

A customizable side-channel modelling and analysis framework in Julia

Simon F. Schwarz
Robinson College



**UNIVERSITY OF
CAMBRIDGE**

*A dissertation submitted to the University of Cambridge
in partial fulfilment of the requirements for the degree of
Master of Philosophy in Advanced Computer Science*

University of Cambridge
Computer Laboratory
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
UNITED KINGDOM

Email: sfs48@cam.ac.uk

May 5, 2021

Declaration

I Simon F. Schwarz of Robinson College, being a candidate for the M.Phil in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: ??

Signed:

Date:

This dissertation is copyright © 2021 Simon F. Schwarz.

All trademarks used in this dissertation are hereby acknowledged.

Abstract

Write a summary of the whole thing. Make sure it fits in one page.

Contents

1	Introduction	1
1.1	Motivation	2
1.2	Aims	3
1.3	Structure	3
2	Background & related work on side-channel attacks	5
2.1	Background	6
2.2	Traces	8
2.2.1	Collection	8
2.2.2	Preprocessing	9
2.3	Attacks	10
2.3.1	DPA	10
2.3.2	CPA	13
2.3.3	SPA	16
2.3.4	Template Attacks	16
2.3.5	Advanced Attacks	16
2.4	Countermeasures	20
2.4.1	Masking	21
3	Background on Julia	25
3.1	Types	25
3.2	Multiple Dispatch	25
4	Implementation	27
4.1	Ciphers	28
4.2	Custom Types	28
4.2.1	Integer-like types	28
4.2.2	Logging	28
4.2.3	Masking	28
4.3	Attacks	28

4.4	Documentation	28
5	Evaluation	29
6	Summary and Conclusions	31
6.1	Summary	31
6.2	Future Work	31

List of Figures

1.1	Real-world side-channel analysis: An oscilloscope measuring the power of a microchip.	2
2.1	Power consumption during the execution of an AES encryption. The first peak is caused by loading the data, the subsequent ten peaks correspond to the ten rounds of AES. Data source: [1]	8
2.2	A recorded power trace, and the point in time where the first AES S-Box lookup is performed.	10
2.3	Power consumption over all traces at the first S-Box lookup.	10
2.4	Power consumption over the two partitions at first S-Box lookup.	11
2.5	DOM over time between the two positions. The spike marks the time of the first S-Box lookup.	12
2.6	DOM over time for two key guesses. The spike for the correct key guess is at the position where the first S-Box output is calculated.	13
2.7	Comparison between the power prediction for the correct key, and the actual measured power at the point of the first S-Box computation ($t = 298$). Both are strongly correlated ($\rho = 0.87$).	15
2.8	Correlation for correct key and for incorrect key over time. The correct key exhibits a spike in correlation at the computation of the first S-Box output ($t = 298$)	16
2.9	Correlation for all keys	17

List of Tables

Chapter 1

Introduction

TODO photo

Figure 1.1: Real-world side-channel analysis: An oscilloscope measuring the power of a microchip.

1.1 Motivation

“Classical” cryptanalysis tries to analyze the security of cipher algorithms based on their mathematical specification. However, this analysis cannot account for security issues arising from the realization of an algorithm. Such issues are very common and can either originate from software or hardware flaws.

For example, the NSA used its TEMPEST [2] program as early as WWII to measure EM emissions to reconstruct secret messages written on a typewriter. More recently, such attacks on popular cryptographic algorithms like AES have been published [3, 4]. Lastly, attacks like SPECTRE [5] or Meltdown [6] have gained popularity by abusing hardware flaws. All those attacks have in common that they utilize additional information gathered via an *unintended side-channel*, like EM emissions, timing information, or power consumption. *Side-channel analysis (SCA)* is the study of the real-world realization of ciphers that takes side-channel information into account. Hence, SCA tries to close the gap between the formal security guarantees of a cipher, and its real-world security.

An example setup for SCA could be an oscilloscope connected via shunt-resistor to the power supply of a processor. The oscilloscope then repeatedly measures the power consumption in small intervals during the execution of a cryptographic algorithm. This setup is depicted in Figure 1.1. However, a setup like this is expensive and requires advanced knowledge to use.

