622.755 – Crypto Engineering Winter Term, 2020/21

Handout 2: Differential Attacks: DPA on AES

Elisabeth Oswald

1 General Comments

This handout supports the delivery of the Crypto Engineering unit. For each week, there are slides, some videos and sometimes handouts. I suggest if there is a handout to read the handout first. It will give you some background information and explain important concepts, give examples, and suggest some exercises as well as hints for further reading. Anything under further reading will enhance your understanding, but is not strictly required to be able to get on with the coursework.

The videos are designed to talk you through the notes as well as the exercises and make explicit links to the coursework. I will often use analogies in videos to explain things, whereas the notes, whilst explaining the same ideas, will be more formal.

For this week you need to have a detailed understanding for AES and implementing it such as provided in Cryptography (Summer Term) and in the previous weeks of this course.

There are three associated videos available from YouTube via an unlisted playlist: `https://www.youtube.com/playlist?list=PLQ5hY1xU49CBkXsaK5TgUiQsy4jAZ7-xp`. The videos are numbered, and you are looking out for numbers 13–15.

The text, and the pictures in this handout are largely taken from the book that I co-authored [2]. I am trying to get the AAU library to purchase the electronic copy.

2 What you should get out of this.

This handout is about a detailed study of differential side channel analysis with a focus on power attacks on AES. After engaging with this content you should be able to:

- name characteristic components of the power consumption
- reason about what to expect when visually inspecting a trace
- explain in detail how a differential attack works: steps, choices, implications thereof
- explain the concept of a signal to noise ratio and it's relevance to differential attacks
- derive an approximation for the number of needed traces for an attack to succeed with high probability

d_0	d_4	d_8	d_{12}
d_1	d_5	d_9	d_{13}
d_2	d_6	d_{10}	d_{14}
d_3	d_7	d_{11}	d_{15}

k_0	k_4	k_8	k_{12}
k_1	k_5	k_9	k_{13}
k_2	k_6	k_{10}	k_{14}
k_3	k_7	k_{11}	k_{15}

Figure 1: State and Key Layout (numbering convention) for AES

3 AES

AES encrypts data of a fixed block length (128 bits) under a key, which can either have 128, 192, or 256 bits. It is common practice to use the abbreviation AES-128, or simply AES, in order to refer to AES with 128-bit keys. There are few differences between AES-128, AES-192 and AES-256: the number of rounds increases and the key schedule changes accordingly. We focus on AES-128 (aka AES) only: to extend what we describe in this handout to AES with longer keys, one has to adapt attacks to cope with the fact that only 128 bits of those longer keys leak in a single round.

3.1 Structure of the AES Encryption

In AES the input data and the key are represented as an array of bytes with four rows and four columns, see Figure 1. The data array is typically called the *state*. AES operates as a typical iterated cipher: a round transformation (in the form of an SP network) is repeatedly applied to the state. The round keys are also computed in an iterative fashion, and the round key data is analogously arranged to the state. It is important to notice how the elements of the state are indexed: we run through *column*-wise.Decryption works similarly, and there are different ways to implement it (e.g. via the equivalent inverse cipher). If any of these statements do not sound familiar to you, it would be best to revise the notes about AES implementations that are part of this unit.

There are a number of implementation strategies to map the high-level description A to some concrete implementation goal. In software, in the absence of a dedicated instruction set (e.g. like on the M3 core that your attack target is based on), one has the choice between an 8-bit implementation (with reasonable memory footprint), or a high-throughput 32-bit implementation (e.g. the T-table implementation option that comes with a considerable memory footprint). We will focus our attention on an 8-bit implementation (i.e. no T-tables), although it is perfectively feasible (and sensible) to utilise 32-bit instruction for some components of the round. This option reflects that attack target, which is part of your coursework (it has SubBytes as well as elements of MixColumns precomputed).

Alg. A: AES Encryption

Provided an input m and key k , this delivers an AES encryption c of the input as a result.

```
input: m, k
output: c
state = m
rk = KeyExpansion(k)
AddRoundKey(state , $rk_0$)
for r = 1 step 1 to 9
            SubBytes(state)
            ShiftRows(state)
            MixColumns(state)
            AddRoundKey(state , $rk_r$)
 end
 SubBytes(state)
 ShiftRows(state)
 AddRoundKey(state , $rk_{10}$)
c = state;
```

4 A First Look at AES Traces

The focus of this week is on *Differential* (power/EM) attacks on AES. If AES was implemented in a way that its' execution times depended on data, then of course also timing leakage could be used in the same overall attack strategy. As your attack target however makes it relatively easy to produce an AES implementation that is constant time, we will focus purely on power/EM.

4.1 Power traces

The total power consumption of a device depends on the power consumption of all the logic cells and their interconnects making up the circuit. Hence, the total power consumption essentially depends on the number of logic cells in a circuit, the connections between them, and the technology based upon the cells are built. Thus architectural design (software and hardware), cell level and transistor level specificities all contribute to how "leaky" a circuit/device is.

As sketched in one of the video tutorials, logic cells are very often implemented using CMOS (complementary metal-oxide-semiconductor). In the tutorial I discuss (on a high

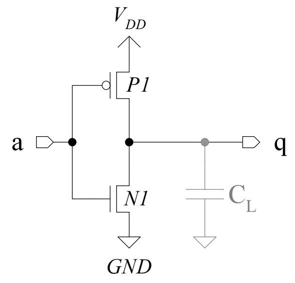


Figure 2: CMOS inverter.

level, based on a CMOS inverter) when and why CMOS cells dissipate power. The discussion of an inverter is representative for all other cells, because all CMOS cells are built based on complementary pull-up and pull-down networks (a network is just a fancy name for a bunch of connected gates). The inverter simply has two transistors $P1$ and $N1$, see Figure 2, whereas more complex gates just require more PMOS and NMOS transistors. In the inverter $P1$ is conducting and $N1$ is insulating if the input a is set to GND. Vice versa, $P1$ is insulating and $N1$ is conducting if the input a is set to V_{DD}. In both cases, there is no direct connection between the V_{DD} line and the GND line. All capacitances in the circuit are modelled as a single "capacitor" C_L. A detailed explanation of all characteristics of this inverter, including a power simulation can be found in Chapter 3 of the DPA book[2].

The power consumption of the inverter (and therefore any other CMOS gate) is comprised of two components, see also Tab. 1. The first component is the static power consumption, which is the power that is consumed if there is no switching activity in a cell. The second component of the power consumption is the dynamic power consumption P_{dyn}, which is due to the switching of internal signals or output signals of a cell. The total power consumption of a cell is the sum of both static and dynamic power consumption. Typically, the dynamic power consumption is much more dominant than the static one. However, in very small technologies this relationship is switched. The target device that we use in this unit is not based on a very small technology, hence the dynamic power consumption is dominant.

Table 1: Output transitions of the CMOS inverter and the corresponding power consumption.

