]

    @staticmethod
    def hash(block):
        block_string = json.dumps(block, sort_keys=True).encode()
        return hashlib.sha256(block_string).hexdigest()

    def proof_of_work(self, last_proof):
        proof = 0
        while not self.valid_proof(last_proof, proof):
            proof += 1
        return proof

    @staticmethod
    def valid_proof(last_proof, proof):
        guess = f'{last_proof}{proof}'.encode()
        guess_hash = hashlib.sha256(guess).hexdigest()
        return guess_hash[:4] == "0000"
```

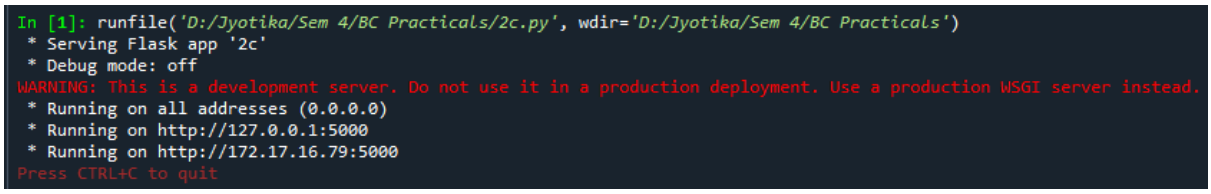
```
# ------------------------------
# Flask API Setup
# ------------------------------
app = Flask(__name__)

node_id = str(uuid.uuid4()).replace('-', '')
blockchain = Blockchain()



@app.route('/mine', methods=['GET'])
def mine():
    last_block = blockchain.last_block
    proof = blockchain.proof_of_work(last_block['proof'])

    blockchain.new_transaction(sender="0", recipient=node_id, amount=1)

    block = blockchain.new_block(proof)

    response = {
        'message': "Block mined successfully!",
        'index': block['index'],
        'transactions': block['transactions'],
        'proof': block['proof'],
        'previous_hash': block['previous_hash'],
    }
    return jsonify(response), 200



@app.route('/transactions/new', methods=['POST'])
def new_transaction_api():
```

```
    values = request.get_json()

    if not values:
        return 'No data received', 400

    required = ['sender', 'recipient', 'amount']
    if not all(k in values for k in required):
        return 'Missing fields', 400

    index = blockchain.new_transaction(values['sender'], values['recipient'],
values['amount'])

    return jsonify({'message': f'Transaction will be added to block {index}'}), 201



@app.route('/chain', methods=['GET'])
def full_chain():
    return jsonify({'chain': blockchain.chain, 'length': len(blockchain.chain)}), 200



@app.route('/nodes/register', methods=['POST'])
def register_nodes():
    values = request.get_json()
    nodes = values.get('nodes')

    if nodes is None:
        return "Error: Please provide a list of nodes", 400

    for node in nodes:
        blockchain.register_node(node)
```

```
    return jsonify({'message': 'Nodes registered successfully', 'nodes':
list(blockchain.nodes)}), 201



@app.route('/nodes/resolve', methods=['GET'])
def consensus():
    replaced = blockchain.resolve_conflicts()

    if replaced:
        return jsonify({'message': 'Chain was replaced', 'chain': blockchain.chain}), 200
    else:
        return jsonify({'message': 'Chain is authoritative', 'chain': blockchain.chain}), 200



# ------------------------------
# Run the Node
# ------------------------------
if __name__ == '__main__':
    try:
        if 'ipykernel' in sys.modules:
            port = 5000
        else:
            import argparse
            parser = argparse.ArgumentParser()
            parser.add_argument('-p', '--port', default=5000, type=int, help='Port to run the
node on')
            args = parser.parse_args()
            port = args.port

        app.run(host='0.0.0.0', port=port, debug=False, use_reloader=False)
```

```
    except Exception as e:
        print("ERROR starting the node:", e)
```

OUTPUT:



PRACTICAL NO (2D)

AIM: Demonstrate mining using geth on your private network.

CODE:

