Technische Universität München

Fakultät für Informatik

Master's Thesis in Computer Science

Design of a Framework for Side-Channel Attacks on RFID-Tags

Entwicklung eines Frameworks für Seitenkanalangriffe auf RFID-Tags

Author: Hagen Fritsch

Supervisor: Prof. Dr. Claudia Eckert

Advisors: Dipl.-Inf. Daniel Angermeier
Benedikt Heinz

Submission Date: 12. November 2010

Abstract

RFID systems are not yet typical targets for side-channel attacks, although they are just as likely vulnerable as their contact-based counterparts. Based on a real world scenario, a framework to assist execution of such attacks on RFID systems is developed resulting in a powerful and versatile framework that is already used to analyze the results of basic measurement setups. These results are sufficient to reveal concrete power traces, but due to deficient measurement setups, power analysis attacks were impossible. However, measurement setups were not part of the thesis and the functionality of the framework could be successfully verified using simulated traces. These also yielded requirements for such measurement setups in which the carrier impact needs to be drastically reduced in order to allow any practical DPA attack. Such a setup seems however practical and future work is likely to show such attacks.

Zusammenfassung

RFID-Systeme waren bisher kein typisches Ziel für Seitenkanalangriffe, obwohl sie dafür vermutlich genauso anfällig sind, wie ihre kontaktbasierten Pendants. Anhand eines realen Szenarios entstand im Rahmen dieser Arbeit ein leistungsstarkes und vielseitiges Framework zur Unterstützung der Durchführung solcher Angriffe auf RFID-Systeme. Es wurde bereits zur Analyse von Aufzeichnungen einfacher Messaufbauten eingesetzt und die daraus gewonnenen Ergebnisse reichen aus, um konkrete Stromprofile aufzudecken. Aufgrund der mangelhaften Messaufbauten, die jedoch nicht Teil dieser Arbeit waren, waren Stromverbrauchsanalyse-Angriffe unmöglich. Die Funktionalität des Frameworks konnte allerdings mittels simulierter Stromprofil-Traces erfogreich verifiziert werden. Diese ergaben darüber hinaus Anforderungen an Messaufbauten für derartige Angriffe: Um überhaupt einen praktischen DPA Angriff zuzulassen, müsste der Einfluss des Trägersignals in den Aufzeichnungen drastisch reduziert werden. Solch ein Messaufbau erscheint allerdings durchaus realistisch, weswegen zukünftige Arbeiten zu dem Thema sicherlich solche Angriffe aufzeigen werden.

CONTENTS

CHAPTER

ONE

THEMATIC INTRODUCTION

Nowadays, RFID systems are known to almost everyone. With the variety of possible applications, these systems have invaded our lives, with most people carrying multiple smartcards with them, many of which are already contactless smartcards based on RFID technology. With the advance of these technologies, they also become interesting for attackers and withstanding attacks becomes a crucial property. In the past years, numerous RFID systems have been misused or their security has been broken or weakened, because of a lack of awareness or the use of homebrew cryptosystems with serious vulnerabilities. For example, the Hitag2 system has been broken in 2009a due to weaknesses in the cipher's initialization (Plötz and Nohl, 2009a). The Mifare classic system has been broken due to a too simple cipher and several architectural mistakes (Garcia et al., 2008; de Koning Gans et al., 2008). The Legic Prime family of RFID cards features a proprietary protocol with a cipher that can only be regarded as obfuscation, because the key is sent unencrypted over the air interface after the activation of the card (Plötz and Nohl, 2009b). Research is ongoing and as RFID cards are attractive targets, more protocols will be revealed and further flawed crypto systems will be broken. However, manufacturers learned from past mistakes and are now producing chips with more modern cryptography. This is possible due to advances in semiconductor manufacturing processes that nowadays allow to build gates in much smaller structures consuming less power. Consequently more gates can be put on a smartcard's microchip, allowing for the modern cryptography that was earlier infeasible on such systems.

Besides mathematical analysis, it is already known for a long time, that systems can be breached using side-channel attacks. Most prominent of those are differential power analysis attacks, which are very powerful and are already known to easily reveal the key of unprotected contact-based smartcards, even if modern cryptographic algorithms are used. These attacks have typically been conducted by measuring the power consumption of the device in question. Applying DPA attacks to RFID systems is a relatively new concept, although only a logical consequence. However, it is inequally harder to do so, as the contactless smartcards are passively powered, so that measurements of the power consumption are not directly possible. Therefore this thesis' task is to develop a framework to aid preprocessing, analysis and visualization of the recorded traces. Furthermore the framework can assist in the investigation of possible measurement setups and evaluation of their quality, but the development and improvement of such measurement setups are outside this thesis' scope.

1.1. Methodology

In order to acquire detailed knowledge on the requirements for the framework, it is part of this thesis' task to run a real world scenario. Several attack targets will be chosen for the execution of experiments. The optimal goal would be to run a set of measurement and use

the workflow model of the framework to break a target's cryptographic key. A couple of tools were already available for preprocessing or analysis that were previously used in wired DPA scenarios. It is thus expected to evaluate the usefulness and practicability of these tools and to record the requirements for new tools or workflows if necessary.

It is assumed that the power analysis measurement setup needs to be tuned in order to get the best results. As such, several measurement runs with possible different setups have to be executed and their results have to be analyzed, compared and visualized, all of which are tasks the framework shall be capable of, at least in assisting the workflow to reduce the manual work to a minimum. These tasks will show the usefulness of the framework and put it under real test conditions.

Several preprocessing steps will be tried, to find out, how to yield the most relevant information for power analysis attacks. In order to actually run an attack, correlations for plain- and ciphertexts will be calculated to locate interesting points in the power traces. A visual inspection for cryptographic patterns might aid this process. Finally it was planned to generate hypothesis on the very implementation of the cipher, which would eventually lead to the revelation of the card's secret key. However, this could not be achieved, due to the lack of quality in the available measurement setups in which none of trace recordings contained enough usable information to perform such an attack.

1.2. Overview

The thesis is divided into six chapters. This first chapter serves as an introductory chapter. The second and third chapter build the theoretical block discussing side-channel attacks and giving an overview on the relevant aspects of RFID systems. In Chapter 4, a typical attack workflow is described together with discussions and descriptions of the actual implementation of the framework-specific parts of such a workflow and its functionality. More prerequisites needed for the implementation of the framework and most importantly the surrounding test environment for the actual attacks such as the protocols involved, follow in Chapter 5. Eventually, Chapter 6 describes the measurement setups and the conducted experiments, together with their results and an evaluation, which is continued in the last chapter rounding up the thesis with further ideas.

CHAPTER

TWO

SIDE-CHANNEL ATTACKS

Despite the possible provable security of a cipher or protocol, its actual implementation might still be subject to attacks based on leaked side-channel information. Side-channels were often overseen in the past due to a lack of awareness and their relationship on different layers of a security system involving both hardware and software, i.e. parts that are usually engineered by different people (Kocher et al., 1999). Closed systems such as smart cards, whether contactless or not, may leak information about the operations or the processed data on side-channels. The most popular and easily exploited of these side-channels are the time passed during the execution of an operation on the device as well as the power consumption or electromagnetic radiation during such an operation. The next sections introduce these side-channels and their exploits to provide a basic understanding for the attacks and analysis capabilities that the framework will need to support and which will eventually be run against RFID tags during the experimental part of this thesis.

2.1. Timing Analysis

Operations executed by a device might differ in their execution time due to a variety of reasons (Kocher, 1996). The most obvious one is found in the software layer: data dependent conditional operations. However, underlying layers such as caches or the actual hardware impose even more differences, of which the software engineer might not be aware. CPU instructions often take varying amounts of time for different operations or even have data dependent runtimes, as it is the case for operations that terminate early if one operand is zero.

A typical example for an attack that exploits different runtimes can be found for password protected devices that use a `strcmp` call to check the password. As the function returns as soon as the first character mismatch is found, the timing of this operation is dependent on the number of correct characters. As such, the brute-force range for a n byte password can be reduced from 256^n to $256 \cdot n$ since one will notice a different timing as soon as one tries a password that starts with the right character.

Countermeasures are easily implemented by making sure, that the runtime of the function is always the same. For the above example, a fixed `strcmp` function would compare *all* characters of the string, thus taking the same time no matter how many characters match.

One might assume that timing attacks only work for these straightforward bugs, but are not as relevant for real systems. This is however not the case: More complex attacks will exploit data-dependent execution times that only leak fuzzy information such as the hamming weight of a key using differential attack styles that will be described in Section 2.2.3 in the context of differential power analysis. Timing vulnerabilities were found in implementations of all

major crypto systems (e.g. Kocher, 1996; Schindler, 2000) and thus have to be taken seriously, especially as they can be exploited without any special or expensive equipment. However, as these attacks are well known and countermeasures are easy and cheap to implement, modern security devices are unlikely to leak any exploitable information in the timing domain. Still, several experiments will be conducted in the experimental part of this thesis (Section 6.1) to verify this assumption.

2.2. Power Analysis

When processing data, a microchip consumes power. As one can probably guess, this power consumption depends on the kind of operations executed and on the data processed by the device. For now it shall be assumed, that the operation-specific power consumption accounts for the major part, while processed data just slightly influences the whole power consumption. A more detailed background describing these dependencies will be given in Section 2.2.1. Due to the operation-specific power consumption, it is often possible to recognize algorithmic states (e.g. rounds in a block-cipher) or even single operations, just by visually inspecting a power trace, i.e. a recording of the power consumption of a device for a certain operation.

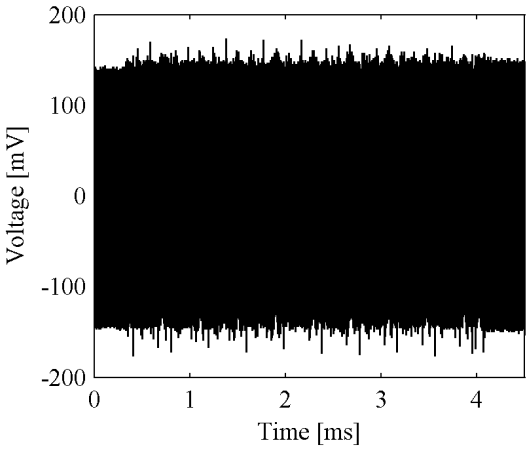

Figure 2.1.: Power profile of a AES encryption performed by a microcontroller (Mangard et al., 2007)

As an illustration, Figure 2.1 shows the power trace of an AES-128 operation, in which the ten rounds of the cipher are distinguishable at a closer look.

Since the processing of power traces is one of the most important functions of the framework, foundations and attacks will be detailed in the following sections.

2.2.1. The power model

In order to understand assumptions made on the power consumption, it is crucial to model it, based on the actual hardware design involved. Almost every microchip relies on CMOS logic which consists of cells made up of transistors. Each calculation is a state-change in the microchip and each change in a transistor's state consumes power since capacitors in the wiring are charged or discharged. Additionally, shortcut currents may flow for a short time in a transistor network transition. See Mangard et al. (2007) for further details on currents and noise sources.

Since attackers do not normally have access to a chip's netlist, they work with extremely simplified power models. A very simple model would be to assume, that the power consumption is proportional to the number of transistor transitions. This already disregards a wide variety of effects, such as different lengths of wiring and therefore its different capacities, but also that a transition from 0 to 1 consumes the same amount of energy as its counterpart. Despite its huge amount of inaccuracy, this power model is already very powerful and describes power traces fairly well. But still, attackers do not have a detailed knowledge about the transistors on the chip. They can however already distinguish operations to a great amount of detail, since the number of transistors transitions depends very much on the type of operation executed. Operations that take several clock cycles can thus be recognized by a distinguishable pattern (see Figure 2.1), which is the basis for the simple power analysis attacks described in Section 2.2.2.

Unfortunately for the attacker, data dependencies are not as clearly visible and other forms of attacks are required. Depending on the hardware implementation of the chip, different power models may apply for data dependent information. A typical one bit register will only consume power if a transition occurs. As such, a commonly used power model for registers is the hamming distance of two values, which counts the number of bits in which these values differ. If the new value is loaded into the register, only bits that are different from the original value will induce a state change and therefore only these will consume power.

Additionally in microcontrollers, data buses often use precharge logic in which the power consumption is dependent on the hamming weight of a value (i.e. the number of bits that are 1). Section 2.2.3 shows how these minimal information leaks can be exploited in differential power analysis attacks which are one of the key applications of the framework.

2.2.2. Simple Power Analysis

Historically, *simple power analysis* (SPA) has been developed before differential power analysis and its usages and applications are limited today. However, SPA serves as a good example to illustrate the severe impact that a clearly readable power profile might have on the security of a device. It also shows why data dependent analysis is much more difficult and motivates the more advanced differential attacks.

Typically, the attacker's goal is to extract the secret key of a device. In an optimal scenario, the power consumption at a certain moment in time would dependent on a key-bit and the attacker could distinguish the bit by looking at the trace. This is usually not possible, since the data-dependent power consumption is much less distinguishable than the operation-specific part (see Section 2.2.1). Additionally, the key-bits are usually not processed alone but in combinations of bytes or words and several noise sources complicate measurements even more. However, in algorithms that perform conditional operations based on secrets, it may very well be possible to distinguish patterns in the power trace and to reconstruct the key. This kind of attack is known as simple power analysis. A well known victim of such a type of attack is the square-and-multiply algorithm (Kocher et al., 1999), as it is used in modulo exponentiation such as in RSA (Messerges et al., 1999), but the basic principle also applies to similar algorithms such as the double-and-add version found in ECC (Caron, 1999). In RSA, an operand is either squared (key bit is 0) or squared and multiplied with another value (key bit is 1) based on some key information. The operations deal with large integers and have a distinguishable power profile, since their processing is split over many clock cycles. Consequently the power trace reveals the order of operations performed (see Figure 2.2). As each time an multiplication is performed (in addition to the squaring operation) corresponds to a 1 bit in the key and each omission of the multiplication corresponds to a

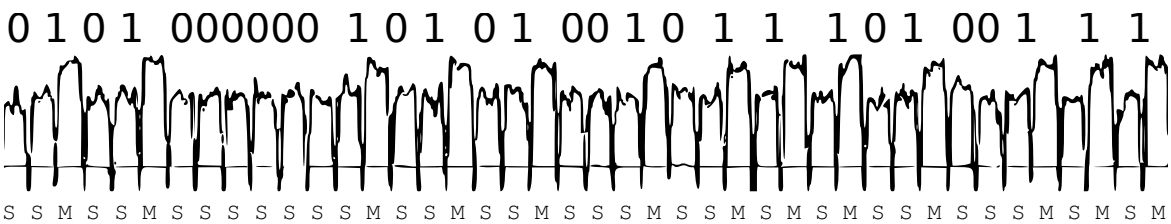

Figure 2.2.: Power profile of distinguishable square (S) and multiply (M) operations (based on Rohatgi, 2010)

0 bit, an attacker is likely able to reconstruct the entire secret key by visually inspecting a single power trace.

If a simple distinction is not possible because of noise, the attacker might still be able to run several measurements with the same data and reduce the noise by averaging the traces.

2.2.3. Differential Power Analysis

In order to attack data dependent power consumption, other means of analysis are required. Technically, *differential power analysis* (DPA) works by using statistical methods on numerous measurements to confirm or reject a certain hypothesis. Such a hypothesis is typically: key bit n is 1.

DPA attacks are extremely powerful and even work on very noisy recordings, which is why they are the most popular type of power analysis attacks (Mangard et al., 2007) and the most likely one to be used in RFID side-channel analysis as well. Consequently, functionality to support and perform DPA attacks are the core of the framework. This section will describe the most important principles of DPA attacks in more detail.

Notation: Conforming with Mangard et al. (2007), the following notation is used. The attacker records a set of traces $t_i = (t_{i,1}, \ldots, t_{i,T})$ with t_i being the ith trace recorded and T denoting the total trace length. For each trace, the device is fed with some known input data $d = (d_1, \ldots, d_D)$ (or some known data is read from the device) where again d_i corresponds to the ith trace recording i.e. to t_i. Accordingly, D refers to the total number of traces.

For example assume, that a device processes a bit of input data d_i at a certain time-offset j. A hypothetic power consumption would be, that the device consumes slightly more power if the bit is 1. In DPA scenarios, unknown components (such as the operation specific power consumption or power consumptions of unknown data) are typically modeled as Gaussian noise, thus a model for the power consumption could assume it to be proportional: $t_{i,j} \sim d_i + N_{i,j}$ with the noise components of this trace $N_i = (N_{i,1}, \ldots, N_{i,T})$, i.e. at offset j, the average $\lim\limits_{D\to\infty} \text{avg}((N_{1,j}, \ldots, N_{D,j})) = c$ is constant.

