Side-Channel Analysis

Assignment 2

March 2019

1 Part 1 - Attack against hardware-based cryptography

1.1 Assignment Description

The purpose of part 1 is to implement an attack against **hardware-based cryptography** (e.g. on an FPGA device). For this part, the encryption algorithm used is AES-128. Its state is kept in a register, which will be overwritten on each round. The structure of the algorithm is presented in figure number 1 below.

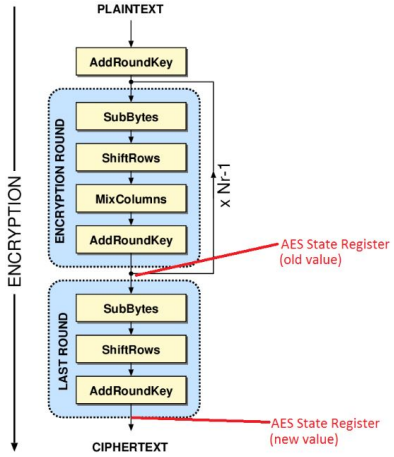

Figure 1: Basic structure of the AES algorithm [1]

The sequence of operations for the last round is shown below, in figure number 2.

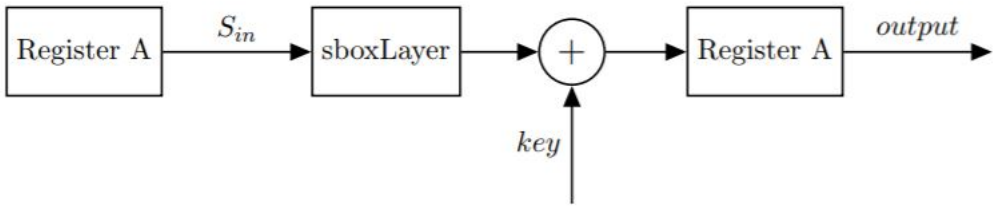

Figure 2: AES operations in the last round

The variables in the image above contain 8-bit values (bytes). Using CPA, the correct byte from the last round key will be recovered. Note that you need to invert the S-box in order to create predictions about the value Sin.

1.2 Input and Output

1.3 Input

The attack uses multiple ciphertexts (i.e., s 10k 8-bit outputs), stored in the vector *outputs* and multiple power measurements (i.e., 10k aligned power traces, each one with 2k time samples), stored in the vector *traces*.

Sources. The traceset (10k traces, 2k samples each) can be found here:
`https://mega.nz/#!b9MXSbAS!Cnh4lm6S0xqXHgp2eQQX0qzXbcQZiljwuOxjixuENvI`

The output bytes (10k) can be found here:
`https://mega.nz/#!3OFwhLiC!Qf3-7jYlKnwrybTGBp2UzZLAhphXwoGAp3Hf9aiY9lM`

1.4 Output

The attack aims to recover the secret key (i.e. find the correct key byte) by using the available power measurements and the ciphertexts. In other words, the 8-bit portion/chunk of the last round key, *key*, needs to be recovered.

1.5 Implementation

The implementation of this attack is described in this section and the source code can be obtained at: source code.

The attack will be performed during the last round of encryption, presented in figure number 2. In this case, attacking the final round of encryption has the advantage of removing the MixColumns operation from the side-channel attack selection function, as this is not executed in the final round.

In case of the AES implementation, the inputs of the S-box in the last round are stored in the same register as the ciphertext. Hence, at the beginning of the last round this register will store the inputs of the S-boxes and at the end of the last round it will store the ciphertext. [2] As a result, bit flips can be counted using the Hemming distance and then, the expected number of bit transitions can be correlated with the measured power.

1.6 Power-Prediction Matrix

The power-prediction matrix is where the hypothetical values for each plaintext (input) corresponding to each possible guess (256 key guesses in total) are stored. Since inputs are not available, the S-box will be inverted in order to create predictions about the value *Sin* (i.e., the input). The information available is in this case the key values and the ciphertext values.

In this case study, examining hardware implementations, the Hamming distance will be applied instead of the Hamming weight model. Specifically, the Hamming distance between the old and the new register state, i.e. the number of bit flips caused when the old register state gets updated with the new register state will be computed. Each value from the power prediction matrix (No_inputs X No_keys), $power\_pred_{i,j}$, will be equal to the Hamming distance between the respective values that are stored in the register in the last round.