```
# Install required library
pip install web3
# Install Ganache https://trufflesuite.com/ganache/

# Import libraries
from web3 import Web3
import time

# --------------------------------
# Connect to Ganache
# --------------------------------
GANACHE_URL = "http://127.0.0.1:7545"
w3 = Web3(Web3.HTTPProvider(GANACHE_URL))

print("Connecting to Ganache...")

if not w3.is_connected():
    print("❌ Cannot connect to Ganache. Make sure Ganache is running.")
    exit()

print("✅ Connected to Ganache")

# --------------------------------
# Get Accounts
# --------------------------------
accounts = w3.eth.accounts
```

```
sender = accounts[0]
receiver = accounts[1]

print("\nAccounts:")
print("Sender  :", sender)
print("Receiver:", receiver)

# ------------------------------
# Check Balance
# ------------------------------
sender_balance = w3.from_wei(w3.eth.get_balance(sender), 'ether')
receiver_balance = w3.from_wei(w3.eth.get_balance(receiver), 'ether')

print("\nInitial Balances:")
print("Sender  :", sender_balance, "ETH")
print("Receiver:", receiver_balance, "ETH")

# ------------------------------
# Send Transaction
# ------------------------------
print("\nSending transaction...")

tx_hash = w3.eth.send_transaction({
   'from': sender,
   'to': receiver,
   'value': w3.to_wei(1, 'ether')
})

print("Transaction Hash:", tx_hash.hex())
```

```
# Wait for mining
receipt = w3.eth.wait_for_transaction_receipt(tx_hash)
print("Transaction mined in block:", receipt.blockNumber)

# --------------------------------
# Updated Balances
# --------------------------------
sender_balance = w3.from_wei(w3.eth.get_balance(sender), 'ether')
receiver_balance = w3.from_wei(w3.eth.get_balance(receiver), 'ether')

print("\nUpdated Balances:")
print("Sender  :", sender_balance, "ETH")
print("Receiver:", receiver_balance, "ETH")

# --------------------------------
# Latest Block Info
# --------------------------------
latest_block = w3.eth.get_block('latest')

print("\nLatest Block Info:")
print("Block Number :", latest_block.number)
print("Miner        :", latest_block.miner)
print("Timestamp    :", latest_block.timestamp)
print("Transactions :", len(latest_block.transactions))
```

OUTPUT:

```
In [1]: runfile('D:/Jyotika/Sem 4/BC Practicals/2d.py', wdir='D:/Jyotika/Sem 4/BC Practicals')
Connecting to Ganache...
✅ Connected to Ganache

Accounts:
Sender   : 0xE13a4A77aD9552103446a32bb0Aa3C191140BEfB
Receiver: 0x5b5fA6c898EC63803Cc9c44a4810ACC9a01b203e

Initial Balances:
Sender   : 98.999976744140625 ETH
Receiver: 101 ETH

Sending transaction...
Transaction Hash: 055ef7569d9d18e5dbe456c6a8fb8d778309a04dc19d69155a8755d58ad64820
Transaction mined in block: 2

