Lab 1 - First-order Side-Channel Attack on AES SBOX

# Introduction

The purpose of this lab is to demonstrate the capabilities of Side-Channel Attacks (SCA) on a well-established encryption algorithm, namely AES. In AES, the secret key and the given data (plaintext) need to be of specific length (128-bit, 192-bit or 256-bit AES), resulting in an output (ciphertext) of the same length. **AES-128** will be used.

The simplest mode of use is **Electronic codebook (ECB) mode**, meaning that the data is separated into blocks of fixed length (in the case of AES-128, blocks are of 16 bytes length) that will be encrypted handled independently of one another.

*Diagram

Description automatically generatedDiagram

Description automatically generatedWhat are the advantages and disadvantages of that? Consider another mode, CBC.*

Figure . AES ECB operation (left) versus AES CBC operation (right).

The success of the algorithm depends on the secrecy of the key. In order to “break” the cipher, you need to obtain the correct key. Assume the simplest attack from software perspective, brute force attack. The worst case requires the examination of the total number of keys. If a key consists of 128-bits, a total of 2128 possible keys need to be examined.

Better formulated attacks can in theory lower the complexity to 2126. Quantum attacks will probably break the AES-128 (meaning AES-192 and AES-256 are no longer safe), but they do not yet pose a threat. For now, AES-128 ECB achieves computational security, meaning the security is based on the fact that current technology is not capable enough to break the algorithm at an acceptable time.

When applied to hardware, the algorithm can be easily broken with an attack aiming the physical vulnerabilities of the device. Side-channel attacks (SCA) exploit the platform’s emanations (electromagnetic, power, etc), which are related to the data being computed.

Power-analysis SCAs can be divided in two main categories : Simple Power Analysis (SPA) and Differential Power Analysis (DPA). SPA attacks involve analyzing the power consumption of a device during cryptographic operations, with the aim of pattern identification that correspond to different parts of the key and use this information to recover the key. While this requires only a small amount of such measurements to extract the information, it also demands a detailed knowledge of the operations involved in the cryptographic algorithm examined, and how they correspond to the pattern the power traces form.

Chart

Description automatically generated

Figure . SPA power trace analysis on AES. The encryption's rounds patterns are visible in the given trace

On the other hand, DPA attacks are a more advanced form of SPA that involve analyzing multiple power traces corresponding to different inputs to the cryptographic device. It has prevailed over SPA, as it can overcome practical limitations such as the lack of knowledge over the cryptographic mechanisms and noise. DPA attacks are based on the statistical analysis of a large number of power consumption measurements derived from the processing of different data (such as a series of known plaintexts) by the same cryptographic module.

For this lab, we are going to perform a DPA on SBOX-based AES. As you can see in the following figure (Fig.2), the platform under attack (STM32) is programmed to encrypt every given plaintext and return the ciphertext to the computer. A probe is set over the platform to measure the power trace, as demonstrated in the oscilloscope’s screen (yellow trace). The trace is also sent to the computer for the analysis at a later stage. An additional trace (purple line) is also measured. This is an internal trigger programmed in the device to indicate the computation under attack, in this case the first round’s SubBytes on the first state’s byte.

A picture containing text, indoor, computer, desk

Description automatically generated

Figure 3. Power analysis setup for trace acquisition

# Attack

As mentioned before, we are aiming for the first rounds SubBytes. Considering the AES encryption steps, we know that each AES round utilizes a different round key, generated from the secret key of the algorithm. For the first round, the round key is equal to the secret key. Hence The first AddRoundKey (XOR) is performed over a plaintext and the secret key.

The SubBytes operation is a particularly attractive target for side-channel attacks because it is highly nonlinear and consumes a significant amount of power. This means that it produces power traces with distinct and easily identifiable patterns, which can be used by an attacker to deduce the secret key being used in the cryptographic operation.

Therefore, for any known input (plaintext) we can assume the secret key (key hypothesis) applied to it to generate a known SubBytes output (the SBOX is publicly known). We can also simulate the power behavior of the computation (Power Trace Hypothesis), given a powel model. Then, we can compare our simulated power traces to the real ones through correlation (Correlation Power Analysis – CPA is used as an alternative name for the process).

Diagram

Description automatically generated

Figure . Diagram of CPA attack

Graphical user interface, chart

Description automatically generated

Figure . First power trace plotted (left) versus all power traces plotted (right) for AES SBOX computation

You are provided with the following files:

* attack\_data\_all.mat : contains the
  + *datapoints* variable, which are 10.000 acquired power traces, sampled. You can see them plotted in Figure 3, with the horizontal axis representing the data points and the vertical the respective power leakage.
  + *plaintexts\_SCA* contain all the plaintexts MATLAB randomly generated to send to MCU, in decimal format. For example, the first trace depicted in Figure 9 (a) is the power consumption behavior for the encryption of the first byte of the plaintext { 0x5b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c }, XORed with the respective key byte. For the first round, this corresponds to the first byte of the master key, as noted before.
  + Finally, *result{}* contains the key, the plaintexts and the derived ciphertexts in hexadecimal form.
* constants.mat contain two useful matrices:
  + the AES SBOX 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.

Each possible SubBytes byte (plaintext XOR key) obviously presents a different power behavior during encryption. This is related to data computed. Hence, we can exploit that power leakage to extract information about the secret key. The more data we have in disposal, the better our results will be. A question posed often in literature is what is the minimum number of traces required to make a successful key extraction, and is used as a security metric for an implementation.

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

# CPA creation

The first step of this attack is to create a simulation of this behavior for every possible input.

*Task 3 : Generate SubBytes Possible Inputs (V)*

First create a matrix S which will contain all data of D, XORed with every hypothetical key.

Increase all S values by one because MATLAB indexes don’t consider 0.

Pass S through SBOX to create V.

*Task 2 : Generate the Key Hypotheses Vector (K)*

Create a vector which will contain all possible key byte values.

*Task 1 : Create Data (D) Vector*

Create a vector that will contain the byte under attack of the plaintext, for all SCA\_plaintexts.

Consider for which byte computation we gathered the traces for.

Up until now, we have V, which contains all possible SubBytes results for our known plaintexts and hypothesized keys. Our aim is to correlate the observed power behavior with the one we hypothesize in order to retrieve the key. For this purpose we use the Hamming Weight model. The Hamming Weight of a byte is a value that represents the number of 1’s in it. The possible Hamming Weights of 8-bit values then range from 0 to 8.

*Task 4 : Generate the hypothetical power consumption H*

Using the HW matrix, create a matrix H for the Hamming Weights of all values of V.

Having two datasets, H -our final hypotheses- and datapoints, we will examine the relationship among them – the degree of correlation. A high degree of correlation indicates that a key hypotheses use presents very similar power behavior compared to the real traces.

*Task 5 : Generate the correlation matrix R*

Use corr() function in MATLAB, for the appropriate datasets. Use Pearson’s correlation.

*Task 6 : Find the key*

Find the maximum value in R, as well as the row and column it is located. The correct key should be 0xFC.

Plot the data so that the horizontal axis contains the key values and the vertical axis the correlation values.

# 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 D,K,V,H and R (what their size represent, what data they contain).
2. Describe the power model used. What other hypothetical power models can be used and what changes in data acquisition would that require?
3. Describe the changes required in the attack platform (tinyAES and dpa\_attack script) in order to retrieve the key byte used for the 2nd plaintext value.
4. Use Hamming Distance power model instead of Hamming Weight.
5. What is the minimum number of traces necessary to find a single byte with the given power model for this demo? How the change of power model affects that number?
6. Compare the complexity of the CPA with brute-force attack on AES.