Task #2: Implementing the Data Encryption Standard (DES)

In this task, you are required to implement a complete DES encryption and decryption program from scratch, following the algorithm ***step-by-step*** as discussed in class.

Instructions:

- You must implement each **major step of the DES algorithm** as a separate function:
  - *Initial Permutation*
  - *Round Function*: *Expansion Permutation*, *S-Box substitution*, *and P-Box permutation*.
  - *Final Permutation*
  - *Key Schedule* (subkey generation for all rounds)
- Your implementation must be **modular and self-contained**:
  - Place all DES-related functions and logic in a single file named: ***task2_des*** (e.g., ***task2_des.py***)
  - This file should **not** contain any code for user input, output, or interaction.
- In a separate script (you may name it ***task2_run_des.py***), create an interactive console program that:
  - Prompts the user to select the operation: **Encrypt (E)** or **Decrypt (D)**.
  - Prompts the user to input a **64-bit plaintext or ciphertext** in **hexadecimal format**.
  - Prompts the user to input a **56-bit DES key** in **hexadecimal format**.
  - Calls your functions from ***task2_des.py*** to perform the encryption or decryption step-by-step.
  - Displays the resulting ciphertext or plaintext in hexadecimal format.
- Create a third script named: ***task2_des_avalanche_analysis***. This script imports your DES functions from ***task2_des.py*** and runs the following experiment to measure the avalanche effect:

  Choose a random 64-bit plaintext $\mathbf{P_1}$ and a random 56-bit key $\mathbf{K_1}$.

  Compute the ciphertext $\mathbf{C_1 = DES\_encrypt(K_1, P_1)}$.

  a) Plaintext Bit Flip:
     - Flip **one random bit** in $\mathbf{P_1}$ to obtain $\mathbf{P_1}'$.
     - Compute $\mathbf{C_2 = DES\_encrypt(K_1, P_1')}$.

  b) Key Bit Flip:
     - Flip **one random bit** in $\mathbf{K_1}$ to obtain $\mathbf{K_1}'$.
     - Compute $\mathbf{C_2 = DES\_encrypt(K_1', P_1)}$.

  Repeat the aforementioned experiment **10 times**, display a summary table showing how many bits differed between $\mathbf{C_1}$ and $\mathbf{C_2}$ in both cases, and comment on the observed avalanche effect.

Deliverables:

1) ***Source code*** files (e.g., ***task2_des.py***, ***task2_run_des.py***, and ***task2_des_avalanche_analysis.py***).
2) A ***brief documentation*** that includes:
   - Overview of your implementation.
   - Sample input/output for encryption and decryption.
   - Avalanche effect results and interpretation.
   - Any assumptions made.

<u>Task #3: Breaking Alice and Bob's Encryption System – Meet-in-the-Middle Attack on Triple DES</u>

Alice and Bob are communicating using Triple DES (**3DES**) to exchange top-secret messages. Their encryption process follows this structure:

$$C = Enc_{DES}\left(K_1, Dec_{DES}\left(K_2, Enc_{DES}(K_1, P)\right)\right)$$

Where:

- P is the plaintext message.
- C is the resulting ciphertext.
- K_1 and K_2 are two independent DES keys (each 56 bits excluding parity).

As a skilled cryptanalyst, you have gained access to the API used by Alice and Bob's encryption system. You can now interact with the server through the ***query_server*** function provided in the ***task3_client.py*** file. This function allows you to submit a plaintext (in hexadecimal) and receive the corresponding ciphertext, encrypted with the secret keys K_1 and K_2.

Structure of *query_server*(student_id, plaintext_hex):

- student_id: A string containing your student ID (e.g., "1212049").
- plaintext_hex: A 16-character hexadecimal string representing a 64-bit plaintext.

The function returns a 16-character hexadecimal ciphertext, computed using the 3DES scheme and the hidden keys K_1 and K_2.

Use this chosen plaintext capability to perform a **Meet-in-the-Middle (MITM) attack** and **recover the secret keys K_1 and K_2**. Luckily, Alice and Bob's system suffers from weak key generation. Each key uses only **12** bits of entropy, with the remaining **44 bits fixed to zero**. That means:

$$\mathbf{0x00000000000000} \leq K_1, K_2 \leq \mathbf{0x00000000000FFF}$$

This substantial reduction in key search space makes the attack feasible on standard personal machines.

Instructions:

- Implement the MITM attack as discussed in class using your own DES from **Task 2**.
- The attack may take **hours** to complete, so implement **checkpointing** to save and resume progress using tools like pickle.
  - On restart, the program should load the saved state and continue.
  - Use progress indicators (e.g., tqdm) to track tested keys and remaining candidates.

Deliverables:

1) ***Source code*** file of your MITM attack (e.g., ***task3_mitm.py***).
2) A ***brief documentation*** that includes:
   - A clear explanation of your attack strategy, including how it works and proof of correctness.
   - The total number of queries sent to the encryption server to recover the correct key pair.
   - The total DES encryption and decryption operations performed during the attack.
   - The final recovered keys:
     - K_1 and K_2 in 14-digit hex format (excluding parity bits).
     - K_1 and K_2 as full 16-digit DES keys (including parity bits).