Updated Balances:
Sender   : 97.999956377100365 ETH
Receiver: 102 ETH

Latest Block Info:
Block Number : 2
Miner        : 0x0000000000000000000000000000000000000000
Timestamp    : 1768978855
Transactions : 1
```

PRACTICAL NO 3

Practical 3(A)

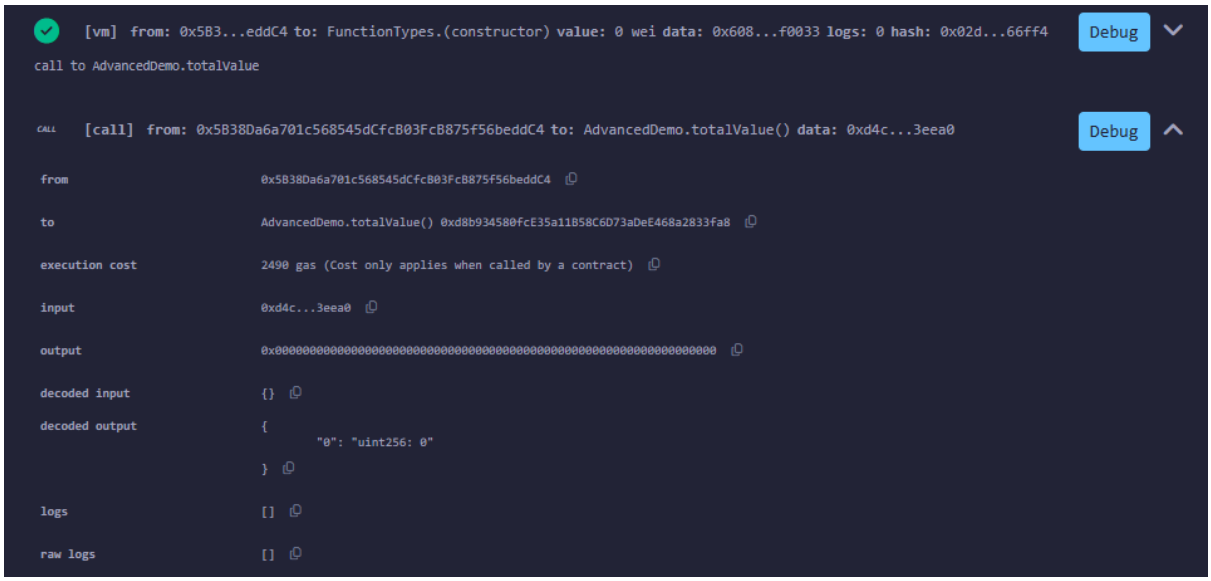
AIM: Write a Solidity program that demonstrates various types of functions including regular functions, view functions, pure functions, and the fallback function.

CODE:

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


contract FunctionTypes {
  uint256 public count = 0;


  // 1. REGULAR (State-Changing) FUNCTION
  // This function modifies the blockchain state and costs Gas.
  function increment() public {
    count += 1;
  }


  // 2. VIEW FUNCTION
  // Can READ state variables but CANNOT modify them.
  // Free when called externally; costs gas if called by another contract.
  function getCount() public view returns (uint256) {
    return count;
  }


  // 3. PURE FUNCTION
  // Neither reads nor modifies state. It only uses its own arguments.
  // Perfect for math or utility calculations.
  function add(uint256 a, uint256 b) public pure returns (uint256) {
    return a + b;
```

```
    }

    // 4. RECEIVE FUNCTION
    // A special function triggered when Ether is sent with no data.
    receive() external payable {
        count += 10; // Logic executed when contract receives ETH
    }

    // 5. FALLBACK FUNCTION
    // Triggered when a function call doesn't match any existing function name,
    // or when data is sent to the contract that it doesn't understand.
    fallback() external payable {
        count = 999;
    }
}
```

OUTPUT:



Practical 3(B)

