Task5

Correlation Power Analysis (CPA) Implementation

In this step, I implemented a Correlation Power Analysis (CPA) attack on the first round of the DES cipher to recover the 6-bit subkey candidates for all eight S-boxes. The script starts by generating *N* random 64-bit plaintexts and capturing *N* corresponding power traces from the ChipWhisperer board, using fixed ADC settings (offset = 3800, samples = 400, decimate = 1). Each plaintext is sent once to the target device, and its corresponding power trace is stored. These traces are later reused for all key guesses and S-boxes to avoid repeated captures.

The CPA algorithm models how power consumption depends on the intermediate values of the DES round. For each S-box (from S1 to S8) and for every possible 6-bit subkey guess (k = 0...63), the script computes the S-box output of round 1 for all plaintexts (p_0 ... p_{n-1}). From each S-box output, it calculates the **Hamming weight**, which represents the number of bits equal to '1' and serves as a simple model of the device's power usage. This produces a data structure that combines all plaintext–key pairs:

```
(p0, k0)  → HW(...)
(p1, k0)  → HW(...)
...
(p(n-1), k0)  → HW(...)
...
(p0, k63) → HW(...)
...
(p(n-1), k63) → HW(...)
```

Each column of this structure corresponds to one key guess and contains *N* hypothetical power values (Hamming weights). The script then compares each key's Hamming weight vector with the real measured power traces. For every key guess, it calculates the **Pearson correlation coefficient** between the predicted

power values and each sample point in the traces. This measures how strongly the estimated power matches the actual hardware power consumption at that time. The sample position where this correlation is the highest in absolute value is considered the best point of leakage for that key guess.

Finally, the script stores the maximum absolute correlation and the corresponding sample index for all 64 key guesses of each S-box. After sorting by correlation strength, the top five keys with the highest correlation are reported as the best candidates for each S-box. These subkeys can later be combined to reconstruct the full DES round key.

```
chmod +x cpa.py
./dpa.py 2000
# or
python3 cpa.py 200
```

```
=== CPA top 5 keys per S-box ===

S-box 1:
  #1: key=0x27 (dec=39), max_abs_corr=0.551539, sample=49 (original ≈ 3849)
  #2: key=0x1D (dec=29), max_abs_corr=0.378878, sample=49 (original ≈ 3849)
  #3: key=0x0B (dec=11), max_abs_corr=0.322788, sample=51 (original ≈ 3851)
  #4: key=0x2A (dec=42), max_abs_corr=0.306056, sample=49 (original ≈ 3849)
  #5: key=0x22 (dec=34), max_abs_corr=0.300858, sample=77 (original ≈ 3877)

S-box 2:
  #1: key=0x2F (dec=47), max_abs_corr=0.605202, sample=49 (original ≈ 3849)
  #2: key=0x23 (dec=35), max_abs_corr=0.295200, sample=50 (original ≈ 3850)
  #3: key=0x20 (dec=32), max_abs_corr=0.294440, sample=49 (original ≈ 3849)
  #4: key=0x17 (dec=23), max_abs_corr=0.275846, sample=80 (original ≈ 3880)
  #5: key=0x1B (dec=27), max_abs_corr=0.272713, sample=6 (original ≈ 3806)

S-box 3:
  #1: key=0x19 (dec=25), max_abs_corr=0.469219, sample=89 (original ≈ 3889)
  #2: key=0x1D (dec=29), max_abs_corr=0.318470, sample=146 (original ≈ 3946)
  #3: key=0x09 (dec= 9), max_abs_corr=0.313989, sample=142 (original ≈ 3942)
  #4: key=0x1B (dec=27), max_abs_corr=0.305733, sample=142 (original ≈ 3942)
  #5: key=0x21 (dec=33), max_abs_corr=0.293713, sample=174 (original ≈ 3974)

S-box 4:
  #1: key=0x0E (dec=14), max_abs_corr=0.529196, sample=157 (original ≈ 3957)
  #2: key=0x1E (dec=30), max_abs_corr=0.467276, sample=157 (original ≈ 3957)
  #3: key=0x36 (dec=54), max_abs_corr=0.404159, sample=157 (original ≈ 3957)
```

```
  #4: key=0x26 (dec=38), max_abs_corr=0.396176, sample=157 (original ≈ 3957)
  #5: key=0x3A (dec=58), max_abs_corr=0.378984, sample=22 (original ≈ 3822)

S-box 5:
  #1: key=0x13 (dec=19), max_abs_corr=0.420292, sample=125 (original ≈ 3925)
  #2: key=0x1A (dec=26), max_abs_corr=0.328361, sample=83 (original ≈ 3883)
  #3: key=0x20 (dec=32), max_abs_corr=0.296806, sample=170 (original ≈ 3970)
  #4: key=0x3C (dec=60), max_abs_corr=0.293844, sample=0 (original ≈ 3800)
  #5: key=0x35 (dec=53), max_abs_corr=0.289183, sample=113 (original ≈ 3913)

S-box 6:
  #1: key=0x2E (dec=46), max_abs_corr=0.342465, sample=125 (original ≈ 3925)
  #2: key=0x13 (dec=19), max_abs_corr=0.337430, sample=160 (original ≈ 3960)
  #3: key=0x31 (dec=49), max_abs_corr=0.291988, sample=114 (original ≈ 3914)
  #4: key=0x3E (dec=62), max_abs_corr=0.281428, sample=33 (original ≈ 3833)
  #5: key=0x0D (dec=13), max_abs_corr=0.279607, sample=169 (original ≈ 3969)

S-box 7:
  #1: key=0x21 (dec=33), max_abs_corr=0.561832, sample=169 (original ≈ 3969)
  #2: key=0x0F (dec=15), max_abs_corr=0.329776, sample=169 (original ≈ 3969)
  #3: key=0x23 (dec=35), max_abs_corr=0.320745, sample=169 (original ≈ 3969)
  #4: key=0x14 (dec=20), max_abs_corr=0.299129, sample=62 (original ≈ 3862)
  #5: key=0x19 (dec=25), max_abs_corr=0.284162, sample=125 (original ≈ 3925)

S-box 8:
  #1: key=0x03 (dec= 3), max_abs_corr=0.467150, sample=101 (original ≈ 3901)
  #2: key=0x1D (dec=29), max_abs_corr=0.318814, sample=61 (original ≈ 3861)
  #3: key=0x1E (dec=30), max_abs_corr=0.312545, sample=127 (original ≈ 3927)
  #4: key=0x2C (dec=44), max_abs_corr=0.306080, sample=27 (original ≈ 3827)
  #5: key=0x07 (dec= 7), max_abs_corr=0.301414, sample=64 (original ≈ 3864)
```

