Attacking AES using CPA

Embedded Security Project

Andreas Löfwenmark and Chih-Yuan Lin
Department of Computer and Information Science
Linköping University
Sweden

December 14, 2015

1 Introduction

This report summarizes the project work of the Embedded Security course. Our assignment was to implement a differential power analysis (DPA) side-channel attack on the Advanced Encryption Standard (AES) algorithm using correlation power analysis (CPA).

2 Method

The method we used for performing the DPA attack is based on the strategy outlined by Mangard et al. [1, Ch. 6]. It consists of the following steps:

1. Choosing an intermediate result of the executed algorithm.
2. Measuring the power consumption.
3. Calculating hypothetical intermediate values.
4. Mapping intermediate values to power consumption values.
5. Comparing the hypothetical power consumption values with power traces.

In this project we chose the S-box outputs in the first round, since the generation of first round result is simple and not error-prone given the plaintext. We did not need to perform any power consumption measurements as power traces were given to us[1]. To map the intermediate values to power consumption values we used the Hamming Weight model. We used correlation power analysis to compare our calculated hypothetical power consumption values with the provided power traces. The correlation is calculated using Pearson's correlation coefficient (Equation 1). The intuition is that a correct key hypothesis results in a good correlation between the model and real power consumption.

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E\left[(X-\mu_X)(Y-\mu_Y)\right]}{\sqrt{E\left[(X-\mu_X)^2\right]}\sqrt{E\left[(Y-\mu_Y)^2\right]}} \tag{1}$$

[1] `https://www.assembla.com/spaces/chipwhisperer/wiki/Example_Captures`

2.1 Simulation

Before starting real CPA attack, we ran the simulation attack first. The purpose of running simulation attack is to verify our implementation of given intermediate value and power model with a known effective single trace. After verification, we then extended our work to real multiple traces.

The Simulation attack is conducted with the known key 198. Figure 1 shows the simulation result when the S-box of the encryption algorithm is not correctly implemented. The incorrect implementation is losing an important property of S-box, the non-linearity. Compare to the correct and final version in Figure 2. Loss of non-linearity makes the correlation of other keys higher, though the peak is still in key 198. Therefore, we assume the result of CPA should display a characteristic like Figure 2.

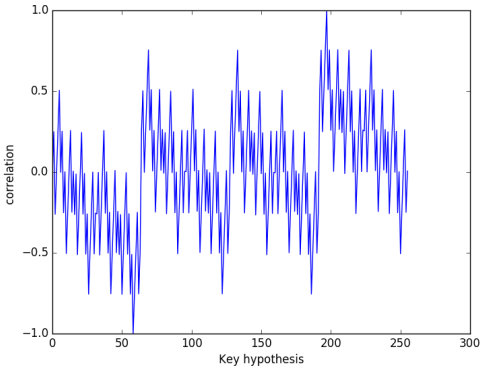

Figure 1: Bad simulation

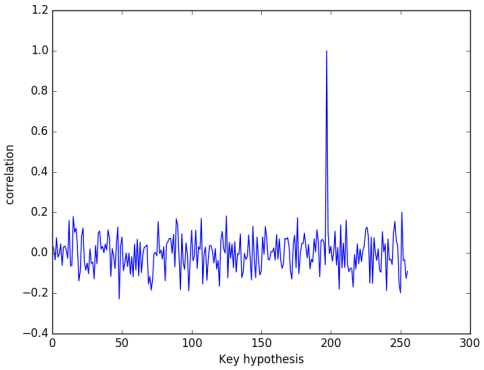

Figure 2: Good simulation

2.2 Implementation

The Python implementation for the CPA attack can be found on GitHub[2]. The program takes one argument, the base name ({base}) of the power traces. A number of files are expected to exist in NumPy[3] (.npy) format, {base}_traces.npy, {base}_textin.npy and {base}_knownkey.npy.

The pseudo code of the main function is shown in Algorithm 1 and the key guessing function is shown in Algorithm 2.

Algorithm 1 Pseudo Code for CPA attack

```
1: procedure CPA
2:     Load power traces, plaintext and knownkey
3:     for each byte i in AES key do
4:         GuessKeyByte(i)
5:         Print guessed key
6:     end for
7:     Print knownkey
8: end procedure
```

[2] https://github.com/hwdev/Embedded-Security
[3] http://www.numpy.org

Algorithm 2 Pseudo Code for guessing a key byte

```
function GuessKeyByte(byteNum)
    for keyGuess in range(0, 256) do
        for all traces do
            Get S-box output using plaintext[byteNum] and keyGuess
            Calculate hypothetical power consumption (Hamming Weight) on S-box output
        end for
        for all traces do
            Calculate correlation coefficients using Equation 1
        end for
    end for
    return index of maximum correlation (our guessed key)
end function
```

3 Results

To test our implementation, we downloaded the file avr_aes128_10000.zip[4] containing 10,000 traces with 3,000 point in each trace.

Figure 3 and Figure 4 show the correlation for key byte 1 and 2, they both show a characteristic similar to Figure 2 giving us confidence that we have a good implementation using multiple traces and also that we have found the correct key byte.

Our CPA implementation successfully identified the correct key in about 40 minutes.

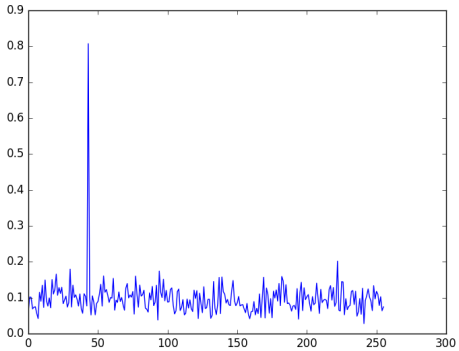

Figure 3: AES key hypothesis, byte 1

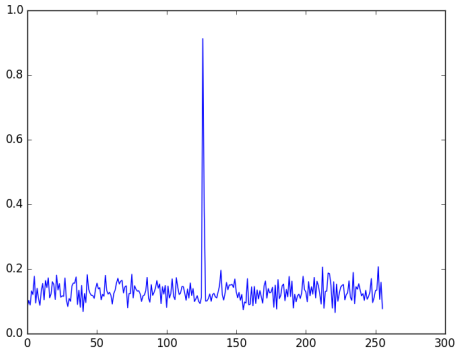

Figure 4: AES key hypothesis, byte 2

References

[1] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks*. Springer US, 2007.

[4] https://www.assembla.com/spaces/chipwhisperer/documents/c9YGi-G5ir44xdacwqjQXA/download/c9YGi-G5ir44xdacwqjQXA