Practical Work : DPA Analysis of an SBox Using Simulations

I. INTRODUCTION

The security of a cryptosystem depends on the specification of the encryption algorithm, but also on the actual implementation. In particular, most effective attacks usually exploit some implementation weaknesses. We will show how a power analysis attack can break the initial operations implemented in the Advanced Encryption Standard (AES).

The Substitution Box is the main non linear component of the AES. It is a byte substitution and it is the first operation computed after the initial key addition. Hence, its result at the first iteration depends only on the values of the initial plain text and of the secret key.

The Differential Power Analysis (DPA) is a powerful attack technique that uses side channel information (namely, data related to power consumption) in order to validate assumptions on the key value. If the cryptographic device is not implemented thoughtfully, then this attack has good probability to succeed. In fact, due to the characteristics of the CMOS logic, the power consumption of the implemented logic is often heavily dependent on processed data, if no specific countermeasures are adopted (e.g., implementation of balanced execution flows).

In this session, we will attack a basic implementation of an AES S-Box. The circuit under analysis computes the modulo-2 addition between the input and the secret key; the result is thus used as the S-Box input.

II. DPA ANALYSIS FLOW

The DPA Analysis flow used in this study is divided into three main steps (as shown on fig1.):

- The creation of the transistor-level description of the design
- Its simulations
- The analysis of the simulated current profiles

1) Creation of transistor-level model

To be able to simulate accurately our design, we have to get to a lower level of description. For this study, we will simulate the transistors of each gate of the description. No Place&Route and parasitic extraction is done here. The generation of this transistor-level description requires different steps.

The first step is the structural synthesis of the design with Mentor Graphics Leonardo (other ASIC oriented synthesis tools also work). See III.A for details about its utilization. The technology used for the synthesis is *AMS c35_CORELIB*. We need to generate the synthesized netlist in Verilog. VHDL version of the netlist can also be generated to simulate it with Modelsim.

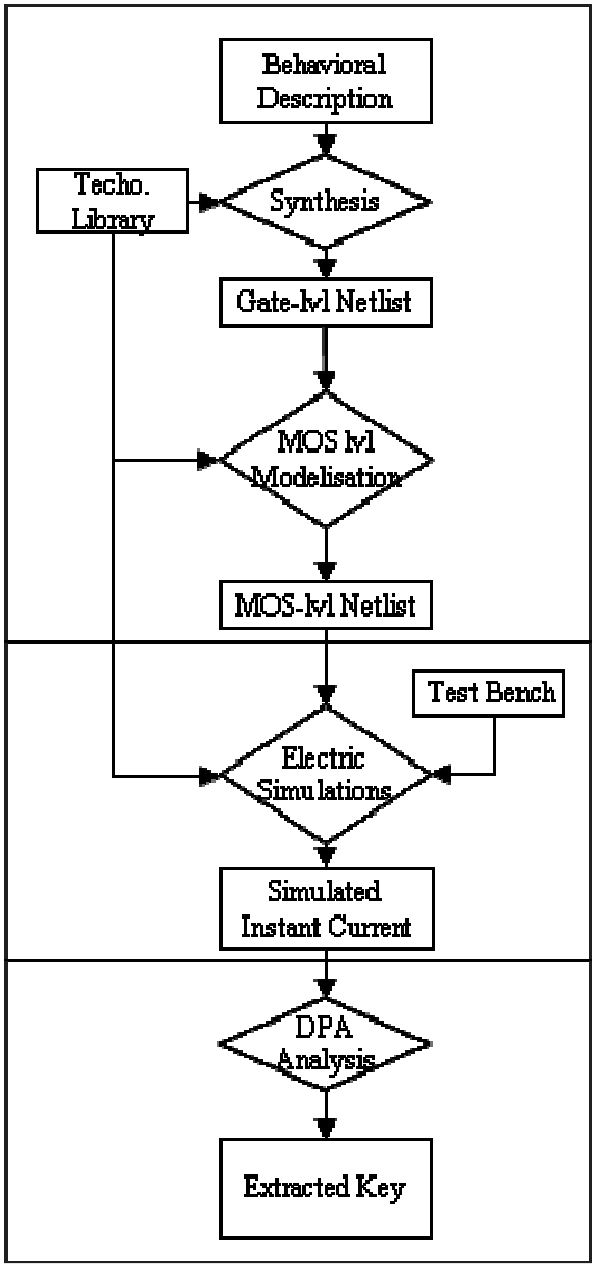

Figure 1 : DPA Analysis flow chart

The second step is the creation of the HSpiceD model of the design. This is done in the Cadence environment with the AMS C35 design Kit. First, we have to create a new library where the Verilog gate-level netlist previously generated is imported. Then, we will have to generate the netlist for HSpiceD simulation and save it as *spice/SBox.spi*. See III.B to understand how to use the useful functions of Cadence tool for the practical work.