By measuring the device process random data, the attacker can now build two sets of traces. The first set will contain all traces in which the bit d_i is 0, thus its average (the infinity limes is omitted from now on) $\text{avg}(t_{i(d_i=0),o}) = c_1$ is constant, while the average in the second set will be another constant: $\text{avg}(t_{i(d_i=1),j}) = c_1 + d_i$ with d_i being due to the model's higher power consumption if the bit is 1. Depending on the number of measurements, the Gaussian noise component is reduced and consequently the two sets will have significantly different averages for each offset j at which d_i or d_i^{-1} is processed. For the remaining offsets j', it holds that $t_{i,j'} = N_{i,j'}$ since no data dependent input is processed, thus the differences in averages for these sets will be close to 0.

Kocher et al. (1999) note, that the two sets will only have the notable difference in their averages, if they are divided according to the input data d_i. If one chooses a random set, the average will approach the average for the whole set, e.g. $\text{avg}((t_{1,j}, \ldots, t_{D,j}) = c_1 + 0.5 \cdot d_i$ if $d_i = 1$ is equally likely as $d_i = 0$.

This can be used to locate the time offsets at which the desired data is processed by the chip, which is very useful in a profiling phase of an attack: interesting parts can be isolated and the trace length can be minimized. In the beginning though, an attacker has only a general idea at which point in time the operation happens. If the processed data is known and varies for each measurement (e.g. some data produced by the device such as a random number or a plaintext fed to the device) the attacker can formulate a hypothesis and using multiple measurements she can brute force the time offset of the data's processing by performing the above steps. The result will be a graph showing the differences of the averages between the two sets: $(c_1 + d_i) - (c_1) = d_i$. If the noise component is small enough, peaks in this graph indicate processing of the desired data. In Section 2.3 it will be elaborated, how this generic technique can further be used to attack the key of a cipher.

As the noise component for RFID traces will be rather high, it becomes interesting to estimate the number of traces required to safely distinguish two such sets. Following Mangard et al. (2007), this number depends on the sampling distribution of the average of the noise component. In Section 2.2.3 a formal way to assess the verification will be shown. For now, it can be noted, that the sampling distribution of the average of a normally distributed random variable $X \sim \mathcal{N}(\mu, \sigma)$ is again normally distributed and can be estimated as follows:

$$\mathrm{E}(\bar{X}) = \mu \tag{2.1}$$

$$\mathrm{Var}(\bar{X}) = \frac{\sigma^2}{n} \tag{2.2}$$

An important consequence of Equation 2.2 is, that the variance can be lowered by increasing the number of traces.

Correlation DPA

A more accurate way of evaluating hypothesis is the use of correlation instead of the difference-of-means approach presented in the previous section. Using correlation, models are no further limited to binary ones, but also allow more complex models such as the hamming weight or hamming distance of a byte or a word. These are very commonly used models in power analysis attacks (see Section 2.2.1).

Correlation describes the degree of a linear connection between two vectors (e.g. the hypothesis and the measured power consumption) and is invariant to scale and offsets of the input vectors. As such, correlation is very well suited for these generic models and the analysis of power traces under varying conditions. The correlation coefficient ρ is defined as follows (Mangard et al., 2007), where $h_k = (h_{k,1}, \ldots, h_{k,D})$ denotes a vector of the hypothetical power consumption and t is still the vector of the measured power consumptions with j referencing the sample position (time offset) as defined in the previous section:

$$\rho(h_k, t'_j) = \rho_{k,j} = \frac{\mathrm{cov}(h_k, t'_j)}{\sqrt{\mathrm{Var}(h) \cdot \mathrm{Var}(t'_j)}}$$
$$\text{with}: \quad t'_j := (t_{1,j}, \ldots, t_{D,j})$$

The correlation coefficient will always be between -1 and 1, where values close to zero indicate no correlation. In order to use correlation for attacks, hypothetical power consumptions

have to be calculated. This works similar to the difference-of-means method without the limitation to a binary model. If the device performs some calculations on an 8-bit value, the method of Section 2.2.3 can only ever locate one bit and thus has to deal with less exact values, as the noise component is stronger. Now, if a hamming-weight model can be assumed, the calculation for the hamming weight of the input data and the actual power consumption is calculated for each sample position of the traces, making use of all 8 bits. This again produces a graph, in which all sample positions, that are uncorrelated to the input data are close to 0. However, for positions that do process the input data, the correlation will be higher. Figure 2.3 gives an example for such a graph.

Again, the quality of the correlation depends on the signal-to-noise ratio (SNR) of the exploitable current p_{exp} in the traces. According to Mangard et al. (2007), the SNR is defined as follows:

$$\mathrm{SNR} = \frac{\mathrm{Var}(p_{exp})}{\mathrm{Var}(p_{noise})}$$

Furthermore they establish the relationship to the correlation with the following equation:

$$\rho(h_k, t'_j) = \frac{\rho(h_k, {t_{exp}}'_j)}{\sqrt{1 + \frac{1}{SNR}}} \tag{2.3}$$

Equation 2.3 gives a hint about the implications of noise on the correlation coefficient. In RFID contexts the noise component is usually very high, thus the SNR is very low. For such SNR values below 1, the impact on the correlation is roughly affected by a factor of $\sqrt{\mathrm{SNR}}$. Consequently, halving the noise (i.e. the standard deviation σ of $\mathcal{N}(\mu, \sigma)$) quadruples the SNR and thus doubles the correlation. But the opposite is true as well, putting RFID analysis on very tough ground. In the experiments of Section 6.4.7 the effects will be demonstrated exemplarily.

Hypothesis tests

In order to formally assess whether the results of difference-of-means or correlation are significantly different from the difference that is introduced by noise, statistical hypothesis tests can be applied. This is useful for automated verification of results. In short, this allows to assess the results' significance by verifying that it is very unlikely (e.g. less than 1% probability) that the mean's differences or correlation are as large just by chance.

Following Mangard et al. (2007), a two-sided test can be used to assert whether the difference of means lies in the critical region as given by the significance interval z_α. Given the set of hypothesis X and the set of samples Y, the test compares $H_0 : \mu_X - \mu_Y = 0$ to $H_1 : \mu_X - \mu_Y \neq 0$. The hypothesis H_0 is accepted if $\mu_X - \mu_Y$ is outside the critical region. It is further concluded, that the number of traces needed to distinguish the means with a confidence of $1 - \alpha$ is given by the following equation:

$$n = 4 \cdot \frac{\sigma^2}{(\mu_X - \mu_Y)^2} \cdot z^2_{1-\alpha}$$

A formula for assessing the required number of traces to significantly distinguish the correlation ρ from a random correlation is given by:

$$n = 3 + 4 \cdot \frac{z^2_{1-\alpha/2}}{\ln^2 \frac{1+\rho}{1-\rho}} \tag{2.4}$$

For small correlations as they are expected in DPA scenarios, the number of traces increases approximately by a factor of 16 if the attainable correlation can be halved by design.

2.2.4. Countermeasures

While SPA attacks can already be prevented fairly easy by avoiding conditional branching based on key material or intermediate values of the cipher (Kocher et al., 1999), it is unequally harder to prevent DPA attacks. However, these measures exist and it is likely, that some of them are being used to protect targets that are to be attacked by the framework. While it is outside the framework's scope to implement advanced methods that can be used to bypass countermeasures, it is still important to understand these protection mechanisms and their impact on the results attainable with aid of the framework. Of the variety of approaches that exist to prevent DPA attacks, two important ones, Hiding and Masking, will be presented in this section. Here, the designers goal is to make the operation dependent portions of a power trace invisible (*hiding*) or unusable (*masking*).

As a consequence of Equation 2.4 and 2.3, the goal is to minimize the DPA attackable signal while maximizing the noise, so that the correlation is lowered and the number of traces required to successfully mount an attack gets very high (i.e. $n > 10^6$). Other measures like modifying the protocol for frequent changes of keys may not always be applicable and are not considered here, although those are capable of removing the DPA attack surface completely.

Hiding

As the name suggests, Hiding *hides* the power profile or data specific power consumption. There is a variety of means to achieve this and research is ongoing. To hide a specific power profile, two major approaches are available, which can also be combined to a certain degree:

Equal power consumption for all operations and data is very difficult to achieve, but still there are means to make power consumption more equal. Such approaches modify the underlying hardware layer. A famous example is dual rail logic, which is an extended version of standard CMOS logic, carrying two lines for each bit, one with 0, the other one with a 1. Each calculation or transition has thus to be executed on both lines and therefore one of the lines will always draw current, whereas the other will not. Consequently the power consumption is not directly dependent on the input value, although still, hard to control physical effects such as wire capacities cause the power to vary slightly. However, devices with this kind of logic are much harder to attack and require significantly more effort. On the flip-side, the additional protection comes at a price, since dual rail logic is more expensive, because it requires more space on the chip and has a higher power consumption.

Artificial noise generated on the chip, for example by random additional calculations, decreases the SNR and thus makes attacks harder.

If the visible power profile can be hidden, both measures are powerful, especially against SPA attacks. However the attacker might still be able to run multiple measurements with the same data and average the results in order to remove the noise component or to measure even small differences more exactly. Hiding of the power trace is thus not sufficient to prevent DPA attacks.

Hiding in time domain randomly scrambles operations or inserts dummy operations in order to make operations like averaging and attempts of trace alignment harder. Dummy operations or clock cycles can easily be inserted, but many techniques exist to overcome these countermeasures (see Kocher et al., 1999 or Mangard et al., 2007 for examples). Where operations are parallelizable, the implementation might be changed to allow the order of

execution to be arbitrarily changed at runtime. This introduces additional noise and thus lowers the correlation.

Masking

A more effective approach is to algorithmically hide those values the attacker is interested in. As will be shown in Section 2.3, the values are typically intermediate values of cryptographic operations. If such values are never directly processed by the microchip, the attacker's efforts will be effectless. To achieve this, the processed values are masked with a random value called *mask*. For symmetric ciphers the mask is typically xored to the real value, producing the masked value, which is then processed by the cryptographic function. Of course, the operations of this function have been adjusted in order to work with masked values. Linear operations are mostly trivial to implement (i.e. for a bitshift on the masked value, the mask has to be shifted as well). Non-linear operations such as SBox replacements are more difficult to implement and might have a negative impact on the cipher's throughput (Mangard et al., 2007).

For asymmetric ciphers, masking is typically achieved using mathematical properties. The mask is again added or multiplied to the initial value in a reversible manner, so that the masking can be removed at the end of the cryptographic operation. A typical masking scheme would be the multiplication of the initial value x with a value $m \cdot r$ that is a multiple of the modulo operand r, e.g.:

$$f(x) = x^p \mod r = (x \cdot mr)^p \mod r \tag{2.5}$$

Completely masking a cryptographic algorithm might take some serious development time. However, there are masking schemes already available for all major cryptographic algorithms due to various publications (e.g. Akkar and Giraud, 2001).

2.3. Hypothesis Generation

For the experimental part, real targets are going to be attacked. As an important aspect for both the attack and the initial analysis is the calculation of fitting hypothesis, the generic principles will be introduced in this section. In order to attack an implementation of a cipher using DPA, a preliminary analysis is crucial. It is important to find out which power model applies to the chip and to be able to construct hypothesis on intermediate values, which are required to run the actual correlation attack.

Some power models were presented in Section 2.2.1 and for an unknown implementation it is typically wise to start with the hamming-weight model, as it is the easiest for analysis. However, if further details are already known, a custom model might apply. Depending on the context, one might already have a clue about the structures on the chip holding the interesting information, be it data buses or registers, along with their respective size. Otherwise one probably has to guess the size of these elements (e.g. 8, 16 or 32 bit registers) and experimentally determine the values in a profiling phase. In such a phase, one compiles a set of several hypothesis (i.e. one for each guess about power model and register width) and posts some data to the device in a known-key scenario, so that one can be sure, that the device processes the input data (plaintext) and the ciphertext at some point. Calculating the correlations for each hypothesis will likely yield which assumptions were right. For a 32 bit register, the 8 bit hypothesis will still be valid, but much less exact than the 32

bit hypothesis which contains less unmodeled noise and thus yields the better correlation values. This profiling method can be extended to find out, how the cipher was implemented, in which order data is processed and which protection methods might be in place, which is very useful for the experiments with unknown devices.

To actually attack a cipher, there are some requirements to launch a DPA attack: intermediate values have to be located that are dependent on the key and on known data (i.e. the plaintext fed to the device or the ciphertext returned from the device). Ideally this intermediate value depends only on a small part of the key (i.e. a single byte) as otherwise the number of hypothesis becomes too big. Such an intermediate is frequently found in ciphers, for example if one byte of plaintext is combined using `xor` with one byte of the key in the first round of the cipher. The attacker can then calculate hypothesis for all possible values of the key byte and run the correlation attack, which should yield one value as the correct one. There is only one drawback for such a placement of the targeted value, which is due to the hamming weight power model: As the key is xored with the plaintext, the output hamming weight will constantly differ in the hamming distance from one key to another with the same difference: I.e. the hamming distance is defined as $hd(a, b) = hw(a \oplus b)$ as such, the distance for the outputs for two different keys (k_1 and k_2) is $hd(d \oplus k_1, d \oplus k_2) = hw(d \oplus d \oplus k_1 \oplus k_2) = hw(k_1 \oplus k_2)$ and thus independent of the data processed. Therefore all key values will yield some correla-

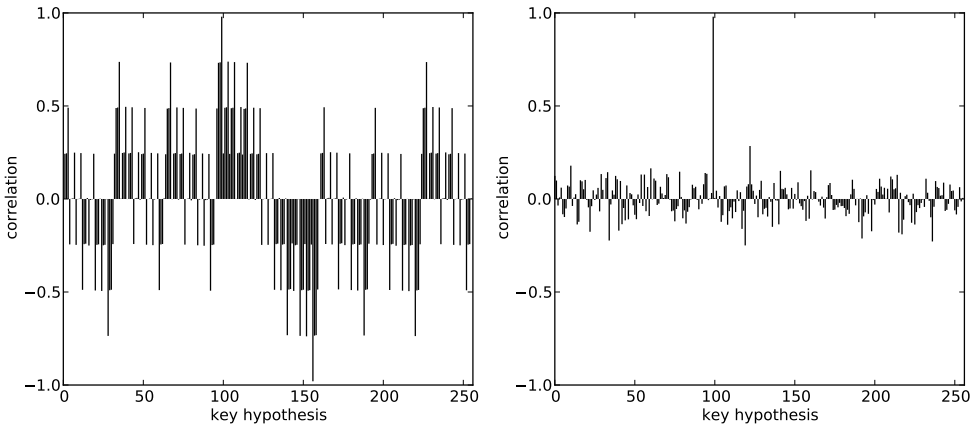

Figure 2.3.: Hamming weight correlation without (left) and with an sbox (right)

tion and it will be rather hard to distinguish the right key from an arbitrary other with just one different bit. The left plot of Figure 2.3 illustrates this phenomenon. To overcome this, it is advisable to choose target values that are dependent on key and data in a non-linear fashion. Typically such relations are found after sbox substitutions. Now the attacked values will be $hw(s(d \oplus k))$. Consequently no other key than the right one will show a significant correlation in the results, as the right plot in Figure 2.3 illustrates. Here the right key is 99, which is visible in both plots, but much easier to spot in the one using the sbox.

CHAPTER

THREE

RFID

This chapter will introduce the most relevant parts of RFID standards and devices to provide a background for the description of protocols and for assumptions made at the physical interface level for power consumption hypothesis. A brief overview on Radio Frequency Identification Devices (RFID) systems will introduce this chapter.

3.1. Overview

RFID is a wireless communication technology. An RFID system usually involves two components: transponders called tags and corresponding reader devices. Tags are typically very small, cheap and passively powered electronic devices carrying data. In their most simple form they just respond with an identification (ID) to the reader device and are attached to products to help automated identification and location. More complex tags, such as the ones this thesis is concerned with, are complete smartcards with a separate microchip, memory and even cryptographic units. Besides from tags that can only be read (mostly ID tokens), reader devices are typically also capable of writing to the tags, which might not be the most intuitive understanding based on their name. The thesis will sometimes refer to RFID reader devices as *proximity coupling device* (PCD) and to RFID tags as *proximity integrated circuit card* (PICC) respectively in accordance with ISO 14443.

RFID tags are in widespread use, counting various applications from theft-protection in stores, car keys, rechargeable money cards or tickets in public transportation up to electronic passports. The latter categories are more advanced tags, that rely on cryptography and embedded secrets for their security. With these presets in mind, side-channel analysis becomes an attractive goal for the attacker, while withstanding these attacks becomes important at the designer perspective.

