**COMSATS University Islamabad, Attock campus Department of Computer Science**

**Graded task #01**

**Information Security**

Brekhna Gul

Sp24-bse-051

12-October -2025

Ma’am Ambreen

**Graded Task 1**

You have implemented DES there is built in implemented DES in python in crypto cipher module use it for encryption/decryption and provide output sample example

***Code:***

# graded task

from Crypto.Cipher import DES

from Crypto.Random import get\_random\_bytes

from Crypto.Util.Padding import pad, unpad

# DES key must be exactly 8 bytes long

key = get\_random\_bytes(8)

def des\_encrypt(data, key):

    cipher = DES.new(key, DES.MODE\_ECB)

    padded\_data = pad(data, DES.block\_size)

    encrypted\_data = cipher.encrypt(padded\_data)

    return encrypted\_data

def des\_decrypt(encrypted\_data, key):

    cipher = DES.new(key, DES.MODE\_ECB)

    decrypted\_data = unpad(cipher.decrypt(encrypted\_data), DES.block\_size)

    return decrypted\_data

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

    # Input data (must be bytes)

    data = b"Secret123"

    print(f"Original Data: {data}")

    # Encrypt the data

    encrypted\_data = des\_encrypt(data, key)

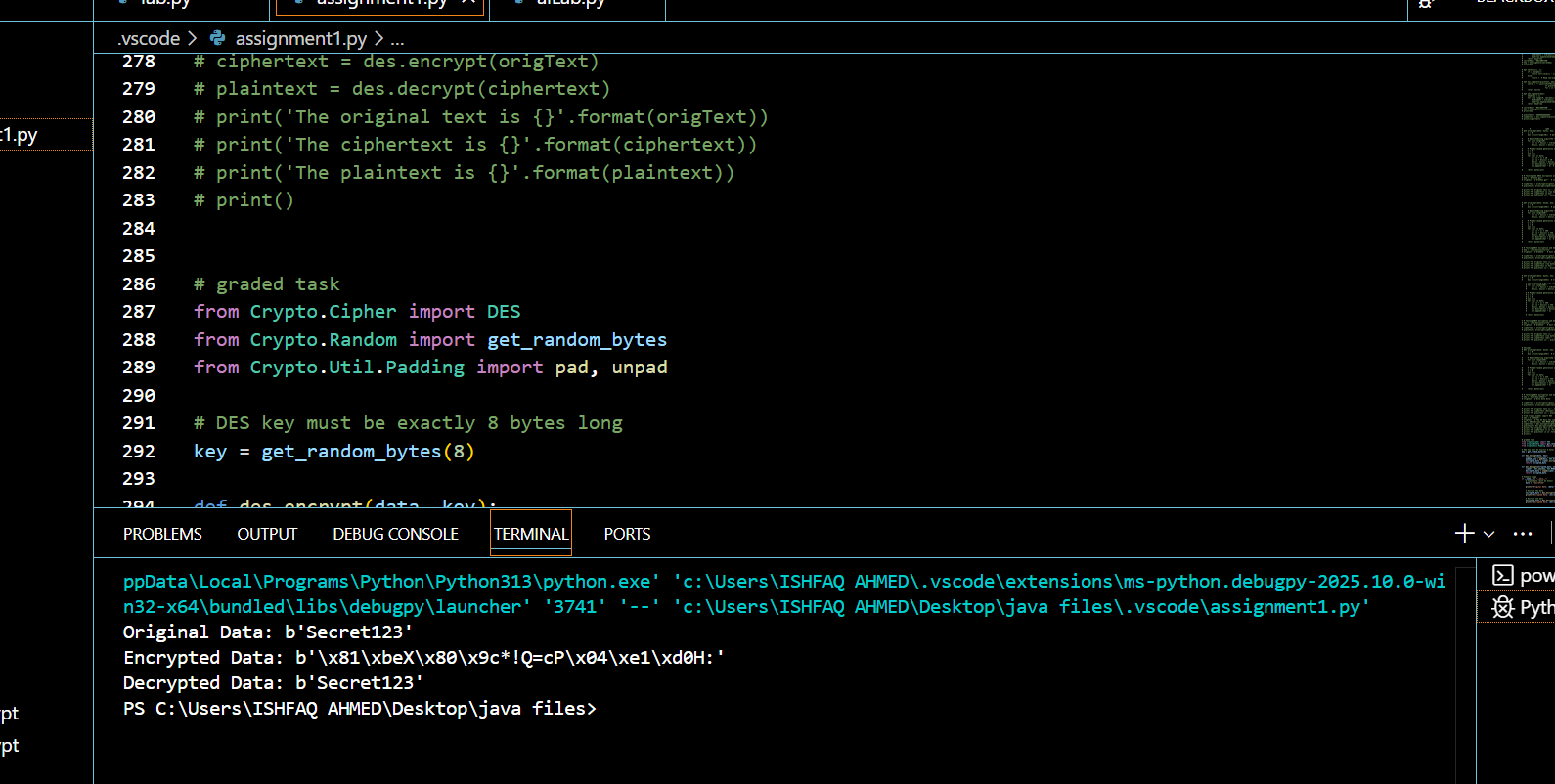
    print(f"Encrypted Data: {encrypted\_data}")