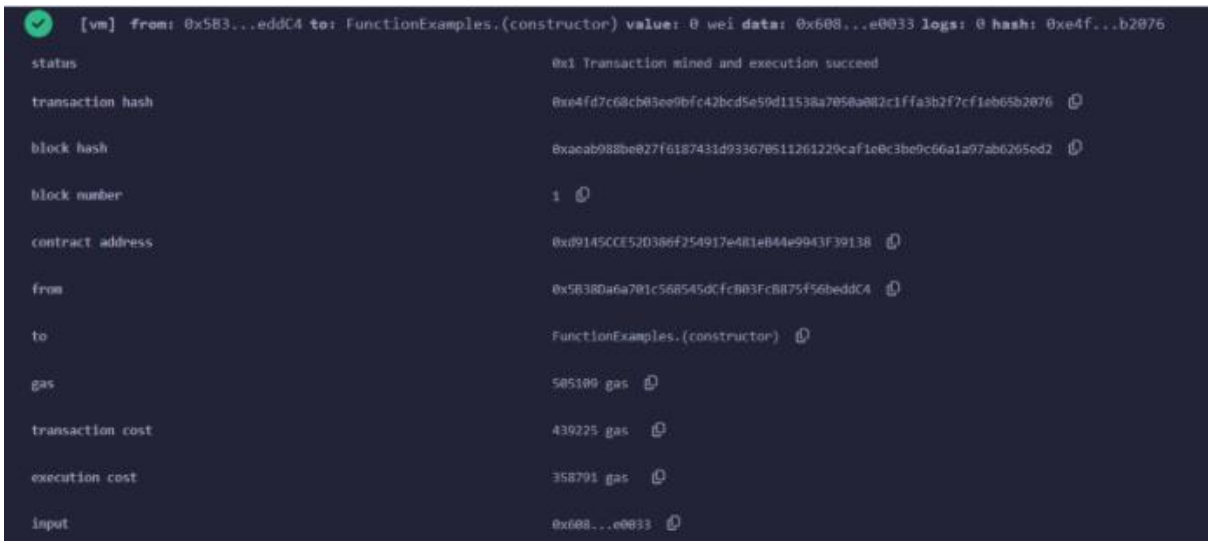
AIM: Write a Solidity program that demonstrates function overloading, mathematical functions, and cryptographic functions.

CODE:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract FunctionExamples {
// ===== Function Overloading =====
// Function 1: Add two numbers
function getSum(uint a, uint b) public pure returns (uint) {
return a + b;
}
// Overloaded Function: Add three numbers
function getSum(uint a, uint b, uint c) public pure returns (uint) {
return a + b + c;
}
// ===== Mathematical Functions =====
// Function to calculate power (exponentiation)
function getPower(uint base, uint exponent) public pure returns (uint) {
uint result = 1;
for (uint i = 0; i < exponent; i++) {
result *= base;
}
return result;
}
// Function to calculate modulo
function getModulo(uint a, uint b) public pure returns (uint) {
require(b != 0, "Division by zero");
return a % b;
}
```