Hence, SCA is currently not very accessible, which leads to less awareness about possible attacks. Ultimately, this can then lead to more vulnerable implementations. This project tries to address this gap by providing a purely software-side solution for generating and analyzing side-channel information,

removing the need for expensive hardware. In particular, this project can be used in the context of teaching side-channel security.

1.2 Aims

This project aims to ease the analysis of side-channel attacks and to eliminate the need for additional hardware. In our framework, a purely software-side solution for trace generations is implemented.

One goal of this project is the analysis of ciphers without major modifications to the original source code. With our framework, only the underlying types of cryptographic functions must be changed, while all of the other source code can be kept. This makes the whole process of analyzing a cipher convenient and easily reproducible.

The project was implemented in the Julia language [7]. Julia is a modern high-performance language based on the *multiple dispatch* paradigm. In combination with Julia's flexible type system, this allows us for implementing our framework in a generic way.

Equipped with a convenient way to record leakage emissions from a cryptographic algorithm, our framework provides various methods for analyzing the recorded data with respect to side-channel security.

1.3 Structure

First, chapter 2 will summarize the background and related work on side-channel analysis, ending with some state-of-the-art attacks. Afterwards, chapter 3 will have a brief look at the background of Julia, especially focusing on Julia's flexible type system. Next, 4 will discuss this project's implementation in detail. In chapter 5 we will have a look at the results, and analyze two popular encryption algorithms with the help of our framework. Lastly, the chapter 6 gives a brief summary of this thesis and highlights potential future work.

Chapter 2

Background & related work on side-channel attacks

In this chapter, we will have a brief look at the motivation and history of side-channel analysis, before dealing with some classical side-channel attacks in detail. In section 2.4 we will explore some countermeasures against different side-channel attacks.

2.1 Background

Cryptanalysis aims to study encryption systems with regard to their security. An important part of cryptanalysis is the “classical” mathematical analysis of an encryption algorithm. This part usually considers only the information directly present in the mathematical specification of the cipher. This information is usually restricted to the input, the output, and the key. For example, a “classical” desirable property for an algorithm would be: “An adversary is unable to efficiently infer the key given only a pair of plaintext and corresponding ciphertext”. Modern cryptographic algorithms, like the Advanced Encryption Standard (AES) [8] or SPECK [9] are generally considered secure in this classical sense.

However, the mathematical analysis (and claims of security made by it) solely rely on the high-level mathematical transformation described by the algorithm. Despite their security in a theoretic scenario, those algorithms are eventually implemented in software, and executed on hardware. Both steps can potentially lead to vulnerabilities that cannot be captured by the high-level mathematical specification. Such weaknesses in implementation or hardware are studied in the area of *side-channel analysis*. In general, side-channel attacks exploit additional information about the execution of a cryptographic algorithm. This information is not part of the cipher’s mathematical specification and is gathered via *unintended side-channels*.

For example, one such unintended side-channel is the power consumption of the processor. Usually, a processor internally represents a binary number using high or low wires. The single wires representing the number then change during arithmetic operations. In most hardware processors, pulling a line to high (or even to low) consumes more energy than not changing

the state at all. Hence, it stands to reason that, for example, computing $(0000)_2 + (0001)_2$ could need less power than computing $(1010)_2 + (1101)_2$. This intuition is later on captured by a power estimation model, which will be discussed in section 2.3.2. Hence, by measuring the power consumed by the processor during cryptographic operations, it can be possible to draw conclusions about the values processed. Ultimately, the goal is to use this knowledge of intermediate values to conclude the secret key.

Apart from power side-channels, examples of such side-channels include, but are not limited to:

- The wall-clock runtime of the algorithm during an en- or decryption process. For example, Brunley and Boneh show in [10] that it was possible to extract private keys from a remote webserver running OpenSSL ([11]) using a timing side-channel.
- The state of the processor cache, which may be modified. Prominent attacks include Meltdown [6] and SPECTRE [5].
- Electromagnetic, acoustic, optical, or other measurable emissions of the underlying device. A prominent example is the NSA's TEMPEST program ([2]), which extracted sensitive information from electromagnetic emissions.

