Lab 4: Side Channel Analysis Attacks Part 3: Differential Power Anaylsis

Igli Duro
Department of Computer Science & Engineering
University of South Florida
Tampa, FL

I. Introduction

For this lab, we evaluate differential power analysis (DPA) as an attack vector for SCA attacks. Power analysis is based off the notion that when different data is processed, even if the operations are the same will require differing amounts of power. A 1 bit will take more power than a 0 bit. DPA uses this knowledge and groups traces of data into bins based off if a particular bit is a 0 or 1. By averaging this data and taking the differential between them, areas of interest can be revealed. This lab was divided into three parts where we calculated the differential power analysis between traces, recovered a key for AES from a single bit of data, and recovering a key for AES from a byte of data.

II. Methods

This lab provided a ChipWhisperer Nano board which used a virtual machine and a Juptyer notebook to interface with it. We were given a set of of notebooks to use to set and program our CW Nano and work through some scenarios. These were notebooks Lab 6A, 6B, and 6C. Each of them has instructions and some partial code for guidance.

In Lab 6A we were tasked with grouping traces and calculating a differential power analysis. We programmed our CW Nano and captured 100 traces. When capturing traces the first bytes were converted to 0xFF or 0x00 depending if the least significant bit(lsb) was a 1. We then grouped traces based off this into two lists, one for 0 and the other for 1. These two groups were then averaged out and the difference was plotted on a graph. Afterwards we repeated the experiment and captured more traces but this time converted bytes to 0x0F instead 0xFF. This was in order to see if there was any significant different in power from a HW distance of 8 and 4.

In Lab 6B we were tasked with recovering the AES key from a single bit of data. This part reused the traces from Lab 6A. We were provided an sbox and built a function, aes_internal that XOR's input data and a key and uses the sbox as a lookup table for the substitution. We then created another function, aes_secret that calls the previous function passing along input data and a hard coded key. This key being 0xEF. We then generated an array of 1000 random integers between 0 and 255. The lsb of each value was then passed along to aes_secret and saved to another array leaked_data. This gave us an array of 0's and 1's which was then graphed. This array would be used to make a guess of the AES key. To do this we created a loop that would try 255 keys with a hypothetical_leakage list. This list would call aes_internal using the array of random numbers and each of the keys. They were then compared to the leaked_data array and the number of the same values was tracked. The correct key used out of the 255 possibilities would have all matching values. Results from this were then outputted. We then repeated the process but instead of just checking one bit, we modified and checked all 8 bits. Results from this were also outputted along with the top 5 matches.

In Lab 6C we were tasked with recovering the AES key from a byte of data and later all bytes of data. We programmed the CW Nano similar and captured some 2500 traces. We then wrote an algorithm for guessing the key, based off 1 byte of data. This algorithm used a similar strategy as in 6A where we grouped up trace arrays based off the lsb. We then averaged the groups, performed a differential on them, and took the max. Generating a SAD value. The idea here was that the correct key would have a noticeably higher value so we can find a best guess. Afterwards we worked on plotting the differences. This used a supplied function and we plotted the differences between bytes using 3 different keys with one of them being the correct one that was found in our guessing function. Then we modified the guessing algorithm to then work through all bytes of data and find the correct key for each byte. We were given a list of the correct keys to verify our guessing algorithm. Lastly we modified the simpleserial-aes.c file that was used to program the firmware and added a random delay. The effects this delay had will be talked about further in the results.

III. Results

A. Reading Check

1. *What are the primary differences between SPA, DPA, and CPA?*
The primary differences are that SPA looks at the power being used and can help identify an algorithm being used. DPA is a more advanced version that looks at the differential of power being used by sorting and averages into bins based a particular bit being a 0 or 1. CPA is similar to DPA, it looks at the expected shape the trace will take, but calculates a correlation coefficient between expected and actual consumption.

2. *In your own words, describe the process for the DPA attack.*
The process of a differential power analysis attack is sorting

traces into bits depending on whether a particular bit is 0 or 1. Afterwards, taking the average of the grouped up traces to remove outliers. And then taking the differential between the two groups to get a more accurate value for the power being used.

