INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Referent: *Vincent Migliore*
vincent.migliore@insa-toulouse.fr

Hardware Security Lab - Introduction to Simple Power Analysis Side-Channel Attacks -

Objectives

This practical work aims at discussing the exploitation of a vulnerability reliying on side-channel attacks based on power or electromagnetic field observation. In particular, we will evaluate the security of a digicode implementation.

1 Introduction

The increasing adoption of cyber-physical and connected objects promises to address a large variety of societal challenges, from simplifying human communications (with legal authorities, public and private communications) to supporting next generation industry (so called Industry 4.0), smart cities and transportations. Since these computing objects become more and more integrated into our daily lives, their physical access is also easier for malicious users. However, such systems, in particular cost-driven ones such as Internet-of-Things (IoT) nodes, are vulnerable to a wide range of cyber-attacks. In addition to usual networked systems' threats, computations are vulnerable to a large range of physical attacks, which exploit some characteristics of the target system including both software and hardware. Among these, side-channel attacks (SCA) can infer secret information from physical observations measured during the execution of sensitive computations. SCA typically exploit physical quantities such as electromagnetic signature and power consumption, behaviour of cache memories, and effects due to speculative execution. SCA have gained momentum with the increased use of cryptography because they represent, with fault injection attacks, the most effective way to break cryptographic implementations. For example, the AES encryption, which is widely used in most computing systems nowadays, is considered secure by traditional cryptanalysis but is highly vulnerable to side-channel attacks. SCA based on power and electromagnetic observations are particularly harmful because they have been successfully applied many different kinds of secured computations; furthermore, they are non-intrusive, hence difficult to detect.

In this practical work, we will attack a target with a Simple Power Analysis (SPA), i.e. an attack based on the observation of power consumption traces without particular statistical analysis. A recent system with a minimal level of security should not be vulnerable to such kind of attack nowadays.

2 Target definition

In this practical work, the target system will be a digicode, i.e. the equivalent of an electronic lock. Instead of a physical key, to unlock the door, a digital key (called a code) is required. A digital key is a sequence of symbols, generally integers from 0 to 9. An example of a digicode is shown in figure 1. In general the number of symbols is fairly small, i.e. 4 to 8 symbols, to be easily remembered. Once a user type a code, the value is compared to a reference (stored in general into the device memory), and only unlock the door if the two codes match.

Figure 1: Digicode example.

3 Evaluation platform

All the experiments will be made on the Chipwhisperer workbench which is a complete platform for power based side-channel attacks evaluation (and glitches, but it is not part of this practical work). A photo of the chipwhisperer workbench is proposed figure 2. The workbench is composed of the following part:

A. The ChipWhisperer Lite (CWLite). It is the device which interfaces a personal computer (that will be used to set the experiments) and the Target (i.e. the device that will be attacked). The CWLite is equipped with an ADC to capture the power consumption of the target.

B. The UFO BaseBoard (in red) and its daughter board (in blue). The UFO BaseBoard is the common board that is used to interface the ChipWhisperer Lite with the daughter board. The daughter board itself is composed of the target (a stm32f3 in this case) and is connected to the BaseBoard with several common interfaces (clock, reset, jtag, gpio, power supply, ...). The daughter board can be changed to target another SoC. The Chipwhisperer Lite provides a daughter board for an XMEGA, but other boards exist (for example, one is equipped with an FPGA, or a RISC-V).

4 Software interface

The practical work sources are located in the moodle and named **Ressources**. The archive is composed of several files and folders. You must unzip the file before any experimentation.

Once unzipped, the folder must contain: a **bin** folder, which contains the python script to communicate with the chipwhisperer as well as the target programming files. A **firmware** folder which contains the source code of the digicode implementation and an empty source code for the hardening phase of the practical work. A **target_prog** folder containing source code files required to compile a code to a given target.

To communicate with the target, you will use the **server.py** python script inside the **bin** folder. To establish the communication and program the target, the command is **./server.py -f *filename***, replacing *filename* with any .hex file located in the same folder as the python script. If you omit the **-f** parameter, you will skip the Target programming and use the existing program. For information, .hex files contain the digicode firmware stored in textual hexadecimal format. For the first experiment, you can load the *test.hex* file.

Once the communication is established, the symbol > will be displayed in the terminal: you are ready to test some codes.

