Side-Channel Analysis - Assignment 1

Dinga Madalina s1033146, Vahdad Alireza S1015567

March 2019

1 Assignment - CPA attack against software-based cryptography

The purpose of this assignment is to implement in detail the Correlation Power Analysis attack (CPA) against software-based cryptographic implementations. For this assignment, the encryption algorithm used is the PRESENT cipher, described at the following link: `http://lightweightcrypto.org/present/present\_ches2007.pdf`. The cipher is implemented on an ARM Cortex-M processor.

2 Input, Output and Assumptions

2.1 Input

The attack uses multiple random inputs or plaintexts (i.e., s 14900 4-bit inputs), stored in the vector *inputs* and multiple power measurements (i.e., 14900 aligned power traces, each one with 6990 time samples), stored in the vector *traces*.

The trace acquisition is performed with an electromagnetic probe placed on top of the chip, which often increases the amount of noise in the signal (in other words correlation peaks will not be very clear!).

2.2 Output

The attack aims to recover the secret key (i.e., the output) by using a large number of the available power measurements. In other words, k (i.e., the 4-bit portion (chunk) of the 1st round key) needs to be recovered.

2.3 Assumptions

Throughout the attack the key k is assumed to remain constant and the input in, while random, has to be known.

3 Implementation

The implementation of this CPA attack, which can be found in the following repository `https://github.com/MadalinaDinga/HwSec---assignments/tree/master/assignment1`, is described in this section.

The attack is performed during the 1st round of encryption, as shown in figure 1.

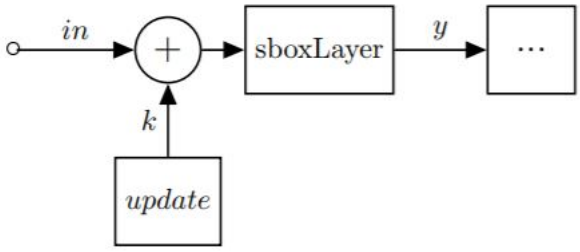

Figure 1: Encryption Using Sbox

All variables in the image above (in, k, y) contain 4-bit values (nibbles). The CPA will be performed in this reduced version of the PRESENT cipher.

The attack point will be the intermediate value y, which depends on k and in. The intermediate value must be a function of the input and the key, which in this case will be the S-box output (i.e., $y = Sbox(in \oplus k)$). The S-box used in PRESENT is a 4-bit to 4-bit S-box sbox:

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
sbox[x]	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Hence, each 4-bit value input is XOR'd with a 4-bit secret key (unknown). This is passed through the S-Box (a look-up table *sbox* is created). The output of the S-Box will be used to check the computed value of the given key, given a number of inputs.

The idea is to compute the value y, for all possible values of the 4-bit key k, $k \in 0..15$ (i.e., 16 possible keys). One of these keys will be the correct one.

3.1 Power-Prediction Matrix

The S-Box values for the inputs and all possible keys will produce the **value-prediction matrix**. This is an intermediary matrix (No_inputs X No_keys). In order to obtain the hypothetical power consumption values for the possible keys, the Hamming Weight will be applied to the S-box values contained in the value-prediction matrix. In order to calculate the Hamming weight of the guess, each possible value 0-16 is converted to binary and the number of 1s contained will represent the value. The implementation uses a Hamming Weight look-up table to store the hamming weight for each possible value.

Finally, a new matrix is obtained (No_inputs X No_keys), which is the **power-prediction matrix**, where the hypothetical values for each plaintext (input) corresponding to each possible guess (16 key guesses in total) are stored.

For performance reasons, the implementation only stores in memory the power-prediction matrix. The Hamming weight model will be applied after computing the S-box values, belonging to the value-prediction matrix. The obtained values will be stored in the power-prediction matrix (power_pred).

3.2 Measurement Matrix

The measurement matrix (a.k.a traces matrix), containing the real measurements, is obtained from the available power measurements collected for n random, but known inputs, where n = 14900.

The obtained matrix will contain one input per line, each with 6990 associated samples on columns, representing the power consumption value at a given time point (i.e., No_inputs X No_samples).

3.3 Matrix Correlation

In order to find the correct key, the hypothetical power consumption values from the power-prediction matrix will be compared with the real measurements, inside the measurement matrix using **Pearson correlation**.