Transition	Type of power consumption
$0 \rightarrow 0$	static
$0 \rightarrow 1$	static + dynamic
$1 \rightarrow 0$	static + dynamic
$1 \rightarrow 1$	static

Power analysis attacks exploit the fact that the power consumption of cryptographic devices depends on the operations they perform and on the data they process. Hence it is useful to have a shorthand for these components when referring more mathematically to the power consumption. For each single power measurement, we refer to the operation-dependent component of the point as P_{op}, and to the data-dependent compontent as P_{data}. Besides P_{op} and P_{data}, each point of a power trace also depends on what is going on the other, possibly data and operation independent "stuff', which we shall simply call "noise" P_{noise}. The components P_{op}, P_{data}, and P_{noise} are additive. Therefore, it is possible to simply model each point of a power trace as the sum of these components:

$$P_{total} = P_{op} + P_{data} + P_{noise}. \tag{1}$$

Now that we have a better sense of how the observable power consumption comes to happen, we can consider what a power trace actually is. Power traces are vectors of voltage values that have been recorded with a digital sampling oscilloscope (i.e. the Pico Scope in your coursework). The measured voltage values are proportional to the power consumption of a cryptographic device because the oscilloscope is connected to an appropriate measurement circuit or EM probe (in your case it is the former). The settings of the oscilloscope determine how many points are recorded per second (or clock cycle of the target device), as well as how many points are stored. The quality of the scope determines how big a range one has for both points per second as well as number of points. The quality of the probe and scope determine how much of the actual signal (i.e. P_{total}) is actually captured, how much this signal is influenced on its' way from the probe to the scope, and thus what signal is actually available for digitisation on the scope. Thus the quality of a measured trace depends not only on the device, and the surroundings, but also every component that is part of the setup.

4.2 Characterising a single trace point

Let's go back at just looking at the properties of a single point in a trace. This makes sense because the kind of attacks that we will study will effectively examine each trace point individually: in statistics terminology these are univariate attacks.

For a single fixed moment of time, as it is represented by a single point in a trace, we shall empirically determine the probability distribution of P_{noise}, P_{data}, and P_{op}. Figure 3 shows a few power traces that are all based on the same operation and data (in other words, P_{op} and P_{data} are constant, and the only variation is due to P_{noise}. It is evident from the figure that, as expected, if a device executes the same instruction with the same data, then the variation in its' power consumption is due to the independent noise only. In the specific setup that we used to capture these traces, the noise level was extremely low. Figure **??** shows a histogram of just a single point (this was derived by utilising 10.000 traces, all based on the same operation and data, where just one single point was extracted). You should be familiar with the shape of this histogram: it is bell shaped, typically for a Gaussian distribution. In a histogram, the areas of the blocks represent

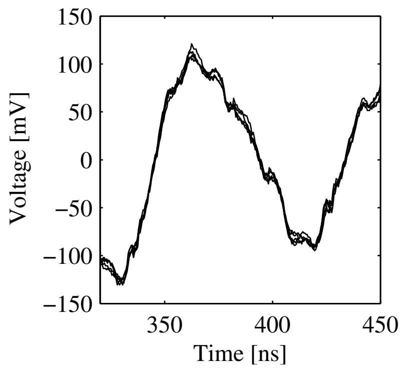


Figure 3: Power if operation and data are fixed. Taken from the DPA book [2].

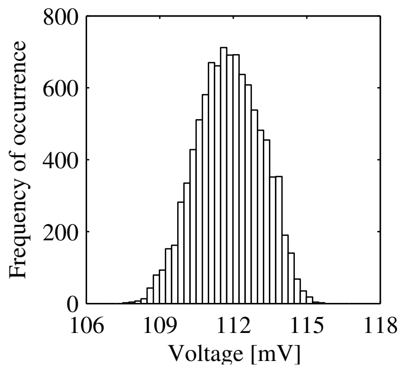


Figure 4: Histogram of the power consumption at 362 ns of Figure 3.Taken from the DPA book [2].

the frequency of occurrence. In Figure 4 all blocks have the same width. Thus, higher blocks represent more frequent power consumption values.

Gaussian (or Normal) distributions occur frequently in practice. The density function describing the normal distribution (2) only depends on the parameters μ (expectation) and σ (standard deviation), where $-\infty < \mu < \infty$ and $\sigma > 0$.

$$f(x) = \frac{1}{\sqrt{2 \cdot \pi} \cdot \sigma} \cdot \exp\left(-\frac{1}{2} \cdot \left(\frac{x-\mu}{\sigma}\right)^2\right) \tag{2}$$

The square of the standard deviation is also called *variance*.

$$\begin{aligned} \mu &= E(X) && (3) \\ \sigma^2 &= Var(X) = E((X - E(X))^2) && (4) \end{aligned}$$

In the experiment visualised in Fig. 4, most points are around 112 mV and only very few points are below 109 mV or above 115 mV. Thus we expect the mean to be close to 112, and sigma to be around $\sqrt{3}$. As evident from the picture, the mean represents the centre of the distribution, and the variance the width of the distribution (hence wider distribution has a higher variance). An important property of the standard deviation is that 68.3% of all x_i are within one standard deviation around the average, 95.5% are within two standard deviations, and 99.99% are within four standard deviations. This explains things like the 2 or 4 sigma rules that one can find in the applied statistics literature.

This little experiment has enabled us to determine the distribution of P_{noise}: clearly it fits very well to a typical normal distribution (remember that all components of the

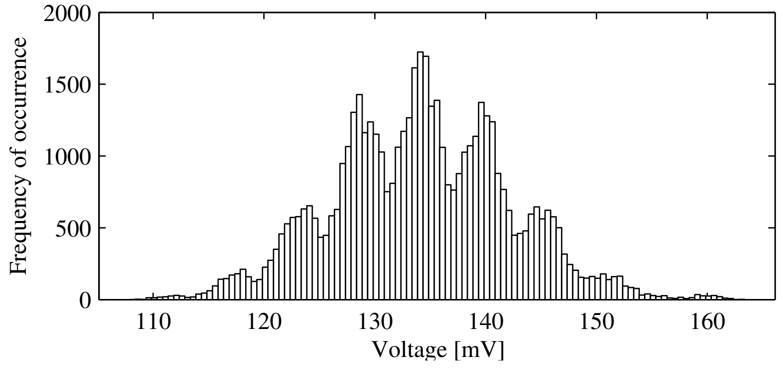


Figure 5: Histogram of the power consumption at 362 ns if different data values are transferred from the internal memory to a register. Taken from the DPA book [2]

analysed point were kept constant, so the only component to cause any variation must have been the noise).

Let's move on then and examine what happens if we keep the operation constant, but vary the data (the noise component is independent of these two and thus will behave the same). Figure 5 shows a histogram of this experiment. The resulting histogram appears to be composed out of nine normal distributions with different heights. The first distribution seems to have its mean at about 112 mV, the second one at about 118 mV, and the third one at about 123 mV, and so on. This makes sense: the device from which these traces were taken was a simple 8-bit micro controller. Thus the different bumps could represent the data dependent behaviour of the 9 different Hamming weights that an 8-bit data value can take. Observing Hamming weight leakage is typically associated with data being transferred across some data bus, and this effect is one of the most widely observed and exploited one in attacks on micro processors. Why might there be an overlap between distributions for different Hamming weights? This can be explained by the fact that the power consumption consists of three components: the operation was kept constant (so there is no variation due to that), the variation in the data we just explained, and then there is the independent noise (this adds to the overall variation and is the reason for the overlap). Thus the distributional characteristics that we see are predominantly because of the distribution of P_{data}, which is given by the distribution of the Hamming weight of the data. The distribution of the Hamming weight is given by a binomial distribution, thus P_{data} is binomially distributed as well (if the leakage is due to the Hamming weight).

