Practical Introduction to Hardware Security

Task 3: Power Analysis Side-Channel Attacks

v1.1

Chair of Dependable Nano Computing, Karlsruhe Institute of Technology
Instructors: Dennis Gnad, Jonas Krautter, Mehdi Tahoori

December 12, 2019

1 Introduction and Supplementary Information

In 1997 and 1999 two very powerful attacks on cryptographic implementations have been shown by Kocher et al. [1] and Biham et al. [2]. These are fault and power analysis side-channel attacks on cryptographic implementations, which work on the power supply voltage level, i.e. they are non-invasive. In this task we will explore power analysis side-channel attacks, more specifically using *Correlation Power Analysis* (CPA) — and with a small twist.

Side-channel attacks passively measure anything they can observe from the system, where we want to concentrate on power consumption, or rather the voltage level (which can relate to power consumption) [1]. So we just measure enough tiny fluctuations in the voltage supplied to the circuit. These tiny fluctuations can leak information about what the circuit is processing, since each transistor that changes its value in a device requires a certain electrical current, written as $i(t)$, current consumption over time. A small voltage drop can result from that used $i(t)$ and other electrical parameters in the device (electrical properties such as inductance L or resistance R in the power supply): $V_{drop} = i \cdot R + L \cdot di/dt$

This voltage drop can be measured, and with the means of the correct statistical approaches (signal processing approaches), the information of the secret key in a cryptographic circuit can be extracted. This is possible without being easily detected inside the circuit. However, typically many 'traces' have to be collected. Where each trace is the voltage fluctuations recorded during one single encryption.

For this task we suggest to first implement the analytical side of the power analysis attack based on CPA (Brier et al. 2004 [3]).

Here comes the twist we have in this lab: Instead of using traditional methods with external test and measurement equipment (oscilloscopes, etc.), which would be too expensive for every student in a lab, we use the FPGA's own internal resources, which is also documented in one of our publications [4].

1.1 Examination

We will first examine the software implementation of CPA and later look into your FPGA implementation to collect the necessary samples. You can work in a team of two, but do not split up the tasks into one person doing CPA and the other FPGA. Both should do both.

The examination will be handled as before: Please send us the respective software. Later on we will ask you for explanations of your code in the lab.

The deadlines will be handled stacked as follows:

- Januar 16, 2020: Hand in Task Part A (Software CPA)
- Januar 23, 2020: Hand in Task Part B (Data collection on FPGA + CPA on your own collected traces)

1.2 Questions

As for all tasks before, if you are stuck, please let us know on any questions. But as computer science master students you should be able to solve standard programming questions yourself (for example using python package documentation, standard syntax questions). Please also do not ask us for help on the correlation results before you have looked at the correlation in a visual form, as noted below.

2 Part A: Implementing CPA

As of here we will not provide you with much more additional details as we have given you in the small introductory lesson. Most details can be read in the original CPA paper by Brier et al. [3], and we suggest to take a look at the website of NewAE [5], which give some more hands-on guide how to perform basic CPA. Additionally to that, please apply the bitwise correlation model as explained in [4]. The shown bit selection can be applied to both the ciphertext or the plaintex-based model. Example data will be provided by us, which you can use to develop the algorithm.

Here are a few additional notes:

- It is sufficient to show the attack working on one of the bytes, and it is sufficient if you can show 2-3 bits that correctly correlate of that byte.
- For the data we give you, the correct key should start correlating obviously after about 2000 traces, so for the start you might only want to work with 3000 traces to reduce waiting times.
- For the FPGA later on, you probably have to use about 50,000-100,000 (=‘100k’) traces of encryptions for successful key recovery.
- Correlation plots usually mark the trace of the correct key candidate in red/black and all others in grey. For example, see the progress plot in Figure 2 or another completed correlation plot in Figure 2. You can check for example the paper in [4] to see other examples of a final correlation or a progress plot.
- You can first try ciphertext-based correlation through an inverted SBox, or plaintext-based correlation through a normal forward SBox.

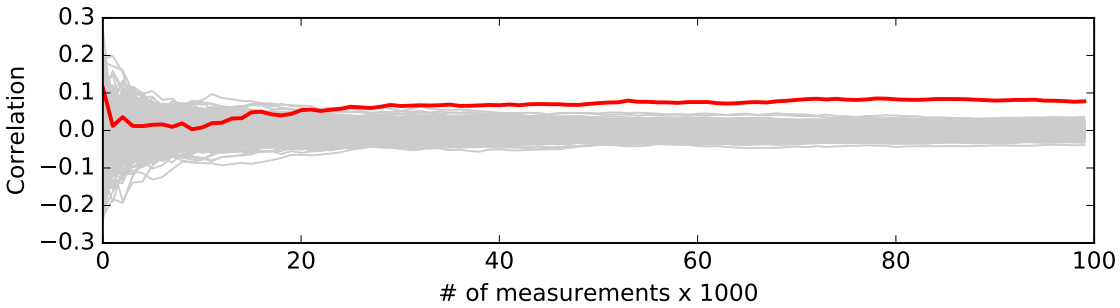

Figure 1: Example correlation diagram showing the progress of correct key bit of a ciphertext-based correlation, distinguishable after about 40-50k traces. In another sort of diagram you can look at the final correlation.