A standard command is composed of the Four digit number tested, followed by several options. Below the available options:

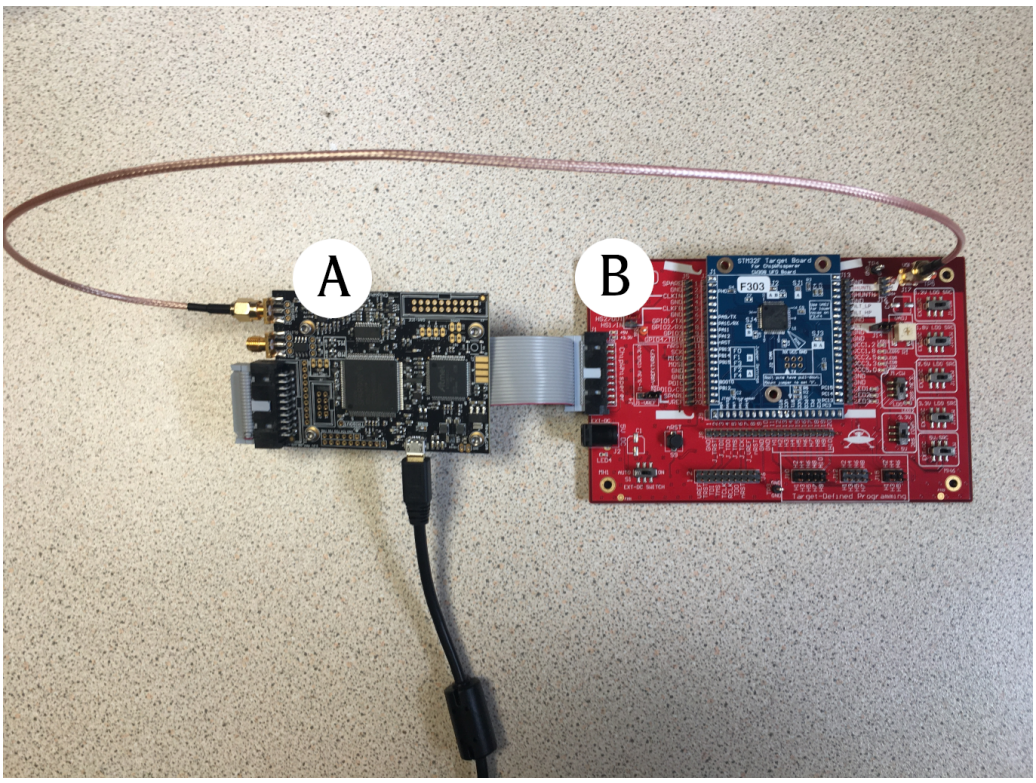

Figure 2: Presentation of the Chipwhisperer Pro.

-c : Capture the power consumption of the target.

-p : Plot the power consumption of the target. Requires **-c** option.

-a : Enter in interactive mode. In this mode, the previous figure is not cleared which allows to see the evolution of the curves. Requires **-c** and **-p** options.

-t : Display the numerical value of the power consumption. Requires **-c** option.

-s (**VERY** optional) : Set the number of samples captured. The default value is 500.

For example, if you want to test the code 1234, capture the power consumption, plot the result in interactive mode and set the number of samples to 1000, the command will be:

1234 -c -p -a -s 1000

or

1234 -cpa -s 1000

If you want to clear the figures in interactive mode, instead of typing a code value, type **clear**.

5 Experiments

The experiments are split into three labs: a first lab will be dedicated to the security assessment of a digicode implementation. A second lab will be the hardening of the digicode implementation. A third lab is the automatisation of the attack using a python script. Notice that there is no imposed order for lab 2 and 3, but lab 2 is more fundamental than the second.

5.1 Security assessment of an implementation

To evaluate the potential vulnerability of the digicode program, you have at your disposal several files:

firmware/base/main.c: This file written in c contains the main loop of the digicode program. The analysis of this file is not mandatory, but can help you understand more precisely how the communication between the chipwhisperer and the target is performed.

firmware/base/verify_code.s: This file written in assembly contains the verification function used by the main loop to verify the code typed. This is the main file that you should work around.

bin/test.hex: This file is a test program (in hex format) that is very cloed to the final challenge but with unlimited tries and a code set to 9999. Use the fact that you know the secret code for your security assessment.

We suggest the following steps for the security evaluation:

1. Analyse the **verify_code.s** file and evaluate the computation time in different scenarios (linked to the number of good digits).
2. Make an experimental protocol to show the vulnerability using the **test.hex** implementation.
3. Verify the vulnerability on a real target.

Once the vulnerability exploitation is understood, you can try to break other implementations with unknown code (*secret-digicode-X.hex*, with X from 1 to 4).

5.2 Hardening

Now you are ready to harden the code against the vulnerability shown previously. In the **firmware** folder, you have a base bone source code into the **hardened-digicode** subfolder. You only have to implement the countermeasure in the verification function described in **verify_code.s**.

To compile and test your program, at the root of the practical work folder, a Makefile is given. To compile your program, type **make**. This will compile the firmware, store it into the **bin** folder and convert it to textual hex format. Now you can test your code using **server.py -f hardened-digicode.hex**.

5.3 Challenge

Now we will try to design a script to automatically perform the attack. A base bone of script called **attack.py** is proposed in the **bin** folder. You have to adapt the script to perform the attack, exploiting the weaknesses observed in the test program. Note that you only have to modify the script where there is a comment (which starts with #). Once completed, you can try your script with the several implementations provided. To change the target implementation, modify the header of the attack script.

Good luck!