Computation of the predicted input S_in:
$S\_in = S^{-1}(outputs_i \oplus k_j)$

Computation of the power-prediction values by applying the Hemming distance of the predicted input S_in and the output:
$power\_pred_{i,j} = HammingDistance(S\_in, outputs_i)$

The Hamming distance uses the Hemming weight of the XOR-ed old and new register state, i.e., counting the values of 1 for S_in (the old state) $\oplus$ output (the new state).

1.7 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 = 10k.

The obtained matrix will contain one input per line, each with 2k associated samples on columns, representing the power consumption value at a given time point (i.e., No_inputs X No_samples).

1.8 Matrix Correlation

In order to reveal the correct key byte, 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. That is, the indices of the highest values of the correlation matrix reveal the positions at which the chosen intermediate result has been processed and the key that is used by the device. [2]

1.9 Results

Running the program using the entire set of available power traces (10k), revealed **key = 71** as the correct key byte. The top 5 candidates were the following: 71 90 216 137 58.

For every time sample, the plot below shows the absolute correlation of all the key candidates over the samples. The top candidate (key 71) is highlighted in blue in figure number 4. In this plot of the correlation matrix there are significant peaks between 1600 and 1800. It can be observed that the plot for the correct key hypothesis (in blue) leads to the highest peak.

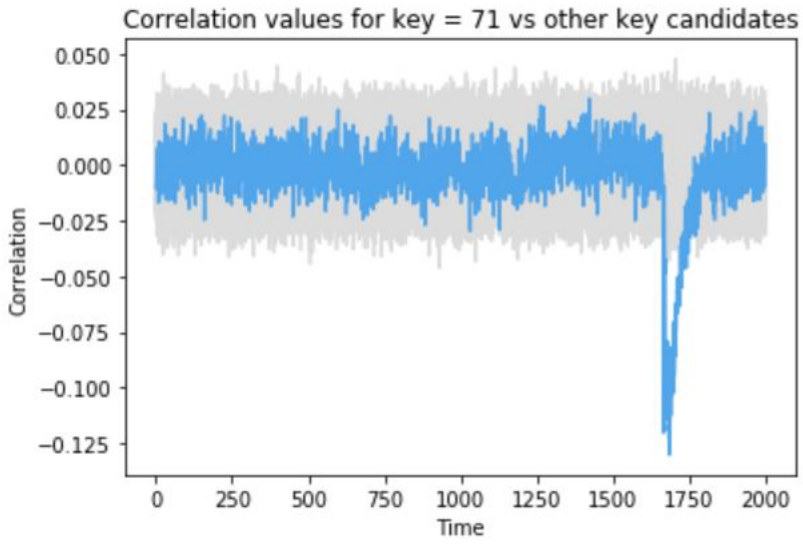

Figure 3: Correlation for key = 71 vs other key candidates

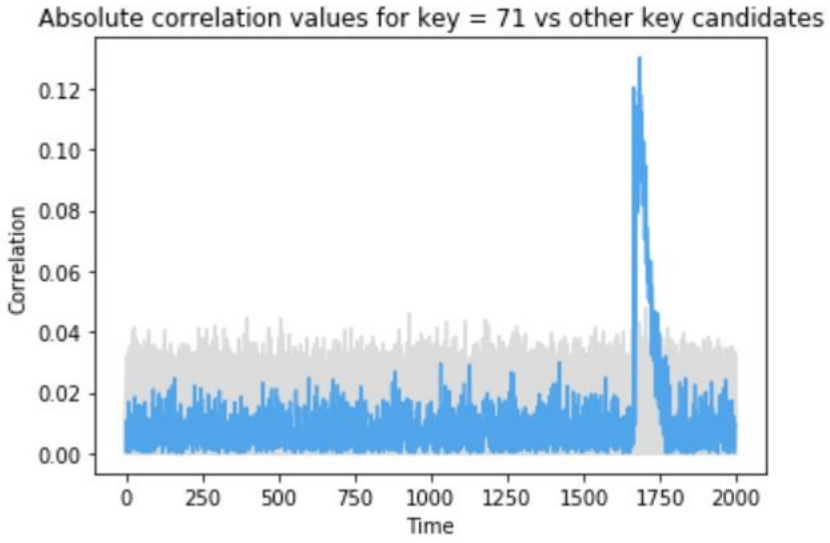

Figure 4: Absolute correlation for key = 71 vs other key candidates

2 Part 2 - Attack against a protected implementation

2.1 Assignment Description

The purpose of the second part of the assignment is to perform an attack against a protected implementation, using a common technique for protecting a cryptographic implementation against side-channel analysis, namely masking. Higher-order DPA attacks imply combining several intermediate values. [2]

In this part, the simulated traceset corresponding to figure number 5 below, will be analysed.