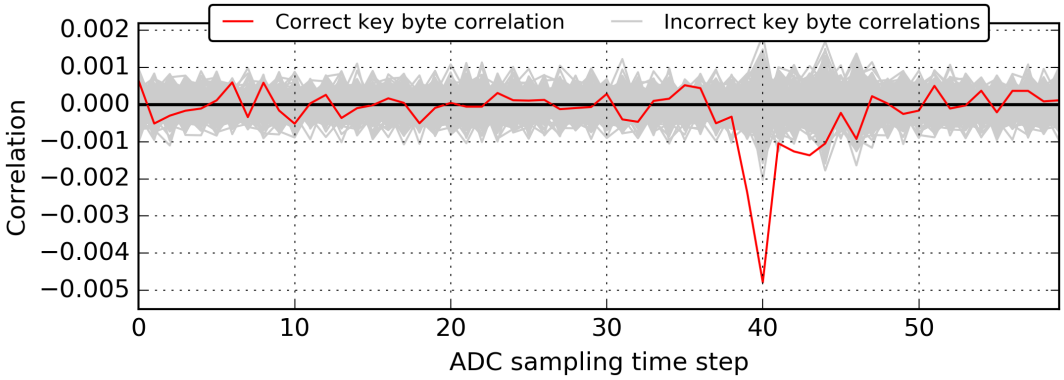

Figure 2: Another example correlation diagram of an AES implementation. The correct key byte correlates fine at an ADC sampling time step of 40

What we expect in your solution for CPA:

- Finally you should have a known-ciphertext-based correlation (meaning you correlate for the respective roundkey). Initially, a plaintext-based one might be easier to implement.
- Ability to plot a progress plot and/or final correlation plot, this can also help you debugging in the process of development.
- Sufficient speed to later be able to work on 100k traces, meaning that for the example traces, the correlation of one byte (with all the 8 bits in parallel) should only take a few seconds. You will need to use multithreading/tasking in one way or another.

3 Part B: Implementing the collection of measurements

The design we provide as a basis for your implementation is already very extensive. It contains all required module instantiations and signals:

1. A BRAM instance which will be used to store sensor values.
2. A sensor based on a delay line to capture the propagation of the clock signal.
3. A decoder to decode the propagation as a 6-bit integer value.

4. A state machine with the following behaviour:
   a) Wait for 16 bytes of plaintext to be received on the UART module.
   b) Perform the encryption of the received plaintext.
   c) Send the 16 bytes ciphertext via UART.
   d) Send 56 bytes of sensor data from the BRAM (which should be enough to capture an AES encryption round) via UART.

Your task is to fill the BRAM with sensor data in *sense_module.v*. Connect the correct signals, activate the writing at the right moment and adapt the BRAM address appropriately. Depending on whether you want to attack the first or the last AES round, you may use the start of the encryption or the *aes_lastround* signal for synchronization.

Moreover, you most certainly have to adapt the initial sensor delay for your specific FPGA in *latticesense.v*. Check the values your sensor delivers: If the values are constant 0, you will have to decrease the *initlen* parameter of the module, whereas *initlen* must be increased, if the values are constant 63. Recompile your design every time you change *initlen* and verify if your sensor values are in a proper range.

Finally, you have to implement a host program to send random plaintexts to the FPGA and collect sensor traces. We provide *queryCipherSense.py* to quickly check the correct function and value range of the sensor, but you need to collect about 100000 traces for a successful CPA. It is recommended, that you use the same data format as in our example data, in that case we may also be able to help you, if you encounter any issues. The data is split up in two CSV files, one containing the key and the respective plaintext in hexadecimal format in each line and one containing the sensor values for the encryption of that plaintext. Check the provided example data or ask us, if anything is not clear.

4 Hints

This chapter will probably be extended over time. So check back regularly. Please also see generic hints in `FPGA_basics/common_errors_faq.pdf`

- It is reasonable to use python for all scripts. Data can be stored in simple .csv files.

For example this saves data from a csv file row-wise into ”rows”, and then plots each row as one trace in a single diagram:

```
#!/usr/bin/python
import numpy
rows = numpy.genfromtxt("filename.csv",delimiter=',').transpose()
plt.plot(rows)
plt.savefig("plot.pdf",dpi=300) # save to a file
plt.show # interactive
```

- If the traces collected from your FPGA do not correlate properly, but from the examples they do, you can try the board from another student or contact us. Some of the FPGAs leak less than others.
- The best correlating bit in the example traces is: Last Round (meaning ciphertext-based correlation), 1st Byte, 2nd Bit

References

[1] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual International Cryptology Conference*. Springer, 1999, pp. 388–397.

[2] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Annual international cryptology conference*. Springer, 1997, pp. 513–525.

[3] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2004, pp. 16–29.

[4] F. Schellenberg, D. R. Gnad, A. Moradi, and M. B. Tahoori, "An inside Job: remote power analysis attacks on FPGAs," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2018*, 2018.

[5] NewAE Wiki. (2018) Correlation power analysis. [Online]. Available: https://wiki.newae.com/Correlation_Power_Analysis