3.2. Communication and Powering

Tags are completely passive devices that usually do not have their own power supply (Finkenzeller, 2003). Communication is therefore initiated by the reader, which generates a radio field that powers the tag and provides its clock signal using inductive coupling for which tags are equipped with a coupling unit (i.e. a coil). This however limits the range of RFID systems by design to ranges between 10cm and 1m.

Figure 3.1 shows the schematics of the power supply for a RFID tag. The capacitors C_r and C_1 are selected to form a parallel resonant circuit on the operating frequency. Furthermore

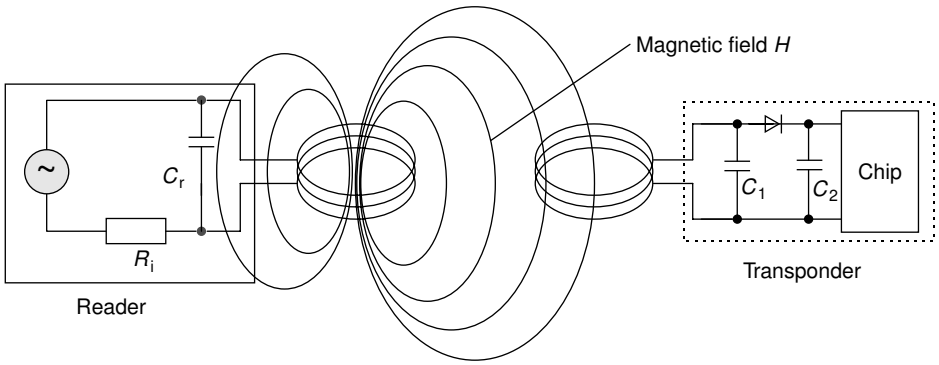

Figure 3.1.: Inductive coupling of a RFID tag from the energy of the magnetic field generated by the reader (Finkenzeller, 2003)

the diode and capacitor C_2 work for the rectification of the induced power to provide a stable current.

After the tag enters the interrogation zone and is powered through the reader, it will be in an active state and respond to requests issued by the reader.

3.3. Modulation and Transmission Protocols

There are several communication standards for low frequency (LF, typically 135 MHz) and high frequency (HF, 13.56MHz) devices for different applications involving different modulation schemes and transmission protocols. Since all protocols relevant for this thesis are based on the HF standard ISO 14443 type A, discussion is limited to this standard, although all ideas and experiments will basically work with any other standard without significant adjustment as well. For an in-depth description of RFID standards and protocols consult Finkenzeller (2003). A brief overview on the transmission protocol involved in the ISO 14443 standard will be given in Section 3.3.1.

The ISO 14443 standard consists of four parts:

ISO 14443-1 Physical characteristics

ISO 14443-2 Radio frequency power and signal interface

ISO 14443-3 Initialization and anti-collision

ISO 14443-4 Transmission protocols

As the standard operates in the HF band, the carrier frequency is fixed at 13.56MHz. Modulation from reader to tag is achieved by using amplitude-shift keying (ASK) modulation, creating gaps in the carrier when data is transmitted. The tag responds on a 848kHz subcarrier again using ASK modulation. These data transmissions therefore cause an increase in the carrier strength in the recorded traces (see Figure 3.2).

Since it might lead to problems, if multiple tags are within the same interrogation zone, the third part of the standard specifies anti-collision procedures and handling of multiple tags at once.

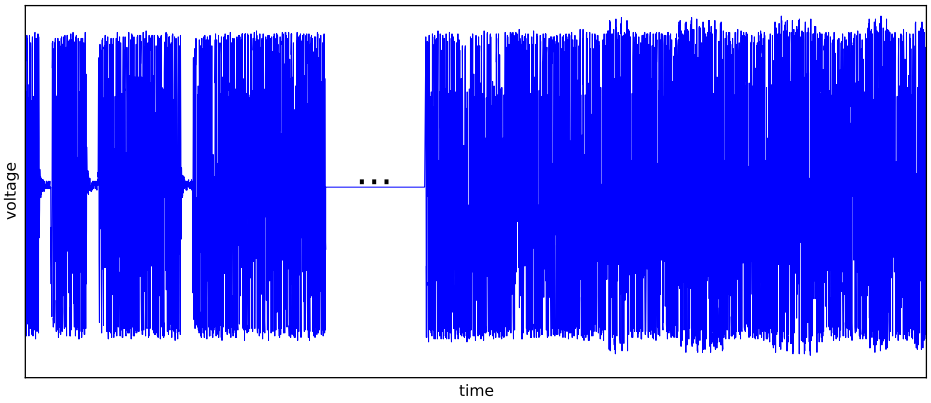

Figure 3.2.: Sample RFID trace showing modulation from reader to tag (left) and vice-versa (right)

3.3.1. ISO 14443

The ISO 14443 standard not only defines all the physical aspects of data transmission (as partially portrayed in Section 3.3), but also handles anti-collision, card selection and mechanisms for higher level layers such as data integrity or data block chaining. Since all experiments in this thesis are conducted in controlled environments, no consideration of anti-collision is necessary. Nevertheless, the protocol implementation needs to be considered for the placement of the trigger signal.

During the card selection process (ISO 14443-3), the card discloses its UID and returns an acknowledgement (SAK) that defines whether the card is ISO 14443-4 compliant or driven by a sole proprietary protocol. The latter is the case for example for Mifare classic cards (see Section 5.3.2) while the former holds for DESFire cards (Section 5.3.1). Such ISO 14443-4 compliant cards perform an additional operation prior to entering the proprietary protocol level: The reader issues a request for answer to select (RATS) command that assigns a logical communication channel to the selected card and returns further information about the card type. The logical communication channel (identified by a CID) allows multiple cards to be in active communication in the same interrogation zone. They will ignore all packets that are not addressed with their CID.

ISO 14443-3 also defines the frame format for a block transmission protocol (see Figure 3.3).

Prologue field			Information field	Epilogue field
PCB	[CID]	[NAD]	[INF]	EDC
1 byte	1 byte	1 byte		2 bytes

Error Detection Code

Figure 3.3.: Packet frame format (ISO 14443-3)

Such a frame is broadcasted on the radio-channel. The card that has been selected to work on channel CID will receive, error-check and process the frame. As the frame-size is limited, a packet may be split into multiple frames. For details consult ISO 14443-4.

CHAPTER

FOUR

FRAMEWORK IMPLEMENTATION

This chapter introduces a standard workflow for the analysis of power / EM traces along with the requirements to the framework and its architecture. Eventually it is shown, how the framework assists preprocessing and analysis of these traces.

4.1. Workflow Overview

The following is an overview on the necessary steps for conducting power / EM analysis experiments. Additional details relevant for the implementation of the steps will be given in the course of this chapter. As time based side-channel attacks only account for a minor and simple part, they'll be omitted for now.

Analyze the protocol In order to measure a device's power consumption, one has to be able to communicate with it in order to force it to do the calculations one wishes to attack. Therefore the protocol has to be known at least partially and interesting points have to be located. This includes finding out the time offsets at which calculations take place using which cipher, keys and values.

Implement protocol and trigger The protocol needs to be implemented so that the attacks can be conducted automatically. Additionally for recording the power consumption using a oscilloscope, it is usually required to provide a trigger signal indicating beginning and end of the attacked calculations.

Record traces In an automated fashion, the device is forced to do some calculations which are then recorded using the digital oscilloscope, and either used directly in an optimized attack scenario or saved for later analysis.

Initial Analysis Usually the attacker has very limited initial knowledge about the internals of the device, thus visual trace inspection in combination with several preprocessing tasks may be used to find information about the device operation, the cipher, the circuit architecture and sometimes even possible countermeasures.

Preprocessing One of the most important parts to aid analysis is trace preprocessing. Depending on the setup, traces may need to be aligned first in order to make any analysis possible. Other preprocessing steps include rectification, filtering, integration or peak extraction and are detailed in Section 4.2.

Hypothesis Generation In order to actually attack a device's power consumption, the cipher needs to be analyzed to provide some meaningful hypothesis that can be used for correlation as described earlier. According to Section 2.2.3, these hypothesis will initially concentrate on the known cipher- and plaintexts in order to verify the traces and to gain further information about the time offset at which computations take place and the quality of the recordings.

Correlation The actual correlation attack is then left with the plain calculation of the correlation of the hypothetic power consumptions with the preprocessed traces. Based on significant peaks, key bytes can be extracted until the whole key is revealed. If multiple candidates remain, a brute-force attack may usually be practical to verify the right key.

4.2. Preprocessing Toolsuite and Visualisation

With the acquirement of the first traces, it was immediately necessary to preprocess them in preparation for initial analysis. Some existing code already provided basic utilities that aided this preprocessing stage. Also several other simple preprocessing tools were created. Trying to use these tools on the recorded traces revealed several problems which eventually lead to the requirements for the preprocessing part of the framework that forms its core.

The remaining parts of this section describe some of the necessary preprocessing steps in greater detail. Further requirements for analysis workflows and the resulting framework architecture are described in the following Section 4.4.

4.2.1. Requirements

Initially scripts, later Makefiles were used to enhance the application of the already existing tools to traces, but this quickly turned out to produce a great deal of overhead if a new process needed to be started for each step of preprocessing. In a standard Makefile driven workflow, each intermediate result would be stored in a separate file. This file had to be stored on disk and could not stay in the kernel's cache, because it is overwritten by the intermediate results of other trace files. It also has to be read again for the next processing step, causing additional unnecessary disk read, write and, also very important, space overhead. As a result, a pipe based architecture seemed most useful. However such an architecture built on standalone tools still has significant drawbacks. Because of these initial mess-ups, the requirement for an efficient, flexible and easy to use scriptable preprocessing library was created.

For reasons of efficiency, the functions need to be written in C and cannot be directly implemented in a scripting language. There are already libraries providing similar functions such as the SciPy library, but as it solely operates on `double` arrays, the space overhead was not neglectable anymore, since the traces are recorded on a 8-bit discrete scale and as such only require an eighth of the space used by SciPy. Furthermore implementing the preprocessing tasks in SciPy would limit the framework's use to the python domain.

Since many tools in the C language domain already existed, an easy integration of the provided functionality in these programs was aspired. Since a subset of the preprocessing functions was already written in C, these functions were already used from C programs in early experiments. But it soon became evident, that the coding overhead and the lack of scripting options was just too big to be of easy use in the C language domain. Thus the toolsuite was modified to exposing its interface using a shared C library which can then be accessed using wrappers from the user's favorite scripting language or by directly calling the provided functions from other C programs.

4.2.2. Rasterization / Alignment

Since the clock of the oscilloscope is not synchronized with the PICC's clock, there is a drift in its processing causing up to ±20% of clock skew for different traces, while a single clock cycle might vary about ±5 to 1%. As such, any processing depending on exact alignment of the traces will fail or at least result in significantly more noisy data. Fortunately the clock signal for the device is derived from the carrier signal which is clearly visible in the recorded trace. In order to fix the clock skew, the rasterization step will make each clock cycle the same length by locating beginning and end of a clock cycle using a least squared difference pattern search approach before interpolating the current clock cycle length to the desired one. There might still be a clock skew within the individual clock cycle, however this will be relatively small may be neglected here. Using the rasterization approach, the individual data points of each traces will correspond to a certain time relative to the device's clock, which is the requirement for further analysis such as time-based averaging or correlation.

See `preprocessor.raster()` in the appendix for an API description.

Pattern Matching

The reference pattern has to be found manually once for each measurement setup by visually inspecting a recorded trace. This pattern is ideally a small subset of a clock cycle with a steep increase or decrease of amplitude. It is then used to identify a fixed position in each clock cycle. The pattern p of length n is shifted along the trace t, calculating the squared difference (variance) d:

$$d_i = \sum_{j=0}^{n-1} (t_{i+j} - p_j)^2$$

Each local minimum in d that is below a certain threshold marks the the fixed position and is then used to interpolate the specific period to the specified length.

4.2.3. Analysis and normalization

Before feeding traces to the correlation engine, it is tried to minimize certain noise effects. The rasterization process already removes a great deal, but still individual traces may vary. As traces are usually recorded over several hours, environment conditions are subject to change and may thus affect the recorded values. A initial analysis step implemented in the framework calculates average, variance as well as minimum and maximum values in a traces. These results aid identification of erroneous traces, but will also show other biases. For example, several trace sets that were recorded during experiments, had varying averages, that were dependent on time and seemed to be due to the oscilloscope. To reverse these effects, the trace's averages can be adjusted by shifting them along the y-axis. Further

normalization is not done, since the variance of traces only slightly differs. Modifying it might also introduce additional unexpected errors.

4.2.4. Average and Variance Graphs

A single power / EM trace is much too noisy to detect any interesting data in it. However several hundreds or thousands of traces combined may give a more accurate picture of the device operations and the recording's quality. Average graphs give a general idea about the device's power profile and may be used to detect patterned operations such as rounds of a block cipher.

$$\mu_t = \frac{1}{n} \sum_{i=0}^{n-1} t_{t,i}$$

Variance graphs show areas where power consumption varies stronger.

$$\sigma_t^2 = \frac{1}{n-1} \sum_{j \in T} (j_i - p_t)^2$$

This might be due to input data and thus help to identify the interesting parts of the power traces, but may also be caused by other non constant data or operations processed by the device. For power analysis on RFID systems this may also be due to protocol or radio specific properties. The magnitude of the variance difference is also an important indicator for quality of the leaked power signal.

Graphs generated using this method will be shown in Chapter 6 to support the analysis and evaluation of the experiments.

4.2.5. Simple Filters

Correlations can be significantly improved if the noise component, which is present in the traces, can be reduced. One possibility to achieve this are digital filters, some of which are implemented in the framework to simplify this task.

Average Filter

An average filter takes n input samples and returns their average. It is used to reduce high frequency noise. Such a filter is shifted along the trace t producing $|t| - n + 1$ new output samples.

$$\mu_i = \frac{1}{n} \sum_{j=0}^{n-1} t_{i+j}$$

The filter can be rapidly calculated by using its recursion equation:

$$\mu_{i+1} = \mu_i - \frac{t_i - t_{i+n}}{n}$$

Peak Extraction and Integration

The integration filter is similar to the average filter, except that the division is omitted, thus the integration filter looses scale and will likely need another target data type as the sum of

several bytes easily exceeds a byte's capacity. The filter is also used to reduce high frequency noise and provides a sliding window for analysis. As the leaked current is modulated onto the trace, the carrier peaks contain the most amount of information (Mangard et al., 2007), but are obscured by noise. Assuming a Gaussian distributed noise source, the integration filter will reduce the amount of noise, while amplifying the leaked signal.

A similar approach, not exactly filtering, but reducing the amount of data of a trace, is peak extraction which is also based on the observation that the leaked information is mainly hidden in the peaks. By extracting only the maxima of a rectified trace, the most valuable information is still present, while the trace's size is reduced by a factor of $^{f_s}/_{2f_c}$ (with f_s being the sampling and f_c the carrier frequency).

Additionally, peak extraction may be combined with a preceeding integration step to reduce the noise component. An additional benefit is the possibility of a normalization step, that can be used to keep the (small) data type, while providing greater accuracy due to the scaling procedure in the normalization. This is possible, because the peaks will be in a very small range of values.

FIR Filter

Finite impulse response (FIR) filters are similar to average filters (the average filter is actually a special case of a FIR filter), but allow a more fine grained control, such as weighting neighboring points less. Filter design is an entire field of study which is not delved into here, especially as these are mainly used for stream processing data. For fine-grained frequency filtering FFT filters are much better suited.

A FIR filter is given as an array of coefficients c and its calculation is straightforward:

$$t'_i = \frac{1}{\sum c_j} \sum_{j=0}^{n} c_j \cdot t_{i+j}$$

4.2.6. FFT filters

To remove certain frequencies, FFT filters are a powerful tool. As an efficient implementation of FFT is not straightforward, the actual calculation is performed by libfftw3 for which the framework provides a wrapper facilitating FFT usage.

4.3. Correlation

As already discussed in Section 2.2.3, the correlation coefficient, which is the preferred tool for finding the matching hypothesis, is defined as follows (Mangard et al., 2007):

$$\rho_{X,Y} = \frac{\mathrm{Cov}(X,Y)}{\sqrt{\mathrm{Var}(X) \cdot \mathrm{Var}(Y)}}$$

While covariance and variance are defined as follows for sample sets X and Y:

$$\begin{aligned} \mathrm{Cov}(X,Y) &= \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) \\ \mathrm{Var}(X) &= \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2 \end{aligned}$$

A naive implementation for calculating the covariation matrix of a set of hypothesis H and a set of traces T with a trace length of S samples, results in $|H| \cdot S$ matrix multiplication in a magnitude of $O(S \cdot |T|)$ in memory requirement and each $O(|H| \cdot S \cdot |T|)$ in time. For uncharacterized devices, trace lengths in correlations can easily reach up to 1M samples, while the number of traces is usually several 10k, resulting in a memory requirement of several GB. Obviously the naive approach is not appropriate.

