Name: Luv Ahuja
Roll number: 23157028
Email ID: luvahuja23@iitk.ac.in
Group: 5

CS 984 Introduction to Hardware Security Assignment 1

Given items.

- Prof/TAs has provided the "AES_Power_Trace_emasters.csv" file which consists of the power traces corresponding AES implementation of SAKURA-G platform that runs an implementation of AES-128 on a Spartan-6 FPGA. Below is the format in which power trace obtained has been feed into the provided excel spreadsheet.

    1. **Column A** – It consists of the plaintext values.
    2. **Column B** – It consists of the Cipher Text values.
    3. **Column C-BXZ** – It consists of the power traces values.
    4. **Column D-M** – It consists of the 10 rounds of the AES implementation power traces values.

- Prof/TAs has also provided the working CPA code on AES-128 implementation to retrieve/recover 0^{th} byte Key using the provided power trace within the "AES_Power_Trace_emasters.csv" file.

Scope of problem

- Retrieval/Recovering of the 6^{th} byte (as per the assigned byte to corresponding group-5) from 10 round AES-128 implementation by applying correlation power attack code on the power consumption trace provided. Using CPA method to extract the 6th byte of the secret key used in AES-128 execution.

Solution Approach/Strategy Opted

I would like to thank Prof. Debapriya Basu Roy who has explained the Correlation Power Attack (CPA) on FPGA hardware topic so fantastically and provided such an amazing slide content/material which has helped me to understand this Assignment problem statement and using below computation model, I took/opted approach towards solving the assignment. Let's understand it in detail.

I understood clearly that there is no mix-column transformation operation if provided power traces of the AES implementation belong to the last round. To identify the 6^{th} byte of 10 round key, we require to perform operations: XORing with key bytes, Inverse ByteSub (S-Box) and Inverse Shift row, Add round Key XOR operation as per defined operation order for the last round.

We focused on finding 2 most important aspects which has major/key role in recovering the 6th byte from AES implementation.

1. Computing the initial values and positions in each location by the power traces
2. Computing Hamming distances during the bit transitioning between the power trace of the corresponding initial and final values.

We utilized the provided code for 0th byte and inserted the logic for performing the AES Inverse shift row operation on the power trace to find/recover the 6th byte. Below are high-level of steps for the approach taken.

1. Group-5 need to recover the 6th Byte of the Key which we assume as K6.
2. We've taken the C6 (Cipher-Text) value from the spread sheet given.
3. Let's identify the R6.
    - As per the below table, C6 value will be carried by the Register R6.
    - Interim position of R6 computed as C6 Xoring with K6.
4. Let's identify the R2.
    - Initial position (Position of current register R6) = Inverse SR
    - Final position (Position of C6) is known after Inverse SR
    - Value in R2 = Inverse S-box (C6).

Note: Since R6 belongs to 3rd row of the Matrix so as per Inverse Shift Row operation rule, R6 will shift 2 bytes. R6 will take position of the R14 and R2 will take position of R10 in the matrix.

5. Computing Hamming Distance when this state transition happens correlates with the associated power trace values.
    - Now we identify a register where we know the initial value and final value.
    - If we find the hamming distance between the two values and store them in the hypothetical matrix.
6. Perform correlation between hypothetical and real power matrix using PCC (Pearson Correlation Co-efficient) gives the key value of the given byte position (6th)

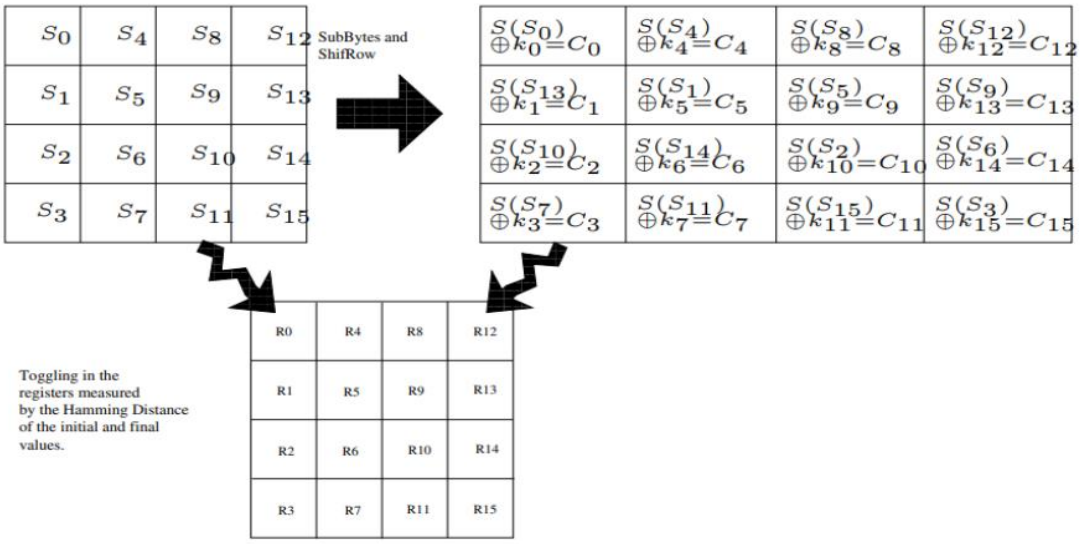

Figure1: Computation Model

Solution Execution Platform:

1. Platform: Windows 10.
2. Software: Visual Studio Code
3. Language: Python version 3.10
4. Recover the 6th byte correct key value via running "CPA_against_6th_byte_Group5.py" (Same has been attached along with zip file)

Solution Implementation

I've taken the leverage of the code provided by the Prof to form the logic for recovering the 6th Byte Key value. The important aspects for finding specific byte value of the Key in 10th round are listed below:

1. No Mix-Column operation due to Power trace belong to the AES implementation within Last Round.
2. As we've been instructed to use the CPA methodology which uses the power traces produced without affecting the state array matrix and its bit locations, so we implied the Inverse Shift Row operation.
3. Operations: **P.T XOR with Key (K6), Inverse Shift Row, Inverse S-BOX and Add round Key XOR.**

With above stated approach, I've modified existing logic (0Th Byte) and introduced the below logic (6th Byte) into the code which has been provided by Prof.

1. In this Power based SCA, we have obtained the device power consumption of the DUT while it performs the AES operations. We take many plaintext and cipher pairs.

2. CPA is a divide and conquer approach. We attacked byte-by-byte and piece together the round key, using which we can get the master key using reverse key scheduling. Although, it was not scope of the problem for this assignment. We attack the last round as we know the output of the last round which is CipherText.
3. Although, in the DOM attack we only take one bit of the data, in CPA, we take the entire hamming weight of the target 8-bit data. By applying the Pearson correlation coefficient, we can compute the correlation between the columns of the Actual and hypothetical power traces matrices.
4. **To extract the byte, we need to figure out which register to attack. The reasoning behind this is that while registers are fixed, the data in them will change.**
5. We need to find the Hamming Distance between the byte of the cipher text and the byte from the state matrix.
6. For me belonging to Group-5, I was asked to extract byte 6th (K6). While observing the figure1, I have reached at the conclusion that this will be R6.
7. Since I need to extract K6, I need to get the byte from the CT using this:

```
            #x=int(ciphertexts[j][slice(2,4)],16)
            # find C6 byte
            x_c6=ciphertexts[j][slice(12,14)]
```

8. Because of the shift operation of the register as it belongs to 3rd row of the matrices so there will be only 2 shifts, the other value can be extracted through:

```
# Final value in R2 is C2 (By table observation)
# Find C2 byte #moved to byte 14th after shift row operation
      x_c2=ciphertexts[j][slice(28,30)]
```

9. With these changes, I've run the code. Please Ref below entire code logic.

Entire logic as follows:

```
for j in range(0,8939):
    for k in range(0,256):
            #x=int(ciphertexts[j][slice(2,4)],16)
            # find C6 byte
            x_c6=ciphertexts[j][slice(12,14)]

            #x=ciphertexts[j][slice(0,2)]
            #XOR C6 with assumed Key gives R6 interim value
            x8=int(x_c6,16)^k

            #This R6 interim value moves to R2 due to Inverse Shift Row operation ( This is based on the table observation )
            #This interim R2 gets subjected to Inv S Box to obtain the initial R2 value
            #print(x8,"\n")
            cipher9[j][k]=InvSbox[x8] #Here we are not considereing InvSR because we are getting 0th byte

            # Final value in R2 is C2 (By table observation)
            # find C2 byte #moved to byte 14th after shift row operation
            x_c2=ciphertexts[j][slice(28,30)]

            #x1=ciphertexts[j][slice(0,2)]
            # Final R2 value is x2 below
            x2=int(x_c2,16)

            # Hamming Distance
            xor1=x2^int(cipher9[j][k])
            hamming_dist=number_of_ones(xor1)
            hypothetical_model[j][k]= hamming_dist
```

Key Position	Line 46 Slice	Line 59 Slice	10th RoundKey
0	0,2	0,2	0xB1
1	2,4	26,28	0x6A
2	4,6	20,22	0x9C
3	6,8	14,16	0x98
4	8,10	8,10	0x07
5	10,12	18,20	0x79
6	12,14	28,30	0xAA
7	14,16	22,24	0xF1
8	16,18	16,18	0xBC
9	18,20	10,12	0x94
10	20,22	4,6	0xAA
11	22,24	30,32	0xE5
12	24,26	24,26	0xB9
13	26,28	18,20	0x6A
14	28,30	12,14	0x15
15	30,32	6,8	0xE0

Full Code for Extraction of the 6th Byte Key Value from 10 Round AES implementation.