TODO history + structure of what comes in the next sections

The study of modern side-channel attacks has been introduced by

- Timing (asymmetric) Kocher in 1996 ([12]),
- SPA, DPA: 1999 [3].
- Template Attacks: Chari, Rao, Rohatgi in 2002. [13])
- CPA: 2004 [4]

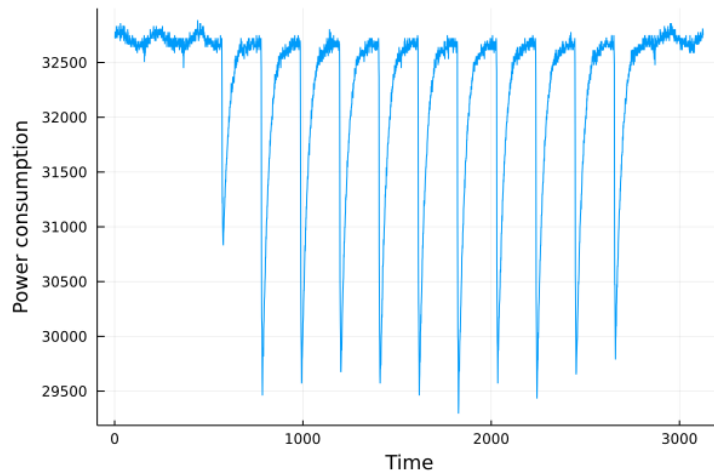


Figure 2.1: Power consumption during the execution of an AES encryption. The first peak is caused by loading the data, the subsequent ten peaks correspond to the ten rounds of AES. Data source: [1]

2.2 Traces

In this thesis, we will focus on side-channels that produce a *trace* of emissions during cryptographic operations.

For example, if a power side channel is used, the measurement of power over time during a cipher operation forms a *power trace*. Figure 2.1 depicts an example of a plotted power trace during the execution of AES. Usually, only relative power consumption is considered when analyzing power traces. Thus, units are relative and are commonly directly taken from the output of the Analog-Digital-Converter used to record the trace.

2.2.1 Collection

Include setup to monitor power side channels, maybe built something by myself?

Power traces are usually collected with an oscilloscope connected via a shunt resistor to the device power line. Such a setup is also shown in figure

TODO ref

TODO figure connection wiring.

For most attacks, multiple traces from the same device and cipher are needed. Usually, those traces should perform the same encryption with the same key, but with *different* input. A scenario where this attack model is suitable is, for example, a Smart-Card that encrypts values it receives. Now, different inputs can be sent to this chip to obtain traces. The exact number of traces needed for an attack depends on a variety of factors including recording quality or potential implemented countermeasures. Kocher suggests in [3] to use 4000 traces, while other authors collect up to 2^{32} traces.

TODO platypus, more creative collection of traces.

Alignment If multiple traces are collected, traces must be *aligned*. There are multiple different approaches to keep traces aligned. If the monitored device provides a signal like a line pulled high as soon as processing begins, this line can be used as a trigger in the oscilloscope. However, a single trigger cannot account for oscillator tolerances of the microcontroller. To cope with this problem, a more advanced approach is to use an external oscillator that provides the same clock signal to the microchip and the oscilloscope. This allows for far more fine-grained measurements. **TODO** ask markus what time they achieved vs. conventional time. publication?

2.2.2 Preprocessing

TODO Filtering, most variance points

Interesting point identification DOM on points (fscore, Anova, r^2) cut out single intervals take 20 percent most interesting points

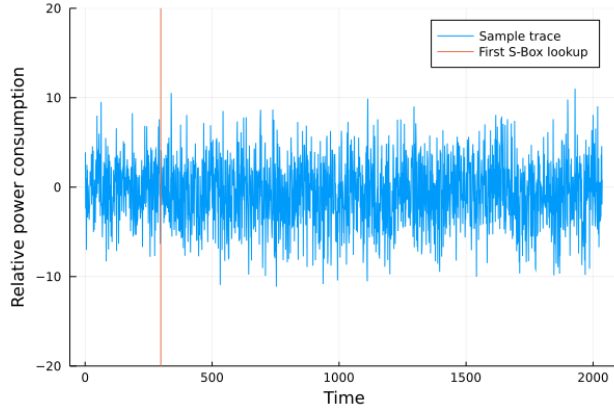


Figure 2.2: A recorded power trace, and the point in time where the first AES S-Box lookup is performed.