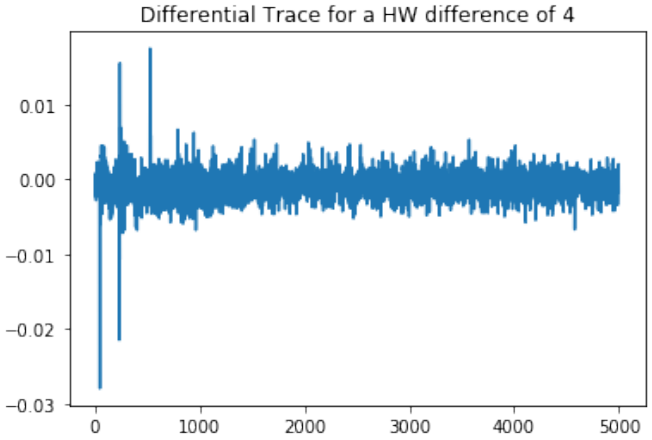
3. *What is one technique for mitigating a basic DPA attack? What are some considerations/limitations to this solution?* One technique for mitigating a basic DPA attack is introducing a random delay somewhere in the operation, this in turn causes issues with capturing and averaging subsequent traces. Introducing delay however does not work in all cases, and can be countered by manipulating data dynamically to re-align the traces.

B. Analysis of Results

For the results of Lab 6A we can see in the graph for the Differential Trace for a HW Difference of 8 there was a noticeable spike towards the beginning, meaning a larger difference of power occurred. In the graph for the Differential Trace for a HW Difference of 4 a similar spike occurs, but is less extreme.

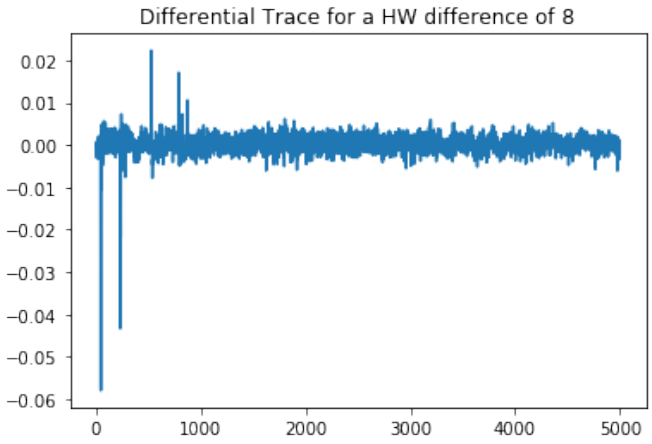
For the results of Lab 6B we can see in the graph for the Patterns of 0's and 1's that it spikes and dips consistently throughout the trace. Along with that AES_Guesser was able to guess the key for all bits of data.

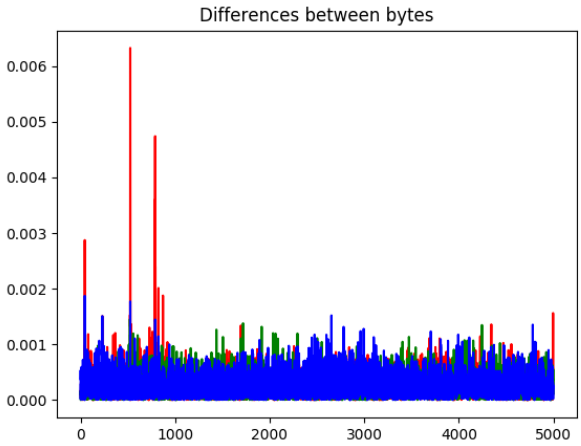
For the results of Lab 6C we can see the power differentials in bytes using 3 different keys, where the red plot was the correct key and the green and blue key were incorrect keys. The red plot/correct key had more extreme spikes from the other two indicating more power was being used. And for our AES_Guesser we were able to find best key guesses for each byte of data. These lined up with the list of correct keys we were provided. Once we ran the experiment for Lab 6C again but adding a jitter we can see that that our key guesses were all different. We'll discuss this in the next section.