    # Decrypt the data

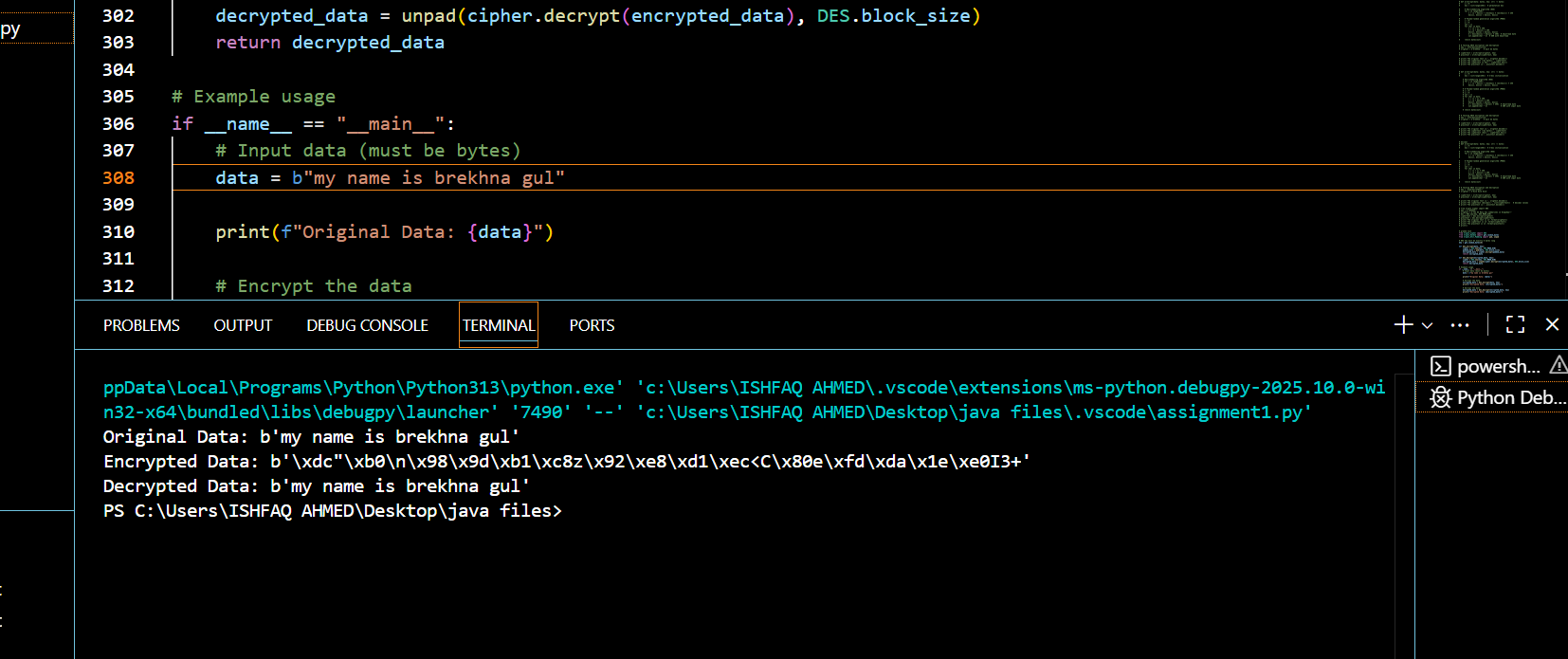
    decrypted\_data = des\_decrypt(encrypted\_data, key)

    print(f"Decrypted Data: {decrypted\_data}")