We have seen that the leakage of a single point can be described by a statistical distribution. The total power consumption is a sum of several factors. The noise component can be well approximated by a normal distribution. The component that is due to the operation tends to be constant in the kind of attacks that we focus on and thus does not contribute to our investigation. The data dependent component may be described by a binomial distribution, if the leakage is due to the Hamming weight. Otherwise it may have a different distribution, and in practice we often find that describing it via a normal distribution works just fine.

4.3 Visual inspection of a trace

Every algorithm that runs on a cryptographic device is executed in a sequential manner. Even in hardware, cryptographic (round) functions are too complex to be executed in a single step. The mathematical description of an algorithm has fist to be translated into some (high-level) language. This description is eventually translated (by a compiler) into instructions that are supported by a device.

In our concrete example, AES, there are ten rounds. Each round consists of four round transformations, which are called AddRoundKey, SubBytes, ShiftRows, and MixColumns, each of which takes in one or several bytes of the state. Suppose that AES is implemented in software like in your course work. In this case, the round functions will be translated into the instructions that are available on the processor that sits on the scale board (an ARM M3). The instruction set of this processor consists of arithmetic instructions (such as addition), logical instructions (such as exclusive-or), data transfer instructions (such as move), and branching instructions (such as jump). This instruction set is rather typical for similar (small) processors.

Each instruction works on a number of bytes and utilises different components of the processor, such as the arithmetic-logic unit, memory (external or internal RAM or ROM), or some peripheral (such as a communication port). These individual components of the processor differ in functionality and implementation. Therefore, they have a characteristic power consumption, which leads to a clear pattern in the power trace. The possibility to distinguish instructions within a trace can lead to a serious security problem if the sequence of instructions directly depends on the key (as we have seen in the case of RSA).

Let's perform a visual inspection of a power trace of an AES implementation on a simple microcontroller (not the M3 on the SCALE board). In that AES implementation there are no data dependencies in the instruction flow, thus an attack such as on RSA will not work. Figure 6 shows a power trace of a full AES encryption run on the simple microcontroller. The 9 full rounds are clearly visible, as well as the last round in which MixColumns is skipped.

As a next step we zoom into the first two rounds, and then a single round of this trace, and finally into a short sequence of low level instructions. Figure 7 zooms in on the first two rounds. The first round is located between 0.06 ms and 0.73 ms, the second round is located between 0.84 ms and 1.52 ms. If you struggle locating the rounds, then look out for three tiny peaks in the upper half of the picture. Figure 8 shows just

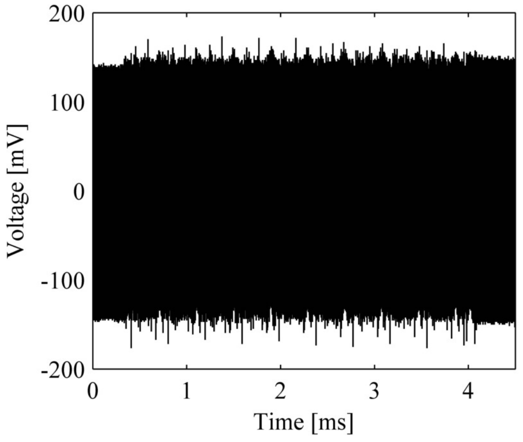


Figure 6: Full AES encryption.

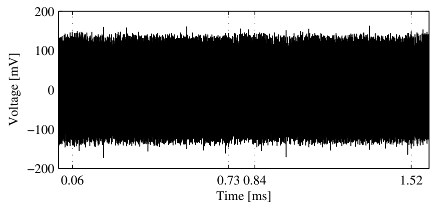


Figure 7: Two rounds of AES (no compression)

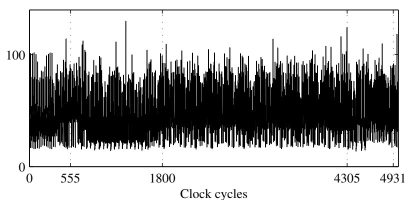


Figure 8: One round of AES (with compression)

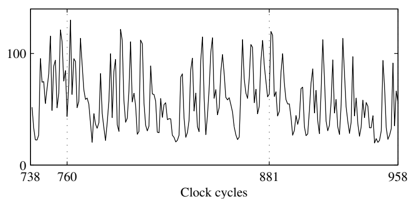


Figure 9: AddRoundKey, SubBytes, and ShiftRows(twice).

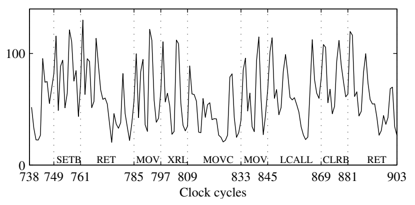


Figure 10: AddRoundKey, SubBytes, and ShiftRows with annotation.

```
LCALL SET_ROUND_TRIGGER
MOV A,ASM_input + 0     ; load a0
XRL A,ASM_key + 0       ; add k0
MOVC A,@A + DPTR        ; S-box look-up
MOV ASM_input, A        ; store a0
LCALL CLEAR_ROUND_TRIGGER
```

Figure 11: The sequence of assembly instructions that corresponds to Figure 10.

a single round where we have “compressed” the trace (by identifying each clock cycle and replacing it with a single representative value). The round starts at clock cycle 555 and ends at clock cycle 4 305. There are three different patterns visible in this picture. Many typical AES implementations in software first perform the round functions that operate on individual bytes and then perform MixColumns, which operates on a column (4 bytes) of the AES state. Your implementation might follow the same principle, or you may have chosen to process the state column by column (thus interleaving a single column of MixColumns with the other round functions).

If the key generation is done on-the-fly, it will have to occur in between the rounds so the next round key is ready. The implementation of AES that we are examining does just that. Therefore, the first pattern in Figure 8 corresponds to the sequence of AddRoundKey, SubBytes, and ShiftRows operations, the second pattern corresponds to MixColumns and the third pattern corresponds to the generation of the round key.

Within the first pattern, one can count 16 tiny peaks (on the bottom of the trace) that are closely located next to each other. Each peak corresponds to a sequence of AddRoundKey, SubBytes, and ShiftRows. In Figure 9, we zoom in on Figure 8 and show the power consumption of the first two peaks only. The first peak is located between clock cycle 760 and 881. The shape of the trace shortly after clock cycle 760 is very similar to the shape of the trace after clock cycle 881. To help interpreting this trace, I have added the sequence of assembly instructions in Figure 11.

First, the function `SET_ROUND_TRIGGER` is called to set a trigger signal. The `SET_ROUND_TRIGGER` function executes a `SETB` (set bit) and a `RET` (return) instruction. Then one byte of the AES state is processed. Afterwards, the trigger signal is cleared by calling the `CLEAR_ROUND_TRIGGER` function, which executes a `CLRB` (clear bit) and a `RET` instruction. On this microcontroller, different instructions require different numbers of clock cycles for their execution. `SETB`, `MOV`, and `XRL` require 12 clock cycles whereas the other instructions require 24 clock cycles. The M3 processor that you are using is much more efficient: it uses a single or two clock cycles per instruction. Figure 10 shows an annotated sequence of instructions corresponding to the assembly.

5 Differential power analysis of AES

Standard DPA attacks exploit the data dependent component of the power consumption. Such attacks often use a large number of power traces and analyse the power consumption at a fixed, single moment of time as a function of the processed data.