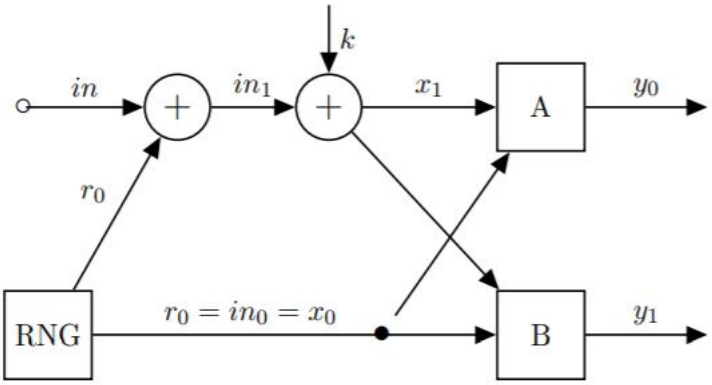

Figure 5: Cipher - KeyAddition and Substitution layers

2.2 Input and Output

2.3 Input

The attack uses multiple inputs (i.e., 2k 4-bit inputs), stored in the vector *inputs* and multiple power measurements (i.e., 10k aligned power traces, each one with 2k time samples), stored in the vector *traces*.

Sources. The input nibble (2k) can be found here:
`https://mega.nz/#!PsNTDKzL!XARLcy5xODY1aNlA2lcVjrQQVPkb6NTZHF8gLf4JO38`

The traceset (2k traces, 10 samples each) can be found here:
`https://mega.nz/#!P5ETlB7a!edwAeSDKPrOtpmobyEBHO35T9RSWGgIC-N7zvLigQ7c`

2.4 Output

The attack aims to recover the secret key (i.e. find the correct key byte) by using the available power measurements and the input nibbles. The absolute correlation value will be analysed for every key candidate, demonstrating the leakage spike of the correct candidate.

2.5 Masking Technique

2.5.1 Splitting Phase

The attack will be performed against a protected cryptographic implementation, which uses masking to randomize the key-dependent intermediate values. In this technique, the input is split in two values: in0 and in1. After the splitting, two shares are obtained:
$in1 = in \oplus r0$, r0 - randomly generated number;
$in0 = r0$.

The two shares in0 and in1 can be recombined via a XOR operation into the original input plaintext in (i.e. $in = in0 \oplus in1$), proving the correctness of the scheme.

2.6 KeyAddition and Substitution Phases

After the splitting, the KeyAddition and Substitution layers of the cipher are used, as shown in figure number 5. In the KeyAddition phase, the key value will be XORed with in1.

In order to maintain the correctness of the masking scheme (i.e. ensure that $y = y_0 + y_1$), the structure of the Substitution layer will be altered by creating two lookup tables:

- Lookup table A is a random and secret lookup table, mapping an 8-bit input to a 4-bit output. It receives values x_0, x_1 and produces $A(x_0||x_1)$, where —— denotes concatenation. The output is y_0.
- Lookup table B maps an 8-bit input to a 4-bit output. It receives values x_0, x_1 and produces $S(x_0 \oplus x_1) \oplus A(x_0||x_1)$. The output is y_0.

In this way, the Substitution layer is carried out correctly. To be noted that all intermediate values shown in the figure are 4-bit values (nibbles).

Focusing on the Substitution layer, the lookup table S is the Sbox used in the PRESENT cipher, mapping a 4-bit input to a 4-bit output as specified in `http://lightweightcrypto.org/present/present_ches2007.pdf`.

2.7 Implementation

The implementation of this attack is described in this section and the source code can be obtained at: source code

2.8 Leakage

The leakage model for this simulation is the identity model, where leakage(value) = value. Also, the values y_0 and y_1 are processed in different points of time making univariate techniques to fail in recovering the key.

The traceset only contains the leakage that has been produced after the Substitution layer, i.e. the leakage stemming from processing y0 and y1. which means the leakage occurs from processing y_0 and y_1.

