Lab 4 Handout

*Hardware Attacks Countermeasures on AES*

# Introduction

In previous labs we saw the power of modern hardware Attacks. Specifically:

* Side-channel Attacks take advantage of the power leakage of an embedded device. By proper power trace acquisition and analysis, we were able to extract part of the secret key of a cryptographic mechanism.
* Differential Fault Analysis induces a fault injection to a computation, resulting in faulty output. Then through the proper analysis, we were able to extract part of a round key.

Thus, the introduction of countermeasures becomes a vital aspect of developing embedded applications. In this lab, we are going to implement simple solutions that lower the effectiveness of such attacks.

# Project Setup

For this lab, we will use the **tinyAES code**. The development platform we will use is the **Keil Studio Online** (<https://studio.keil.arm.com/auth/login/>) and the **NUCLEO-F103RB** development platform.

To setup the project:

* Login or create an account
* Create a new project: File -> New… -> Mbed Project
* Select the “**mbed-os-example-blinky-5**” as a template.
* Name the project tinyAES and click Add Project

From there, you can drag-and-drop the given codes (main.cpp, aes.cpp and aes.h) to the root folder of the project.

The project can be built by either:

* Building the project and adding the generated .bin file in the connected device (should appear on File Explorer as a drive)
* Run the project directly, if the device is connected through the Device Manager of Keil Studio.

In order to verify the projects operates correctly, you need to establish a communication with the Matlab environment. This can be done by using the simple\_traces\_acquisition.m script. Change the COM\_PORT\_NUMBER = "COMx"; accordingly.

# SCA attack countermeasures

In previous labs, we performed the analysis of power traces derived from AES SubBytes execution. In this lab we will introduced three countermeasures that would make the analysis attack more difficult (but not impossible).

## Part 1 : Introduction of random delays

The introduction of a random delay during the SubBytes execution would affect the alignment of the traces. In order to do so, we will use the time functionality of the development platform.

**Task 1. (Optional) Add time libraries**

On the aes\_delay.cpp code, add the time library

**Task 2. Create delay\_SubBytes() function**

Find the delay\_SubBytes() function on the code and fill the ??? so that:

 1. The random number generator is initialized.

 2. A random value r is set with that value between 0 and 999.

 3. Execution of getSboxValue() waits for r microseconds.

**Task 3. Use delay\_SubBytes() function**

Replace all SubBytes() uses with delay\_SubBytes(). Re-load the code to the embedded device and run simple\_trace\_acquisition.m script to verify its correctness.

## Part 2 : Randomization of calculations

Either duplicate the project to save the previous countermeasure or use the same project. Open the random\_aes.cpp file. For this countermeasure we will randomize the execution order of the state matrix replacement performed on SubBytes. This is possible because the replacement of one value is independent of the others. Again, this countermeasure changes the alignment of SubBytes execution.

**Task 1. Create index matrices**

On the aes\_random.cpp code, go to random\_SubBytes(). Create two arrays containing values from 0 to 3. Those correspond to the new indexes of rows and columns of the state matrix.

**Task 2. Create randomize() function**

In order to randomly shuffle the elements of those two arrays, use the randomize() function. The function uses the Fisher–Yates shuffle Algorithm:

 1. First, initialize the random number generator

 2. Create a loop whose index i starts from the last element (n) and moves to 0

 3. Create a value j whose value is randomly set from 0 to i

 4. Swap the input’s array i value with the j value. Use a temp variable to do so.

**Task 3. Use the randomize value for the rows and columns array**

Randomize the elements of the rows and columns arrays back on random\_SubBytes() function. Use randomized arrays’ values to derive the new rows and cols index values. Use them as index for SubBytes replacement.

**Task 4. Use random\_SubBytes() function**

Replace all SubBytes() uses with random\_SubBytes(). Re-load the code to the embedded device and run simple\_trace\_acquisition.m script to verify its correctness.

## Part 3: Insertion of a fake computation

Either duplicate the project to save the previous countermeasure or use the same project. Open the fake\_aes.cpp file. For this countermeasure we will perform a “dummy” computation of Subytes.

**Task 1. Create a fake\_state**

Create a 2-dimensional fake\_state of 4x4 elements and the corresponding data structure (consult the state\_t definition). Also create a fake\_state\_line vector of 0 zero elements.

**Task 2. Add randomize() function**

Add the randomize() function created in the previous step.

# Deliverables

**Task 3. Create shuffling() function**

Go to shuffling() function and implement the following steps:

1. Create a vector itable containing values from 1 to 32. **This table will be used for indexing of both state and fake\_state**.

 2. Randomize its order using randomize().

 3. Copy state to fake\_state using nested loops

 4. For all the values of state and fake state:

 a. Create a variable m, which contains the itable value, divided by 17

 b. If m is zero, remove 1 from m

 c. Also, create two new variable which will contain

i. The result of m divided by 4 (k)

ii. The remainder of m divided by 4 (l)

 e. Perform the substitution of state[k][l]

 f. If m == 1, remove 17 from m.

g. Follow step c(i) and c(ii) and perform the inverse substitution on the fake\_state[k][l]. The functionality is available in the code)

The deliverable of this lab will be **a report**. Describe the methodology followed in the lab. In addition, answer the following questions regarding the attack:

1. Explain the logic of the third countermeasure. What would be the effect on the trace acquisition?
2. Which, in your opinion is the most effective countermeasure in terms of area, latency and security
3. Do those countermeasure offer absolute security? Describe methods to bypass them