```
// ===== Cryptographic Function =====
// Function to hash a string using keccak256
function getHash(string memory input) public pure returns (bytes32) {
return keccak256(abi.encodePacked(input));
}
}
```

OUTPUT:



Practical 3(C)

AIM: Write a Solidity program that demonstrates various features including contracts, inheritance, constructors, abstract contracts, interfaces.

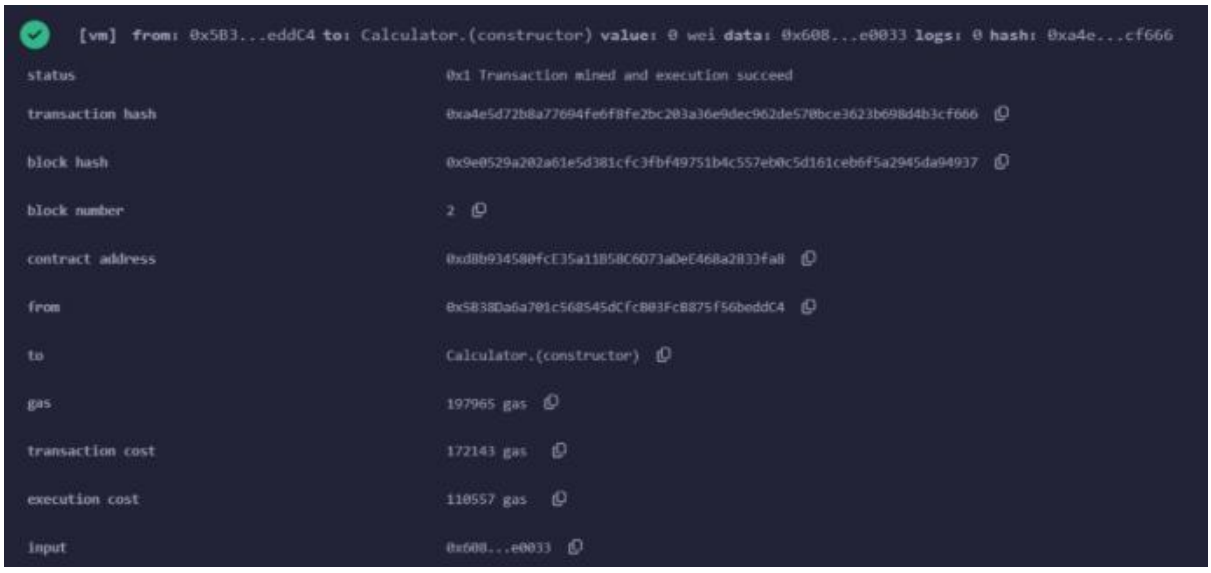
CODE:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
// ------------------------------------
// Interface Example
// ------------------------------------
interface ICalculator {
function add(uint a, uint b) external pure returns (uint);
function subtract(uint a, uint b) external pure returns (uint);
}
// ------------------------------------
```

```
// Abstract Contract Example
// ------------------------------------
abstract contract Shape {
uint public sides;
constructor(uint _sides) {
sides = _sides;
}
function getSides() public view returns (uint) {
return sides;
}
function area() public view virtual returns (uint);
}
// ------------------------------------
// Base Contract (Calculator Implementation)
// ------------------------------------
contract Calculator is ICalculator {
function add(uint a, uint b) public pure override returns (uint) {
return a + b;
}
function subtract(uint a, uint b) public pure override returns (uint) {
return a - b;
}
}
// ------------------------------------
// Derived Contract (Inheritance + Constructor + Abstract Implementation)
// ------------------------------------
contract Square is Shape, Calculator {
uint public length;
constructor(uint _length) Shape(4) {
length = _length;
```

```
}
// Overriding abstract function from Shape
function area() public view override returns (uint) {
return length * length;
}
}
// ------------------------------------
// Another Derived Contract demonstrating Multi-level Inheritance
// ------------------------------------
contract Cube is Square {
constructor(uint _length) Square(_length) {}
// Additional function
function volume() public view returns (uint) {
return length * length * length;
}
}
```

OUTPUT:



Practical 3(D)

AIM: Write a Solidity program that demonstrates use of libraries, assembly, events, and error handling.

CODE:

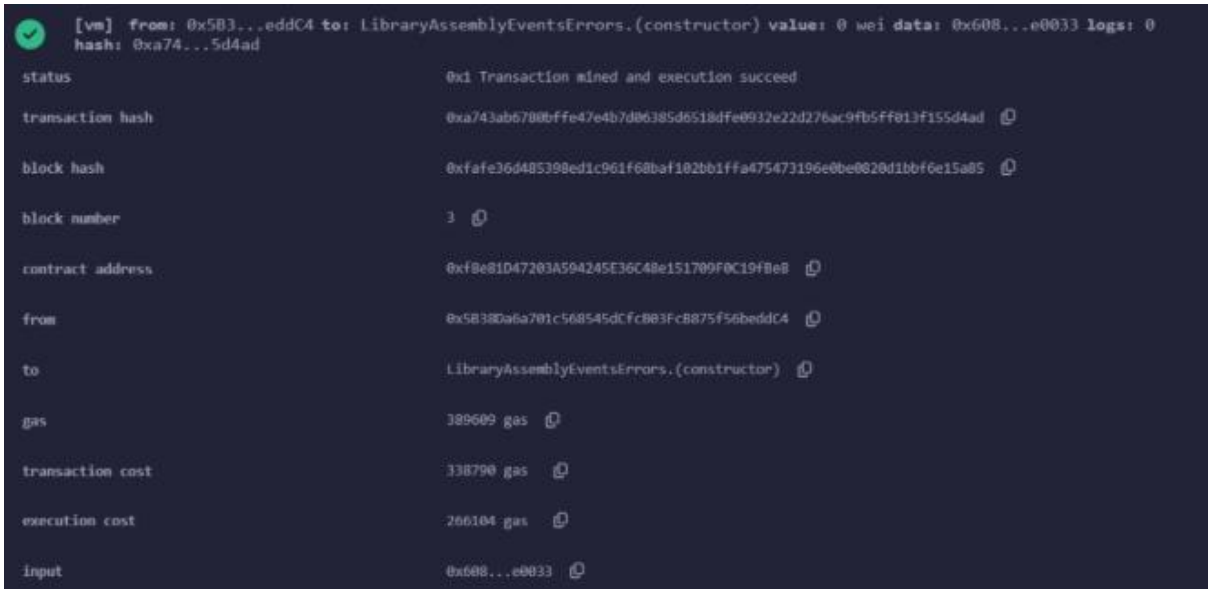
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

// ------------------------------------
// Library Example
// ------------------------------------
library MathLib {
function multiply(uint a, uint b) internal pure returns (uint) {
return a * b;
}
}

// ------------------------------------
// Custom Error Example
// ------------------------------------
error DivisionByZeroError();

// ------------------------------------
// Main Contract
// ------------------------------------
contract LibraryAssemblyEventsErrors {
using MathLib for uint;

uint public result;

// Event Example
```

```
event CalculationResult(string operation, uint value);
event AssemblyOperation(uint indexed input, uint output);

// Function using Library and Events
function multiplyUsingLibrary(uint a, uint b) public {
result = a.multiply(b);
emit CalculationResult("Multiplication", result);
}

// Function demonstrating Error Handling (require, revert, assert, custom error)
function divide(uint a, uint b) public returns (uint) {
if (b == 0) {
revert DivisionByZeroError(); // Custom Error
}

uint divisionResult = a / b;
assert(divisionResult >= 0); // Should always hold true for uint division
emit CalculationResult("Division", divisionResult);
return divisionResult;
}

// Function demonstrating Inline Assembly
function squareUsingAssembly(uint x) public returns (uint) {
uint output;
assembly {
output := mul(x, x)
}
result = output;
emit AssemblyOperation(x, output);
return output;
```

}

}

OUTPUT:


[vm] from: 0x5B3...eddC4 to: LibraryAssemblyEventsErrors.(constructor) value: 0 wei data: 0x608...e0033 logs: 0 hash: 0xa74...5d4ad
status
0x1 Transaction mined and execution succeed
transaction hash
0xa743ab6780bffe47e4b7d8638d6518dfe0932e22d276ac9fb5ff813f155d4ad
block hash
0xfafe36d485398ed1c961f68baf102bb1ffa475473196e0be0820d1bbf6e15a85
block number
3
contract address
0xf8e81D47203A594245E36C48e151709F0C19fBe8
from
0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to
LibraryAssemblyEventsErrors.(constructor)
gas
389609 gas
transaction cost
338790 gas
execution cost
266104 gas
input
0x608...e0033

PRACTICAL NO 4

AIM: Write a program to demonstrate mining of Ether.

CODE:

```
import hashlib
import json
from time import time


class BlockchainNode:
    def __init__(self):
        self.chain = []
        self.pending_transactions = []
        # Create the genesis block (the first block)
        self.create_block(proof=100, previous_hash='1')

    def create_block(self, proof, previous_hash):
        block = {
            'index': len(self.chain) + 1,
            'timestamp': time(),
            'transactions': self.pending_transactions,
            'proof': proof,
            'previous_hash': previous_hash or self.hash(self.chain[-1]),
        }
        self.pending_transactions = []  # Reset pending transactions
        self.chain.append(block)
        return block