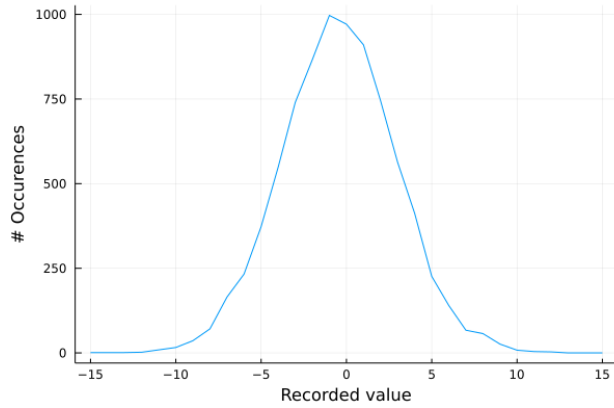


Figure 2.3: Power consumption over all traces at the first S-Box lookup.

2.3 Attacks

2.3.1 DPA

Differential Power Analysis (DPA) was first outlined by Kocher [3] in 1999. First, this approach collects multiple aligned traces. The traces should record a cipher operation on a different, random input to the cipher each time. Note that, however, the cipher algorithm and the key must remain static for this attack to work.

Consider a set of aligned traces recording an en- or decryption. Hence, at a

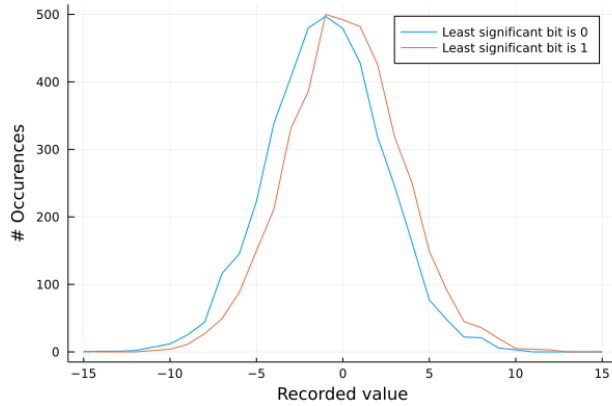


Figure 2.4: Power consumption over the two partitions at first S-Box lookup.

fixed time, the same operation has been executed while recording each trace. For example, if the recorded device performs an AES encryption, there is a timepoint where the first S-Box output is computed. Figure 2.2 shows a sample recorded trace and the timepoint of the computation of the first S-Box output. Considering all traces, we can plot the frequency a specific value is recorded at the first S-Box lookup. Such a frequency diagram is depicted in figure 2.3.

The fundamental idea of DPA is now to partition the set of all collected traces into two subsets according to a guess of the intermediate value. For example, we can partition the traces from figure 2.3 based on “Is the least significant bit of the first S-Box output 0?”. The frequency of the two induced subsets is plotted in figure 2.4. Notably, the two subsets are distinguishable and have a different mean.

While analyzing traces, the exact point of the computation of the first S-Box output is not known a priori. Hence, it is necessary to not only compare the difference of means at a single position, but rather consider it at all possible positions. Figure 2.5 plots the difference of means over the whole trace length, where the partitioning is performed upon the least significant bit of the first S-Box output. Naturally, this difference spikes at the computation of this value, and approaches zero elsewhere. This is a good indicator that a value depending on the first S-Box output actually occurs in our trace.

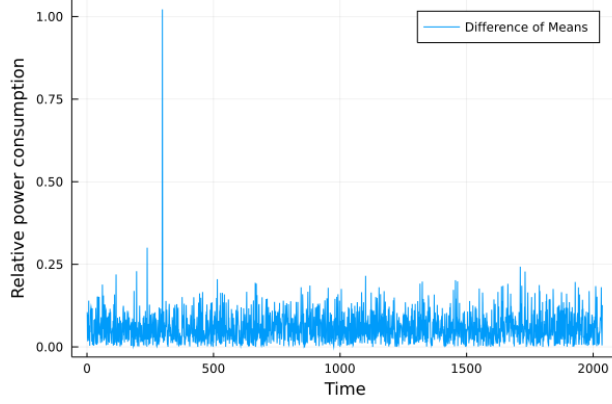


Figure 2.5: DOM over time between the two positions. The spike marks the time of the first S-Box lookup.

