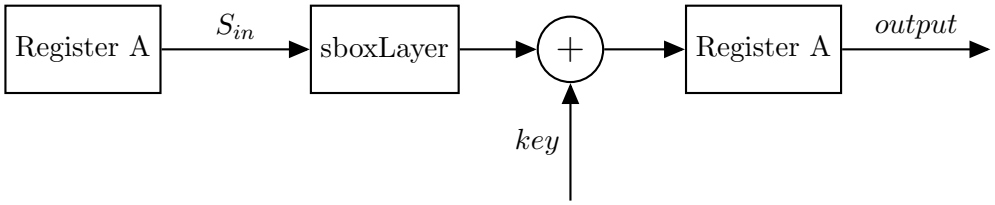
Side-Channel Analysis Assignment 2

Hardware Security – Spring 2019

March 19, 2019

1 Part 1: SCA on Hardware

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.
The attack will be performed during the *last* round of encryption, as shown below. Attacking the last round of encryption has the advantage of removing MixColumns from the side-channel attack selection function.
The sequence of operations for the last round is shown below. Note that the Hamming weight model is not usually applicable in hardware implementations. Thus, it must be replaced with 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.



The variables in the image above contain 8-bit values (bytes). Using CPA you need to recover a correct byte from the last round key. Note that you need to invert the Sbox in order to create predictions about the value S_{in}.

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

The output byte can be downloaded here:
`https://mega.nz/#!30FwhLiC!Qf3-7jYlKnwrybTGBp2UzZLAhphXwoGAp3Hf9aiY9lM`

Plot the correlation of all the key candidates over the samples and find the correct key byte.

2 Part 2: Higher-order Attack – Serial Processing

The purpose of the second part of the assignment is to perform an attack against a protected implementation. A common technique to protect a cryptographic implementation against side-channel analysis is *masking*, which randomizes the key-dependent intermediate values. For instance, the input plaintext in is split into 2 values in_0, in_1. For the splitting, we need to generate a random number r_0, compute $in_1 = in \oplus r_0$ and set $in_0 = r_0$. We can verify the correctness of the scheme by observing that the *shares* in_0 and in_1 can be recombined via a XOR operation into the original input plaintext in (i.e. $in = in_0 \oplus in_1$). After the splitting, we proceed with the KeyAddition and Substitution layers of the cipher as shown in the following picture. In order to maintain the correctness of the masking scheme (i.e. ensure that $y = y_0 \oplus y_1$), we need to alter the structure of the Substitution layer and introduce lookup tables A and B. All intermediate values shown in the figure are 4-bit values (nibbles). Focusing on the Substitution layer, lookup

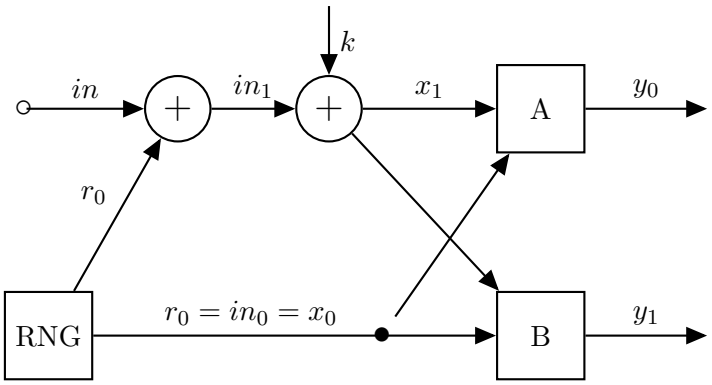


table S is the Sbox used in the PRESENT cipher, mapping a 4-bit input to a 4-bit output as specified in `http://lightweightcrypto.org/present/present_ches2007.pdf`. Lookup table A is a random and secret lookup table, mapping an 8-bit input to a 4-bit output. In the figure, table A receives values x_0, x_1 and produces $A(x_0||x_1)$, where $||$ denotes concatenation. The lookup table B maps an 8-bit input to a 4-bit output. It receives x_0, x_1 and produces $S(x_0 \oplus x_1) \oplus A(x_0||x_1)$. We can then verify that $y = y_0 \oplus y_1$, i.e. the Substitution layer is carried out correctly.

In this part, you need to analyze the simulated traceset that corresponds to the figure above. The input nibble is available here:
`https://mega.nz/#!PsNTDKzL!XARLcy5x0DY1aNlA2lcVjrQQVPkb6NTZHF8gLf4JO38`
The observed traceset is available here (2k traces, 10 samples each):
`https://mega.nz/#!P5ETlB7a!edwAeSDKPr0tpmobyEBHO35T9RSWGgIC-N7zvLigQ7c`

1. The traceset contains only the leakage produced after the Substitution layer, i.e. the leakage stemming from processing y_0 and y_1. The leakage model used in the simulation is the *identity* model, i.e. $leakage(value) = value$. The values y_0 and y_1 are processed at different points in time, i.e. they leak in two different time samples and univariate (single-sample) techniques will fail to recover the key.

2. In order to perform a 2nd-order attack in this scenario, you need to perform a pre-processing step where different sample points are combined. Thus, you need to create a new traceset that contains all possible *multiplications* between the available samples.
   For example, trace $\mathbf{t} = (t_0, t_1, t_2)$ will be replaced by $\mathbf{t'} = (t_0 * t_1, t_0 * t_2, t_1 * t_2)$. In general, the dimension of $\mathbf{t'}$ will be $\binom{m}{k}$, where m the number of samples ($m = 10$) and k the attack order ($k = 2$). (Hint: use the *nchoosek* MATLAB function).

3. 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 y.

4. Thus, after the trace pre-processing step, you can compute the value-prediction matrix for value y, using all possible 4-bit key candidates and all available 4-bit input values. (Hint: no need for power-prediction matrix, since we work under the identity leakage model).

5. Perform a correlation power analysis and find the correct key candidate. For every sample of the new (processed) trace, plot the absolute correlation value for every key candidate, demonstrating the leakage spike of the correct candidate.

The deliverables are: the source code for part 1 and 2 and a very brief report containing the correlation graphs that demonstrate the correct key.