CPA_against_6th_byte_Group5.py

Solution Results

1. After running above code (consisting of the AES last round operations) **to recover the 6th byte key value came out to be = 170**. PFB the resulting output from the code.

```
PROBLEMS 2   OUTPUT   DEBUG CONSOLE   TERMINAL

PS C:\VS Code Python Project> python -u "c:\Users\luv.ahuja\Desktop\IIT Kanpur\Q3\CS984 Hardware Security\Assignment\Assignment-1\CPA_against_6th_byte_Group5.py"
[['21A7D2BD873F22ECB0F4840E2BD69177' 'D9959FDF641778A83D3D45DC3E282F5D'
  116 ... 116 116 117]
 ['C0A06A4EE3A233EE582C28FFC83ABA0D' 'E412BCFE4AA01B0A2D382149AD3F2906'
  116 ... 114 114 114]
 ['07D3580C7695EC206ACB7476E361190C' '6BB092EF6950C6B0117FE63E7FAA5C6D'
  114 ... 114 113 116]
 ...
 ['8C88E806D1A495630494FA0BE0E14E31' '15A46DFB62A25F463ECA8A4E8F9291E4'
  116 ... 115 115 116]
 ['ABAA5DBDB09170C216CA9D2BC8080E2F' 'FA7C3ABF50C13DB1B9E3E6BDE56BAF84'
  115 ... 114 114 115]
 ['15135C3508DD752CBBA00D0F7D14D911' '990DCC7D385EAF77112F4AF324AF46EE'
  116 ... 116 117 117]]
170
```

2. Above code execution has inferred the plotting of the graphical representation also.

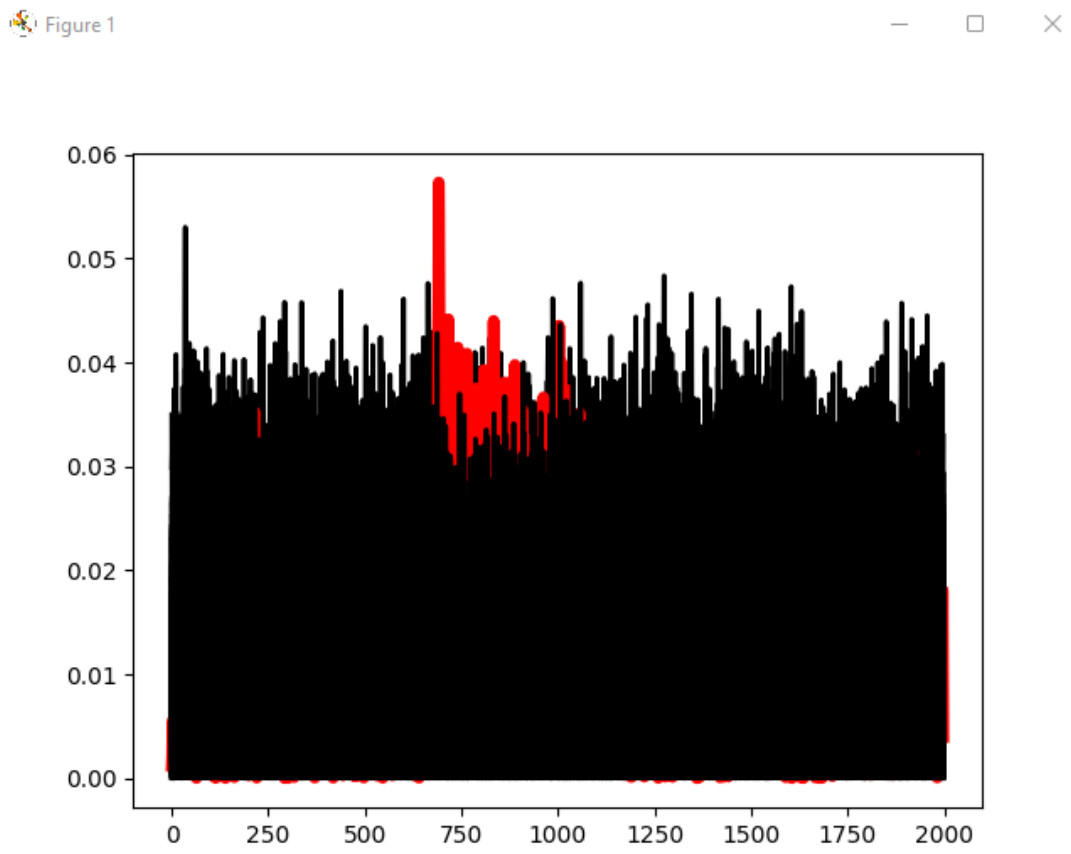

Figure2: Plot Representation