The implementation uses the Python function corrcoef, from the NumPy library (`https://docs.scipy.org/doc/numpy/reference/generated/numpy.corrcoef.html`) to obtain the Pearson correlation coefficients between two columns in the two compared matrices. This uses the formula: $r = \frac{\sum_{i=1}^{n}(h_i-\bar{h})(t_i-\bar{t})}{\sqrt{\sum_{i=1}^{n}(h_i-\bar{t})^2(t_i-\bar{t})^2}}$

Where:
i - i iterates over the traces
h - hypothetical values (from power prediction matrix)
t - trace values (traces/ measurement matrix)

The highest correlation among the absolute values of the obtained Pearson correlation reveals the key.

4 Results

Running the program using the entire set of available power traces (14900), revealed key number 6 as top candidate key. The top candidates are the following: 6 13 0 5 12 10 4 8 3 7 14 15 9 1 11 2

For every time sample, the plot below shows the absolute correlation value for every k candidate. The top candidate is highlighted in blue in figure 2.

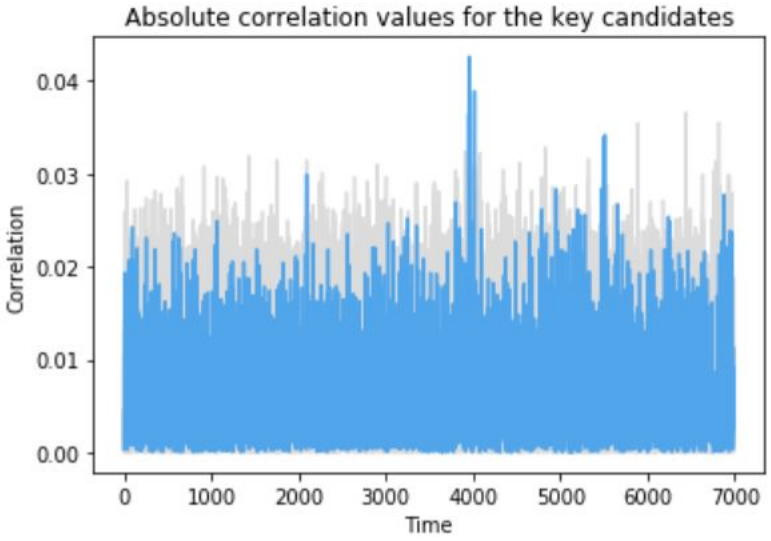

Figure 2: Absolute correlation values for the key candidates

The attack was run with 500, 1k, 2k 4k, 8k and 12k power traces, out of 14900 aligned traces, each one with 6990 time samples. Candidates were ranked from best to worst for each of these runs, based on the correlation values. The correct candidate ranking for all these attacks is presented in the plot below.

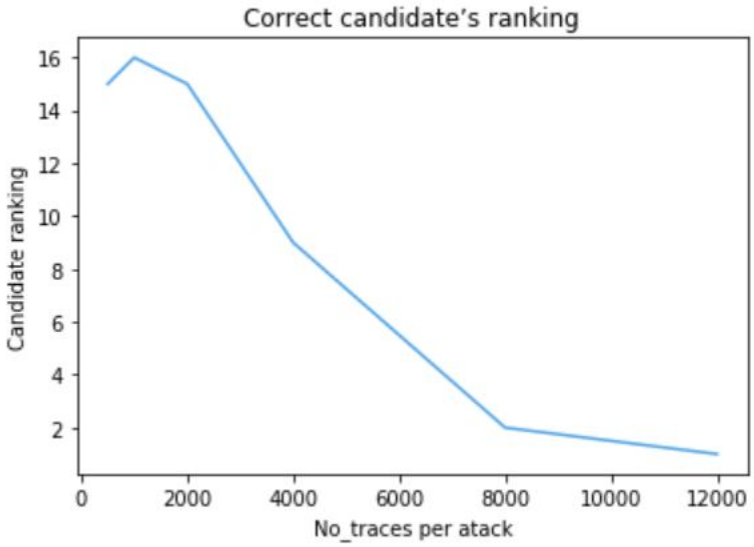

Figure 3: Ranking of the correct candidates

This plot, representing several experiments with different amounts of data shows that the larger the number of data is, the better the ranking of the correct candidate. Specifically, in our case, using the given input data (i.e., plain texts and traces), candidate number 6 was chosen as top candidate in the first run, which used all available power traces (14900 power traces in total). As observed in the figure, lower amounts of data ranked this top candidate worse, ranging from position 16 (for 2k power traces), to 1 (for 12k power traces).

5 Conclusion

In summary, the number of power traces used to perform the CPA attack is important when selecting the top candidate key since it can highly influence the results. As for the implementation of the attack, the Hamming weight model seems to perform fairly well in calculating the hypothetical values for each candidate key.