2.9 Trace Pre-Processing

In order to perform a 2nd-order attack in this scenario, a pre-processing step is applied, where different sample points are combined. Thus, a new traceset containing all possible multiplications between the available samples will be created. That is, m = 10 samples taken k = 2 at a time, where m is the number of samples and k is the attack order.

As a result, assuming that y_0 leaks in sample t_i and y_1 leaks in sample t_j , then, sample t_i * t_j will correlate with the key-dependent value.

2.10 Value-Prediction Matrix

This step involves computing the value-prediction matrix for value y, using all possible 4-bit key candidates and all 4-bit input values.

2.11 Matrix Correlation

Correlation power analysis will be performed against the value-prediction matrix and the actual trace values, in order to find the correct key candidate.

2.12 Results

Running the program using the entire set of available power traces (2k traces, of 10 samples each), revealed **key = 3** as the correct key byte.
The top 5 candidates were the following: ...

For every sample of the new trace, the absolute correlation value for every key candidate was represented in the plot below (figure number 7), demonstrating the leakage spike of the correct candidate.

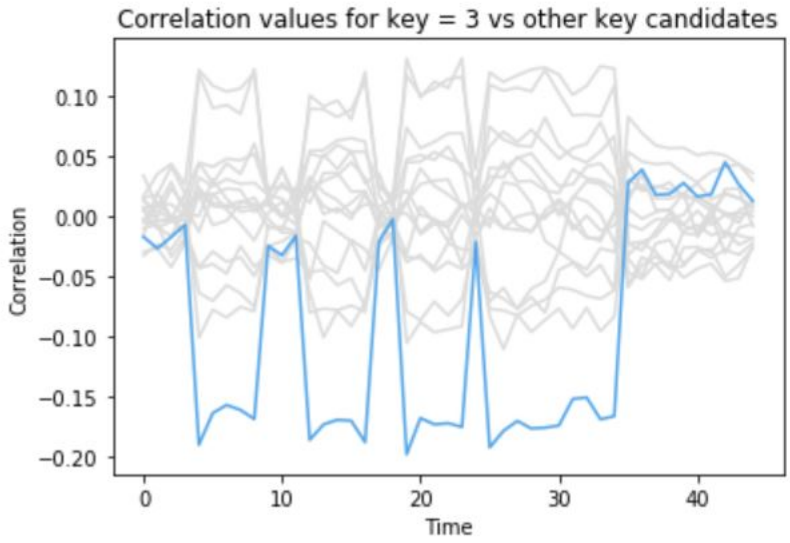

Figure 6: Correlation for key = 3 vs other key candidates

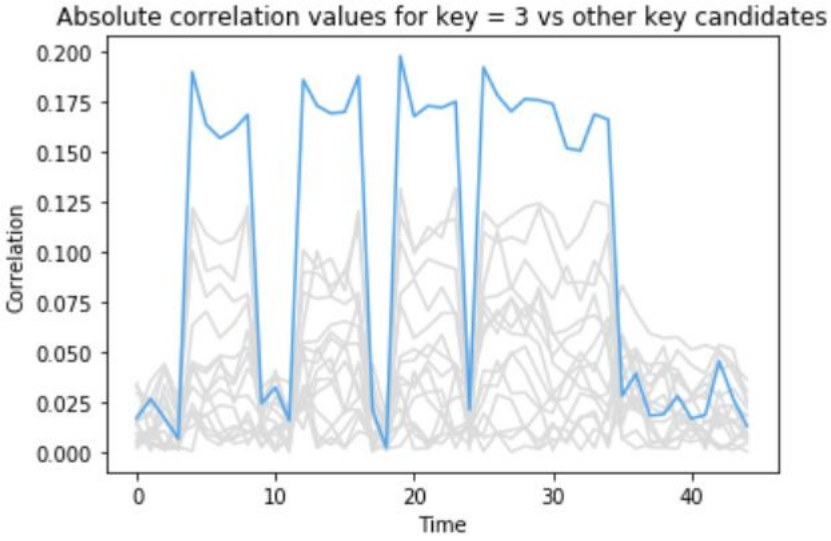

Figure 7: Absolute correlation for key = 3 vs other key candidates

3 Conclusion

References

[1] F. K. Gürkaynak, *GALS system design: side channel attack secure cryptographic accelerators*, 2006.

[2] *Differential Power Analysis.* Boston, MA: Springer US, 2007. [Online]. Available: https://doi.org/10.1007/978-0-387-38162-6_6