# Modern Cryptology, Project homework 2 - Power Analysis on AES

Vicente Bartual Ferran, Bence Mány

March 2022

## 1 Description of the attack

In this assignment we are going to perform a power analysis attack on AES. This is a side-channel attack, which means that instead of trying to break the math and logic behind the algorithm, we are focusing on some physical aspects of the hardware executing the encryption.

### 1.1 Physical background

To explain how the attack is performed, a brief explanation of how processors work at a physical level is needed. When encryption (and for this matter, any sort of calculation) is executed inside a processor, there are a lot of components inside it with different functions. Let's focus on one in particular: the data bus, which allows communication between different components. A data bus is formed by a number of lines, which define its size. In each line, the bits are represented with different voltages: a 1 corresponds with a high level (for example, 5V), while a 0 corresponds with a low level (0V). These bus lines have some small capacitance, like very small capacitors. To control the line level, there are a couple of switches on each line connecting it to either high or low voltage, depending on the bit being transmitted. Every time the line changes from 0 to 1 (from low to high voltage), it must charge itself, which will produce a spike in the power consumption. This spike is produced independently in every line of the data bus, so it will be proportional to the number of lines changing to high voltage. But this actually isn't how modern processors work. Since the time to charge/discharge a line puts a limit on how fast the processor can work, in modern processors, the lines are kept in a mid-level voltage just before their value is defined each clock cycle. The result is that now there will be a spike (smaller, but a spike nonetheless), for each 1 on the data bus, even if a line had a 1 in the previous clock cycle, since it went back to this mid-level voltage before the current cycle. This means that there will be a spike proportional to the number of 1s in the data bus, that can be obtained measuring the power consumption.

### 1.2 The side-channel attack

The attack is performed as follows: First, different inputs are sent to the target device, and the power consumption is measured while the device encrypts the data. The data is stored in a table (*T-table*), which contains the power trace measurement samples for each input. The exact sample time when the encryption occurred is unknown, but the attack itself will discover it.

In this attack, the encryption executed by the device consist on XORing the input with the key, and then performing an S-box substitution, corresponding to the first round of AES. The next step is calculating, for every possible key and every input, the resulting value after the S-box operation. This values are the ones related with the power traces from the first step, so the interesting thing about them is not the value itself, but the number of 1 bits (*Hamming-weight*) in them. Let's define a new table, (*H-table*) in which for each possible key, the Hamming-weight of every output is stored.

As stated above, there is a relation between the number of 1s of the resulting encryption and the power consumption for a given input. Taking this into account, the last step is calculating the correlation between the Hamming-weights of the calculated outputs and the measured actual power consumption samples. Then repeat this for each possible key candidates. The key-sample pair that produces the highest correlation will give the most probable key, and that time sample is the most likely time when the encryption was executed.

For this assignment, we are given a list of 600 1-byte inputs, and 55 power samples measured for each of these inputs while they were encrypted on an ATmega16 microcontroller. This data correspond to the first step of the attack.

## 2 Implementation

Our implementation includes the following functions:

1. **read_traces**: It reads the power data file. In the file, in each line we have 55 power traces that correspond to a single input value. However, this data is going to be used for the correlation, in which we want the values of all the inputs for a given sample time. The function parses the data to store it in this way, returning a 2-dimension array in which the inner lists contain the 600 values corresponding to a single sample time, and the outer list has 55 inner lists.

2. **read_inputs**: Reads the inputs data file. Returns a list containing the 600 input values.

3. **hamming_weight**: Calculates the Hamming weight (number of 1 bits) of a byte.

4. **calculate_h_table**: Calculates the H-table from the input list. Each input value is XORed with every possible key and then is substituted using the AES S-box. The final values of the H-table are the Hamming-weights of

those results. In the same way as **read_traces**, the results are grouped in lists containing the values corresponding to the 600 input values.

5. **pearson_correlation**: Calculates the Pearson correlation coefficient between the two input data sets, which must have the same length. Returns a floating point value in the range of [-1; 1]. 1 relates to perfect linear correlation, -1 is perfect inverse linear correlation, 0 means the data sets don't correlate at all.

6. **key_predict**: Finds the highest correlation for the set of coefficients. Returns three values: the highest correlation coefficient, the most likely key candidate and the time sample that produced it.

7. **main**: The starting point. It uses the previously described functions to calculate the correlation between every key and every sample time. Finally prints the most likely key, along with the correlation and the sample time that produced it.

The program will run according to the description of the attack:

1. Creating the T-table: reading the power consumption traces from file

2. Creating the H-table: reading the inputs from file, then XORing the inputs with the possible keys, performing the AES S-box substitutions, and finally calculating the Hamming-weights of the results.

3. Calculating the Pearson correlation coefficients: each column of the H-table is being compared with each column of the T-table.

4. Determining the most likely key and time sample for the encryption by finding the highest correlation coefficient.

During this project, the data sets used were **T3.dat and input3.dat**. The most likely key candidate turned out to be **0x20** (32), with a correlation of 0.318 for the 31st sample time. The code can we executed by running ”python power_analysis.py”.