Recall that we partitioned the traces based on “Is the least significant bit of the first S-Box output 0?”. However, in an attack scenario, we cannot directly know this value. In AES, the defining equation for the S-Box output in the first round is:

$$I_i = S[X_i \oplus K_i]$$

where I_i is the S-Box output for the i -th byte, K_i is the i -th key byte, and X_i is the i -th input byte. Now, note that S is the static (and public) AES substitution box, and X_i is known for each trace. Thus, the S-Box value only depends on K_i , which is a single key byte.

Now, the already established method can be used as an oracle: For each possible key guess $K_i \in \{0, \dots, 255\}$, calculate I_i for each trace, and partition the traces based on the least significant bit of I_i . If the guess for K_i is correct, then I_i is correct as well. Hence, at the calculation of the first S-Box output, we expect the difference of means between both partitions to be non-zero. Thus, this time-point will show up as a “spike” if the DOM is plotted over time.

In contrast, for a wrong guess of K_i , there will be no correlation between I_i and the actual trace values, since I_i is never computed during the recording. Hence, given enough traces, the difference of means approaches zero at all

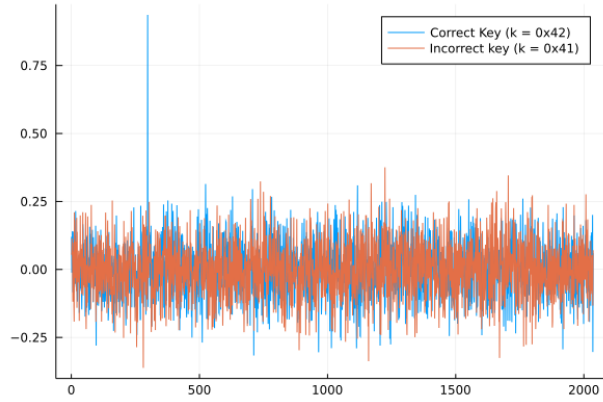


Figure 2.6: DOM over time for two key guesses. The spike for the correct key guess is at the position where the first S-Box output is calculated.

points, and there will be no “spike” in computation. Figure 2.6 compares typical DOMs for a correct key guess with an incorrect key guess.

Given an oracle that can determine if a key byte guess K_i is correct trivially breaks AES: For each key byte, there are 256 different possibilities. For each possibility, the oracle can be queried, resulting in $16 \cdot 256$ queries in total. This is a vast improvement to a naive brute-force attack, which would need 2^{128} tries.

2.3.2 CPA

Correlation Power Analysis was published by Brier, Clavier, and Olivier in 2004 [4]. It tries to mitigate problems with DPA caused by only taking a single bit into account. Instead, in CPA, a *leakage model* is established. This model predicts side-channel values during the processing of different values.

Leakage models

For example, a leakage model for a power side-channel attack would be a power consumption estimate. Implicitly, such a model was already established for DPA by assuming: “Processing a value with LSB 1 takes more (or less) energy than one with LSB 0”. However, this model can be improved

by, for example, taking other bits into account. This idea is formalised in a leakage model. Such a model is a function receiving a value that is processed R , and should return a side-channel estimate W_R .

Hamming weight model For example, a simple model is the Hamming weight. Formally, the Hamming weight of a b -bit number $R = \sum_{j=0}^{b-1} d_j 2^j$, $d_j \in \{0, 1\}$ is $H(R) = \sum_{j=0}^{b-1} d_j$. Then, our leakage model is

$$W_R = H(R)$$

This model captures a processor that consumes power for every set bit, for example, by pulling the corresponding wire up.

Hamming distance model A generalized version of the Hamming weight model is the Hamming distance model. This model captures the idea that power leakage depends on switching bits, i.e. it consumes power to pull a line high or low. If the state before the computation of R is D , then the Hamming distance model is

$$W_R = H(R \oplus D)$$

Computing correlation

Given an estimate for the side-channel value, the goal of CPA is to check if the measured values indeed correlate to the prediction. For a reasonable leakage model, we would expect a linear correlation, which is captured by Pearson's correlation coefficient

$$\rho_{W,H} = \frac{\text{cov}(W,H)}{\sigma_W \sigma_H}$$

It holds that $-1 \leq \rho_{W,H} \leq 1$, where $\rho_{W,H}$ is close to ± 1 for a good linear correlation.