Adjusting the formulas for computer algorithms results in the following, effectively reducing the requirements for calculating the variance to the sum and the squared sum of the vector.

$$\begin{aligned}
\mathrm{Var}(X) &= \frac{1}{n}\left(\sum x_i^2 - 2\bar{x}\sum x_i + \sum \bar{x}^2\right) \\
&= \frac{1}{n}\left(\sum x_i^2\right) - \left(\frac{1}{n}\sum x_i\right)^2 \\
\mathrm{Cov}(X,Y) &= \frac{1}{n-1}\left(\sum x_i y_i - \bar{x}\sum y_i - \bar{y}\sum x_i + n\bar{x}\bar{y}\right) \\
&= \frac{1}{n-1}\left(\sum x_i y_i - n\bar{x}\bar{y}\right)
\end{aligned}$$

The amount of hypothesis is usually comparatively small (typically one byte is attacked, resulting in 256 hypothesis), thus its average $\bar{y}$ and standard deviation σ_y can be precomputed with neglectable efforts. To reduce the global calculation overhead, the correlation equation is also reduced to simple terms:

$$\rho_{x,y} = \frac{1}{n-1} \cdot \frac{\sum x_i y_i - \bar{y}\sum x_i}{\sigma_y \cdot \sqrt{\frac{\sum x_i^2}{n} - \left(\frac{\sum x_i}{n}\right)^2}} \tag{4.1}$$

Thus the remaining reduced requirements for the calculation are as follows resulting in a total space requirement of $O(|H| \cdot S)$.

1. precomputed hypothesis values ($\bar{y}$ and σ_y)
2. a multiplication sum for each hypothesis and trace ($\sum x_i y_i$)
3. sum and square sum for each trace-sample ($\sum x_i$, $\sum x_i^2$)

Since the sums can be updated on the fly, traces can now be processed sequentially and the runtime is now reduced to $O(|H| \cdot S \cdot |T|)$ once, while the expensive calculation of equation 4.1 only happens once for each entry of the result matrix, i.e. $O(|H| \cdot S)$. The process is also highly parallelizable, thus for a small number of hypothesis (≤ 64), the limiting factor in this algorithm tends to be the harddisk bandwidth for reading the traces.

4.4. Framework Architecture

As a result of the experience with the old tools (see Section 4.2.1) the following list of requirements was created:

A pipe based architecture allows the chaining of multiple processing steps and the respective reuse of intermediate results without having to store intermediate results on disk.

Expose basic functionality in C domain for reasons of speed, efficiency and reuse in other applications.

Allow interactive and scriptable use of the framework and provide a high-level interface for easy use, adaption and experimentation.

Consequently, the preprocessing steps described in the previous sections were implemented in a C library, although in their standalone form, as provided by the library, they are still rather complicated to use. Therefore the high-level framework architecture was defined to allow for the easy use of preprocessing and correlation functionality in DPA scenarios, emphasizing the workflow aspect of these tasks. Figure 4.1 gives an overview of this architecture.

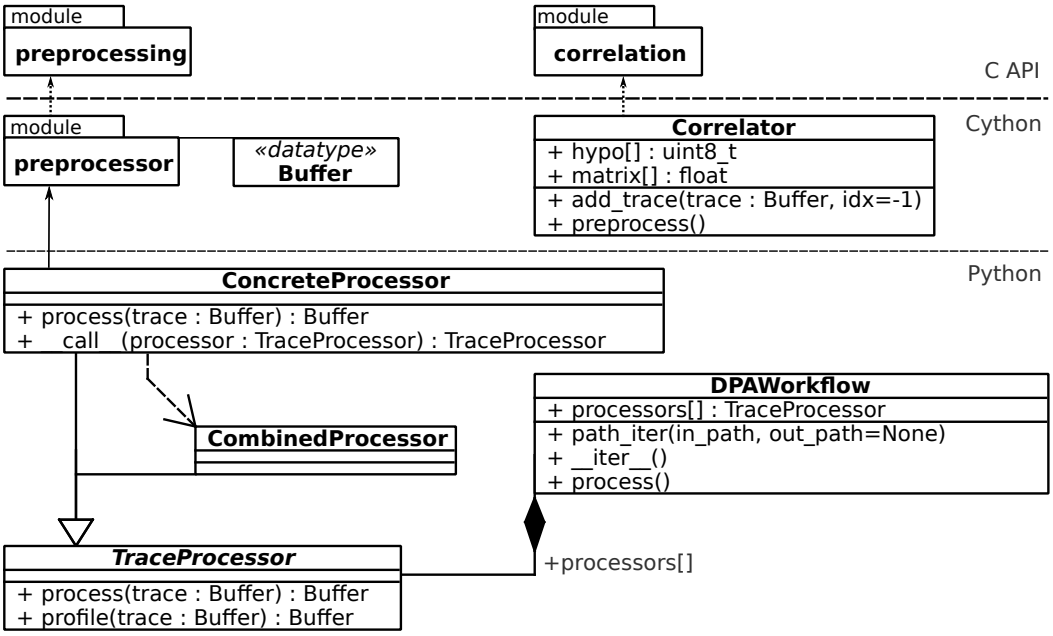

Figure 4.1.: Architecture of core functionality

4.4.1. Type Independence

One further requirement for the preprocessing toolsuite is the type independence. Traces are typically recorded as unsigned chars (`uint8_t`) and several preprocessing steps might need better accuracy, while memory consumption should be limited to a minimum, especially if results have to be written back to the harddisk. On disk, such multiple thousands of traces easily consume a couple of 100 GiB. To achieve type independent functionality, a C++ template approach seemed most useful. Unfortunately though, C++ modules have serious problems when it comes to using these modules from external languages. Consequently, the preprocessing toolsuite is implemented in C, where the templating functionality is achieved using preprocessor macros. Some additional glue- and boilerplate code is auto-generated from the C source file.

On a higher abstraction level, the preprocessor.Buffer class holds the type for each buffer, so that the wrapper function can then select the corresponding implementation from the preprocessing module.

4.4.2. Implementation

For the higher level implementation, it was decided to implement the functionality in python for several reasons. Python has excellent wrapping capabilities of C code (`cython`), allows

for quick and easy implementation not only of the library but also of actual software that uses the provided library, comes with a shell that aids experimental inspection of results in initial analysis scenarios and last but not least includes many more powerful tools with SciPy and the matplotlib for visualization of the results. Other tools (such as octave or an existing plotting utility) were initially used for visualization of the results, but were either slow, inflexible or had serious usability issues (e.g. a bug in gnuplot currently prevents scaling and zooming).

The modules were thus implemented in cython and python, along with documentation and unit tests. However, for any given scenario that might need to use the preprocessing toolsuite independently of python, the basic functionality is still in the C modules, which can be used from almost any other language as well.

4.4.3. Correlation Workflow

The correlation calculation is accelerated as described in Section 4.3 and based on Equation 4.1. This functionality is provided in a separate module, which is again wrapped by a python module to provide a simple interface (documented in Appendix A.2). The procedure works as follows:

Hypothesis have to be defined manually for each trace and key, based on assumptions on the power model, the cipher etc. (see Section 2.3 for details). Typically the trace recording's logfile would be read to create the hypothesis based on the input / output of the device.

Traces have to be added. The traces can be the result of a workflow task, if there are multiple workflow results, multiple Correlators can be used to evaluate multiple results simultaneously. Since the Correlator is thread-safe, multiple traces can be added at once.

In a final step, the actual correlation is computed returning the correlation matrix suitable for display and evaluation.

4.4.4. Highlevel Workflow

In order to simplify the whole process in an workflow, typical programming tasks of the analysis stage are handled by the framework, more precisely by the DPAWorkflow class. As illustrated in Figure 4.1, each processing function is wrapped by a ConcreteProcessor that is able to process individual traces to produce a certain output. These processors can be chained and combined and thus form a pipe-based architecture that is used by a DPAWorkflow. Such a workflow instance iterates over all input files and handles management of reading traces and passing of intermediate results along the pipe of processors.

The sole processing of the traces would not be sufficient, as the data needs to end up in sinks for evaluations. These sinks are VoidProcessors that do not produce any output, but still process traces and gain information from then. Standard sinks calculate the correlation of preprocessed traces with a set of hypothesis or generate average and variance traces whose visualizations lead to graphs such as the ones in Chapter 6.

Further information on the workflow internals is available in Appendix A.3.

CHAPTER

FIVE

PREPARATION

This chapter covers the experiments' surroundings. To prepare the actual attacks, several targets were chosen, thus a specific description of these targets will provide the necessary preconditions for mounting the attacks. The exact experiments' setups and results are described in Chapter 6.

5.1. Tool Selection

Attacks may sound easy in theory, but may need more elaborate work in practice. One of the challenges when analyzing and attacking RFID chips is to choose the right tool for operation. A typical RFID reader is usually not sufficient for side-channel attacks, as it lacks certain features, such as the ability for very accurate time measurements of operations or for forced variation of these timings. Also it is indispensable for all experiments that require recordings using a oscilloscope, to generate a trigger signal in a very specific state. Earlier studies and projects on RFID devices sometimes used self-built readers for these tasks, but today several open source devices such as the OpenPCD or the proxmark3 exist, providing all the necessary freedom for these operations. Both devices support the ISO 14443-A protocol suite, which is the protocol of the chosen attack targets (see Section 5.2). Since the OpenPCD features a cleaner and better structured source tree, it is well suited for productivity uses, but limited to the HF ISO standards in contrast to the proxmark3 board which is basically a software defined radio (SDR) for HF and LF RFID protocols, thus very well suited for protocol analysis and implementation of arbitrary new protocols. While there were no immediate plans to extend the attacks to non-ISO 14443-A compliant cards, the prospect of not having to start from scratch for arbitrary other protocols seemed worthwhile. Also in spring 2010, the proxmark3 software was in more active development supported by a bigger community than the OpenPCD, thus the proxmark3 was the choice for the attacks described here, although other devices might have worked just as well. Despite some bugs and instabilities, which were due to poor software design, the proxmark3 was an excellent choice and enabled the execution of all the experiments with moderate efforts. The few bugs and annoyances that were encountered during the development could be located and fixed.

5.2. Attack Targets

In recent years, several major RFID cards, like Mifare Classic (Garcia et al., 2008; Nohl et al., 2008) or Legic Prime (Plötz and Nohl, 2009b), were attacked by reverse engineers exploiting flaws in the card's hardware, its proprietary ciphers and the protocols. With the advance of more or less modern cryptography on RFID smart cards, other means of analysis will

become the target of hackers. Compared to their predecessors, newer cards are developed with security much more in mind. As such, well-known public ciphers such as DES or AES are used in these cards' protocols which seem safe and robust for now.

Experimentally breaking a modern card's security is the most likely practical application of the framework, and consequently the analysis and tests focused on such a modern card, in order to develop the requirements for the framework with a practical target that allows the identification of real life use-cases. Additional emphasis has been put on the analysis of the Mifare Classic system, as it is proprietary, but very well known and researched. Furthermore, the card has been opened in the reverse engineering process and pictures of the chip are available that could be beneficial for the analysis, e.g. by possibly allowing fine grained positioning of an EM probe.

Beside this simple target for testing purposes, the actual experiments were meant to focus on its successor: the Mifare DESFire card. The card seems secure on a protocol level, but it is assumed, that it does not feature strong protection against power/EM analysis attacks, as there is an extended version (the DESFire EV1) explicitly protected against these kind of attacks.

With the emergence, that no immediate results are to be expected from these targets, a third target was chosen for evaluation and comparison of power analysis of classic power supplies to induction based ones as they are found in the RFID cards. For this a dual interface card was chosen. The programmable JCOP 41 can be accessed using a classic contact-based smartcard interface, but also using RFID and is thus a good target for such comparisons. The card supports a wide range of cryptography modules already in hardware, which is especially hardened against DPA attacks, but since the device can be freely programmed and executes a virtual machines, it is assumed, that it still leaks data dependent information. Even if this were not the case, it is assumed, that some kind of power profile can be extracted, which can be used to compare it to the RFID recorded one.

5.3. Target Protocols

In order to initially conduct the experiments, the targeted protocols have to be implemented and since the framework shall be capable of working with many different protocols, the respective protocol implementation needs to be easy. As motivated in Section 5.1, this is the case for most open-source reader devices, although the proxmark3 has the additional advantage, that nearly all operations can be executed on the board and one is in complete control of the most aspects of the protocol implementation including modulation details and timing. To verify, that custom protocols can be implemented, which is also a necessity for eventually testing the framework against any card of that protocol, the DESFire protocol was implemented exemplarily. The next section describes this protocol in more detail, as many aspects of the specs were not publicly available. Some additional information on the implementation of the other protocols, especially concerning the Mifare Classic card will be given towards the end of this chapter.

5.3.1. Mifare DESFire

The Mifare DESFire protocol is built upon the ISO 14443-A standard (see Chapter 3) embedding its own proprietary protocol on top of it. Unfortunately, the protocol is not publicly specified and documentation is scarce. However, a couple of documents found in the files section of the proxmark3 website (slides on a Mifare Workshop which have unfortunately now been removed) as well as several sources on the internet were very helpful in implementing the parts necessary for the attack of the device. In particular sample traces found on Ridrix's Blog (Ridrix, 2009a, b) clarified open questions and the Series 40 Nokia 6212 NFC SDK (Nokia, 2008) includes some Java classes revealing additional missing protocol details after decompilation.

DESFire Protocol Overview

The DESFire protocol runs on top of ISO 14443-4 and is a request-response protocol in which the PCD issues a command and awaits a reply by the PICC. Each DESFire command starts with a command code byte followed by a command-specific number of argument bytes.

A DESFire card is organized in applications and data files. Each application holds up to 14 keys that can be assigned different rights regarding the application or its files. A typical application will have a key with read permissions only, while another one, whose confidentiality may be more important, will also have write permissions. In order to execute restricted operations, a key has to be selected. This is done by executing an authentication procedure to ensure that both parties have knowledge of the same key, and to establish a session key which may be used for encrypted communication or message authentication.

The key number 0 of the default application (0) is the master key and allows to create or destroy other applications as well as to change the card's master key.

DESFire (3)DES Operation

DESFire cards have a special (3)DES operation mode in such as they only perform encrypt operations, even when decrypting data. As for DES it holds that $\text{Decrypt}(\text{Encrypt}(x)) = \text{Encrypt}(\text{Decrypt}(x))$, thus such a mode is possible if the reader only uses decrypt operations,

even if encrypting data. See Figure 5.1 for an example illustrating the difference between standard operation and the DESFire specific one.

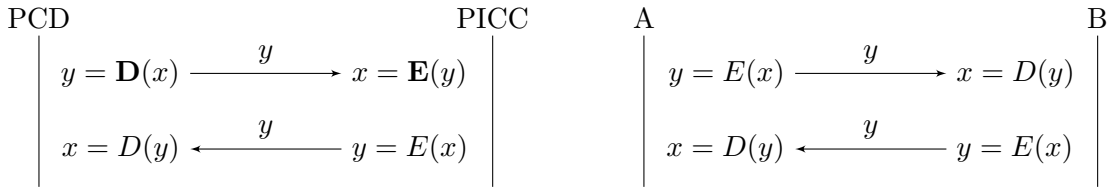

Figure 5.1.: Difference between DESFire specific DES operation (left) and the normal one (right)

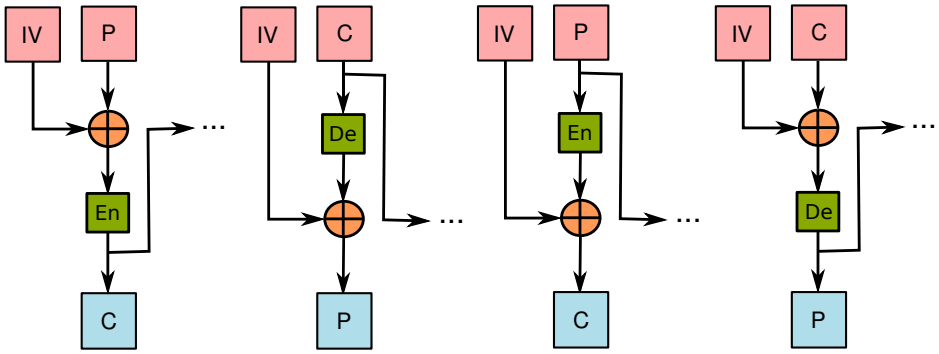

Figure 5.2.: Standard and DESFire specific Cipher Block Chaining modes

Another consequence of this operation mode is that CBC modes (Cipher Block Chaining) have to be adjusted, such that there is an abnormal handling for packets received by the PCD. Figure 5.2 illustrates the different modes. In traditional setups, only the first two modes are used. In the DESFire case, the PICC uses mode 1 when sending and mode 3 when receiving, whereas the PCD uses mode 2 when receiving and mode 4 when sending.

