University of Stuttgart
Germany

Lab Course Side-channel Analysis on FPGAs

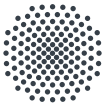
WP3.1.2

Practical Correlation Power Analysis Exercise Sheet

by Maël Gay

La Région Auvergne-Rhône-Alpes
Erasmus+
Co-funded by the Erasmus+ Programme of the European Union

Prof. Dr. Ilia Polian
Dr. Maël Gay
Hardware-oriented Computer Science



University of Stuttgart
Germany

Hardware-oriented Security
Solution of Exercise Sheet 1

Exercise 1

In this exercise, we will program the SAKURA G board, which is composed of two FPGAs, one for communication and another one for data processing, with a Small Scale AES 444 (SSAES444) implementation.

In the case of the communication FPGA, the smaller Xilinx LX-9, it is already pre-programmed with the required input-output interfaces for USB and internal communications. Please do not modify this FPGA. If you are interested, the configuration files are however provided at the following link **[insert download link here]**.

The main FPGA, the Xilinx LX-75 one, will be used for encryption. The top module "sakura_g_sc_aes_444.v", the interface to the communication FPGA "host_if.v" and the constraint file "sakura_g_sc_aes_444.v" should not be modified. You can should have a look at those file to understand them. The file "sc_aes_444_table_ecb.v" is the one that you'll need to modify in order to implement your SSAES444.

Open the Xilinx ISE software and create a new project. In this project, load all the source files available to you and modify the "sc_aes_444_table_ecb.v" by adding your SSAES implementation. Make the appropriate changes to match the given inputs and outputs.

Once your implementation is ready, make sure the JTAG programmer is connecting to the correct port on the board (that is the one for the LX-75 FPGA). Then, launch the Impact software from within ISE and program the FPGA with your design.

You should now have a working SSAES implementation on the SAKURA G board. Download and open the C++ Visual Studio solution available at **[insert download link here]**. This is used to control both the FPGA and the oscilloscope for power measurements.

Connect the oscilloscope's probes to the board. That is to say the power measurement probe on the J3 port of the board and trigger probe on one of the output pin of the Xilinx LX-75 FPGA. Ask your supervisor if you are unsure.

Once everything is connected, you can modify the "main.cpp" file, and more precisely the section "Oscilloscope parameters", to fine tune the area you want to measure. Please ask your supervisor for verification of those parameters. This may require several test runs. The goal being to fit a whole encryption on the screen of the oscilloscope.

Finally, once you have the approval of your supervisor, proceed to measure a sufficient amount of power traces.

Solution:

Please refer to the solution files and the associated READMEs in the corresponding archive for the correct code.

Exercise 2

Publishing date: Month Day, Year

The goal of this exercise sheet is to perform a Correlation Power Analysis (CPA) on a Small Scale AES implementation (SSAES). For this purpose, we will use Matlab and modify an existing script.

First, to perform a CPA, some power traces are necessary. They are available for download at **[insert download link here]**. The power traces were recorded using a Teledyne Lecroy WaveSurfer 3024z oscilloscope, measuring the power consumption of an SSAES implemented on a SAKURA-G FPGA board (Xilinx Spartan 6 LX-75). The archive contains the necessary Matlab scripts, a CSV file of the power consumption traces and two text files of the corresponding plaintext/ciphertext.

The Matlab scripts load_io.m and load_traces.m are used to load the data. The last script, cpa.m, is the one that you should modify to perform the CPA attack. The first part of the script loads the data using the provided functions. You should not modify the script before the section delimited by the comment: **"Exercise 1 - Area Selection"**.

From this section and until the next one (i.e. comment **"Exercise 2 - Key Recovery"**), is where we will select the area to attack in order to facilitate our CPA, as the first step in any power analysis is to identify the appropriate portion of the data. The provided traces are already centred on one SSAES encryption. You can uncomment the provided example of trace plotting to display one complete power trace.

What can you identify on the recorded power traces ? According to what you identified and/or by referring to the lecture slides, suggest a portion of the traces suitable for a CPA. Validate this area with your supervisor and then modify the "offset_initial" and "segmentLength" variables below the **"Initial CPA area selection"** comment to restrict the trace to this area. The reduction is already performed by the code below those variables, until the next section.

As a quick reminder, the goal is to recover the master key. With this in mind, you should focus on an area of the trace where the key is used. In other words, you should focus on an intermediate value of the encryption that you are capable to compute (with some key guesses).

Once the selection was validated by your supervisor, you can proceed to the next exercise, in which you will implement the actual CPA.

Solution:

Please refer to the solution files and the associated READMEs in the corresponding archive for the correct code.

Exercise 3

In this exercise, we will implement a Correlation Power Analysis (CPA). Below is a small reminder on the required steps to perform a CPA. Please refer to the lecture slides for a complete explanation on CPA.

For each key byte, the attack steps are:

1. Make a guess for the byte
2. Compute the corresponding intermediate result for every encryption
3. For each data point, compute the correlation coefficient between the modelled and measured power consumption.

4. Recover the key byte candidate: highest correlation (e.g. extract the maximum)

To implement the attack in the provided Matlab script, modify the script in the section **"Exercise 2 - Key Recovery"**. The required steps are also provided as comments in this section of the Matlab script.

The master key for the encryption was "FEDCBA9876543210". With your implementation of CPA, did you manage to recover this key ? If yes, verify your implementation with your supervisor. If not, do not hesitate to ask your supervisor for some guidance.

You have now recovered the correct secret key. What improvement to your implementation of a CPA can you suggest ?

Solution:

Please refer to the solution files and the associated READMEs in the corresponding archive for the correct code.