The final step of the creation of the ready-to-use transistor-level model (*spice/SBox_Mod.spi*) of the SBox is adding the definition of the VDD voltage in the netlist. For that, just add the line `VVdd vdd! 0 5.0v` before the line *.END* .

2) *Electric simulations*

Once we have the ready-to-use transistor-level model of the design, we can feed an electric simulator with it. We have chosen Synopsys Nanosim, which is a fast electric simulator (much faster than Spice). In order to obtain the current profiles of the design with given stimuli, we have to supply some extra data: a vector file, which contains the stimuli, and a configuration file, which indicates to the simulator what to do (especially which values to measure). These two files have to be given to the simulator with the transistor-level netlist. Output files obtained from Nanosim are some kind of container. A post processing is necessary to convert the needed current waveform into a format easily read by a C application. More details are given in III.C.

3) *DPA Analysis*

This is the final step of the study: the goal is to find the secret key which was used during the previous simulations. This also can be done with actual measures on the chip to attack it. In our flow, this step is realized with a homemade C application. This tool returns the key found by the algorithm on the basis of the provided data.

III. CAD ENVIRONMENT FOR THIS SESSION

Each CAD tool used in this session also needs its configuration script. They are in your project directory, *shrc_**. But, you won't have to source them manually; the script *script_InitTP.sh* does it for you. Just source this one.

A. *Mentor Graphics Leonardo*

Leonardo is available only in script-mode, which means that you will need to generate the netlist from the command line. Run the following command line in order to get the synthesized netlist in Verilog format:

spectrum vhd/SBox.vhd synth/SBox.v -target=c35_CORELIB

There is no need to configure advanced options. You can also generate a VHDL netlist to perform post-synthesis simulations in Modelsim.

B. *Cadence with AMS c35 Design Kit*

To start the Cadence environment, start the command *amsc35b4*. At the startup, several windows open as shown in figure 3: the main window is the little one at the bottom of the screen. A dialog box asks you to choose for technology options, choose the option already checked (C35B4C3) and validate.

The Library Manager, already opened on the figure 3, will probably be closed at your first startup. To make it appear, just go to *Tools->Library Manager* in the menu of the main window. In this library manager, you will have to create your own library: *File->New->Library*. This opens a new window asking you the name of your library *SBox_AMS_CDS*, then another dialog box appears to select the associated technology: choose an already existing one, then browse to find *Tech_c35b4*.

The importation of your *Verilog* design will be made trough the *File->Import->Verilog* dialog box in the menu of the main window. Figure 4 shows this dialog box. You have to set some values in it: *Target Library Name* to *SBox_AMS_CDS*, *Reference Libraries* to *sample basic CORELIB*, *Verilog Files to Import* to *synth/SBox.v*, *Power Net Name* to *vdd!* and *Global Nets* to *vdd!*.