For sake of clarity and simplicity, the following sections will refer to encryption as the operation being performed when sending data (i.e. the PICC encrypts when sending, the PCD decrypts). Consequently, decryption will refer to the operation performed when receiving data. Unless stated otherwise, all data is processed in the according CBC mode with the initialization vector defaulting to 0.

DESFire cards can operate in 3DES or fall back to single DES mode, which happens automatically if selecting a key $K_{3DES} := \{K_A, K_B\}$ with $K_A = K_B$, since 3DES uses three single DES operations:

$$3DES(x) := E(D(E(x, K_A), K_B), K_A)$$

If K_A and K_B are the same, it holds that $D(E(x, K_A), K_A) = x$ resulting in:

$$3DES(x) := E(D(E(x, K_A), K_A), K_A) = E(x, K_A)$$

DESFire Protocol Commands

The following describes a couple of commands exemplarily that played a crucial role in the execution of the experiments or in understanding yet unknown aspects of the protocol. The command to read an encrypted file for example helped to understand the encryption, MAC and padding scheme that is used in these operations, which was required to implement the

CHANGE_KEY command. To do so, a plain file was created and its access settings were changed to only allow encrypted reads. Using the READ_FILE command, an encrypted packet would be read and could be decrypted and analyzed to find out, which padding is used and how the MAC is calculated.

Not all commands that have been implemented are strictly relevant for this thesis and therefore some commands and additional details are omitted in this listing. For a more thorough command and feature reference, please refer directly to the DESFire implementation for the proxmark3 device, which has been open sourced as part of this work.

Authentication One of the key aspects of the card's protocol is the authentication procedure that selects a specific key. Figure 5.3 illustrates the process: The PCD sends a REQUEST_AUTHENTICATION command specifying the key number to be used for authentication. Both parties will now use the specified key for their 3DES operations. The PICC responds to the request with an encrypted 8 byte nonce in a AUTHENTICATION_FRAME packet. Being decrypted and left-rotated by one byte by the PCD, it continues authentication in another AUTHENTICATION_FRAME packet by encrypting the modified nonce and generating and encrypting its own 8 byte random number. After verifying the information, the PICC finishes authentication by returning the nonce. Again it decrypts and left-rotates the nonce by one byte, decrypts the result and returns it to the PCD, which then verifies the results.

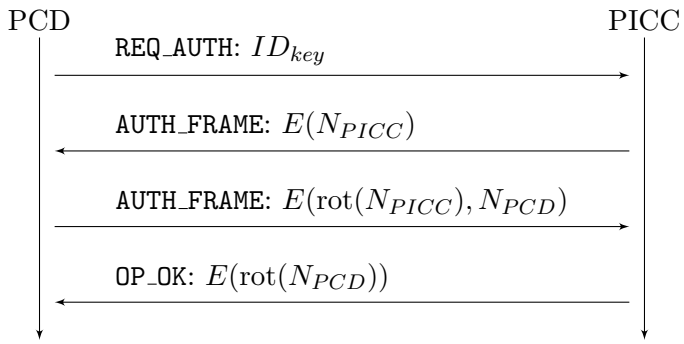

Figure 5.3.: Authentication procedure in the Mifare DESFire protocol

Both parties then establish a 16 byte session key that is derived from the exchanged nonces as follows, where $N_{PCD,1}$ refers to the first 4 bytes of the nonce generated by the PCD:

$$K_{Sess} := N_{PICC,1}|N_{PCD,1}|N_{PICC,2}|N_{PCD,2}$$

The mixing is necessary, since the PCD could force the device to operate in single DES mode if the session key were defined as $K_{Sess} := N_{PICC}|N_{PCD}$ and the PCD chose the same nonce as the PICC. In case the original key is a single DES key, the last 8 bytes of the session key will be set to its first 8 bytes thus forcing single DES operation.

Application Creation and Selection For experiments, it makes sense not to work on the master key / master application, which could potentially render the card unusable. Therefore it is useful to be able to create and select applications on which further experiments can be conducted.

An application is identified by a 24bit application id (AID). The AID is always transmitted in little endian byte order (i.e. the first byte transmitted holding the least significant byte of the AID). To create an application, a CREATE_APPLICATION command is sent:

1	byte	CMD	CREATE_APPLICATION
3	bytes	AID	application id
1	byte	KeySettings	Upper 4 bits: key change key Flags: 8 := allow config change 4 := allow deletion 2 := allow file list 1 := allow master key change To freeze keys, set this byte to `0xf0`
1	byte	KeyCount	number of keys in use

Accordingly, an application can be selected by sending a `SELECT_APPLICATION` command:

1	byte	CMD	SELECT_APPLICATION
3	bytes	AID	application id

The application is then activated and all following operations occur in this application's context.

Read File Files are identified by a number and can be read using the `READ_DATA` command:

1	byte	CMD	READ_DATA
3	bytes	FID	file id
3	bytes	Offset	
3	bytes	Length	number of bytes to read

Depending on the file's settings that are also stored on the card, the response is either plain, uses a Message Authentication Code (MAC) for integrity or is encrypted using 3DES with the established session key.

In order to gain test vectors and sample data, a file with known content has been written to the card. Using the `CHANGE_FILE_SETTINGS` command the security level of the file was then changed, such that its content is either returned with a four byte MAC or completely encrypted.

1	byte	CMD	CHANGE_FILE_SETTINGS
3	bytes	FID	file id
1	byte	SecurityLevel	PLAIN, MAC, 3DES ...
2	bytes	AccessSettings	4 nibbles: 3: read/write key 2: `0xe` 1: read key 0: write key

Encryption As a result of the observations of reading an encrypted file with known content, it is concluded, that for encrypting data the plaintext is first prepared by appending a two byte CRC and padding the whole blob to a multiple of eight bytes. The result is then encrypted using the session key and the corresponding CBC mode (see Section 5.3.1). This could be verified by testing the functionality using the `WRITE_DATA` command.

Message Authentication Furthermore, it is concluded, that data that is protected by a MAC contains a four byte MAC that is appended to the original plaintext. Observations showed, that in order to generate the MAC, the zero padded plaintext is enciphered in CBC

mode (mode 1 in Figure 5.2). The first four bytes of the last cipher block then correspond to the MAC.

Key Change In order to attack others keys but 0x000..., keys need to be changed. This can be achieved using the `CHANGE_KEY` command:

1	byte	CMD	CHANGE_KEY
1	byte	SLOT	key slot (0-13)
N	bytes	key data	(see description)

There are two modes for changing keys. The normal mode applies if the changed key is the one with which the PCD was just authenticated or if any key is allowed to change keys. In these cases no proof of knowledge of the old key is necessary and key-data is computed by encrypting the new key according to the previous paragraph on encryption.

In all other cases, key-data is computed as follows: The old key and the new key are combined bytewise using XOR and a two byte CRC of the result is appended. Another two byte CRC of the new key is then appended, before the data is zero padded and encrypted in CBC mode. This is just as before, but without implicitly appending an additional CRC to the data.

Evaluation

Finding out the details to implement the DESFire command set was a tedious amount of work and required many experiments and blind guesses. However, the proxmark3 proved as a useful platform to support experiments on yet unknown command sequences and eventually allowed to implement these commands in a usable manner. As expected, the implementation of custom protocols is an easy task and the proxmark3 provides the necessary flexibility to implement even complex or non-standard protocols, which is one of its strengths and one of the reasons, the tool has been chosen in the first place.

5.3.2. Mifare Classic

The older Mifare Classic card is also built upon ISO 14443. Again the protocol is not publicly specified, but since it has already been completely broken (Garcia et al., 2008, 2009), sufficient amounts of protocol information are available to actually mount the attack. For example, the proxmark3 codebase already contains functionality to send an authentication request in order to record the card's nonces as implemented by de Koning Gans et al. (2008)). This information is used to break the card's key based on weaknesses in the cryptographic algorithm (c.f. Garcia et al., 2008, 2009). But the functionality can also be reused for the experiments, simplifying the development. The basic card architecture and authentication mechanism will be described here, as it is the base for locating interesting cryptographic operations and therefore also needed for the placement and implementation of the trigger.

Card and Memory Structure

The card has a similar but different setup as the more advanced DESFire card. It consists of 16 sectors that are composed of four 16 byte blocks. The last block of each sector contains keys and access conditions for each of the four blocks. Each sector comes with two 48bit keys, each having different rights on the blocks as defined in the access control block. For

example, there might be a public read-capable key and a private write-capable key. Upon successful authentication with one key, its corresponding access conditions are activated. The card provides simple data manipulating functions such as read, write, increment or decrement.

Authentication

Authentication is performed in a mutual challenge response protocol. Several weaknesses have been discovered in the proprietary cipher. Accordingly the authentication procedure is very well documented (e.g. in Garcia et al., 2008 or Garcia et al., 2009):

The card transmits a `AUTH_A` (`AUTH_B`) command with the corresponding block-number to request authentication for the specified block with key A (or B respectively).

1	byte	CMD	AUTH_{A|B}
1	byte	BLOCK	block number

Then the exchange of four-byte challenges and responses is started as illustrated in Figure 5.4. For details refer to Garcia et al. (2009). The tag sends its nonce n_T to the reader that

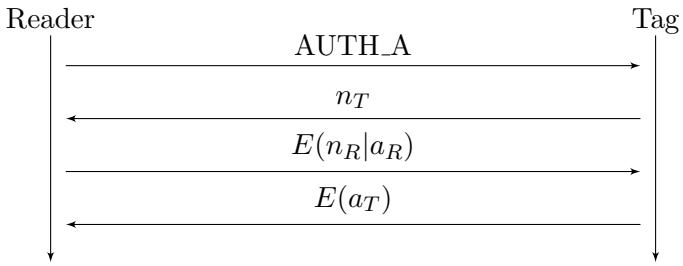

Figure 5.4.: Authentication procedure in the Mifare Classic protocol

computes the appropriate answer $a_R = f(n_T)$ and proves knowledge of the shared key, by encrypting it with the key-stream generated by the crypto-1 cipher that was earlier initialized with the shared key. The process is repeated for the reader's nonce n_R.

5.3.3. JCOP

For the comparison experiments, the dual-interface card JCOP 41 was chosen, which was the only dual-interface card available. Again the card's contactless interface is based on ISO 14443 (see Section 3.3.1).

For the contact-based interface, a RS232 based terminal was used, for which software was already available. On top of both layers, the card is run by the blockoriented T=1 protocol as specified in ISO 7816-3. Only the most necessary parts of this protocol were implemented for both interfaces to allow communication and execution of the experiments.

As the card is freely programmable, a simple program was written for the device, which the card would execute for each request to this application.

CHAPTER

SIX

EXPERIMENTS AND RESULTS

This chapter describes the experiments that were conducted to generate data for assessing the frameworks requirements and for verifying its functionality under real conditions. Since there was no attack surface for time-based side-channel attacks, as described in the next subsection, the work focuses very much on recording and evaluating the power profiles of devices for DPA attacks. Several measurement setups were tried in the prospect of acquiring a strong power/EM signal despite the high noise induced by the carrier. Besides the actual experiments on real data, additional experiments with simulated data were conducted in order to assess the minimum SNR requirements and the effects induced by the several noise components individually. However, as these experiments on simulated data can only model the chip's behavior to a certain degree, its results are best understood as a hint to show capabilities and limitations.

6.1. Timing Experiments

The reader board allows to measure time very exactly, at least up to the carrier frequency which 1:1 corresponds to the clock rate of the attacked smartcard. As such, time measuring functionality was implemented into the proxmark3 software, that counts the number of clock cycles until the first bit of the answer is received. Several measurements were executed with different data. All targets though, seem to be protected against timing attacks, as the timing for each command is constant. The only noteworthy observations are, that the Mifare DESFire card takes a little bit longer for the first authentication request to an application. Subsequent requests to the same application are quicker, most likely due to some caching functionality. The same applies to the JCOP card, the response of which is significantly quicker, if the exact same command is issued a second time immediately after the original command. Additionally, the card is comparatively slow, thus an implementation that has some operation specific timing, will be easy to attack even with less exact measurements.

6.2. Measurement Setup Overview

DPA attacks have been successful with a huge variety of measurement setups, with even the most sloppy experiments still yielding results. The experiments conducted in this thesis generally follow the setup illustrated in Figure 6.1. The numbers correspond to respective actions and are order chronologically. The proxmark3 is controlled by the computer (1), creates the HF field at the antenna (2) and initializes the card up to execution of the command of interest, at which point a trigger is posted on one of the device's LEDs (4). The oscilloscope is attached to the trigger and records the probe's information (5) upon activity

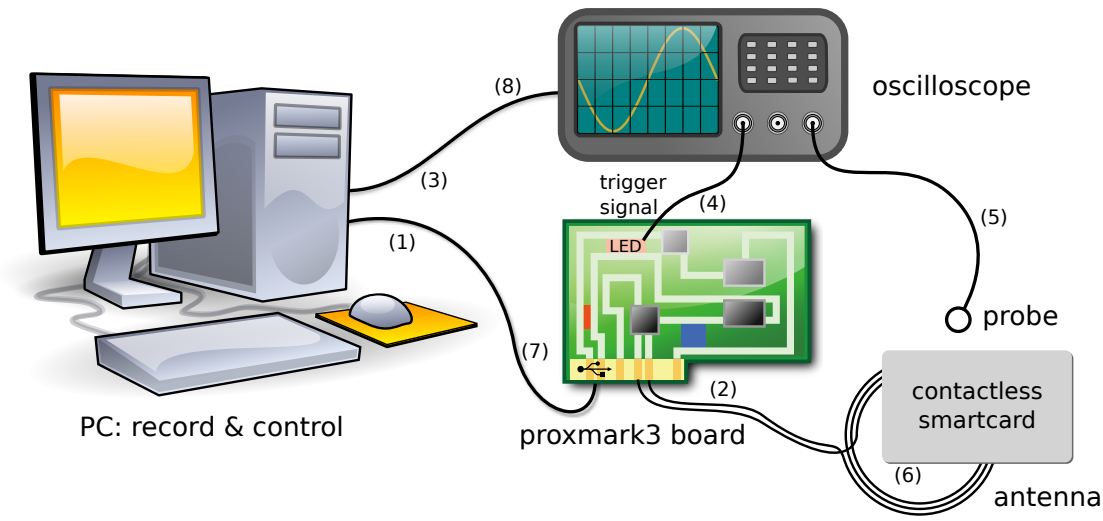

Figure 6.1.: Illustration of the general experiment setup

of the trigger signal (4), for which it has been set up (3) by the controlling software running on the PC. The PC will also download and store the recorded trace from the oscilloscope (8) after the proxmark3 received (6) and transmitted (7) the answer from the card.

6.3. Target Protocols and Trigger Placement

The experiments on the Mifare Classic and Mifare DESFire cards were designed in such a way, that the observed operation includes cryptographic operations with a constant key, but different plain- / ciphertexts. This section will briefly describe the measurement implementation and observation points.

6.3.1. Mifare Classic

For the Mifare Classic card, the authentication works using a mutual challenge response protocol (see Section 5.3.2). For each trace to be recorded, a new ISO 14443 connection is established according to Section 3.3.1. Upon transmission of the authentication request, the trigger is set and stays active until the card responds. Afterwards the card is disconnected, because the authentication procedure did not complete.

6.3.2. Mifare DESFire

The Mifare DESFire protocol was already described in Section 5.3.1. Authentication is again based on a mutual challenge response protocol. Since the card allows multiple authentications in a row, the connection to the card is only established initially. The software then runs a series of authentication requests for the same application and key. The trigger is again set after sending the authentication request and is released with the reception of the card's answer.

Experiments conducted by Kasper et al. (2009) placed the trigger at a different position in the protocol, i.e. after the reader sent its nonce and its answer to the tag. As the initial plan was to attack the cipher's intermediate results using correlation, one might be worried, that the tag does not do any cryptographic operation for the placement chosen in this

thesis' experiment, because it might just send random data as a challenge and perform the more expensive cryptographic operations only in the second phase of the authentication procedure. For a correlation attack on intermediate values this might be fatal. However, as the measurement setups turned out not to be good enough for such attacks, this was neglectable. Furthermore, the results would still reveal a correlation for the ciphertext (which corresponds to the random data sent by the card), but not for the initial nonce plaintext (which the card would only compute in the second phase). For calculating the hypothesis, it has therefore to be kept in mind to not only attack the initial nonce plaintext, but also the encrypted ciphertext sent by the card. In such a scenario, attacks on the cryptographic operations could only be made in the second phase of the authentication.