IV. Discussion

As mentioned in Section III, adding a jitter delay to our experiment for Lab 6C our key guesses were all different. This completely thwarted our algorithm as is and would require

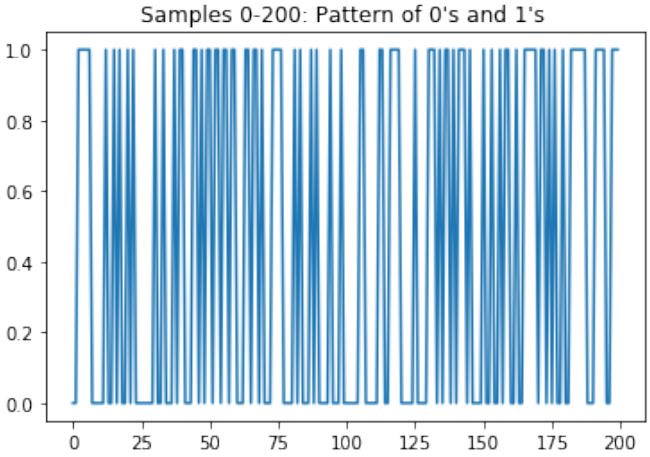




addition work and manipulating of data to correct our guessing algorithm. Without the jitter delay our algorithm was able to guess the correct keys. In Lab 6B we were also able to guess the correct keys by looking at bits of data. And in the graph for the pattern of 0's and 1's we can see a fairly even distribution. Lastly in Lab 6A comparing the two differential traces with a HW difference of 8 to a HW difference of 4 we see expected results as a larger difference would mean more 1's and therefore more power being drawn.

V. Conclusion

In this lab, differential power analysis was used to obtain secret keys from an AES function. This was done in three different parts. Where we did a DPA on traces, guessed the key used on a bit of data, and guessed the key used in a byte of data. We were successfully able to obtain the keys in the latter two parts and graphed the power differential in the first part. Afterwards we reran the last part but introduced a delay. The impact of the delay threw off our guessing algorithm and were unable to obtain the correct keys.



Attacking Subkey: 100% 16/16 [03:18<00:00, 12.42s/it]

```
Best guess 2B for subkey 0 : 0.00632520732510794
Best guess 7E for subkey 1 : 0.00632193512201118
Best guess 15 for subkey 2 : 0.006824519763175399
Best guess 16 for subkey 3 : 0.005734247325996145
Best guess 28 for subkey 4 : 0.00692025701926012
Best guess AE for subkey 5 : 0.0058401813750620785
Best guess D2 for subkey 6 : 0.006404160829848307
Best guess A6 for subkey 7 : 0.006411695458442002
Best guess AB for subkey 8 : 0.00624769638058037
Best guess F7 for subkey 9 : 0.006249305784959955
Best guess 15 for subkey 10 : 0.0057885107622734355
Best guess 88 for subkey 11 : 0.005941292593698452
Best guess 09 for subkey 12 : 0.0065424418057472705
Best guess CF for subkey 13 : 0.005612170056294247
Best guess 4F for subkey 14 : 0.0072878677275669684
Best guess 3C for subkey 15 : 0.006288810791100663
```

```
Best guess EB for subkey 0 : 0.03073372202623887
Best guess 39 for subkey 1 : 0.02201854550158693
Best guess D5 for subkey 2 : 0.021641671182808297
Best guess 0C for subkey 3 : 0.021484456261701687
Best guess 72 for subkey 4 : 0.0200363016871418
Best guess 72 for subkey 5 : 0.022843233805237104
Best guess EF for subkey 6 : 0.02157937972272029
Best guess 60 for subkey 7 : 0.01984869734041736
Best guess 88 for subkey 8 : 0.021096804205643672
Best guess 9B for subkey 9 : 0.019622607403394943
Best guess D5 for subkey 10 : 0.022510996970082922
Best guess 32 for subkey 11 : 0.02217071819603754
Best guess 99 for subkey 12 : 0.023731934346018863
Best guess C1 for subkey 13 : 0.019446956547340175
Best guess 64 for subkey 14 : 0.02165581579998413
Best guess 0D for subkey 15 : 0.019038311326062866
```