

# ASSIGNMENT – 3

Name: Vamshikiran Morlawar

Roll no: 22111066

## **Group Details:**

Group No. 4

Vinay Agrawal (22111068)

Manish Kumar Ghildiyal (22111039)

Ayush Kothiyal (22111015)

Vamshikiran Morlawar (22111066)

Arabinda Karmakar (22111011)

## **Python Tool Version used: 3.8.0**

### **Question 1:**

**DOM Attack:** In this assignment, you will be provided with the power traces of AES. The power traces are stored in a CSV file, where each row indicates the power consumption of one AES execution. For every row, the first entry is plaintext, the second entry is ciphertext, and all the subsequent entries are power consumption values. Your task is to write a code for Difference of Mean Attack, and use that code on the given power traces to recover the 4th and 5th bytes of the secret key used in the AES execution. The demo code is for finding out the 0th byte of the secret key.

Note: You will be provided with 10 CSV files, you have to use one according to your group number.

(e.g., if your group number is 1, then use HW power trace 1.csv)

## Logic:

Here, we had to implement DOM attack, and use the power traces provided to recover 4<sup>th</sup> and 5<sup>th</sup> bytes of secret key used in the AES execution. We have example code for recovering 0<sup>th</sup> byte. So, we need to run the same code twice to get the 4<sup>th</sup> and 5<sup>th</sup> bytes. Now, for the 0<sup>th</sup> byte recovery we shifted the 128-bit cipher text to 120 bits towards right in order to get the 0<sup>th</sup> byte in last 8 bits and hence, got the 0<sup>th</sup> byte recovered.

Now, for extending the same concept for 4<sup>th</sup> and 5<sup>th</sup> byte, secret key we have to get the first 4 and 5 bytes of data of cipher text in order to recover the intended byte. So, to recover 4<sup>th</sup> byte first, we shift the 128-bit cipher text, (120-32) i.e. 88 bits towards right to get the first 4 bytes at the end. Now, we do 'logical and' operation on the last 8 bits of the shifted cipher text with all 8-bit 1's to eventually get the 4<sup>th</sup> byte recovered.

Similarly, to recover 5<sup>th</sup> byte of secret key, we shift the 128-bit cipher text, (120-40) i.e. 80 bits towards right to get the first 5 bytes at the end. Now, we do 'logical and' operation on the last 8 bits of the shifted cipher text with all 8-bit 1's to eventually get the 5<sup>th</sup> byte recovered.

Code implementation of same:

```
shift_regain_left = (8*j)-40
ct_temp = (ct >> (120+shift_regain_left))
ct_temp &= 0b11111111
```

Where, j is the loop variable taking values 0 and 1. 0 for the 4<sup>th</sup> byte recovery and 1 for the 5<sup>th</sup> byte recovery.

**Key Bytes to recover: 4<sup>th</sup> and 5<sup>th</sup>**

**CSV file used: HW\_power\_trace\_4.csv (Group 4)**

**Recovered 4<sup>th</sup> byte secret key: 87**

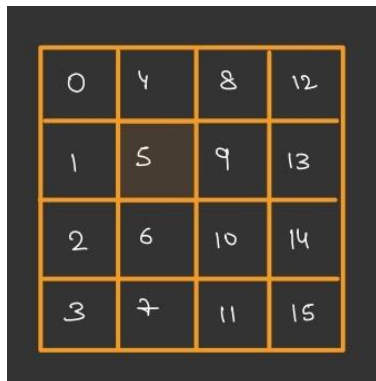
**Recovered 5<sup>th</sup> byte secret key: 62**

## Question 2:

**CPA:** In this assignment, you will be provided with the power consumption of the last round of AES. The power traces are obtained from SAKURA-G platform that runs an implementation of AES-128 on a Spartan-6 FPGA. The power trace is stored in a CSV file, where each row indicates the power consumption of one AES execution. For every row, the first entry is plaintext, the second entry is ciphertext, and all the subsequent entries are power consumption values. Your task is to write a code for Correlation Power Attack, and use that code on the given power trace to recover the target byte assigned to your group.

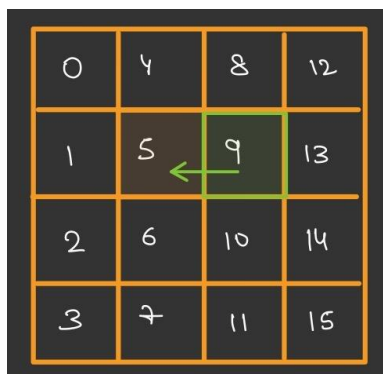
## Logic:

We have to attack the 5th byte according to the diagram above. The matrix has all the 128 bits arranged byte wise in a 4x4 format.



0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

The last round has the following stages: SubByte, ShiftRow and AddRoundKey. We are given the cipher text so first we take the 5th byte from the ciphertexts[i][j] matrix and XOR it with the corresponding key (k).



0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

Now shifting this element to the right 1 time, that is to the 9th element of the ciphertexts matrix because of the 5th byte being in the 1st row within the matrix. Then apply the InvSbox substitution corresponding to the 9th element to find the hamming distance by counting the number of ones. The key which gives the highest peak for most of the test cases with respect to this value will be considered our key.

Code implementation of same:

```
for j in range(0,8937):
    for k in range(0,256):
        x=ciphertexts[j][slice(10,12)]
        x8=int(x,16)^k
        cipher9[j][k]=InvSbox[x8]
        x1=ciphertexts[j][slice(18,20)]
        x2=int(x1,16)
        xor1=x2^int(cipher9[j][k])
        hamming_dist=number_of_ones(xor1)
        hypothetical_model[j][k]= hamming_dist
```

**Target Byte assigned: 5<sup>th</sup> Byte (Group 4)**

**Value of recovered 5<sup>th</sup> byte: 121**