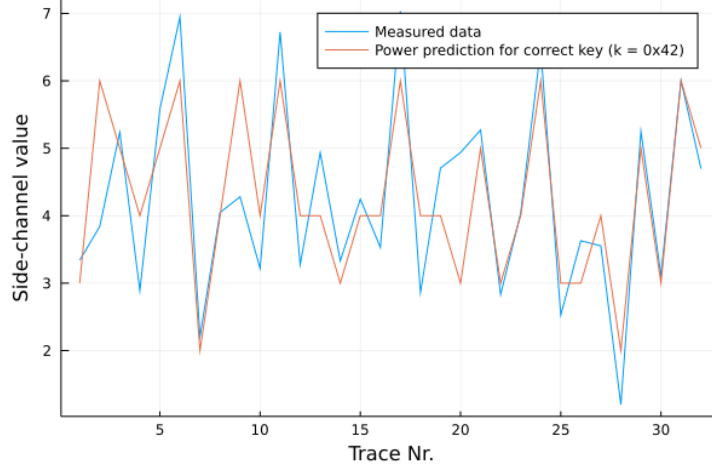


Figure 2.7: Comparison between the power prediction for the correct key, and the actual measured power at the point of the first S-Box computation ($t = 298$). Both are strongly correlated ($\rho = 0.87$).

Example: Attack against AES

Given a power estimation model, we can target the first S-Box output as in DPA. Let W_R denote the power estimation to calculate value R . Consider only the i -th position in AES ($1 \leq i \leq 16$). Then, for each different input $X_i \in \{0, \dots, 255\}$, we can calculate $W_{S[X_i \oplus K_i]}$ if K_i is known. Since $K_i \in \{0, \dots, 255\}$ is a single key byte, this value can be brute-forced.

For each trace, $W_{S[X_i \oplus K_i]}$ is a trace-specific power estimation. Now, at the point where $S[X_i \oplus K_i]$, we expect the measured values to correlate with the power estimation. Figure 2.7 shows the power estimation for the correct key, and the measured values at the point where $S[X_i \oplus K_i]$ is calculated. Note that the prediction and measured data correlate very well.

However, the point where the first S-Box computation occurs is a priori not known. Hence, the correlation is established for every point of time in our traces. Figure 2.8 shows correlation over time for the correct key, as well as for an incorrect key. The correlation with the estimate for the correct key spikes at the point where the first S-Box output is calculated. Note that the correlation with the correct key also has minor spikes afterwards, which

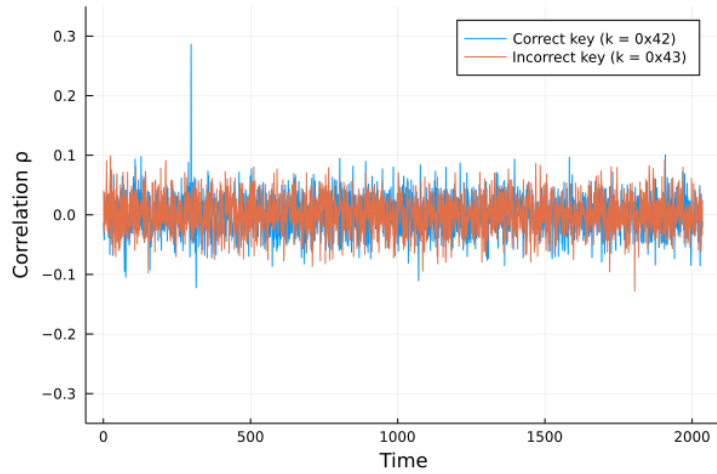


Figure 2.8: Correlation for correct key and for incorrect key over time. The correct key exhibits a spike in correlation at the computation of the first S-Box output ($t = 298$)

result from the computation of values directly dependent on the targeted S-Box output.