    @staticmethod
    def hash(block):
        # We must make sure the Dictionary is Ordered, or we'll have inconsistent hashes
        block_string = json.dumps(block, sort_keys=True).encode()
```

```
        return hashlib.sha256(block_string).hexdigest()

    def proof_of_work(self, last_proof):
        """
        Simple Proof of Work Algorithm:
        - Find a number p' such that hash(pp') contains leading 4 zeroes
        """
        proof = 0
        while self.valid_proof(last_proof, proof) is False:
            proof += 1
        return proof

    @staticmethod
    def valid_proof(last_proof, proof):
        guess = f'{last_proof}{proof}'.encode()
        guess_hash = hashlib.sha256(guess).hexdigest()
        return guess_hash[:4] == "0000"

# --- Demonstration ---
node = BlockchainNode()

print("Mining block 1...")
last_block = node.chain[-1]
proof = node.proof_of_work(last_block['proof'])
previous_hash = node.hash(last_block)
block = node.create_block(proof, previous_hash)

print(f"Block Mined! Hash: {node.hash(block)}")
print(f'Entire Chain: {json.dumps(node.chain, indent=2)}")
```

OUTPUT:

```
In [1]: runfile('D:/Jyotika/BC Prac 4.py', wdir='D:/Jyotika')
Mining block 1...
Block Mined! Hash: 70b231ed4118a058eab004296222b65808b1a3f6c5a0c5de23f27f5707d905cd
Entire Chain: [
  {
    "index": 1,
    "timestamp": 1771536730.8407454,
    "transactions": [],
    "proof": 100,
    "previous_hash": "1"
  },
  {
    "index": 2,
    "timestamp": 1771536730.8947856,
    "transactions": [],
    "proof": 35293,
    "previous_hash": "802638442434e0d60ab1e7724ed28b23e62c1a1937691e3e32a9343e64daeabb"
  }
]
```

PRACTICAL NO 5

AIM: Demonstrate the running of the blockchain node

CODE:

Step 1-> Create a folder named ethermine and a JSON file named genesis.json and write the following lines in it.

{

"config": {

"chainId": 3792,

"homesteadBlock": 0,

"eip150Block": 0,

"eip155Block": 0,

"eip158Block": 0

},

"difficulty": "2000",

"gasLimit": "2100000", "alloc":

{

"0×0b6C4c81f58B8d692A7B46AD1e16a1147c25299F": { "balance":

"9000000000000000000"

}

}

}

genesis.json ethnode_steps.txt

```
{
    "config": {
    "chainId": 3792,
    "homesteadBlock": 0,
    "eip150Block":0,
    "eip155Block":0,
    "eip158Block":0
    },
    "difficulty": "2000",
    "gasLimit":"2100000",
    "alloc":{
    "0x3A7b442afa94ba96396DF86336172947Fa9C48BE":
    {
    "balance" : "90000000000000000000"
    }
    }
}
```

Step 2-> Run command geth account new –datadir

C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical\ethermine testnet-blockchain

```
C:\Users\Achsah>geth account new --datadir C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical
\ethermine
INFO [04-20|20:03:09.337] Maximum peer count                       ETH=50 LES=0 total=50
Your new account is locked with a password. Please give a password. Do not forget this password.
Password:
Repeat password:

Your new key was generated

Public address of the key:   0x77CB2BdBC0f1743bC73E92fla8b1AB80BEDB35AE
Path of the secret key file: C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical\ethermine\key
store\UTC--2023-04-20T14-33-26.959134300Z--77cb2bdbc0f1743bc73e92fla8b1ab80bedb35ae

