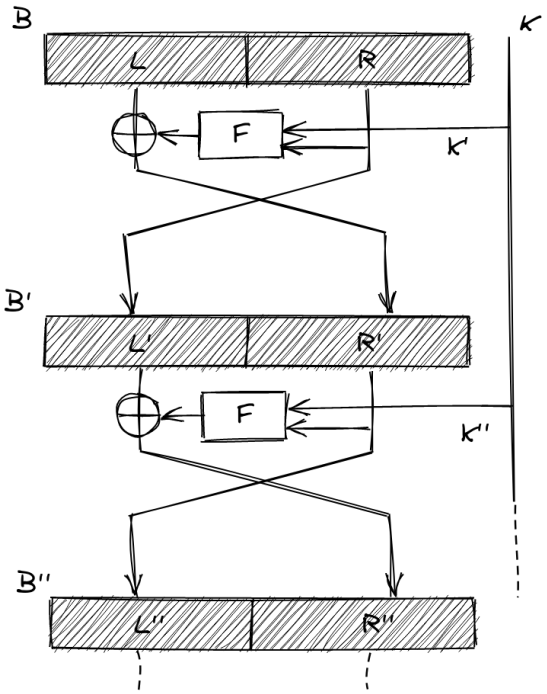
Task2

In Task 2, we generated two sets of 100 plaintexts, where set A had values of the form 0x00000000XXXXXXXX and set B had 0xFFFFFFFFYYYYYYYY, with random 32-bit lower halves. Using the ChipWhisperer-Lite and STM32 target, we captured one power trace per plaintext with 13420 samples, decimate = 4, and offset = 0. Each trace was saved as a .npy file in set_A and set_B directories. We then averaged all traces in each set to obtain set_A_average.npy and set_B_average.npy, computed abs(tAavg − tBavg), and plotted it together with one example trace. The resulting difference trace showed a clear spike at the boundary between the key generation and the start of the Feistel rounds, indicating where the plaintext is first processed.

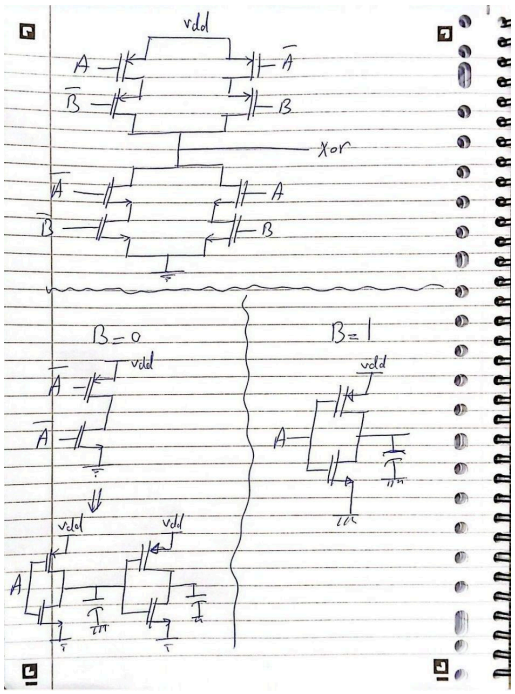
To explain why this happens, we need to look at the DES diagram and the Feistel structure. In the first round, the most significant 32 bits (MSB) of the data are XORed with the output of the F function.



In our experiment, the 32 MSB bits in set A are all zeros, while in set B they are all ones. This means that one of the inputs to the XOR gate during the first Feistel round is either all zeros or all ones, depending on the set.

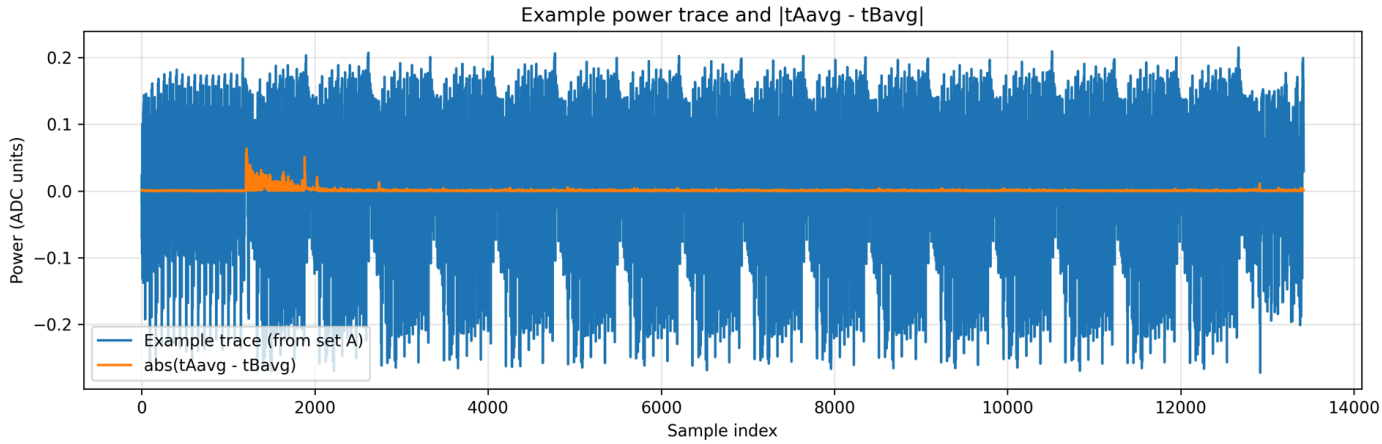
If we look at the CMOS-level implementation of the XOR gate, we can understand why this causes a difference in power consumption. In the diagram, the top part shows a standard CMOS XOR circuit. The lower part of the figure shows two simplified cases: on the left, the XOR when input B is zero, and on the right, when input B is one. When B = 0, the XOR gate behaves like two NOT gates in series. So, when the input changes, two gate capacitors must charge or discharge. However, when B = 1, the XOR behaves like a single NOT gate, meaning only one capacitor needs to charge or discharge. This results in a noticeable difference in dynamic power consumption between the two cases.

Therefore, since in set A the MSB bits are all zeros and in set B they are all ones, the CMOS switching activity is different when the first Feistel round starts. This difference creates the distinct spike we observed in the power trace, marking the beginning of the Feistel operation in DES.



Important Note:

We know that DES begins with an initial permutation that rearranges the input bits before the Feistel rounds start. This means the original 32 MSB bits of the plaintext are spread across the entire 64-bit block, but some of them always end up in the higher 32 bits used in the first round. As a result, even though the bits are permuted, the difference between all-zero MSBs in set A and all-one MSBs in set B still affects the data handled in the first Feistel round. When we run the experiment 100 times for both sets, this consistent difference in bit patterns leads to a measurable difference in power consumption, clearly visible in the averaged traces.



The three scripts in Task 2 work together to generate plaintexts, capture power traces, and process the results. `generate_64bit_sets.py` creates two sets of 100 random 64-bit values, set A with upper 32 bits as zeros and set B with upper 32 bits as ones. `task2_generate.py` reads these values, sends each to the STM32 target via ChipWhisperer, and captures one trace per input, saving them as `.npy` files in `set_A` and `set_B` folders. Finally, `task2_process.py` averages all traces in each set, computes the absolute difference between the averages, and plots it to highlight where the plaintext is processed in the DES operation.