Lab 2 - Second-order Side-Channel Attack on AES

# Introduction

In the previous lab we saw how side-channel leakage can easily lead to the extraction of AES key operating over a device. This has led to the development of SCA countermeasures, with the main categories being:

* *Hiding countermeasures*, which aim to alter the expected leakage pattern of a vulnerable operation. Hiding mainly involves hardware alterations. In software, such countermeasures can be applied over:
  + The time dimension, namely as insertion of dummy operations or shuffling of operations
  + The amplitude dimension, with the addition of more than one parallel processing modules.
* *Masking countermeasures*, which aim to remove the relation among the data that are going to be processed by the vulnerable operation and the resulting leakage. Masking alters the original data in a way that, after the intended operation is finished (and has generated leakage that is not related to their original form), can be removed to retrieve the intended output. Multiple forms of masking exist, such as:
* Boolean Masking : data ⊕ mask = data’
* Multiplicative Masking : data \* mask = data’

The number of masks used corresponds to the *order of security*. The above schemes offer a 2nd-order protection against SCA. This means that, in order to break them, we need an attack of equal order, whose complexity exponentially grows.

In this lab, we are going to examine the logic behind a 2nd-order attack over a masked implementation of AES SUBBYTES.

# Masked AES Algorithm Inspection

The AES algorithm the lab is based on implements a masking logic. Firstly, 6 mask values, m0, m1, m2, m3 (MixColumns input masks), m4 and m5 (SubBytes input and output masks) are randomly generated. In addition, m6, m7, m8, m9 (MixColumns output masks), are derived from m0, m1, m2 and m3 respectively, giving us a total of 10 masks, which will be reused throughout the cryptographic operation

mask[6] = mul\_02[mask[0]] ^ mul\_03[mask[1]] ^ mask[2] ^ mask[3];

mask[7] = mask[0] ^ mul\_02[mask[1]] ^ mul\_03[mask[2]] ^ mask[3];

mask[8] = mask[0] ^ mask[1] ^ mul\_02[mask[2]] ^ mul\_03[mask[3]];

mask[9] = mul\_03[mask[0]] ^ mask[1] ^ mask[2] ^ mul\_02[mask[3]];

The remasking operation is performed over both the round keys and the plaintext, through the following function:

static void remask(state\_t \* s, uint8\_t m1, uint8\_t m2, uint8\_t m3, uint8\_t m4, uint8\_t m5, uint8\_t m6, uint8\_t m7, uint8\_t m8)

{

for (int i = 0; i < 4; i++)

{

(\*s)[i][0] = (\*s)[i][0] ^ (m1 ^ m5);

(\*s)[i][1] = (\*s)[i][1] ^ (m2 ^ m6);

(\*s)[i][2] = (\*s)[i][2] ^ (m3 ^ m7);

(\*s)[i][3] = (\*s)[i][3] ^ (m4 ^ m8);

}

}

Note here how the masks are essentially reused within the state. If combined correctly, the traces of the SubBytes calculation of every 2 bytes using the same mask (di ⊕ ki ⊕ m) will result in a combined trace for the following calculation:

(di ⊕ ki ⊕ m) ⊕ (dj ⊕ kj ⊕ m) = di ⊕ ki ⊕ m ⊕ dj ⊕ kj ⊕ m = di ⊕ ki ⊕ dj ⊕ kj

Which is our attack hypothesis.

# 2nd-order attack

## Trigger placement and trace acquisition

As demonstrated above, higher-order attacks exploit the joint leakage of a number intermediate values that are being processed by the cryptographic algorithm. The logic can be exploited since masks tend to be reused for efficiency, and by combining the proper intermediate values, it is possible to remove the masking and extract the key-related information.

Therefore, it is important to acquire traces that correspond to calculations of values with the same mask. As per the 1stt-order attack, the values the attack is going to be performed over are the outputs of the first round SubBytes. It is easily noted that two consecutive SubBytes operate on bytes that are masked with the same value, since the state is examined column-wise. Again, a trigger signal needs to be included, this time surrounding both calculations under examination.