There exists a general attack strategy that is used by all DPA style attacks. This strategy consists of five steps, which I am now taking directly from [2].

Step 1: Choosing an Intermediate Result of the Executed Algorithm The first step of a DPA attack is to choose an attack target: this must be some intermediate result of the cryptographic algorithm that is executed by the attacked device. The chosen intermediate result (or target function, or selection function) needs to be a function of some known data d and a small part k of the key.

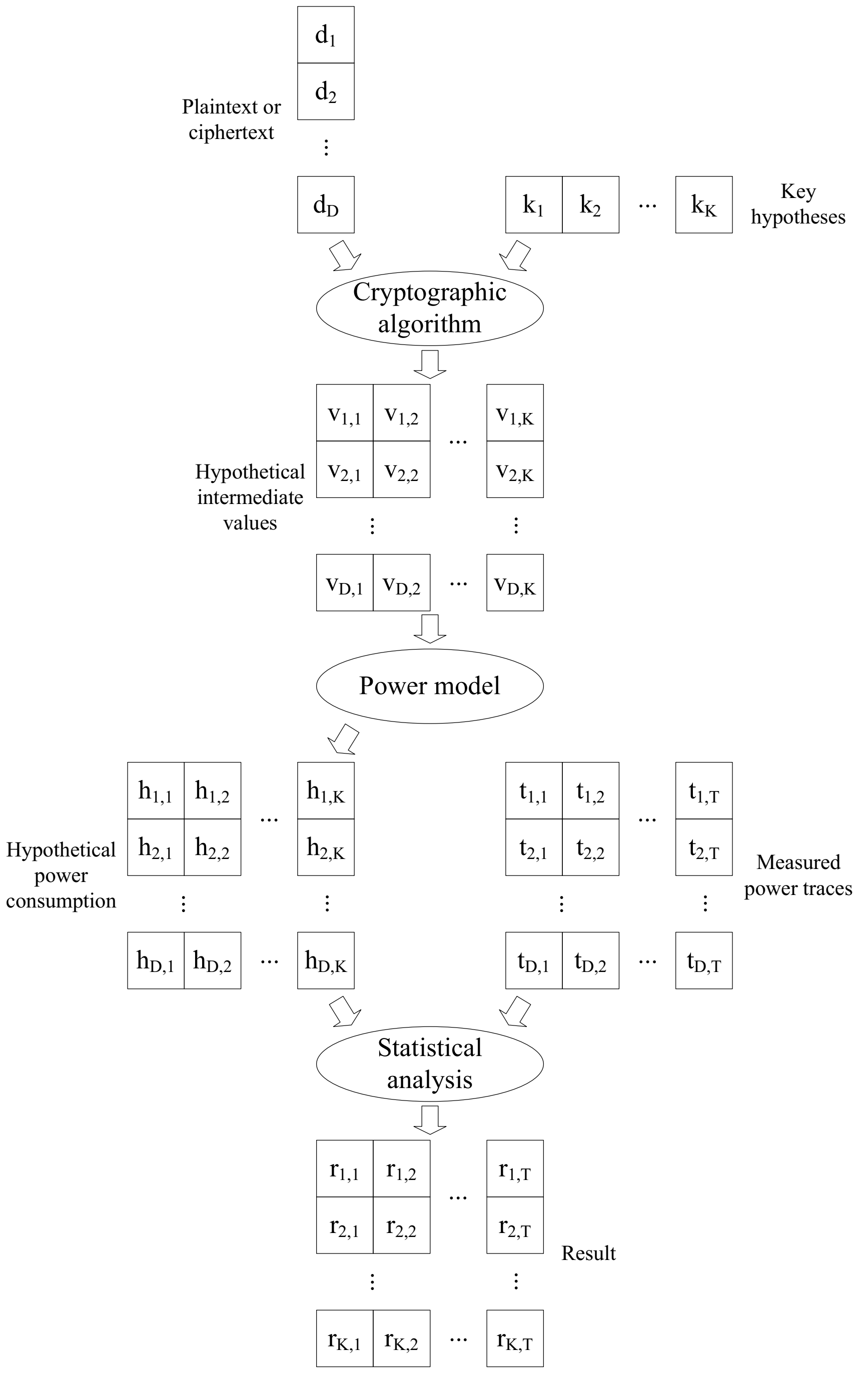
Step 2: Measuring the Power Consumption The second step of a DPA attack is to measure the power consumption of the cryptographic device while it encrypts or decrypts D different messages (these maybe plaintexts or ciphertexts). For each of these encryption or decryption runs, the attacker needs to know the corresponding data value d that is involved in the calculation of the intermediate result chosen in step 1. We write these known data values as vector $\mathbf{d} = (d_1, \ldots, d_D)'$, where d_i denotes the data value in the i^{th} encryption or decryption run.

During each of these runs the attacker records a power trace. We refer to the power trace that corresponds to data block d_i as $\mathbf{t}'_i = (t_{i,1}, \ldots, t_{i,T})$, where T denotes the length of the trace. The attacker measures a trace for each of the D data blocks, and hence, the traces can be written as matrix $\mathbf{T}$ of size $D \times T$. It is important for DPA attacks that the measured traces are correctly aligned. This means that the power consumption values of each column $\mathbf{t}_j$ of the matrix $\mathbf{T}$ need to be caused by the same operation. In practice this may not be the case and then one has to process power traces carefully. There are a number of alignment and general signal processing techniques out there, but they are beyond the scope of this unit.

Step 3: Predicting Intermediate Values The next step of the attack is to calculate a *hypothetical intermediate value* for every possible choice of k. We write these possible choices as vector $\mathbf{k} = (k_1, \ldots, k_K)$, where K denotes the total number of possible choices for k. In the context of DPA attacks, we usually refer to the elements of this vector as key hypotheses. Given the data vector $\mathbf{d}$ and the key hypotheses $\mathbf{k}$, an attacker can easily calculate hypothetical intermediate values $f(d, k)$ for all D encryption runs and for all K key hypotheses. This calculation (5) results in a matrix $\mathbf{V}$ of size $D \times K$. The first part of Figure 12 illustrates this calculation step.

$$v_{i,j} = f(d_i, k_j) \quad i = 1, \ldots, D \quad j = 1, \ldots, K \tag{5}$$

Column j of $\mathbf{V}$ contains the intermediate results that have been calculated based on the key hypothesis k_j. It is clear that one column of $\mathbf{V}$ contains those intermediate values that have been calculated in the device during the D encryption or decryption runs. Remember, $\mathbf{k}$ contains all possible choices for k. Hence, the value that is used in the device is an element of $\mathbf{k}$. We refer to the index of this element as ck. Hence, k_{ck} refers to the key of the device. The goal of DPA attacks is to find out which column of $\mathbf{V}$ has been processed during the D encryption or decryption runs. As soon as we know which column of $\mathbf{V}$ has been processed in the attacked device, we immediately also know k_{ck}.