6.3.3. JCOP

For the JAVA card, ISO 7816 application protocol data units (APDU) have been implemented. Similar to the previously described DESFire card, the connection is established only once. Upon selection of the user defined application, a series of challenges is sent. After each challenge, the trigger is set until the card replies. Unfortunately, the card takes a very long time to respond, as such the recorded trace does not comprise the full time-span of the card's calculation. However, as can be seen in the results in Section 6.4.6, the time-span is sufficient to illustrate the power profile.

The card seems to have a basic caching mechanism causing significantly shorter delays if the same command is repeated. To overcome this, the parameters for the command were changed for each iteration.

In the contact-based setup, the trigger was placed on the I/O lines and manually adjusted in the oscilloscope to fit the calculation period.

6.4. Measurements

The measurement setup was not an actual part of this thesis and unfortunately there was no dedicated setup available for this task. However, some experiments had to be conducted and as such several setups were tried, although a lack of expertise on this analogue area limited the possibilities. For the experiments, there were many ideas about which current or radiation to measure at which place. As a tiny electromagnetic probe was available, even fine-grained local EM measurements (Section 6.4.5) were possible and a card that was earlier dissolved in acetone allowed for direct measurements at the antenna of the contactless smartcard's chip (Section 6.4.4) .

A selection of these setups will now be described separately, along with its characteristics, results and conclusions gained by using the framework that played a crucial role in the analysis and processing of the raw data.

6.4.1. Preprocessing steps

Several different preprocessing steps were tried and combined to get the best results possible. While the rasterization process is a necessary step, it also induces some additional errors as the alignment is done on a period basis (40-80 samples) which still allows for a couple of samples shift within one period. This shift tends to be significant, as the carrier amplitude

changes rapidly and peaks are usually short. Interpolation and rounding add to further inaccuracy.

As such it turned out, that the most valuable data is found in the traces peaks, which conforms with findings shown in Mangard et al. (2007). While rasterization makes sense for traces that are less noisy, the actual useful method for experiments that still contain the carrier signal turned out to be peak integration, which is a combination of the integration procedure described in Section 4.2.5 and a following peak extraction. The integration step basically acts as an average filter to reduce local high frequency noise. Due to the peak extraction process no rasterization is necessary and as such no further errors are introduced which decreases the overall variation of the trace. This approach is also slightly faster, because rasterization needs to compute the squared difference to the reference pattern for each offset in the trace. The power profile for the conducted operation will then be visible on the average graphs which can also be used to assess the quality of the measured data.

6.4.2. Differential Probe

The first row of experiments was conducted using a differential probe at the reader's antenna at which voltage drops were assumed to be noticed if the DESFire card consumes more power. This was also the easiest accessible location to place a probe since the proxmark3 board used to conduct the experiments already provides these ports.

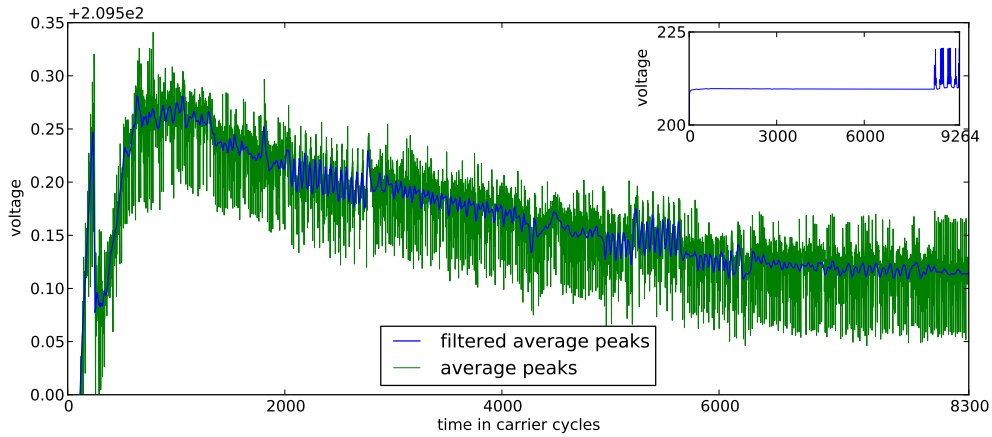

Figure 6.2.: Differential probe (58132 traces). Average graph.

Figure 6.2 shows the average graph of the peak integration ($n = 6$) of all recorded traces. As this graph contains a recurring pattern of a length of 16 carrier cycles, it can be assumed that one operation on the card takes 16 carrier cycles. Consequently an according average filter creates the blue line, which clearly shows the specific power profile of the card even in this flawed measurement setup. However, the noise induced by the air interface and the quantization is just too big to allow any DPA correlation. It stays zero even for plain- or ciphertexts in multiple scenarios.

6.4.3. Simple Coil Probe

Since the EM probe in the lab is rather focused for very local measurements, a simple coil of the chip's size was used in another set of experiments on the same card to receive a broader signal. Unfortunately some external factors influenced the measurement setup, as

is illustrated in the left part of Figure 6.3 showing the average of each trace. Consequently

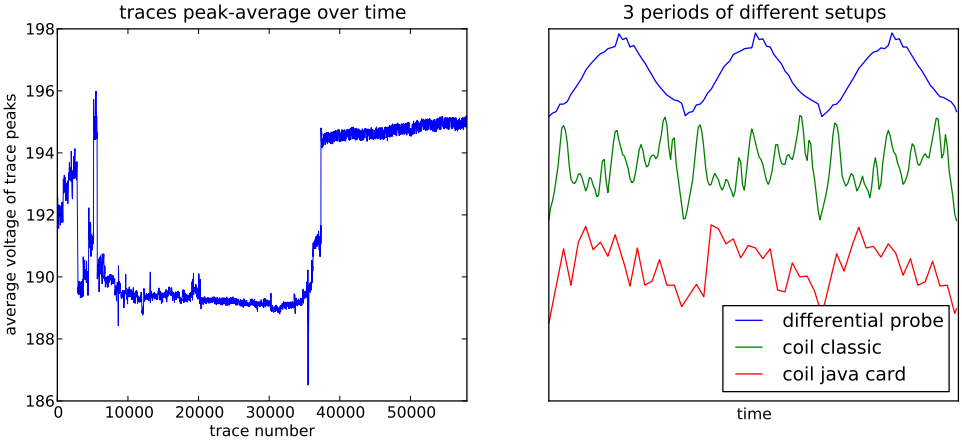

Figure 6.3.: Average of each trace of the simple coil measurement and zoomed power profile of a clock cycle

for calculating the power profile, only the results of the traces ≥ 38500 were chosen. Since the traces of these results do not follow the typical sinus pattern that was observed in all other experiments (see the right part of Figure 6.3), it seems likely, that the probe does not actually measure the power profile of the card. In the presence of these assumptions it is

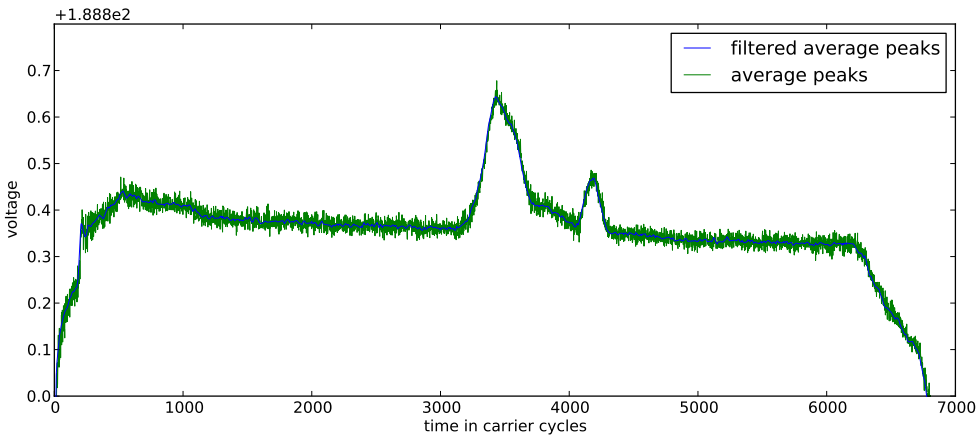

Figure 6.4.: Simple Coil Probe. Average graph.

no surprise, that the graph in Figure 6.4 does not mimic the power profile of the differential probe experiment at all. As a single period's peak is very short, instead of peak integration only peak extraction was used to create the power profile of the figure.

6.4.4. Differential Probe at Smartcard's Antenna

The lab provided a raw smartcard chip, so that the probe could be attached directly to the card's antenna. The only available such chip was a Mifare Classic card, so these experiments were not conducted with the DESFire card. The experiments were executed for two different setups: In the first, the probe was directly attached to the antenna, whereas the second experiment had an additional resistor of 10Ω implemented, since typically measuring the voltage drop across a resistor is used to measure the power consumption of the device.

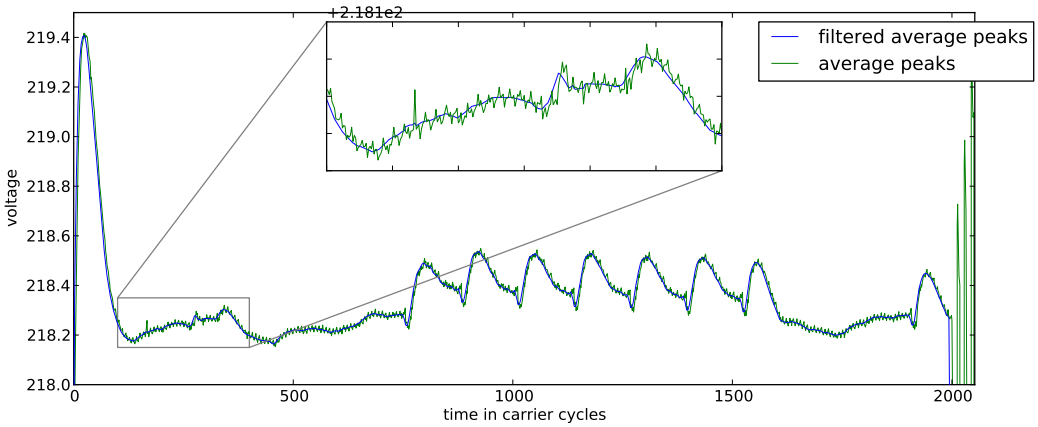

Figure 6.5.: Diffprobe at antenna (16635 traces). Average graph.

The results were obtained using the same method as for the differential probe (Section 6.4.2). Accordingly the graph in Figure 6.5 shows a comparatively clear power profile of a Mifare Classic chip during authentication. Again, a recurring pattern of 8 carrier cycles length can be found (the zoomed region in the figure is to make this pattern more visible). Applying an average filter of this length produces the blue bar.

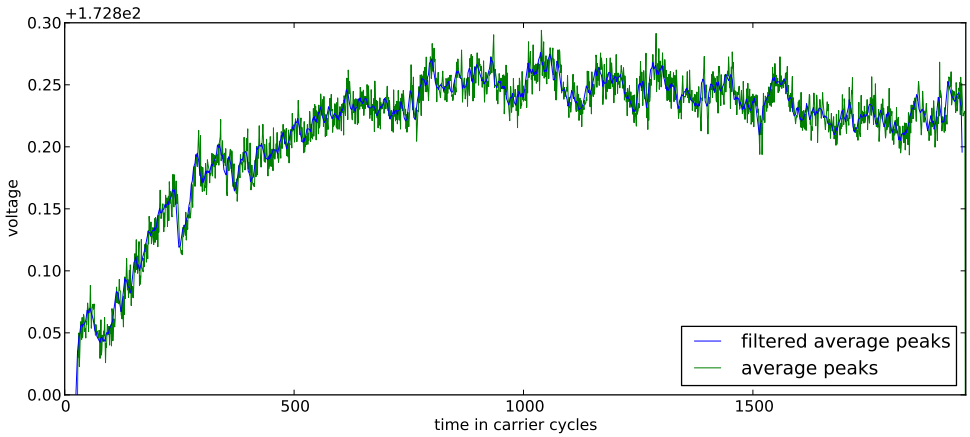

Figure 6.6.: Diffprobe at antenna with resistor (20500 traces). Average graph.

The results of the second experiment, in which an additional resistor was used, are illustrated in Figure 6.6 which is a very clear indication for a failed measurement setup: The individual differences are much smaller and have a huge noise component. No clear power profile can be seen, no pattern is visible. Measuring without a resistor therefore yields the better results, still they are not sufficient to calculate a correlation on the plaintext.

6.4.5. Local EM Probe

Another set of experiments tried to measure the EM radiation with a high precision EM probe. Since no further information was available about the chip, there was no clue about where to measure exactly, so the chip was divided into four quadrants and a new set of measurements (with about 26,000 traces each) was conducted for each quadrant.

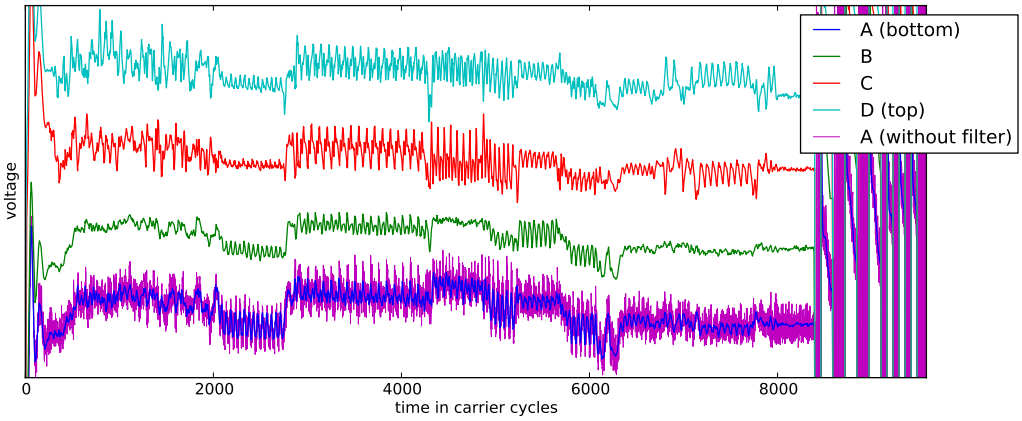

Figure 6.7.: Local EM probes: 4 quadrants. Average graphs.

Figure 6.7 shows the four power traces acquired with the EM probe in the four quadrants of the chip. The offset has been slightly adjusted to fit the figure. All traces show a common generic profile. However, they differ in details. Some operations are better visible in one trace, whereas nearly invisible in another (e.g. the pattern in trace B at offset 7500). This gives a clue on the location of functionality on the chip. One can even guess about certain functionality based on the visual pattern. For example, the attacker will be very much interested in locating the 3DES engine. As no clear pattern for a $3 \cdot 16$ round can be located, it can be assumed, that the chip operates in single DES mode, because certain keys like the one in the experiments allow this mode, conforming with the protocol that was described in Section 5.3.1. The most likely location for such a 16 round pattern will be the one around offset 4000.

6.4.6. Comparison Experiment

In order to better assess the quality of the measurements, a row of experiments was conducted for a dual interface smartcard. Numerous traces where recorded using the simple coil probe described in Section 6.4.3. Additionally, a couple of reference traces were recorded, measuring the power consumption in a contact-based setup. As the dual interface card is protected against DPA attacks, even in the contact-based setup, only a very bare power profile is visible. However, recognizing this profile in the EM results would confirm, that one actually measures the power consumption, and allows to assess the quality.

Figures 6.8 and 6.9 make clear, that the power traces leaked through multiple measurements on the EM channel are strongly related to the actual power consumption of the card, as a couple of recurring key patterns can be recognized. The graphs also indicate, that a relatively big difference in the actual power trace ($\approx$ 25 unit points) only reflects about 0.025 unit points in the EM trace, which is about factor 40 below the resolution of the oscilloscope, thus subject to serious quantization errors not to mention additional noise.

This already gives the very specific hint, that the plaintext correlations could not be achieved in any of the experiments, because the measurement setups are just not good enough. In the next section, the quality requirements will be detailed further.