**Expected output 1:**



**Expected output 2:**

****

**Graded Task 2 — Meet-in-the-Middle (MITM) Attack on Double-DES**

**Code:**

# --- helper: make an 8-byte DES key from a small integer (0..255) ---

def make\_des\_key(x):

    assert 0 <= x < 256

    # keep 7 bytes fixed, vary only last byte -> tiny keyspace for demo

    return b'\x01\x23\x45\x67\x89\xab\xcd' + bytes([x])

# --- DES ECB encrypt/decrypt with padding ---

def des\_encrypt(key, data):

    cipher = DES.new(key, DES.MODE\_ECB)

    return cipher.encrypt(pad(data, DES.block\_size))

def des\_decrypt(key, data):

    cipher = DES.new(key, DES.MODE\_ECB)

    return unpad(cipher.decrypt(data), DES.block\_size)

# --- double DES: C = E\_{k2}( E\_{k1}(P) ) ---

def double\_des\_encrypt(k1, k2, plaintext):

    return des\_encrypt(k2, des\_encrypt(k1, plaintext))

# --- tiny MITM attack ---

def mitm(plaintext, ciphertext):

    keyspace = 256

    table = {}

    # Phase 1: E\_{k1}(P) for all k1

    for k1\_i in range(keyspace):

        k1 = make\_des\_key(k1\_i)

        I1 = DES.new(k1, DES.MODE\_ECB).encrypt(pad(plaintext, DES.block\_size))

        table.setdefault(I1, []).append(k1\_i)  # store possibly multiple k1 indices

    # Phase 2: D\_{k2}(C) and lookup

    candidates = []

    for k2\_i in range(keyspace):

        k2 = make\_des\_key(k2\_i)

        I2 = DES.new(k2, DES.MODE\_ECB).decrypt(ciphertext)

        if I2 in table:

            for k1\_i in table[I2]:

                candidates.append((k1\_i, k2\_i))

    # Verify

    verified = []

    for k1\_i, k2\_i in candidates:

        k1 = make\_des\_key(k1\_i)

        k2 = make\_des\_key(k2\_i)

        if double\_des\_encrypt(k1, k2, plaintext) == ciphertext:

            verified.append((k1\_i, k2\_i))

    return verified

# --- demo run ---

if \_\_name\_\_ == "\_\_main\_\_":

    P = b"SecretMsg"  # plaintext

    # choose real secret keys from tiny keyspace

    k1\_true = random.randint(0, 255)

    k2\_true = random.randint(0, 255)

    K1 = make\_des\_key(k1\_true)

    K2 = make\_des\_key(k2\_true)

    print("True keys (ints):", k1\_true, k2\_true)

    print("K1 hex :", binascii.hexlify(K1).decode())

    print("K2 hex :", binascii.hexlify(K2).decode())

    C = double\_des\_encrypt(K1, K2, P)

    print("Ciphertext (hex):", binascii.hexlify(C).decode())

    found = mitm(P, C)

    if found:

        print("Recovered pairs (k1\_int, k2\_int):")