Plaintext or ciphertext
d_1
d_2
d_D
k_1
k_2
k_K
Key hypotheses
Cryptographic algorithm
Hypothetical intermediate values
$v_{1,1}$ $v_{1,2}$ $v_{1,K}$
$v_{2,1}$ $v_{2,2}$ $v_{2,K}$
$v_{D,1}$ $v_{D,2}$ $v_{D,K}$
Power model
Hypothetical power consumption
$h_{1,1}$ $h_{1,2}$ $h_{1,K}$
$h_{2,1}$ $h_{2,2}$ $h_{2,K}$
$h_{D,1}$ $h_{D,2}$ $h_{D,K}$
Measured power traces
$t_{1,1}$ $t_{1,2}$ $t_{1,T}$
$t_{2,1}$ $t_{2,2}$ $t_{2,T}$
$t_{D,1}$ $t_{D,2}$ $t_{D,T}$
Statistical analysis
Result
$r_{1,1}$ $r_{1,2}$ $r_{1,T}$
$r_{2,1}$ $r_{2,2}$ $r_{2,T}$
$r_{K,1}$ $r_{K,2}$ $r_{K,T}$

Figure 12: Block diagram illustrating the steps 3 to 5 of a DPA attack.

Step 4: Mapping Intermediate Values to Power Consumption Values The next step of a DPA attack is to map the hypothetical intermediate values $\mathbf{V}$ to a matrix $\mathbf{H}$ of *hypothetical power consumption values*, see Figure 12. For this purpose, the adversary either relies on a standard leakage model (such as the Hamming weight), or they determine a more accurate model via a process called profiling (templating).

Step 5: Comparing the Hypothetical Power Consumption Values with the Power Traces After having mapped $\mathbf{V}$ to $\mathbf{H}$, the final step of a DPA attack can be performed. In this step, each column $\mathbf{h}_i$ of the matrix $\mathbf{H}$ is compared with each column $\mathbf{t}_j$ of the matrix $\mathbf{T}$. This means that the attacker compares the hypothetical power consumption values of each key hypothesis with the recorded traces at every position. The result of this comparison is a matrix $\mathbf{R}$ of size $K \times T$, where each element $r_{i,j}$ contains the result of the "comparison" between the columns $\mathbf{h}_i$ and $\mathbf{t}_j$. The comparison is done based on a statistical distinguisher. In practice the correlation coefficient tends to be an excellent choice for this purpose. The idea is then that a higher value $r_{i,j}$ indicates a better match between the columns $\mathbf{h}_i$ and $\mathbf{t}_j$. The key of the attacked device can hence be revealed by checking which key hypothesis leads to the best (highest) match. The indices of the highest values of the matrix $\mathbf{R}$ also reveal the positions at which the chosen intermediate result has been processed and the key that is used by the device.

If all values of $\mathbf{R}$ are approximately the same, this indicates the attack hasn't succeeded. Reasons for failure include an insufficient amount of traces, a bad choice for an intermediate value, or an inappropriate power model. Or, there simply is a bug in your code.

6 Configuring an attack: an example for software

We choose the output byte of the first AES S-box in round one (step 1). This intermediate result is a function of the first byte of plaintext and the first byte of the secret key. Accordingly we sampled the power consumption of a simple microcontroller during the first AES round. We took traces corresponding to 1 000 different plaintexts, which we saved alongside. Hence we have a matrix $\mathbf{T}$ of power consumption values (step 2). We calculated hypothetical intermediate values based on the 1 000 known plaintexts. This means that we have calculated the values $v_{i,j} = S(d_i \oplus k_j)$, where $d_1, \ldots, d_{1\,000}$ are the first byte of each of the 1 000 plaintexts and $k_j = j - 1$ with $j = 1, \ldots, 256$. The matrix $\mathbf{V}$ has hence a size of $1\,000 \times 256$ values (step 3). Next we mapped this matrix to hypothetical power values using the Hamming weight leakage model, leading to a matrix $\mathbf{H}$, $h_{i,j} = HW(v_{i,j})$ (step 4). We used the correlation coefficient to determine the linear relationship between the columns $\mathbf{h}_i$ and $\mathbf{t}_j$ for $i = 1, \ldots, K$ and $j = 1, \ldots, T$ (step 5).

This attack configuration results in a matrix $\mathbf{R}$ of estimated correlation coefficients

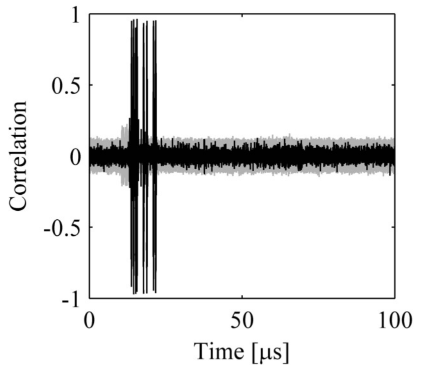


Figure 13: All rows of **R**. Key hypothesis 225 is plotted in black, while all other key hypotheses are plotted in gray.

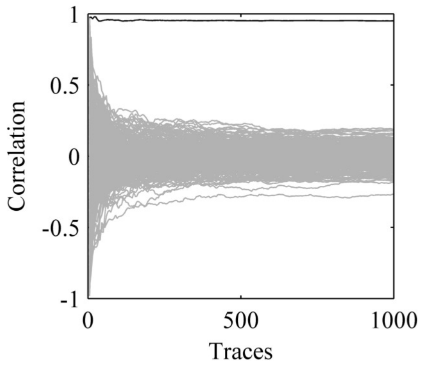


Figure 14: The column of **R** at $13.8\,\mu$s for different numbers of traces. Key hypothesis 225 is plotted in black.

$r_{i,j}$. Using the just introduced notation we can therefore estimate r_{ij} as given in (6). In (6), the values $\bar{h}_i$ and $\bar{t}_j$ denote the mean values of the columns $\mathbf{h}_i$ and $\mathbf{t}_j$. Figure 13 shows the result of this attack. The plot for key 225 again contains the highest peaks.

$$r_{i,j} = \frac{\sum_{d=1}^{D}(h_{d,i}-\bar{h}_i)\cdot(t_{d,j}-\bar{t}_j)}{\sqrt{\sum_{d=1}^{D}(h_{d,i}-\bar{h}_i)^2\cdot\sum_{d=1}^{D}(t_{d,j}-\bar{t}_j)^2}} \tag{6}$$

7 Analysing Attack outcomes

At this point, we have a framework for understanding and also practically configuring differential style attacks. Given that a number of choices are available for the different steps, a number of obvious questions arise: how do we judge the quality of an attacks; what may the impact of different choices for the different steps be on the quality; can we predict, based on trace characteristics, how well attacks should work; and is there a "best" attack?

Answers to these questions are in fact a matter of ongoing research, and I will aim here to give you a fair summary of what we (= the community of researchers and practitioners) know for certain.

7.1 Signal to Noise ratio

We introduced earlier to concept of data and operation dependency via making them explicit as variables in the total power consumption. Besides these components we also explicitly named a third component: noise (i.e. variation in power that is due to other things going on in the circuit). In differential style attacks we are targeting the same operation (i.e. this variable is constant) across the different traces. The only variation in these traces is due to data (or "the signal") and noise. Intuitively we would expect that the ratio between signal and noise will have a strong impact on how well attacks work in practice.

In the (statistical) signal processing, the signal-to-noise ratio (or SNR) is defined as the ratio of the average power of a signal (meaningful information) to the average power of background noise: SNR $= \frac{P_{\text{signal}}}{P_{\text{noise}}}$.