- You can share your public address with anyone. Others need it to interact with you.
- You must NEVER share the secret key with anyone! The key controls access to your funds!
- You must BACKUP your key file! Without the key, it's impossible to access account funds!
- You must REMEMBER your password! Without the password, it's impossible to decrypt the key!
```

Step 3-> Run command geth account new --datadir

C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical\ethermine

```
C:\Users\Achsah>geth --datadir C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical\ethermine i
nit C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical\ethermine\genesis.json
Fatal: invalid genesis file: math/big: cannot unmarshal "\"3792\"" into a *big.Int

C:\Users\Achsah>geth --datadir C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical\ethermine i
nit C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical\ethermine\genesis.json
INFO [04-20|20:23:47.707] Maximum peer count                       ETH=50 LES=0 total=50
INFO [04-20|20:23:47.717] Set global gas cap                       cap=50,000,000
INFO [04-20|20:23:47.720] Using leveldb as the backing database
INFO [04-20|20:23:47.720] Allocated cache and file handles         database=C:\Users\Achsah\Document
s\MScIT\sem4\blockchain_practical\ethermine\geth\chaindata cache=16.00MiB handles=16
INFO [04-20|20:23:47.741] Using LevelDB as the backing database
INFO [04-20|20:23:47.765] Opened ancient database                  database=C:\Users\Achsah\Document
s\MScIT\sem4\blockchain_practical\ethermine\geth\chaindata\ancient/chain readonly=false
INFO [04-20|20:23:47.767] Writing custom genesis block
INFO [04-20|20:23:47.773] Persisted trie from memory database      nodes=1 size=147.00B time="636.4µ
```

Step 4->Run command geth --identity "localB" --http --http.port "8280"--http.corsdomain "*"

- http.api"db,eth,net,web3" –datadir

"C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical\ethermine" --port "30303" -

nodiscover --networkid 5777 console. This command will enable geth console.

```
C:\Users\Achsah>geth --identity "localB" --http --http.port "8280" --http.corsdomain "*" --http.api
"db,eth,net,web3" --datadir "C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical\ethermine" --
port "30303" --nodiscover --networkid 5777 console
INFO [04-20|20:29:41.383] Maximum peer count                       ETH=50 LES=0 total=50
INFO [04-20|20:29:41.389] Set global gas cap                       cap=50,000,000
INFO [04-20|20:29:41.392] Allocated trie memory caches             clean=154.00MiB dirty=256.00MiB
INFO [04-20|20:29:41.396] Using leveldb as the backing database
INFO [04-20|20:29:41.396] Allocated cache and file handles         database=C:\Users\Achsah\Document
s\MScIT\sem4\blockchain_practical\ethermine\geth\chaindata cache=512.00MiB handles=8192
INFO [04-20|20:29:41.412] Using LevelDB as the backing database
INFO [04-20|20:29:41.420] Opened ancient database                  database=C:\Users\Achsah\Document
s\MScIT\sem4\blockchain_practical\ethermine\geth\chaindata\ancient/chain readonly=false
INFO [04-20|20:29:41.423] Disk storage enabled for ethash caches   dir=C:\Users\Achsah\Documents\MSc
IT\sem4\blockchain_practical\ethermine\geth\ethash count=3
INFO [04-20|20:29:41.424] Disk storage enabled for ethash DAGs     dir=C:\Users\Achsah\AppData\Local
\Ethash count=2
INFO [04-20|20:29:41.426] Initialising Ethereum protocol           network=5777 dbversion=<nil>
INFO [04-20|20:29:41.427]
INFO [04-20|20:29:41.430] ------------------------------------------------------------------------------
```

Step 5-> Run the command

miner.setEtherbase('0xC050FE4d9bAc591d29538e2FD9cCA848B29489D0') in the geth console

Step 6-> Run the command miner.start() to start mining

Step 7-> Below screen shorts are the mining processes running on your local machine

Step 8-> To stop the mining process Ctrl+D