Name : Tejas Ingole

Roll no : 232010012

```
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives.hashes import SHA256
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
import os
import random

# --- RSA Key Generation ---
def generate_rsa_key_pair():
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
    )
    public_key = private_key.public_key()
    return private_key, public_key

# --- Diffie-Hellman Parameter Setup ---
def diffie_hellman_key_exchange():
    print("Enter a large prime number (P) for Diffie-Hellman (default: 23):")
    P = int(input() or "23")  # Use 23 if no input is provided

    print("Enter a generator (G) for Diffie-Hellman (default: 5):")
    G = int(input() or "5")  # Use 5 if no input is provided

    # Alice's private and public values
    a = random.randint(1, P - 1)
    A = pow(G, a, P)

    # Bob's private and public values
    b = random.randint(1, P - 1)
    B = pow(G, b, P)

    return P, G, a, A, b, B

# --- RSA Encryption and Decryption ---
def rsa_encrypt(public_key, message):
    return public_key.encrypt(
        str(message).encode(),
        padding.OAEP(
            mgf=padding.MGF1(algorithm=SHA256()),
            algorithm=SHA256(),
            label=None
        )
    )

def rsa_decrypt(private_key, ciphertext):
    return int(private_key.decrypt(
        ciphertext,
        padding.OAEP(
```

```
            mgf=padding.MGF1(algorithm=SHA256()),
            algorithm=SHA256(),
            label=None
        )
    ).decode())

# --- AES Encryption and Decryption ---
def encrypt_message(message, key):
    iv = os.urandom(16)
    cipher = Cipher(algorithms.AES(key), modes.CFB(iv))
    encryptor = cipher.encryptor()
    ciphertext = encryptor.update(message.encode()) + encryptor.finalize()
    return iv, ciphertext

def decrypt_message(ciphertext, key, iv):
    cipher = Cipher(algorithms.AES(key), modes.CFB(iv))
    decryptor = cipher.decryptor()
    plaintext = decryptor.update(ciphertext) + decryptor.finalize()
    return plaintext.decode()

# --- Main Function ---
def main():
    # Generate RSA key pairs for Alice and Bob
    print("Generating RSA key pairs for Alice and Bob...")
    alice_private_key, alice_public_key = generate_rsa_key_pair()
    bob_private_key, bob_public_key = generate_rsa_key_pair()
    print("RSA key pairs generated!")

    # Diffie-Hellman Key Exchange
    print("\nStarting Diffie-Hellman Key Exchange...")
    P, G, a, A, b, B = diffie_hellman_key_exchange()
    print(f"Diffie-Hellman Parameters: P={P}, G={G}")
    print(f"Alice's public value: {A}")
    print(f"Bob's public value: {B}")

    # Exchange public keys securely using RSA
    print("\nExchanging Diffie-Hellman public keys securely...")
    encrypted_A = rsa_encrypt(bob_public_key, A)
    decrypted_A = rsa_decrypt(bob_private_key, encrypted_A)

    encrypted_B = rsa_encrypt(alice_public_key, B)
    decrypted_B = rsa_decrypt(alice_private_key, encrypted_B)

    print(f"Encrypted A sent to Bob: {encrypted_A}")
    print(f"Decrypted A by Bob: {decrypted_A}")
    print(f"Encrypted B sent to Alice: {encrypted_B}")
    print(f"Decrypted B by Alice: {decrypted_B}")

    # Key Derivation
    print("\nDeriving shared key...")
    alice_shared_secret = pow(decrypted_B, a, P)
    bob_shared_secret = pow(decrypted_A, b, P)

    assert alice_shared_secret == bob_shared_secret, "Shared secrets do not match!"
```

```
    shared_key = alice_shared_secret.to_bytes(16, 'big')  # Convert to bytes for AES
    print(f"Shared secret key (used for AES): {shared_key}")

    # Message Encryption and Decryption
    print("\nEnter a message for Alice to encrypt and send to Bob:")
    message = input("Message: ")

    print("\nEncrypting the message...")
    iv, ciphertext = encrypt_message(message, shared_key)
    print(f"Ciphertext (encrypted message): {ciphertext}")

    print("\nBob decrypts the message...")
    decrypted_message = decrypt_message(ciphertext, shared_key, iv)

    assert message == decrypted_message, "Decryption failed!"
    print("\nMessage exchange successful!")
    print(f"Original Message: {message}")
    print(f"Decrypted Message: {decrypted_message}")

# Run the program
if __name__ == "__main__":
    main()
```

```
Generating RSA key pairs for Alice and Bob...
RSA key pairs generated!

Starting Diffie-Hellman Key Exchange...
Enter a large prime number (P) for Diffie-Hellman (default: 23):
47
Enter a generator (G) for Diffie-Hellman (default: 5):
7
Diffie-Hellman Parameters: P=47, G=7
Alice's public value: 27
Bob's public value: 16

Exchanging Diffie-Hellman public keys securely...
Encrypted A sent to Bob: b'\xae-5\x9eF\x1ac\xd7\xa5\xf3\x17D\xcf\xa92\x82\xc5&\xe
Decrypted A by Bob: 27
Encrypted B sent to Alice: b"3\x80v\xd5\xbe\x0e\r\xc6y\xb9\xf6\x044\x18Y\xe0C\x1b
Decrypted B by Alice: 16

Deriving shared key...
Shared secret key (used for AES): b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x

Enter a message for Alice to encrypt and send to Bob:
Message: THE WORLD

Encrypting the message...
Ciphertext (encrypted message): b'\xad\xc2\xc5\xc6\xbc\x01\x08p\x06\x9f'

Bob decrypts the message...

Message exchange successful!
Original Message: THE WORLD
Decrypted Message: THE WORLD
```