But what is the (average) power of a signal? The power $p(t)$ of an (electrical) signal is defined as the product of current $i(t)$ and voltage $v(t)$. Ohm's law relates voltage to current and resistance $R: v(t) = Ri(t)$. Therefore we have that $p(t) = i(t)v(t) = Ri^2(t)$. Average power is then that power over time t, which in case of a discrete signal (this is what a scope gives us) is given as $RE(i^2(t))$. Because one can mean centre traces without loss of information, we have that $E(X) = 0$ and therefore $E(X^2) = VAR(X)$, thus we can define the SNR (setting $R = 1$) generally as

$$SNR = \frac{Var(Signal)}{Var(Noise)} \tag{7}$$

In case of a differential style attack, the *signal* corresponds to what the adversary predicts, and therefore exploits. This may sometimes only be a fraction of P_{data}. Therefore we make this explicit and use the notation P_{exp} to refer to the power consumption that is due to the prediction. The noise component that we identified before, which is independent of the signal, retains it's meaning. It is important to keep in mind that there may hence be a fraction of the data dependent power consumption which is not captured by an adversary and this makes calculating SNRs potentially tricky in practice. The specific SNR then for a differential attacker is:

$$SNR = \frac{Var(P_{exp})}{Var(P_{noise})} \tag{8}$$

It is possible, if the leakage distribution of P_{exp} is known (i.e. one has done a statistical analysis), to calculate the SNR for trace points. This calculation is straightforward if $P_{data} = P_{exp}$, and I just show you some results in Figure 15. The first plot of Figure 15 shows 9 average traces of the simple microcontroller that forms the basis for all experiments in these notes. These 9 traces were obtained by sampling the power consumption of a memory transfer with uniformly distributed data. From the randomly sampled data, I then collected all traces associated with 0, 1, 2, ... Hamming weight, averaged over each class, and then presented the resulting average traces. You can see that the traces differ in several regions of the figure, in particular when the signal rises or falls sharply (i.e. near a clock edge). The data dependency, i.e. the signal, is exactly in these moments where these differences occur.

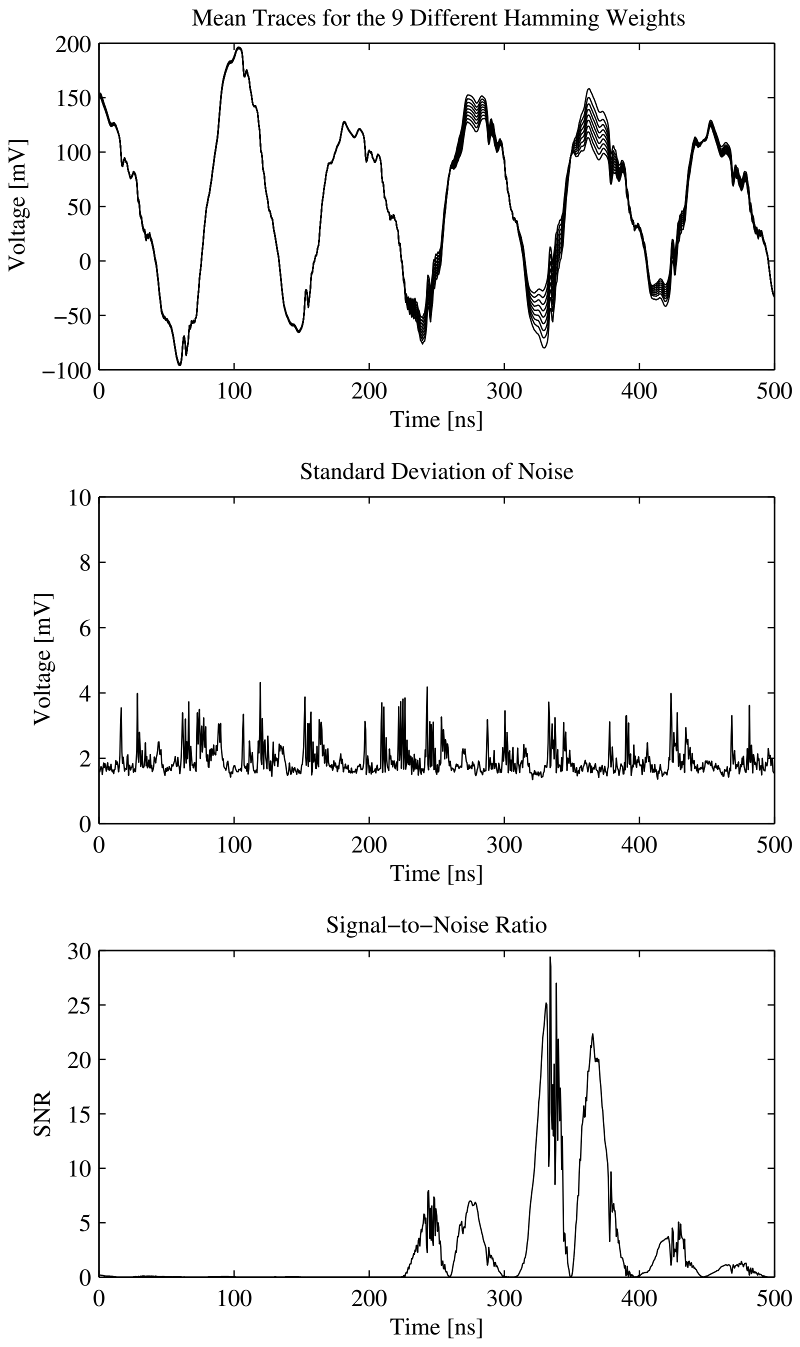


Figure 15: The signal levels, the standard deviation of the noise, and the SNR when attacking a uniformly distributed 8-bit data value on our microcontroller.

The second plot shows the standard deviation of the power traces for the case that the microcontroller processes some fixed data, i.e. this plot shows the standard deviation of the electronic noise P_{noise}. Obviously the noise isn't the same at the different moments in time, and it shows a patter that coincides with the clock edges (the faster signals change in a circuit the more power is consumed and the more variation there is).

The third plot shows the SNR. The SNR is highest where we expect it to be: in the points where the average traces differ most, the SNR is highest.

The SNR definition as we are going to use it in this handout will turn out to be useful to predict and understand DPA outcomes. However it does require to estimate the distribution of P_{exp}, which is not always possible (e.g. the M3 on the SCALE board is considerably more complex than the device that was used to generate the pictures here, where P_{exp} is a function of not just the current but also previous data values). In such cases an alternative definition for the SNR has been used in the community which uses the reciprocal of the coefficient of variation, i.e., the ratio of mean to standard deviation of a signal: $\text{SNR} = \frac{\mu}{\sigma}$, where μ is the mean of P_{data} and σ is an estimate of the standard deviation of the noise.

7.2 Distinguisher statistic

In this handout we have introduced what is called a "standard DPA attack" scenario as defined in [1]. A succinct and slightly more formal summary () of this strategy is as follows. We assume that the power consumption P of a cryptographic device depends on some internal value (or state) $F_{k^*}(X)$ which we call the *target*: a function $F_{k^*} : \mathcal{X} \to \mathcal{Z}$ of some part of the known plaintext—a random variable $X \overset{R}{\in} \mathcal{X}$—which is dependent on some part of the secret key $k^* \in \mathcal{K}$. Consequently, we have that $P = L \circ F_{k^*}(X) + \varepsilon$, where $L : \mathcal{Z} \to \mathbb{R}$ describes the data-dependent component and ε comprises the remaining power consumption which we modelled as independent random noise. The attacker has N power measurements corresponding to encryptions of N known plaintexts $x_i \in \mathcal{X}$, $i = 1, \ldots, D$ and wishes to recover the secret key k^*. The attacker can accurately compute the internal values as they would be under each key hypothesis $\{F_k(x_i)\}_{i=1}^{D}$, $k \in \mathcal{K}$ and uses whatever information he possesses about the true leakage function L to construct a prediction model $M : \mathcal{Z} \to \mathcal{M}$.

