**IS LAB ASSIGNMENT 2**

**COMSATS UNIVERSITY ISLAMABAD**

**(ATTOCK CAMPUS)**

**SUBMITTED BY**

**MISBAH UR REHMAN**

**REGISTRATION NO**

**SP24-BSE-036**

**Graded Task 1 — DES encryption/decryption (use built-in implementation)**

We use PyCryptodome's Crypto.Cipher.DES. DES requires an 8-byte key. We use ECB mode for simplicity and Crypto.Util.Padding to pad plaintext to DES block size. The script prints original plaintext, ciphertext (hex), and decrypted plaintext.

**Code**

# graded\_task1\_des\_example.py

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

key = get\_random\_bytes(8)

def des\_encrypt(data: bytes, key: bytes) -> bytes:

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: bytes, key: bytes) -> bytes:

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

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

return decrypted\_data

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

data = b"Secret123" # example plaintext (bytes)

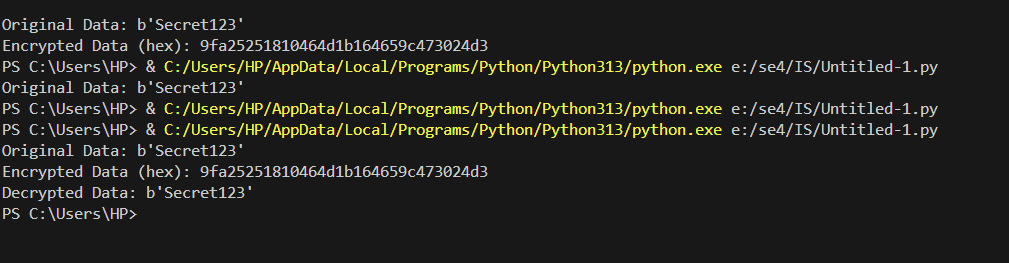
print("Original Data:", data)

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

print("Encrypted Data (hex):", encrypted\_data.hex())

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

print("Decrypted Data:", decrypted\_data)



**Graded Task 2**

Double-DES encrypts C = E\_{K2}(E\_{K1}(P)). Given known (P, C), an attacker can:

1. For each K1 guess compute I1 = E\_{K1}(P) and store I1 -> K1.
2. For each K2 guess compute I2 = D\_{K2}(C) and look up I2 in stored table.  
   If I1 == I2, (K1, K2) is a candidate key pair. This reduces complexity from O(2^{2n}) to O(2^n) time and memory (for n-bit keys). For demonstration we use **reduced 16-bit** per key (fast on desktop) then expand to 8-byte DES keys to run DES operations.

# graded\_task2\_mitm\_demo\_fixed\_final.py

"""

Meet-in-the-Middle demo for 2-DES using reduced 16-bit keys for each DES key (demo only).

This version avoids formatting KeyError by escaping braces.

"""

from Crypto.Cipher import DES

import time

# Helper: construct an 8-byte DES key from a 16-bit integer (demo only)

def build\_des\_key\_from\_int(k16: int) -> bytes:

    b = k16.to\_bytes(2, 'big')

    return b + b + b + b  # 8 bytes total

def des\_encrypt\_block(key: bytes, block: bytes) -> bytes:

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

    return cipher.encrypt(block)

def des\_decrypt\_block(key: bytes, block: bytes) -> bytes:

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

    return cipher.decrypt(block)

def demo\_mitm\_reduced\_space():

    # 8-byte plaintext block (DES block size)

    P\_block = b"ABCDEFGH"

    # True secret reduced keys (16-bit each) for demo

    K1\_true\_16 = 0x1234

    K2\_true\_16 = 0xBEEF

    # Build full 8-byte DES keys (demo construction)

    K1\_true = build\_des\_key\_from\_int(K1\_true\_16)

    K2\_true = build\_des\_key\_from\_int(K2\_true\_16)

    # Compute C = E\_{K2}(E\_{K1}(P))

    I\_true = des\_encrypt\_block(K1\_true, P\_block)

    C\_block = des\_encrypt\_block(K2\_true, I\_true)

    print("MITM demo (reduced 16-bit keys).")

    print("Plaintext block (P):", P\_block)

    print("Ciphertext block (C hex):", C\_block.hex())

    # Escape braces by doubling them so .format won't treat them as placeholders

    print("True K1\_16: 0x{:04X}, True K2\_16: 0x{:04X}".format(K1\_true\_16, K2\_true\_16))

    # Build forward table for E\_{K1}(P)

    max\_key = 1 << 16  # 2^16

    table = {}

    t0 = time.time()

    for k1 in range(max\_key):

        key1 = build\_des\_key\_from\_int(k1)

        I1 = des\_encrypt\_block(key1, P\_block)

        # store first seen k1 for this intermediate value

        if I1 not in table:

            table[I1] = k1

    t1 = time.time()

    # Use escaped braces in the message ({{ and }}) so .format doesn't look for 'K1'

    print("Built table for E\_{{K1}}(P) over 2^16 keys in {:.2f} seconds".format(t1 - t0))

    # Now try K2 guesses: compute D\_{K2}(C) and lookup

    found = []

    t2 = time.time()

    for k2 in range(max\_key):

        key2 = build\_des\_key\_from\_int(k2)

        I2 = des\_decrypt\_block(key2, C\_block)

        if I2 in table:

            k1\_candidate = table[I2]

            found.append((k1\_candidate, k2))

            # break when true pair found (optional)

            if k1\_candidate == K1\_true\_16 and k2 == K2\_true\_16:

                break

    t3 = time.time()

    print("Searched K2 space in {:.2f} seconds".format(t3 - t2))

    print("Found candidate key pairs (k1\_16, k2\_16):", [(hex(a), hex(b)) for a,b in found])

    # Verify the first found candidate by doing full double-encrypt

    if found:

        k1\_test\_16, k2\_test\_16 = found[0]

        k1\_test = build\_des\_key\_from\_int(k1\_test\_16)

        k2\_test = build\_des\_key\_from\_int(k2\_test\_16)

        i\_test = des\_encrypt\_block(k1\_test, P\_block)

        c\_test = des\_encrypt\_block(k2\_test, i\_test)

        print("Verification: Re-encrypted ciphertext matches C?:", c\_test == C\_block)

    else:

        print("No candidate keys found (unexpected in demo).")

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

    demo\_mitm\_reduced\_space()

