# Comsats University Islamabad

(Attock Campus)


---

# ASSIGNMENT 2

**Name : Eman Ejaz** (Sp24- BSE-052)

**Subject:** Information security Lab

**Submitted to:** Mam Ambreen Gul

**Submission Date:** 7- October- 2025

**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:**

```
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}")
```

**Output:**

```
Original Data: b'Secret123'
Encrypted Data: b'\x10?\xc9\xdd\x11\x80\xd9}\xe3\x95\xce\xb7cG\x93\\'
Decrypted Data: b'Secret123'
```

## Lab Graded Task2:

```
from Crypto.Cipher import DES
from Crypto.Util.Padding import pad, unpad
import binascii
import random

# --- 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[I1] = k1_i  # store the integer index

    # Phase 2: D_{k2}(C) and lookup
    candidates = []
    for k2_i in range(keyspace):
        k2 = make_des_key(k2_i)
        I2_padded = DES.new(k2, DES.MODE_ECB).decrypt(ciphertext)
```

```
        if I2_padded in table:
            candidates.append((table[I2_padded], 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).")
```

## Output:

```
ent2.py
True keys (ints): 120 164
K1 hex : 0123456789abcd78
K2 hex : 0123456789abcda4
Ciphertext (hex): 8aab96275a3014625ed51f7db27b808c7cb7e34da10511e3
No keys recovered (unexpected for this demo).
PS C:\Users\pak\Desktop\Python programs>
```