DPA is motivated by the intuition that the model predictions under the correct key hypothesis should give more information about the true trace measurements than the model predictions under an incorrect key hypothesis. A distinguisher D is some function which can be applied to the measurements and the hypothesis-dependent predictions in order to quantify the correspondence between them. For a given such comparison statistic, *Dist*, the *estimated* vector from a practical instantiation of the attack is $\hat{\mathbf{Dist}}_D = \{\hat{Dist}_D(L \circ F_{k^*}(\mathbf{x}) + \mathbf{e}, M \circ F_k(\mathbf{x}))\}_{k \in \mathcal{K}}$ (where $\mathbf{x} = \{x_i\}_{i=1}^{D}$ are the known inputs and $\mathbf{e} = \{e_i\}_{i=1}^{N}$ is the observed noise). Then the attack is *o-th order successful* if

$\#\{k \in \mathcal{K} : \hat{\mathbf{Dist}}_D[k^*] \leq \hat{\mathbf{Dist}}_D[k]\} \leq o$.

The *success rate* of a DPA attack is the probability that the correct key is ranked first by the distinguisher (the *o-th order success rate* is the probability it is ranked among the o first candidates).

An intuitive choice for the comparison statistic $Dist$ is the (Pearson) correlation. It is based on the covariance (which is used to express the linear relationship between two statistical variables). The definition of the covariance is given in (9) and an equivalent form is given in (10). The latter definition (10) shows that the covariance is also related to the concept of statistical dependence. If X and Y are statistically independent, then $E(XY) = E(X) \cdot E(Y)$, and therefore $Cov(X, Y) = 0$. For normally distributed X and Y also the converse holds. This means, if $Cov(X, Y) = 0$, then X and Y are independent.

$$\begin{aligned} Cov(X,Y) &= E((X - E(X)) \cdot (Y - E(Y))) \qquad (9) \\ &= E(XY) - E(X) \cdot E(Y) \qquad (10) \end{aligned}$$

The correlation ρ is essentially a "normalised" version of the covariance (i.e. it can only take values between plus and minus one: $-1 \leq \rho \leq 1$). It is typically unknown and needs to be estimated. The correlation ρ and its' estimator r are defined in (11) and (12) resp.

$$\begin{aligned} \rho(X,Y) &= \frac{Cov(X,Y)}{\sqrt{Var(X) \cdot Var(Y)}} \qquad (11) \\ r &= \frac{\sum_{i=1}^{n}(x_i - \bar{x}) \cdot (y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \cdot \sum_{i=1}^{n}(y_i - \bar{y})^2}} \qquad (12) \end{aligned}$$

7.3 Number of Needed Traces

In DPA attacks, the correlation coefficients between the columns of the predictions $\mathbf{H}$ and traces $\mathbf{T}$ are estimated based on the D elements of these columns. The more traces an attacker acquires, the more precisely the estimated correlation coefficients $r_{i,j}$ of the matrix $\mathbf{R}$ match the actual correlations $\rho_{i,j}$.

Often the pertinent question in the context of DPA attacks is how many traces does an attacker need to acquire in order to determine the key that is used by the device? In case of the correlation coefficient this means: how many traces are needed in order to find out which column of $\mathbf{H}$ has the strongest correlation to a column of $\mathbf{T}$? Although this question looks pretty simple at first sight, giving a precise answer is not straightforward without knowledge of the device's leakage characteristics. In fact, we can only provide a rule of thumb in this case.

Recall that (assuming a good enough device leakage model is used) , a successful DPA means that there is a significant peak for the point in time when the key dependent intermediate is processed (aka row *ck* and column *ct* of $\mathbf{R}$). It may happen that at time *ct* more key hypotheses than just *ck* show some peaks — but we now make the deliberate, simplifying assumption that the peaks associated with other key hypothesis are insignificant: we "pretend" they are 0. Therefore, a DPA attack will be successful if the correlation associated with ck can be distinguished from 0.

The concept of a sampling distribution is helpful in understanding the next ideas that enable us to provide a rule of thumb for the number of needed power traces. To introduce this concept I suggest a thought experiment. Suppose that we would repeat an experiment that samples D traces from a device (with a fixed key k^*) and calculates a DPA attack with some known inputs, i.e. in our thought experiment we produce several distinguishing vectors $\mathbf{r}$. Let's just focus on the estimates for the correct key and correct time. These individual correlation coefficients are unlikely to be exactly identical: we would expect them to be very similar, but because traces are noisy, we wouldn't expect them to be the same. Hence the estimates r are in fact realisations of a random variable. They have a distribution (it's called the sampling distribution), which in the case of correlation turns out to be quite involved and tricky to work with. However, if a sufficiently large number of traces (≥ 30) can be measured, and the (true) correlation is small (i.e.≤ 0.2) a transformation that is due to Fisher can be used to map the random variable R to a random variable Z_1 that has a normal distribution, see (13). The mean of Z_1 is then given by μ, see (14), and the variance is σ^2, see (15). The value n denotes the number of traces. Note that the fraction $\rho/(2 \cdot (n-1))$ approaches zero for large n and can thus be omitted from the formula.

$$Z_1 = \frac{1}{2} \ln \frac{1+R}{1-R} \tag{13}$$

$$\mu = \frac{1}{2} \ln \frac{1+\rho}{1-\rho} + \frac{\rho}{2 \cdot (n-1)} \tag{14}$$

$$\sigma^2 = \frac{1}{n-3} \tag{15}$$

Now that we have reasoned that the any estimate r that we compute is a realisation of the sampling distribution of R, we turn our attention back to the question of how many traces we need such that we can distinguish a non-zero r from 0 with a high probability. We also know that our estimate r gets better the more traces we have, and that a better estimate for r will make it easier to distinguish it from 0.

In statistics, we use *confidence intervals* to telling how good a given approximation is. When we say that we have a 0.99 confidence interval for some estimate, then we mean that our construction delivers an interval that contains that estimate with probability 0.99.

Hypotheses tests are closely connected to confidence intervals. In a hypothesis test, we test whether a certain hypothesis that we make is true or not. For instance, we can test whether r is equal to a certain value r_0 or not. In our use case, we formulate two hypotheses. The first (null) hypothesis is that H_0: $\rho = \rho_0$. The second, alternative hypothesis is H_1: $\rho \neq \rho_0$. A confidence interval contains all the values that are reasonably close to a certain parameter. Hence, if we want to know whether a certain parameter is equal to a certain value, we have to determine whether this value lies in the confidence interval of the parameter or not. In other words, a hypothesis test accepts all those values of the null hypothesis, which lie in the respective confidence interval.