Similar to DPA, likely key bytes are expected to have bigger spikes in the correlation at the targeted timepoint. To infer the correct key, it is sufficient to consider the maximal absolute correlation at any timepoint. Figure 2.9 shows the maximal correlation for all key candidates, with a clear spike at the correct key $k = 0x42$.

2.3.3 SPA

TODO needed?

2.3.4 Template Attacks

2.3.5 Advanced Attacks

All previously explained attacks use the principle of *divide and conquer*: First, the whole key (for example, the 16 key bytes from AES) is divided into smaller pieces, and later combined to form the whole key. This approach is

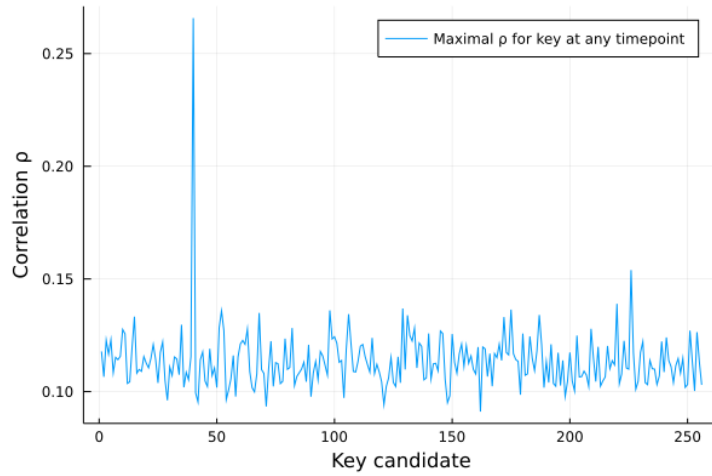


Figure 2.9: Correlation for all keys

very powerful, since almost no knowledge of the exact implementation was used. Moreover, most often only single values and single points in time were targeted. A clear benefit of this approach is, hence, the simplicity of the attacks. However, this poses the question if more information can be obtained from side-channel traces by taking more properties of the cipher into account.

Algebraic Attacks

Algebraic attacks are a class of side-channel attacks. In this attacks, the *algebraic properties* of ciphers are exploited. Clearly, all intermediate values that occur during the encryption process are algebraically related to each other. This relationship is publicly known by the specification of the algorithm. Attacks from this class then try to combine side-channel information from different timepoints with their algebraic relationships. For example, assume the Hamming weight of all intermediate values is known. Then, it is possible to obtain a system of equations from the specification of the algorithm, and from the Hamming weight of all intermediate values. Hence, this system of equations should be overdetermined by the additional constraints on Hamming weight. Those additional constraints can make the resulting system easier to solve, for example, with a SAT- or SMT-Solver.

Some prominent algebraic attacks are:

- *SPA on AES Key-Expansion*: In [14], Mangard provides a way to reconstruct the AES secret key by knowing the Hamming weights of all intermediate values of the key expansion. By exploiting the additional constraints on Hamming weight combined with properties of the AES key expansion, the time complexity of reconstructing the key is greatly reduced compared to a brute-force search. The resulting attack complexity is feasible in practice.
- *Collision attacks* have been introduced by Schramm, Wollinger, and Paar in [15] (back then against DES). Subsequently, collision attacks have been generalized to cover AES in [16], and improved by Bogdanov, Kizhvatov, and Pyshkin in [17]. The overall idea is to exploit collisions of *intermediate* values. Based on equalities between some intermediate values, conclusions about the round keys can be drawn. [17] shows that for each collision, it is possible to infer 8 secret key bits. Following from the birthday paradoxon, collisions are very likely to occur even for few (about 20) traces, which is a significant improvement compared to DPA or CPA, where vastly more traces are needed.
- *Algebraic attacks* are a generalisation of the attacks outlined above. They were introduced by Renaud and Standaert in [18]. In their paper, they show how to use generic SAT-solving approaches to target all intermediate values, not only the ones occurring in the first or last rounds. **TODO** make clear that this list is the history of AA.