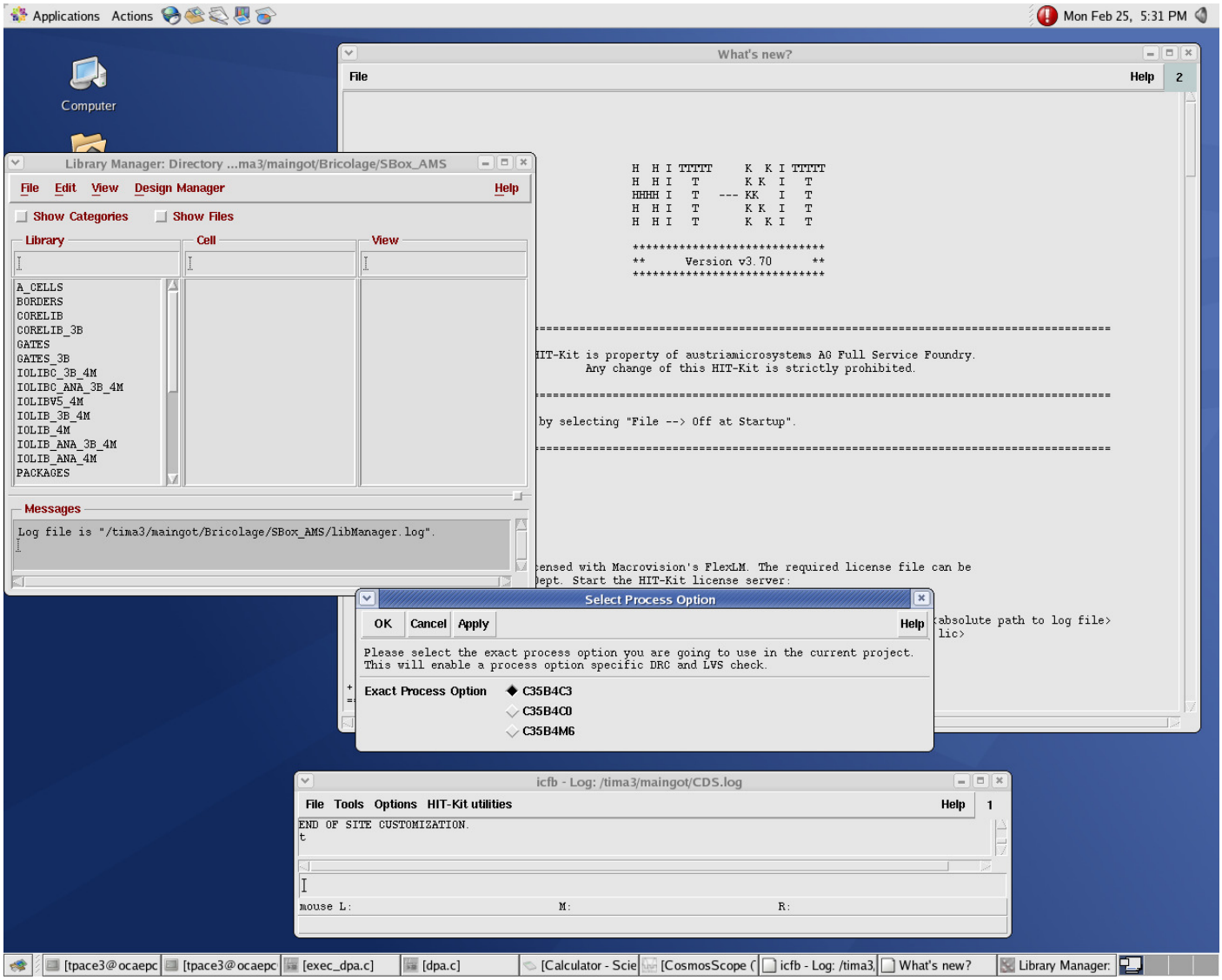

Figure 2 : Cadence starting windows

To generate the transistor-level netlist, your will have to start the simulation tool *Tools->Analog Environment->Simulation*. In this new tool, you have to first set the simulator to *HSpiceD* in *Setup->Simulator/Directory/Host* and to load your design (found in your library *SBOX_AMS_CDS*) in *Setup->Design*. Then you have to generate the netlist with *Simulation->Netlist->Create*. This will open a new window with your netlist; you just have to save it as *spice/SBox.spi*.

Remember: **the final step** of the creation of the ready-to-use transistor-level model (*spice/SBox_Mod.spi*) of the SBox is adding the definition of the VDD voltage in the netlist. For that, just add the line `VVdd vdd! 0 5.0v` before the line *.END* .