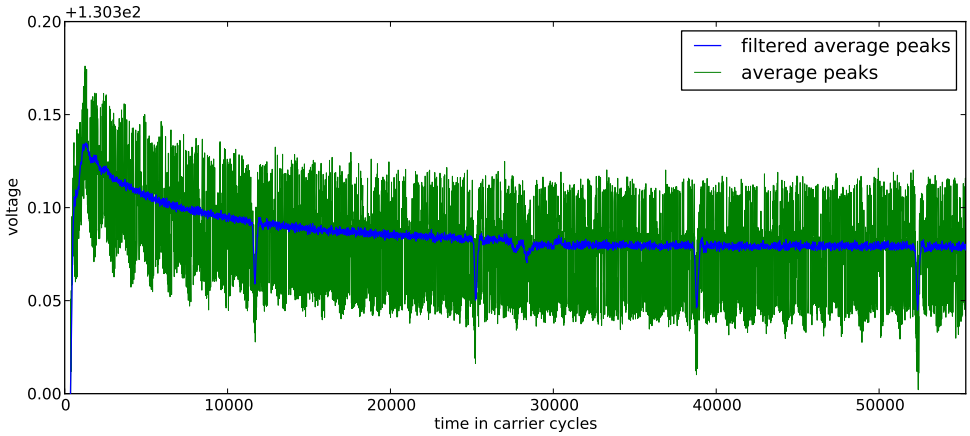

Figure 6.8.: Simple EM probe: power profile java card. Average Graph.

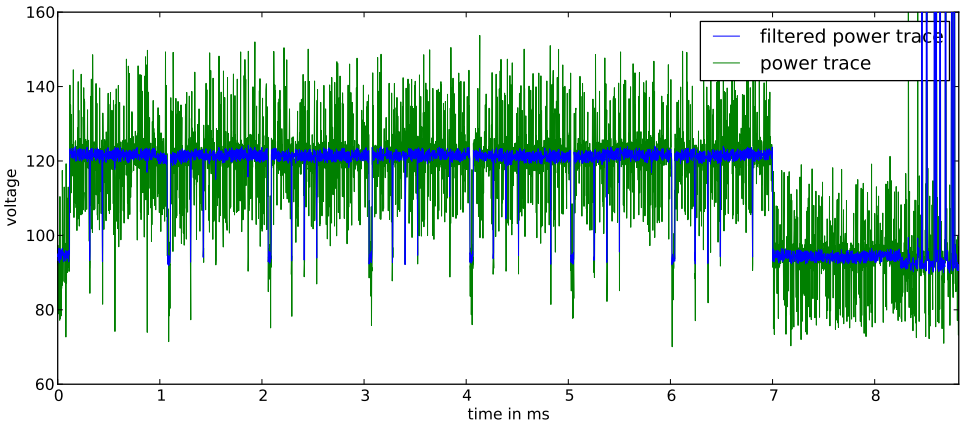

Figure 6.9.: Wired power profile java card. Single Trace.

6.4.7. Simulated Experiments

In order to assess and verify the capabilities of the framework and to find out in which range of parameters the system works, a set of simulated experiments was conducted. A generator for simulated traces is used to generate these traces under varying assumptions. Each trace consists of 50 periods of 40 samples each, based on Equation 6.1. The leakage current c_{leak} is a multiplier for the hamming weight of the data $\mathrm{hw}(d_t)$. It is modulated onto the carrier signal $c_{carrier}$, with only a small amount of normally distributed noise n_i with a strength of 2% of the carrier signal. With each period, the leakage current is increased.

$$t_i = (c_{leak} \cdot \mathrm{hw}(d_t) + c_{carrier}) \cdot \sin\left(\frac{i2\pi}{40}\right) + n_i \tag{6.1}$$

With the known model, the hypothesis for the correlations are again the hamming weights of the data that have been modulated onto the trace. For reference, the hamming weights of random data were calculated as an additional hypothesis which is also shown in the following graphs.

10.000 traces were generated and stored as `unsigned char`s, which includes the additional quantization noise. Afterwards, the correlations were computed and plotted in the graph

in Figure 6.10 which illustrates, that the desired correlation is clearly visible (and distinguishable from the random correlation), once the leakage current reaches about 0.5% of the carrier strength.

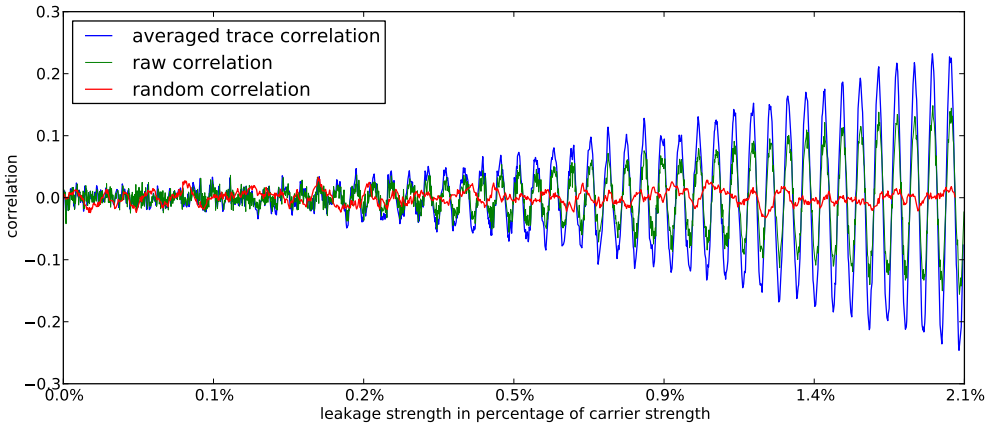

Figure 6.10.: Fake experiment correlations for varying leakage strengths

In a second experiment, the noise component was varied, while the leakage component is fixed to 4% of the carrier signal, a strength that already yielded a strong correlation for the above experiment. Again Figure 6.11 shows the calculated correlations after processing

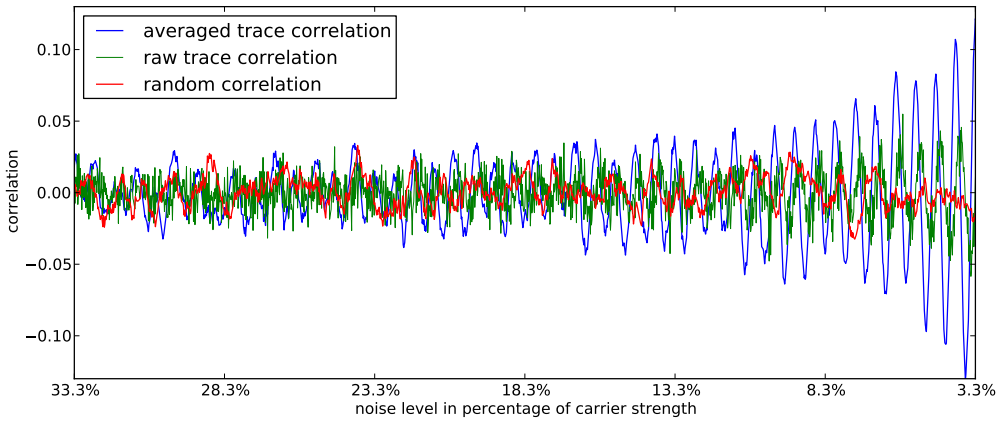

Figure 6.11.: Fake experiment correlations for varying noise levels

10.000 traces. As shown there, if the noise level is stronger than 8% of the carrier signal, it is very hard to get a clear correlation even if the leakage signal is very strong. Also the best correlation is achieved by using an averaging filter of the size of half a period. This is due to the modulation of the signal onto the carrier, where each sample adds a part of the leakage current, while the normally distributed noise is reduced with each additional sample added to the calculation.

Of course these fake experiments have significant flaws, are based on random assumptions and cannot possibly mimic the real world completely. However, they were never intended to do so, but still provide at least hints concerning requirements for future experiments while they also help to classify the other experiments that were already conducted. Combined with the results of the comparison experiment of the previous section, it becomes clear, that

none of the above measurement setups provide the necessary resolution to actually measure data dependent differences in the power profile.

6.5. Future Experiment Ideas

Unfortunately, time and know-how constraints did not permit to conduct experiments with every single idea. However, these will very likely yield much better results than above experiments and might very well be suited to execute an actual DPA attack on contactless smartcards.

The main reason for failure of above experiments is the presence of the carrier signal. If it can be reduced, the results will very likely improve. There are several ways to reduce or even remove the carrier completely:

Analogue filtering Kasper et al. (2009) conducted a similar experiment and used some analogue filtering process to remove the carrier. Basically they amplified the reader's 13.56MHz oscillator signal and performed an analogue subtraction of a phase-shifted version of the signal and the measure data (see Figure 6.12). Using this method, the oscilloscope's quantization range could be used much better.

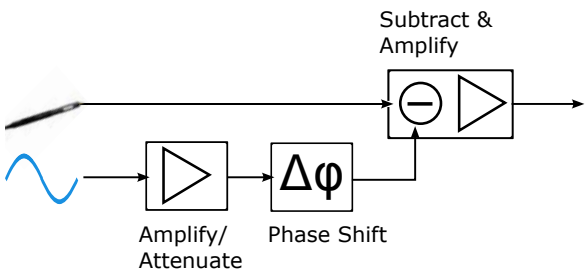

Figure 6.12.: Analogue filtering as performed by Kasper et al. (2009)

Rectification An analogue rectifier might be used to overcome some of the quantization errors, especially in conjunction with a capacitor that works as a simple analogue smoothing filter.

Antenna placement If one has access to a raw chip (e.g. by resolving the card's plastic in acetone) one can probably extend the antennas wires and place the antenna far away from the chip. EM measurements at the chip will then only show a significantly reduced carrier signal.

CHAPTER

SEVEN

CLOSEUP

In the previous chapters, the foundations on conducting side-channel attacks on RFID systems were shown along with the workflow process that is assisted by the framework in order to effectively analyze and evaluate the results. The theoretical background was explained, protocols were exemplarily described to enable the execution of actual experiments, which eventually provided a good confidentiality concerning the effectiveness and practical usability of the framework. Now a last look back will evaluate the achievements and the practicability of DPA attacks RFID smartcards.

7.1. Toolsuite Evaluation

With the initially available tools, trace processing was a very hard and time-consuming task. For example, it took multiple days to experiment with the preprocessing to reveal the power trace of the differential probe setup (Section 6.4.2). Now, the toolsuite allows to try multiple preprocessing steps at once, while processing traces simultaneously on multiple processors without any huge memory requirements. The analysis workflows can easily be programmed and combined, so that the programming tasks are accelerated. This is in contrast to the old C utilities that often required a lot of boilerplate code for management tasks and were not at all flexible. Also the actual usage of the utilities was inefficient and painful, as some descriptions in Section 4.2 already suggested. Furthermore, the overall speed of the execution of preprocessing and analysis steps of a workflow has been drastically accelerated by avoiding the disk storage of intermediate results.

The inefficient and painful use of utilities changed with the framework: For the differential probe experiment mentioned above, a generic analysis workflow can be used or a new one can be written within minutes. The actual computation to get the results roughly takes an hour depending on the number of traces. As such, one spends more time actually evaluating the results than waiting for their calculation to finish.

The framework is stable and fast, because the computationally expensive routines have been implemented in a C module with multiprocessing in mind from the beginning. Furthermore, it is easy to extend the framework and add new functionality: With a couple of lines of code, one can create a new type-independent processing function, the actual usage of which is a piece of cake.

Although none of the cryptographic components of the cards could be attacked to due measurement setup problems, using the framework, specific power profiles could be acquired. Additionally the framework's functionality could be verified by the simulated data experiments. Together with the comparison experiment it provided evidence for the lack of exploitable signal in the recorded traces of these basic measurement setups.

The targeted cards did not have any attack surface that would allow to execute time-based side-channel attacks, as such the frameworks capabilities to support these attack styles could not be evaluated. However, simple time-based SCAs are very easy to implement and do not need the support of the framework. The more complex differential attack styles do not differ too much from differential power analysis attacks and can therefore use the framework's functionality.

The proxmark3 was preferred over other RFID readers for a variety of reasons. A custom protocol has been implemented on this board and it could be shown, that the tool is a good choice for such experiments, fulfilled all requirements, and enables a very accurate and easy placement of trigger signals. This is not at last due to the open architecture and firmware of the board.

A fortunate side-effect of the framework is, that most functionality is not limited to use in RFID side-channel attacks, but can also be used in contact-based setups and might enhance processing and workflows there as well.

7.2. DPA Attacks on RFID in Practice

Since the thesis did not focus on the experiment setups, it is unfortunate that none of the basic setups were sufficiently accurate to reveal a correlation to the plaintext. However, other research succeeded with this task. Notable are Kasper et al. (2009), who used a measurement setup with an analogue filter to actually reveal the DES key of a contactless smartcard. They also succeeded in attaining a plaintext correlation on only a couple of thousand traces without an analogue filter. Although the same preprocessing and filtering was applied to the traces of the measurements in this thesis, the results could not be reproduced: neither the filtered power trace nor the correlation results. While it can only be speculated about the reasons, the most likely explanation is an initially better measurement setup e.g. due to the probe's placement at a special position.

Nevertheless, as their experiments showed, these cards are vulnerable against attacks, although the threshold to mount them is raised significantly. Because no faults were revealed on the protocol or cryptographic level, side-channel attacks are one of the only means of attacking these cards. Mounting such attacks requires special resources, but it still seems practical for a sufficiently motivated attacker.

While DPA attacks are a very powerful tool, some ideas for countermeasures were already presented in Section 2.2.4 and the literature (i.e. Mangard et al., 2007) suggests many ways on how to mitigate the thread. It should not be assumed, that RFID systems are automatically protected against DPA attacks due to the high noise induced by the carrier. Several methods to reduce this effect seem practical and one should consequently presume, that contactless smartcards are just as vulnerable as their contact-based siblings.

LIST OF FIGURES

BIBLIOGRAPHY

Akkar and Giraud 2001
Akkar, M. L. ; Giraud, C.: An implementation of DES and AES, secure against some attacks. In: *Cryptographic Hardware and Embedded Systems–CHES 2001*. Paris, France : Springer, 2001, p. 309–318

Caron 1999
Caron, Jean-Sébastien: Resistance against Differential Power Analysis for El- liptic Curve Cryptosystems. In: *Cryptographic Hardware and Embedded Systems–CHES'99, 1st International Workshop*. Worcester, MA, USA : Springer, 1999, p. 724–724

Finkenzeller 2003
Finkenzeller, K.: *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*. John Wiley and Sons Inc, 2003

Garcia et al. 2008
Garcia, F. ; Koning Gans, G. de ; Muijrers, R. ; Van Rossum, P. ; Verdult, R. ; Schreur, R. ; Jacobs, B.: Dismantling MIFARE Classic. (2008), 97-114. `http://packetstorm.rlz.cl/papers/wireless/2008-esorics.pdf`. – Presented at ESORICS 2008

Garcia et al. 2009
Garcia, F. D. ; Rossum, P. ; Verdult, R. ; Schreur, R. W.: Wirelessly pick-pocketing a Mifare Classic card. In: *Proceddings of the 2009 30th IEEE Symposium on Security and Privacy* IEEE, 2009, p. 3–15. – `http://www.cs.umd.edu/~jkatz/security/downloads/Mifare3.pdf`

ISO 14443 2000
Identification cards – Contactless Integrated Circuit(s) Cards – Proximity Cards. 2000

ISO 7816 1997
Identification Cards – Integrated Circuit Cards with Contacts. 1997

Kasper et al. 2009
Kasper, T. ; Oswald, D. ; Paar, C.: EM Side-Channel Attacks on Commercial Contactless Smartcards Using Low-Cost Equipment. In: *Information Security Applications* (2009), p. 79–93

Kocher 1996
Kocher, P.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: *Advances in Cryptology–CRYPTO'96* Springer, 1996, p. 104–113. – `http://www.cryptography.com/public/pdf/TimingAttacks.pdf`

Kocher et al. 1999
Kocher, P. C. ; Jae, J. ; Jun, B.: Differential Power Analysis. In: *Advances in Cryptology: Proceedings of CRYPTO99* (1999). http://www.cryptography.com/public/pdf/DPA.pdf

de Koning Gans et al. 2008
Koning Gans, G. de ; Hoepman, J. H. ; Garcia, F.: A Practical Attack on the MIFARE Classic. In: *Smart Card Research and Advanced Applications* (2008), p. 267–282

Mangard et al. 2007
Mangard, S. ; Oswald, E. ; Popp, T.: *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007

Messerges et al. 1999
Messerges, T. ; Dabbish, E. ; Sloan, R.: Power Analysis Attacks of Modular Exponentiation in Smartcards. In: *Cryptographic Hardware and Embedded Systems–CHES'99, 1st International Workshop*. Worcester, MA, USA : Springer, 1999, p. 724–724

Nohl et al. 2008
Nohl, K. ; Evans, D. ; Starbug, S. ; Plötz, H.: Reverse-engineering a cryptographic RFID tag. (2008), 185-193. http://www.cs.virginia.edu/~evans/pubs/usenix08/usenix08.pdf

Nokia 2008
Nokia: *Series 40 Nokia 6212 NFC SDK*. http://www.forum.nokia.com/info/sw.nokia.com/id/5bcaee40-d2b2-4595-b5b5-4833d6a4cda1/S40_Nokia_6212_NFC_SDK.html. Version: 2008

Plötz and Nohl 2009a
Plötz, Henryk ; Nohl, Karsten: Breaking Hitag2. (2009). https://har2009.org/program/events/135.en.html

Plötz and Nohl 2009b
Plötz, Henryk ; Nohl, Karsten: Legic Prime: Obscurity in Depth. (2009), December. http://events.ccc.de/congress/2009/Fahrplan/attachments/1506_legic-slides.pdf. – Presented at 26. Chaos Communication Congress

Ridrix 2009a
Ridrix: Mifare Desfire communication example. In: *Ridrix's Blog* (2009). http://ridrix.wordpress.com/2009/09/19/mifare-desfire-communication-example/

Ridrix 2009b
Ridrix: t:kort public transportation card explorations. In: *Ridrix's Blog* (2009). http://ridrix.wordpress.com/2009/09/20/tkort-public-transportation-card-explorations/

Rohatgi 2010
Rohatgi, Pankaj: Protecting FPGAs from Power Analysis. In: *EE Times* (2010). http://www.eetimes.com/design/programmable-logic/4199399/Protecting-FPGAs-from-power-analysis

Schindler 2000
Schindler, W.: A timing attack against RSA with the chinese remainder theorem. In: *Cryptographic Hardware and Embedded Systems—CHES 2000*. Worcester, MA, USA, 2000, p. 109–124

APPENDIX

A

FRAMEWORK DOCUMENTATION (EXCERPT)

A.1. Preprocessor

The preprocessor provides a variety of functions to assist quick loading and processing of traces. A trace is represented by a `preprocessor.Buffer` with a certain data type and can be stored (`preprocessor.write_file()`) or read (`preprocessor.load_file()`) from files on the harddrive.