Confidence intervals can be easily constructed for normal variables. Because of Fisher's transformation, we have that $Z = (Z_1 - \mu)/\sigma \sim \mathcal{N}(0,1)$. We can also use Fisher's transform to map 0 correlation to a normal distributed variable R_0.

$$\begin{aligned}
R_0 \quad \mapsto \quad & Z_0 = \frac{1}{2} \cdot \ln \frac{1+R_0}{1-R_0} \\
& \mu_0 = E(Z_0) = \frac{1}{2} \cdot \ln \frac{1+\rho_0}{1-\rho_0} + \frac{\rho_0}{2 \cdot (n-1)} \\
& \sigma^2 = Var(Z_0) = \frac{1}{n-3} \\
R_1 \quad \mapsto \quad & Z_1 = \frac{1}{2} \cdot \ln \frac{1+R_1}{1-R_1} \\
& \mu_1 = E(Z_1) = \frac{1}{2} \cdot \ln \frac{1+\rho_1}{1-\rho_1} + \frac{\rho_1}{2 \cdot (n-1)} \\
& \sigma^2 = Var(Z_1) = \frac{1}{n-3}
\end{aligned}$$

The distribution of the difference $Z_0 - Z_1$ is a normal distribution $Z_0 - Z_1 \sim \mathcal{N}(\mu_0 - \mu_1, \sqrt{2} \cdot \sigma)$. Hence, $((Z_0 - Z_1) - (\mu_0 - \mu_1))/(\sqrt{2} \cdot \sigma)$ has a standard normal distribution. The two-sided confidence interval for $\mu_0 - \mu_1$ is

$$\left[(Z_0 - Z_1) - z_{1-\alpha/2} \cdot \sqrt{\frac{2}{n-3}}, (Z_0 - Z_1) + z_{1-\alpha/2} \cdot \sqrt{\frac{2}{n-3}} \right]. \tag{16}$$

The number of traces n that is necessary to assert with a confidence of $1 - \alpha$ that the two normal distributions $Z_0 \sim \mathcal{N}(\mu_0, \sqrt{1/(n-3)})$ and $Z_1 \sim \mathcal{N}(\mu_1, \sqrt{1/(n-3)})$ are different is given by:

$$n = 3 + 8 \cdot \frac{z_{1-\alpha}^2}{\left(\ln \frac{1+\rho_0}{1-\rho_0} - \ln \frac{1+\rho_1}{1-\rho_1} \right)^2} \tag{17}$$

Now we can provide a simplified "rule of thumb" for the number of needed power traces. To do so, we set $\rho_0 = 0$. Because we want attacks to work with (near) certainty, we adjust the parameter α to be 0.9999, which means that $z_{0.9999} = 3.719$ [1]. The number of traces n that are needed to mount a successful DPA attack can be approximated as follows:

$$n = 3 + 8 \frac{z_{1-\alpha}^2}{\ln^2 \frac{1+\rho_{ck,ct}}{1-\rho_{ck,ct}}} \tag{18}$$

[1] If you are not familiar with type 1 and type 2 errors and quantiles, then you need to trust me that I made the right choices

Table 2: The calculated number of traces n for different values of $\rho_{ck,ct}$.

$\rho_{ck,ct}$	$n_{\alpha=0.0001}$	$\rho_{ck,ct}$	$n_{\alpha=0.0001}$	$\rho_{ck,ct}$	$n_{\alpha=0.0001}$
0.900	16	0.090	3 400	0.009	341 493
0.800	26	0.080	4 307	0.008	432 206
0.700	40	0.070	5 630	0.007	564 519
0.600	61	0.060	7 668	0.006	768 378
0.500	95	0.050	11 049	0.005	1 106 471
0.400	157	0.040	17 273	0.004	1 728 870
0.300	292	0.030	30 720	0.003	3 073 559
0.200	676	0.020	69 140	0.002	6 915 526
0.100	2 751	0.010	276 606	0.001	27 662 152

This formula may not be very helpful at first because you may argue that nobody knows $\rho_{ck,ct}$ a priori. However, there are a number of estimation techniques that practitioners will use to determine this value. Furthermore, one can view this value as a kind of "security" parameter: designers/implementers will do their best to make $\rho_{ck,ct}$ as small as possible. The "security level" device is then the "remaining" correlation, which can be directly mapped to a number of needed traces for DPA style attacks.

Table 2 shows how the number of traces increases when the correlation decreases. For $|\rho_{ck,ct} \leq 0.2|$ and for small SNRs, the following relations hold:

$$n \approx \frac{28}{\rho^2_{ck,ct}} \sim \frac{1}{SNR} = \frac{Var(P_{noise})}{Var(P_{exp})}. \tag{19}$$

- Halving $\rho_{ck,ct}$ means that four times more traces are needed.
- Doubling the amount of noise reduces $\rho_{ck,ct}$ by $\sqrt{2}$, and hence doubles the number of needed traces.
- Halving the capacitance of the wires processing the attacked intermediate results halves the amplitude of P_{exp}, quarters the SNR, and hence quadruples the number of traces.

8 Food for thought and Exercises

The tasks are designed to get you thinking in more depth about side channels that may exist in AES implementations.

Task A: Reasons for variable running times

It may seem not immediately obvious why some AES implementations would not be constant time. Spend some time researching papers/implementation options that could potentially lead to timing leaks.

Your AES implementation may differ from the one that I based this note on. What (high level) features can you identify in power traces of your implementation?

Task B: SPA of your AES implementation

Aquire some AES traces (keep data and key constant), and average over them to reduce the effect of noise. What features can you identify in the resulting “clean” trace? Can you see rounds, or even round functions? Can you identify low level features?

You may want to judge the quality of your acquired traces. How much variation is there in them?

Task C: SNR of your AES implementation

Aquire some AES trace snippets (i.e. “cut out” some relevant parts like one AddRoundKey or one SubBytes), and compute trace variances and try and estimate the SNR.

For the DPA rule of thumb we made the assumption that the correlation of incorrect key hypotheses is 0.

Task D: Correlation coefficients of incorrect keys

Perform some DPA attacks and examine the correlation coefficients of incorrect key hypotheses. In particular, run a DPA attack that targets the SubBytes output and compare the results with a DPA attack that targets the AddRoundKey output.

Most papers discuss attacks on the SubBytes output function.

Task E: Correlation attacks on MixColumns

How could you adapt your attack strategy to target intermediates or even the output of MixColumns?

Task F: Correlation attacks on AddRoundKey

Adjust your attack so that targets the AddRoundKey function. How well does key recovery work in this case? Why might it be better or worse than an attack on SubBytes or MixColumns?

9 Quiz

If you have worked through these notes and spent some time on the tasks, you should be able to answer the following questions.

1. Explain which intermediate values of AES can be targeted with a key hypothesis that is no larger than 8 bits.
2. Relate the concept of the SNR to the quality power analysis attacks. For this purpose define the signal, noise, and how the SNR relates to the number of needed power traces.
3. What is the DPA "rule of thumb"? When does it apply and when does it not?
4. Imagine a differential style attack on DES. Explain for each of the 5 steps of a differential attack what options there are and how you would suggest to proceed.
5. Imagine an attack on AES-192. How would you be able to recover all of the 192 key bits?

References

[1] S. Mangard, E. Oswald, and F-X. Standaert. One for All – All for One: Unifying Standard DPA Attacks. *IET Information Security*, 5(2):100–110, 2011.

[2] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards* . Springer, 2007.