Find FULL_KEY

We start from the S-box candidates found by the CPA attack and stored in sbox_out.txt. Each S-box gives us a few possible 6-bit key pieces. We combine one candidate from each of the eight S-boxes to build all possible 48-bit round-1 subkeys (K1). Then we reverse the DES key schedule: first, we use the inverse PC2 table to expand each 48-bit K1 into 56-bit candidates, filling in the dropped bits in every possible way. Next, we undo the round-1 shift to get the original 56-bit key before the first round. Finally, we apply the inverse PC1 table to make a full 64-bit key, setting the parity bits to zero.

The script then tests each generated key with the known plaintext 4142434445464748 and checks if it encrypts to ef770c97ad062c75. The key that matches this ciphertext is the correct full DES key.

To run it, install pycryptodome (pip3 install pycryptodome)

FULL_KEY IS: 0x5AE0F272B862DA58

```
$ python3 find_full_key.py
```

```
[INFO] Loaded CANDIDATES_HEX from sbox_out.txt:
  S-box 1: ['0x27', '0x38', '0x2C', '0x25', '0x39']
  S-box 2: ['0x2F', '0x1B', '0x23', '0x20', '0x2D']
  S-box 3: ['0x19', '0x11', '0x2C', '0x03', '0x0C']
  S-box 4: ['0x0E', '0x26', '0x1E', '0x36', '0x02']
  S-box 5: ['0x13', '0x35', '0x26', '0x20', '0x1F']
  S-box 6: ['0x2E', '0x0D', '0x3E', '0x13', '0x27']
  S-box 7: ['0x21', '0x0F', '0x2E', '0x14', '0x1C']
  S-box 8: ['0x03', '0x08', '0x32', '0x02', '0x2C']
[INFO] Using top 3 candidates per S-box.

[+] Found a matching key!
    Key (hex) = 0x5AE0F272B862DA58
[INFO] Tested 249 candidate 56-bit keys (from 1 K1s).
[RESULT] Full 64-bit key (parity bits = 0): 0x5AE0F272B862DA58
```