The system has been designed to be more or less type-independent, allowing to process a trace of a certain type while outputing to another data type. This makes sense for example for the `preprocessor.integrate()` function, that often produces values that are out of range of the source data type. Accordingly, the `preprocessor.average()` function typically outputs floats, so a precission loss will occur unless a conversion is requested. Nearly all functions thus accept a *dst_type* parameter, that controls the output data type and defaults to zero and is then set to the same data type as the input data type.

To see which types have been compiled into this module run:

```
$ pydoc dpalib.preprocessor.types
```

class `AverageCounter()`

The `AverageCounter` processes traces sequentially and calculates a average and variance trace based on the input traces.

```
>>> a = AverageCounter(size=5, type=types.float)
>>> a.add_trace(buffer_from_list(types.uint8_t, [0, 1, 2, 3, 4]))
>>> a.add_trace(buffer_from_list(types.uint8_t, [2, 2, 2, 2, 2]))
>>> len(a)
2
>>> avg, var = a.get_buf()
>>> print avg
[1.0, 1.5, 2.0, 2.5, 3.0]
>>> print var
[1.0, 0.25, 0.0, 0.25, 1.0]
```

`get_buf`

get_buf() -> (avg_buf, var_buf)

returns one float buffer for `Buffer` the average and one for the variance of all processed traces

class `Buffer()`

Provides a interface for wrapping and keeping track of native C buffers

```
>>> buf_a = new_buffer(5, types.int8_t)
>>> buf_a[0] = 255
>>> print buf_a[0]
-1
>>> buf_b = buffer_from_list(types.float, [1.5, 2.25, 3.75])
>>> print buf_b.as_list()
[1.5, 2.25, 3.75]
```

`as_list`

returns a list representation of the whole `Buffer`

`zero`

fills the whole buffer with zeros

`analyze()`

analyze(buf, include_variance=True) -> (average, variance, min, max)

calculates above characteristics for the `Buffer` buf

If *variance* is not needed, calculation is slightly quicker.

`average()`

average(buf, n, skip=1, scale=1, dst_type=types.void) -> `Buffer`

average *n* samples of `Buffer` *buf*, the result is scaled by *scale*

skip is a skip-divisor. i.e. skip=2 returns every 2nd result sample

If *signed_scale* is set, a signed scale will be performed with the value *signed_scale* as the virtual zero (i.e. y = (x - signed_scale) * scale + signed_scale). A good value for *signed_scale* is thus the average of the trace.

By explicitly specifying a *dst_type* the result `Buffer` is forced to this type.

`buffer_from_list()`

buffer_from_list(type, list) -> `Buffer`

allocates a new `Buffer` and fills its values with the ones found in *list*

`diff()`

diff(a, b, absolute=True, dst_type=types.void) -> `Buffer`

calculates abs(a[i] - b[i]) for each sample of `Buffer` *a* and *b*, returns a new `Buffer` of length min(len(a), len(b))

If the output `Buffer` type is unsigned and *absolute* is not set the `Buffer` will be shifted by max_{datatype}/2.

By explicitly specifying a *dst_type* the result `Buffer` is forced to this type.

`filter()`

filter(buf, filter_data, scale=1, dst_type=types.void) -> `Buffer`

applies a FIR filter to the `Buffer` buf

filter_data is a `types.int8_t Buffer` containing the filter coefficients, which are automatically scaled by 1/sum(filter_data)

The result can be scaled by *scale*.

If *signed_scale* is set, a signed scale will be performed with the value *signed_scale* as the virtual zero (i.e. y = (x - signed_scale) * scale + signed_scale). A good value for *signed_scale* is thus the average of the trace

By explicitly specifying a *dst_type* the result `Buffer` is forced to this type.

`free_buffer()`
free_buffer(buf) – explicitly free an allocated `Buffer`. This is usually not needed!

`integrate()`
integrate(buf, n, dst_type=types.void) -> `Buffer`

builds the sum of *n* samples each

By explicitly specifying a *dst_type* the result `Buffer` is forced to this type.

`load_file()`
load_file(filename, type, length=0) -> `Buffer`

reads a file into memory returning a `Buffer`

a non-zero *length* specifies the maximum amounts of bytes read

`new_buffer()`
new_buffer(length, type, ptr=0) allocate a new `Buffer` or generate one from an existing *ptr*

`normalize()`
normalize(buf, min=NaN, max=NaN, adjust_factor=1.2, dst_type=types.void) -> `Buffer`

normalizes a trace with values in]min, max[to fit the whole range of the *dst_type*

If *min* and *max* are not set, they will be computed from the current trace with a border of *adjust_factor* (e.g. [0, 1] is adjusted to [-0.2, 1.2]).

By explicitly specifying a *dst_type* the result `Buffer` is forced to this type.

`peak_extract()`
peak_extract(buf, avg=-1, std_dev=-1, break_count=0, break_length=0, dst_type=types.void) -> `Buffer`

extracts high peaks from `Buffer` buf

If *avg* and *std_dev* are not set they are calculated in an analysis phase first.

break_count specifies the number of pause at the beginning of the trace

If traces are not aligned in the time domain, but include a distinct pattern (i.e. a pause in signal), this can be used to align them.

break_length specifies the minimum length of each pause in samples

By explicitly specifying a *dst_type* the result `Buffer` is forced to this type.

`raster()`
raster(buf, edge, period, dst_type=types.void) -> `Buffer`

aligns a given trace buf using the pattern defined by *edge*

edge is a buffer of the same type as buf

period specifies the length of a period in samples each *period* in `Buffer` buf is linearly interpolated to fit the length specified by *period*

See also `raster_config()` to specify advanced attributes such as trigger_values and expected pause intervals.

By explicitly specifying a *dst_type* the result `Buffer` is forced to this type.

`rectify()`

rectify(buf, avg, dst_type=types.void) -> `Buffer`

rectifies a traces, by calculating the absolute difference to *avg*

By explicitly specifying a *dst_type* the result `Buffer` is forced to this type.

`reorder()`

reorder(buf, period, dst_type=types.void) -> `Buffer`

reorders a rasterized `Buffer` buf so that

- *period* buffers are created, each containing only one sample of each period i.e. the buffer contains only the first sample of each period and so on
- all buffers are concatenated again

By explicitly specifying a *dst_type* the result `Buffer` is forced to this type.

```
>>> reorder(buffer_from_list(types.uint8_t, [1,2,3,4,5,6,7,8,9,10,11]), 3)
[1, 4, 7, 10, 2, 5, 8, 11, 3, 6, 9]
>>> reorder(buffer_from_list(types.uint8_t, [1,2,3,4,5,6,7,8,9,10]), 3)
[1, 4, 7, 10, 2, 5, 8, 3, 6, 9]
```

`scale()`

scale(buf, scale=1.0, signed_scale=0, dst_type=types.void) -> `Buffer`

scales buffers values by *scale* (i.e. buf[i] *= scale)

If *signed_scale* is set, a signed scale will be performed with the value *signed_scale* as the virtual zero (i.e. y = (x - signed_scale) * scale + signed_scale). A good value for *signed_scale* is thus the average of the trace.

By explicitly specifying a *dst_type* the result `Buffer` is forced to this type.

`spline()`

spline(buf, target_size, dst_type=types.void) -> `Buffer`

linearly interpolates a `Buffer` buf to fit a given length

For reasons of efficiency this does not do averaging, if *size* is significantly smaller than *buf*'s size.

spline interpolation is not yet implemented

By explicitly specifying a *dst_type* the result `Buffer` is forced to this type.

`square()`

square(buf, dst_type=types.void) -> `Buffer`

calculates the square of each value in `Buffer` buf

By explicitly specifying a *dst_type* the result `Buffer` is forced to this type.

`write_file()`

write_file(filename, buf, length=0)

dumps the `Buffer` *buf* to a file

a non-zero *length* specifies the maximum amounts of bytes written

A.2. Correlation

class `Correlator()`

Correlator(samples, traces, keys)

creates a new `Correlator` instance used to rapidly calculate correlations in a DPA scenario

samples is the number of samples in each trace, i.e. the trace length

traces is the total number of traces to be processed

keys is the number of hypothesis

```
>>> c = Correlator(2, 3, 1) #create a new correlator
>>> c.hypo[0] = 5           #calculate a hypothesis for each trace
>>> c.hypo[1] = 4
>>> c.hypo[2] = 3
>>> c.preprocess()          #preprocess the hypothesis
>>> from preprocessor import buffer_from_list
>>> c.add_trace(buffer_from_list(types.uint8_t, [10, 0]))
>>> c.add_trace(buffer_from_list(types.uint8_t, [ 8, 30]))
>>> c.add_trace(buffer_from_list(types.uint8_t, [ 6, 15]))
>>> c.update_matrix()       #calculate the correlation
>>> round(c.matrix[0], 2)
1.0
>>> round(c.matrix[1], 2)
-0.5
```

`add_trace`

add_trace(buf, idx=-1)

processes a trace (`dpalib.preprocessor.Buffer` buf) for the `Correlator` by updating intermediate values

if *idx* is set, the trace will be added as trace number *idx* allowing to add traces in arbitrary order

`preprocess`

preprocesses the hypothesis. MUST be called before adding the first trace

`update_matrix`

updates the correlation matrix. MUST be called before accessing the matrix

`dump_matrix()`

dump a octave readable form of the `Correlator.matrix` *m* to the file-descriptor *f*, assuming that *m* is a *keys* x *samples* matrix

A.3. Workflow

class `DPAWorkflow(`*info_dict={}, count=None, base_path='.'*`)`

A helper class for a complete trace analysis workflow

Processes a set of traces in a series of analysis steps. Each step is executed by a trace processor. See the `dpalib.processors` module for details.

In a profiling phase, 100 input traces will be processed sequentially to experimentally determine boundaries and lengths (e.g. for the `dpalib.processors.AverageCountProcessor`). This phase is necessary, because several processors require further information. For example, the `dpalib.processors.NormalizeProcessor` needs information on the minimum and maximum of each trace. However, these values need to be constant for the whole set of traces as the output of the processor would otherwise be useless. Other processors like the peak-extraction processor need information on traces' average and variance which typically do not vary much in a set of traces. Therefore, such information is pre-calculated using a couple of sample traces in a profiling phase. This functionality is supported by respective capabilities of the individual processor classes.

The number of traces to profile can be set with the *profile_size* attribute. A record information dictionary can be passed to the module, which contains information to be used by the seperate processors, but the following keys are also used:

errors a list of trace numbers to ignore. These traces will not be read or processed

profile_traces a list of traces that are meant to be used for additional profiling if the initial 100 are not sufficient

trace_type the data type of input traces. Default: `dpalib.preprocessor.types.uint8_t`

In the actual processing phase the traces are processed in parallel using all available cores if the corresponding trace processors are implemented to release the GIL for the actual processing. This is the case for the preprocessing toolsuite including the correlator.

See this source file for a more practical and thorough application of this class.

```
>>> w = DPAWorkflow(count=100)
>>> w.processors = [TraceProcessor()] # the TraceProcessor() doesn't modify anything
>>> w.process()
```

`path_iter(`*in_path, out_path=None*`)`

`process()`

processes the active trace set

See `DPAWorkflow` for a generic overview of provided functionality.

`store_avg(`*avg, name=''*`)`

A.4. Processors

A collection of trace processors for use in a `dpalib.workflow.DPAWorkflow`.

A trace processor can be easily written by inheriting from `TraceProcessor`:

```
>>> class CustomProcessor(TraceProcessor):
...     def process(trace):
...         return custom_processing_function(trace)
```

Further processors can be listed with:

```
$ pydoc dpalib.processors
```

class `AverageCountProcessor`(*callback=<function <lambda> at 0x3247500>, **kwargs*)

Accumulates traces to create a average and variance trace

Take an argument:

callback a function that is called after calculation of the average has been finished

callback(avg, var, name)

where *name* is the name of the referenced processor

static `averagize`(*processors, callback=None, **kwargs*)

takes a list of processors and creates a AverageCountProcessor instance for each one

`finalize()`

calculates the average and calls the *callback* function

class `CombinedProcessor`(*a, b, name=None*)

A helper class to provide the chaining functionality for a `TraceProcessor`

class `CorrelationProcessor`(*correlator=None, **kwargs*)

Adds each processed trace to the correlation module

correlator a `dpalib.correlation.Correlator` instance that has already been initialized with hypothesis and preprocessed

`correlations()`

retrieves the correlations after processing finished, by cutting the correlation matrix into corresponding chunks

class `NormalizeProcessor`(*dst_type=0, ref=None, save=False, name=None*)

Normalizes trace data globally to fit a smaller data type.

This is useful if trace data does not fit a small data type without further processing. The `NormalizeProcessor` determines min and max values of the traces in a profiling stage and fits them into the new data type.

A common application might be:

```
>>> NormalizeProcessor(dst_type=types.uint8_t)
```

class `PeakProcessor`(*break_length=0, break_count=0, *args, **kwargs*)

Performs peak extraction on the trace.

Peak extraction needs information on average and variance of the trace. The values are determined in a profiling stage.

class `RasterizeProcessor(`*edge, period, trigger=150, pause_trigger=1100, min_pause=0, max_pause=0, header_size=128, **kwargs*`)`

A processor for rasterization of traces. Please consult the thesis for an exact description.

Uses a pattern defined by `dpalib.preprocessor.Buffer` *edge* to make each period the same length (*period*).

Further options can be specified:

trigger edge-comparism threshhold that starts search for a local minimum difference

pause_trigger number of samples to pass without a trigger match, that causes indication for a data pause. This can be used to align the start of traces at pauses in the trace

min_pause the minimum amount of pauses that MUST occur

max_pause the maximum amount of pauses that are allowed to occur before assuming an error

header_size number of samples to skip from the beginning of the trace

FIXME: there can only ever be one active configuration of this running at the same time

class `RectifyProcessor(`**args, **kwargs*`)`

Rectifies a trace

class `TraceProcessor(`*dst_type=0, ref=None, save=False, name=None*`)`

A no-op trace processor.

This is a simple base class for other trace processors doing actual computations, that should inherit from this class and overwrite the `process()` method.

Trace processors can be chained:

```
>>> a = TraceProcessor()
>>> b = TraceProcessor()
>>> c = a(b)
```

c.process(buf) is thus equivalent to a.process(b.process(buf))

`finalize()`

finishes pending tasks

`get_samples()`

returns the estimated number of samples of a output trace based on the profiling phase

`process(`*trace, idx=-1*`)`

processes the `dpalib.preprocessor.Buffer` *trace* and returns the modified version

`profile(`*trace*`)`

performs profiling steps (such as finding min/max values) for the `dpalib.preprocessor.Buffer` *trace*

class `VoidProcessor(`*dst_type=0, ref=None, save=False, name=None*`)`

Base class for processors not producing new traces