Trace acquisition is performed as normal. Alterations in the attack platform are required. First, the trigger signal should also be acquired (in the previous lab it was only used to align the power trace to the window of oscilloscope) to take advantage of its information in later steps. This is possible by utilizing the second channel (CH2) of the oscilloscope. Second, since the acquired signal is almost doubled in size, the first’s channel (CH1) needs to be changed to EXT, which operated at approximately double the sampling rate.

You are provided with the following files:

* attack\_data\_all.mat : contains the
  + *datapoints\_leakage* variable, which are 30.000 acquired power traces, sampled with a sampling rate 1/10. You can see them plotted in Figure 3, with the horizontal axis representing the data points and the vertical the respective power leakage.
  + *datapoints\_trigger* variable, which are 30.000 acquired trigger traces, also sampled. Those will be used to determine the exact points the bytes are calculated within the combined trace. In Fig.1, you can see the leakage of one plaintext examination, combined with its respective trigger signal behavior

Graphical user interface, chart

Description automatically generated with medium confidence

Figure 1. Example power trace and trigger signal for the SubBytes computation of two bytes

* + *plaintexts\_SCA* contain all the plaintexts MATLAB randomly generated to send to MCU, in decimal format.
  + Finally, *result{}* contains the key, the plaintexts and the derived ciphertexts in hexadecimal form.
* constants.mat contain two useful matrices:
  + the AES SUBBYTES which will we use to simulate SubBytes, as well as
  + the Hamming Weight model, which we will use to simulate the power consumption of SubBytes calculations in the following steps.

Finally, you are provided with the dpa\_2\_attack.m script, which you will develop during the lab, following the instructions given bellow.

# Attack

*Task 2 : Partition plaintext\_SCA into D vectors*

Create D\_1, D\_2 containing only the plaintext bytes that correspond to the key bytes under attack.

*Task 1 : Set the keys under attack*

Determine which key bytes this attack considers, based on the masked AES operation

In 1st-order attacks, the possible values of the key are 256. When it comes to 2nd-order attacks, given the hypothesis di ⊕ ki ⊕ dj ⊕ kj , we need to examine a combination of keys. This increases the number of possible key values to 2562 = 65.536. For the purpose of this lab, we will examine a reduced vector of possible key pairs.

*Task 3 : Generate the Key Hypotheses Vector (K)*

Create a vector which will contain 4 key pairs near the pair [252,45]. Use one column for each key byte.

*Task 5 : Create Combinations Pairs vector*

Use nchoosek() function over each T row to gain the combination of every two traces. Extract the absolute difference of the two values.

*Task 4 : Trim the datapoints vector*

Create a vector T containing the datapoints\_leakage traces. Trim the traces so that they’ll contain a symmetrical portion of datapoints from the SubBytes computation. The computation of the first SubBytes starts from datapoint 590 and the computation of the second SubBytes starts from datapoint 1760 Keep the windows small for faster execution.

Up until this point we have created signals equivalent to di ⊕ ki ⊕ dj ⊕ kj. We procceed to create the power behavior hypotheses (H) of di ⊕ ki ⊕ dj ⊕ kj.

*Task 7 : Create the correlation matrix*

Having the T matrix and the H matrix, derive the correlation values, using the corr() function in MATLAB. Use Pearson’s correlation.

Find the maximum value in R. What does this index indicates?

*Task 6 : Create the hypotheses (H) table*

Combine D\_1, D\_2, K(1) and K(2) to:

1. Derive the SubBytes computation of the data and the hypothetical keys.
2. XOR the two SubBytes
3. Derive the Hamming Weight using the HW matrix.

# Deliverables

The deliverable of the 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 matrices you created for this attack.
2. What would be the complexity of the attack if all key combinations where to be examined?
3. **Increase the key values included in K at the maximum of Matlab capabilities and repeat the attack. What do you note in terms of time execution? Is the success of the attack affected?**
4. Find and use other ways to combine the traces.
5. Compare the 2nd order attack with the 1st order one (previous lab) in terms of complexity.