ELEC2760 - Exercise Session #3: Side-Channel Attacks

The goal of this exercise session is to implement three well-known side-channel attacks: Kocher et al.'s (single-bit) Differential Power Analysis (DPA), the Correlation Power Analysis (CPA) and the template attack. These attacks will be applied to the ChipWhisperer Lite board running the AES Furious (unprotected) implementation (of which the .asm and .hex files are provided).

Exercise #1 - Single bit DPA.

DPA has been introduced by Kocher, Jaffe and Jun in 1999. In this attack, an adversary attempts to predict the value of one key-dependent bit computed by the device. In the case of the AES algorithm, this is typically a bit in the output of the substitution layer in the first round. For each possible key byte hypothesis (among the 256 possible ones), the leakage traces are then sorted into two sets depending on the value of a bit in the S-box output. The underlying idea of DPA is that the power consumption should behave differently depending on the value of this bit. So, if the subkey hypothesis holds, the two sets should be noticeably different, whereas for the wrong hypotheses, the traces are not sorted in a meaningful fashion and should give indistinguishable sets. Implement Kocher's DPA in order to recover the correct key corresponding to the largest difference between the mean values of the power consumption between the two sets. To do so, implement the function `dpa_byte` in the file `e1.py`. One limitation of the single bit DPA is that it only uses the value of a single bit. How would you improve that?

Hint: the traces provided correspond to the measurements of full encryptions process. How could you simply improve the performances of your attack?

Exercise #2 - CPA.

CPA generally uses an a-priori selected leakage model (e.g. the Hamming weight of the processed data) and assumes that the power consumption of the target device is linearly correlated with it. In order to exploit such a model, we need another distinguisher, more efficient than a simple difference of means. Pearson's correlation coefficient is a natural candidate for that purpose. This statistic measures how well the leakages can be predicted by a linear function of the model values. The underlying idea is that the subkey hypothesis that predicts the leakages best should be the correct one. This correlation is computed as:

$$r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - \left(\sum x_i\right)^2}\sqrt{n\sum y_i^2 - \left(\sum y_i\right)^2}},$$

where n is the number of traces, x are the leakage values and y are the model values (the predicted Hamming weights). Implement the computation of the Pearson correlation by writing the `pearson_corr` function in the file `e2.py`. In order to implement this efficiently in Python, it is advisable to use matrix/vector operations instead of `for` loops over the time samples. Then, implement CPA targeting the AES S-box input & output (i.e., `cpa_byte_in_sbox` and `cpa_byte_out_sbox`). Which one works best? Why?

Exercise #3 - Template attack.

When an adversary can control a device similar to the one he intends to attack, he can take advantage of it in order to estimate more precise leakage models and exploit more powerful statistical tools. Template attacks are a typical example of such stronger attacks. They require the attacker to first run a profiling phase, in order to characterize the distribution of the leakages for each key and plaintext. Then, during the online attack, he can apply Bayes' law in order to compute the likelihood of the different key candidates. A first step is to identify the time sample that gives you the most information. A reasonable (yet heuristic) choice is to use the time sample that gave you the maximum correlation coefficient in the previous exercise. The second step consists in building 256 templates corresponding to the 256 possible outputs of the first AES S-box, e.g. assuming that they follow a normal distribution. In this case, each template has the form:

$$\mathcal{N}(l|\mu_i, \sigma_i) = \frac{1}{\sigma_i\sqrt{2\pi}} e^{-\frac{(l-\mu_i)^2}{2\sigma_i^2}},$$

where μ_i is the average leakage value corresponding to the S-box output value i, and σ_i its standard deviation. Start by implementing the profiling phase function `online_phase` in the file `e3.py`. Next, the (online) attack phase works as follows. The templates can be used to estimate the probability of each subkey byte. For each power trace measured (with a plaintext byte x), the probability of each subkey byte can be computed as:

$$\Pr[\mathsf{S}(x \oplus k) = i] = \frac{\mathcal{N}(l|\mu_i, \sigma_i)}{\sum_j \mathcal{N}(l|\mu_j, \sigma_j)}.$$

The probabilities corresponding to different traces can then be combined in order for the correct subkey byte to be detected (which happens when it has maximum likelihood). Implement the functions `online_phase` and `ta_byte` in the file `e3.py`. Compare the number of traces required for a successful CPA and for a successful template attack. What are the limitations of the CPA attack and the limitations of the template attack?

Eventually, try to improve the previous template attack by extending it to a multivariate setting. Start with a bivariate attack exploiting the two most informative samples in your traces. Then try to extend to more informative samples. How much do you expect to gain compared to a univariate attack (for the online phase)? And what is the expected impact of increasing the number of dimensions on the profiling complexity of the attack?