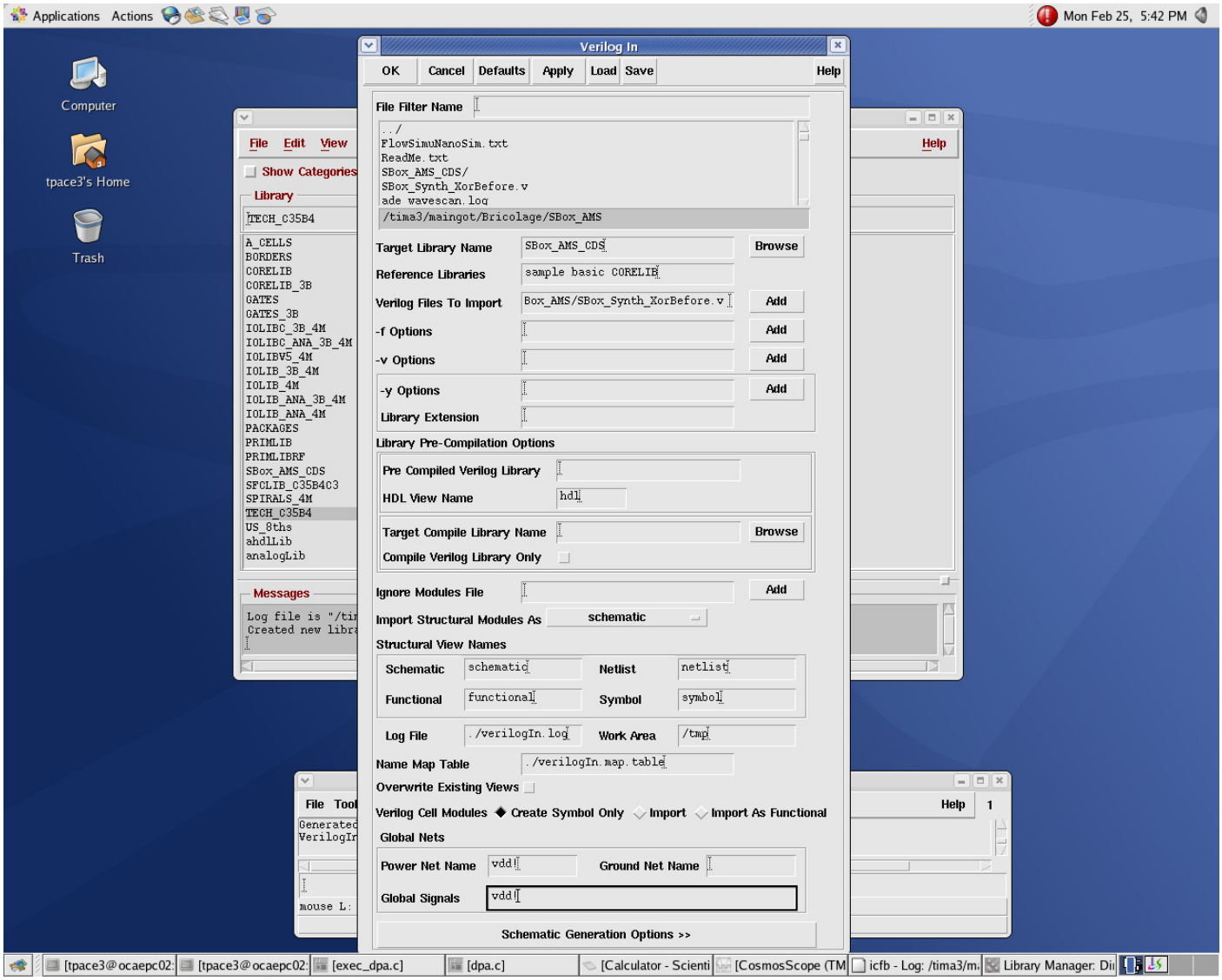

Figure 3 : Cadence - Import Dialog Box

C. Synopsys Nanosim

Nanosim is a fast electric simulator. It needs several files in input to operate properly: the transistor-level netlist, the stimuli file and a configuration file. Some examples can be found in *spice/SBox.vec* and *spice/SBox.cfg* in your project directory. Using these files, the simulation can be run by the following invocation:

nanosim –n spice/SBox_Mod.spi –nvec spice/SBox.vec –c spice/SBox.cfg –out fsdb –t 21ns –o simulations/trace

This command launches the *21ns*-long simulation of the *spice/SBox_Mod.spi* transistor-level netlist using stimuli defined in *spice/SBox.vec*, the configuration from *spice/SBox.cfg*; it will output the file *simulations/trace.fsdb.*

In the project directory, a shell script *script_RunNanosim.tcsh* will run the simulations for each couple (Key,Message) defined in *input_simu_Key.txt* and *input_simu_Mes.txt*. The simulation of each 256 possible

messages for one key takes around 10 minutes (so don't launch the exhaustive simulation during your practical work without modifying the file *input_simu_Key.txt*, it will take more than two days!).

The output file *simulations/trace.fsdb* is a binary file that contains all waveform requested in the configuration file. This waveform container can be visualized using CosmosScope (command *scope*). When opening a plotfile, two new windows appear: the opened plotfiles list and the list of waveforms in each plotfile. Using this graphical tool, it is easy to plot the graph of the current (i(vdd!)), and save it as a text file. This can also be automatically done with the shell script *script_Convertfsbd2Txt.tcsh*. Start the script and wait for the conversion process to complete.

D. *DPA Tool*

The tool is started with the command *exec_dpa*. If the executable file is not present, then you can build the application executing the *make* command, which will compile all the necessary files.

The program accepts two parameters: first index of the bit that will be attacked; second, the path to directory storing all the simulation files (terminated with the / character). It must be started in its local directory.

The tool reads a configuration file (*input_dpa_Mes.txt*) containing the list of the vectors that will be used for the DPA. Then, all the specified files are read and simulation data are collected. For each key guess, it partitions the simulation traces of each message according to the value of the output bit that is attacked. Finally, it evaluates the difference between the averages of each partition and chooses the highest value.

IV. WORK

- Simulated the SBox to understand its functioning (*ModelSim*)
- Synthesize the SBox (*Leonardo*)
- Generate transistor-level netlist for electric simulations (*Cadence*)
- Simulate few traces with different stimuli & compare (*Nanosim, CosmoScope, SBox.vec*)
- Generate all traces for a key you have chosen (*input_simu_Key.txt, scripts*)
- Find the keys used for your sets of traces (*DPA tool* on *simulations/UnknownKey<**Group#**><**A-E**>*)
- Try to change certain parameters of the DPA algorithm to see if it gives the same key (bit_to_test / traces_used) (*exec_dpa, input_dpa_Mes.txt*); test all the bits for at least a couple of unknown keys