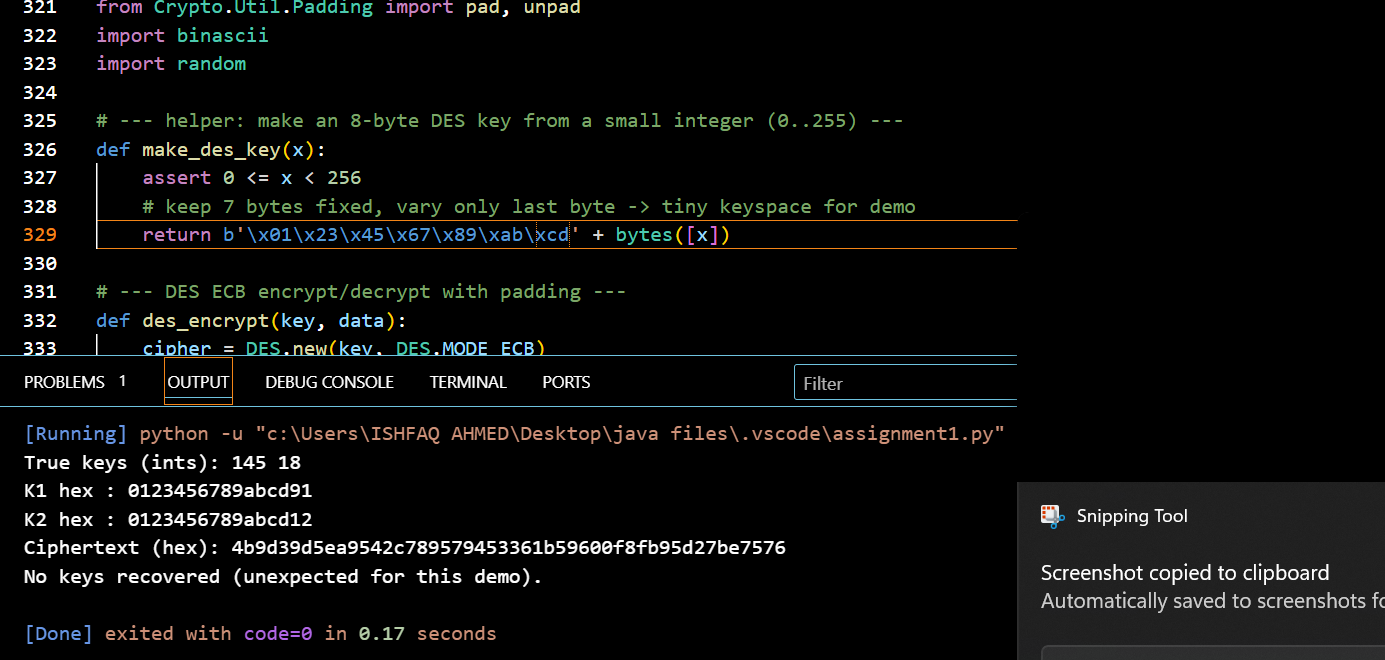
        for a, b in found:

            print(f"  - {a} , {b}  -> k1\_hex={binascii.hexlify(make\_des\_key(a)).decode()} k2\_hex={binascii.hexlify(make\_des\_key(b)).decode()}")

    else:

        print("No keys recovered (unexpected for this demo).")

**expected output:**

****

**Objective:** Visualize and explain how a meet-in-the-middle attack recovers the two keys (K1, K2) used in Double-DES given a known plaintext–ciphertext pair (P, C).

**Answer:**  
Double-DES computes C = E\_{K2}(E\_{K1}(P)). The MITM attack reduces brute force by encrypting P under all K1 values to get intermediate values, decrypting C under all K2 values to get intermediate values, and finding matches. A matching intermediate value I means E\_{K1}(P) == D\_{K2}(C), so (K1,K2) is a candidate pair.

ASCII diagram (paste into Word — looks fine in monospace):

Step-by-step procedure:

1. Obtain a known plaintext–ciphertext pair (P, C).
2. For every possible K1 (iterate full DES keyspace):
   * Compute I1 = E\_{K1}(P).
   * Store (I1, K1) in a lookup table indexed by I1.
3. For every possible K2:
   * Compute I2 = D\_{K2}(C).
   * Check if I2 exists in the I1 table. If yes, retrieve corresponding K1(s) and record candidate pairs (K1, K2).
4. Verify each candidate (K1, K2) by checking E\_{K2}(E\_{K1}(P)) == C. Optionally verify with additional plaintext–ciphertext pairs to eliminate false positives.
5. Return the verified key pair(s).

he meet-in-the-middle attack on Double-DES works by computing intermediate encryption values I1 = E\_{K1}(P) for all K1 and intermediate decryption values I2 = D\_{K2}(C) for all K2, storing I1 values and searching for matches where I1 == I2. A match yields candidate key pairs (K1, K2) which are then verified by re-encrypting P and checking against C. This reduces the search cost to about 2^57 DES operations with 2^56 memory, rendering Double-DES insecure; use AES instead.