CRYPTOGRAPHIC ENGINEERING

REPORT ON DPA ATTACK IMPLEMENTATION

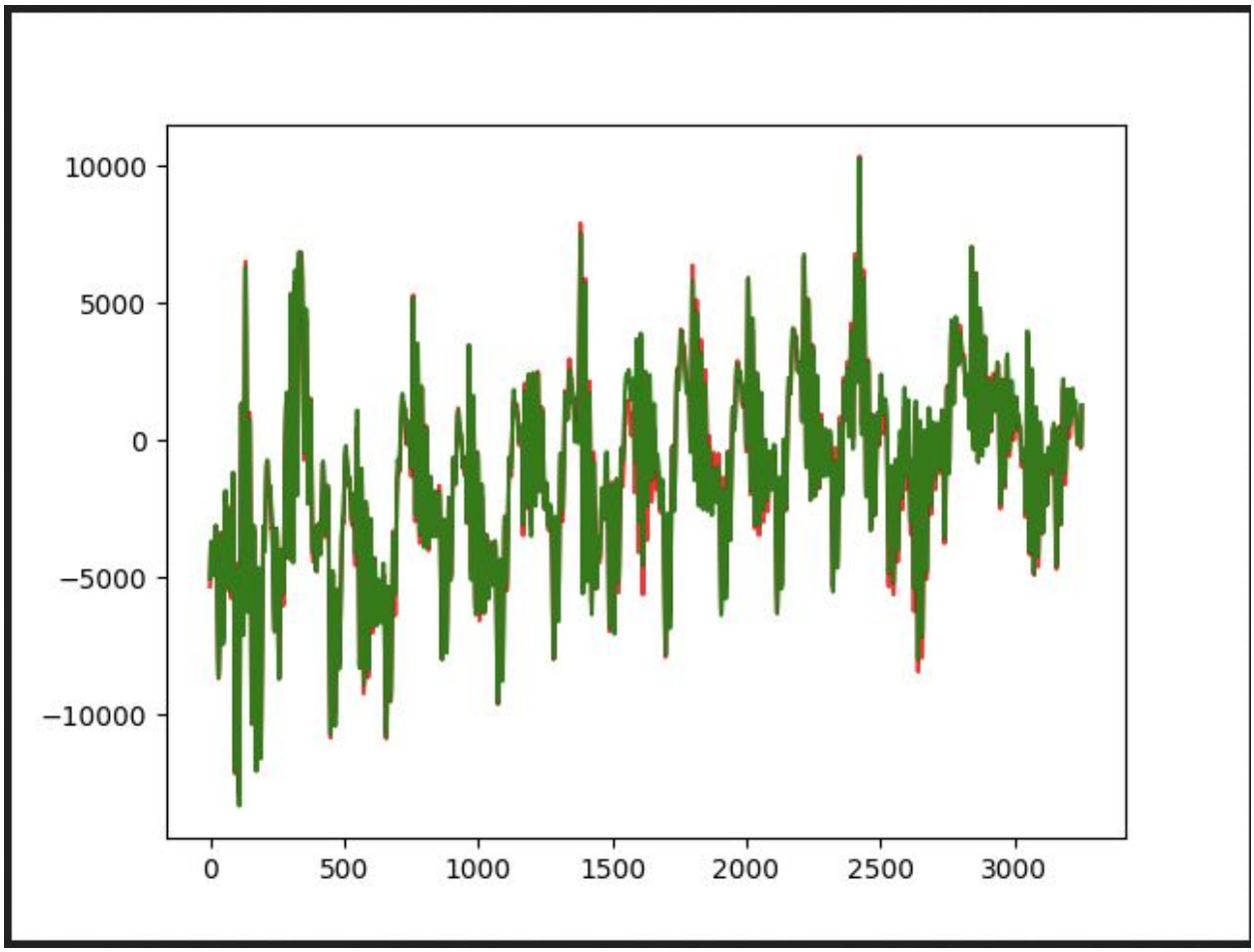

Figure1 : Power Trace Plots for 2 PlainTexts encryption using AES128 with the same Key.

Introduction

In this report we will cover the DPA attack implementation details and several experiments that were conducted. The goal is to use the power traces and try to extract the 16 byte key iteratively. We have referenced the codebase for DPA attacks on AES128 in github chipwhisperer repository. https://github.com/newaetech/chipwhisperer-jupyter

And the method is to use a separator function which partitions all the traces into 2 subsets depending on one fixed bit position. Here we have taken LSB for the experiment and if that bit is 1 its group of traces should on average have higher power consumption during the S-box operation than the other set. This is the main idea for partitioning the traces based on LSB.

Power Traces:

We have taken power traces from DPA contest website www.dpacontest.org/home/. Here we have taken data from contest v2. Now we will be talking about the structuring of the data downloaded. For each key we will have a folder with several CSV files and each CSV file contains power trace for that particular key and the name of the file contains the chosen plain text on which the power trace is captured.

(Eg.wave_aist-aes-agilent_2009-12-30_22-53-19_n=0_k=000000000000003243f6a8885a308d3_m=0000000000000002b7e151628aed2a6a_c=e6a636e30c85f35e980f3546a04daff7). Here the plain text input is 0x 0000000000000002b7e151628aed2a6a(Hexa Decimal Form) And in each csv file we have around 4000 timesteps and its corresponding voltage value.

ATTACK FOR AES FIRST ROUND:

We have used python implementation of this attack and Firstly the S-box mapping is stored in a python list. And the software implementation of the first round of AES is been done. This takes one byte input and one byte key and returns s[input ^ key]. The next step is to split the traces into two sets for each byte guess. So firstly we show how to implement for the first byte key guess and then the similar method follows for the next 15 bytes.

Now for first byte guess key:

We have to iterate on each guess that goes from 0 to 255 , And for each guess we split the traces in to one_list and zero_list where if the LSB output for the first byte from the AES first round is '1' append the trace to one_list and if its zero append it to zero_list. Now we take the average of one_list and zero_list and then take the absolute mean_difference for each

time step and the maximum is stored for that guess key. So the guess key with maximum difference is our final guess for that byte.

// This method is iterated for 16 times to guess 16 bytes of the key.

ATTACK ON LAST ROUND:

The last round of the AES is vulnerable to DPA attacks as it does not contain the mixcolumn operation. So, an adversary can xor the cipher texts with the key guesses. Then invert the subbyte and shift-row operations to obtain the output of the 9th round of AES. Then compute the hamming distance of this output with the cipher text.

In our approach, we considered bit values from 0 to 7 of this difference. Based on this value, we split the traces into piles. Then we considered the absolute difference between the average of the two piles. This is done across all key guesses. The key guess with the maximum value is the right value. Repeat this for all 16 bytes of the key. Then use this key to obtain the actual secret key by inverting the key expansion algorithm.

Run:

Install chipwhisperer from 1 and scared from 5. Then use jupyter notebook to run the ipynb files.

References:

1) https://chipwhisperer.readthedocs.io/en/latest/

2) https://github.com/newaetech/chipwhisperer

3) http://www.dpacontest.org/home/

4) https://eshard.gitlab.io/scared/index.html

5) https://scialert.net/fulltext/?doi=jai.2012.186.19