The major benefit of algebraic attacks is the need for vastly fewer traces compared to divide-and-conquer approaches. However, algebraic attacks also exhibit three major weaknesses:

- First, more knowledge of the internal structure of the targeted device is required. For example, it is necessary to know in which order values are computed to establish algebraic relationships. This poses problems especially when the attacked device is a *black box*, i.e. no further information about the software or hardware is known

- Second, algebraic attacks have little *error tolerance*. In the attacks outlined above, the inferred constraints on intermediate values were treated as “hard” constraints. However, in a real-world scenario, some of those values may be wrong. This, most likely, would produce an unsatisfiable system of equations.
- Third, depending on the exact scenario, a worse runtime in comparison to divide and conquer attacks must be anticipated.

Soft analytical side-channel attacks

Soft analytical side-channel attacks (SASCA) were introduced by Veyrat-Charvillon, Gérard, and Standaert in [19]. SASCA tries to address the last two problems of algebraic attacks, namely runtime and error tolerance.

Concretely, SASCA encodes the “soft” probabilistic side-channel information into an algebraic algorithm. In contrast, in classical algebraic attacks, the probability of constraints is usually discarded, and the most likely option is chosen. Hence, the underlying algebraic algorithm has access to another source of information, namely the probability of the additional constraints. This information is encoded to be solvable with the Belief-propagation algorithm.

This construction allows to combine the best of both worlds. In practice, this can reduce the number of required traces compared to template attacks by a factor of $2^3 - 2^4$.

TODO more SASCA??

Further attacks

TODO anything further?

2.4 Countermeasures

All presented side-channel attacks exploited the fact that additional side-channel information was present. Furthermore, it was necessary that the emitted side-channel data depended on cryptographic secrets. Hence, countermeasures have two starting points to remediate such attacks:

First, it is possible to reduce emitted side-channel information. Approaches from this category, for example, are

- **Shielding:** By using appropriate physical enclosures, emissions to the outside can be blocked. This can be especially effective for electromagnetic, thermal, or acoustic emissions.
- **Internalizing:** By incorporating potential side-channel leakage vectors directly inside the targeted chip, SCA can become significantly more difficult. For example, an integrated power supply can eliminate power analysis, while an integrated oscillator could prevent over-/underclocking attacks.
- **Jamming:** It is possible to actively add noise to potential side-channels. For example, random delays in computations can mitigate timing side-channels as well as power side-channels that rely on aligned traces.

Second, side-channel attacks can be prevented by *uncorrelating* the emitted data from cryptographic secrets. Here, multiple approaches exist:

- **Constant-time cryptography:** The number of instructions, or even the exact sequence of instructions executed does not depend on secrets, but is always the same. This effectively prevents timing side-channels, but is rather hard to realize.
- **Blinding:** In some cipher algorithm, user input can be blinded before en-/decryption. Here, blinding means applying an attacker-unknown permutation to the input. With blinding, it is harder for an attacker to know which values were actually passed to the cryptographic primitive.
- **Masking** is described in the next chapter.

2.4.1 Masking

The *masking* technique was first described by Goubin and Patarin in [20]. It tries to eliminate side-channel dependencies on cryptographic secrets by randomizing all processed values. This is achieved by splitting values into two or more *shares*. Only when all shares are combined, the original value can be reconstructed. If more than two shares are used, this technique is also called *higher-order masking*.

Now, during the execution of a cryptographic algorithm, operations are always performed on all shares separately. Hence, the original intermediate values never occur in memory. Now, if multiple traces are collected, the randomness by splitting the values should be different for every trace. Hence, no direct correlation between the secret, the input, and the measured data should be present.

Boolean Masking

One method of splitting a value into two shares is *boolean masking*. Here, a value V is split into two shares A_V and M_V such that

$$V = A_V \oplus M_V$$

Now, each operation involving any intermediate values V and V' should only operate on A_V and M_V separately. In other words, the value $V = A_V \oplus M_V$ should never be computed except at the very end of our algorithm. Clearly, some operations can easily be split to operate on both shares. For example, the XOR operation $V = X \oplus Y$ can be written as follows:

$$\underbrace{X \oplus Y}_{=V} = \underbrace{A_X \oplus A_Y}_{=A_V} \oplus \underbrace{M_X \oplus M_Y}_{=M_V}$$

Hence, A_V and M_V can be calculated without involving both shares for each component. Similar equations for other bitwise operations can be found in section 4.2.3.

TODO array lookup

Arithmetic Masking

Boolean masking provides a convenient way for masking bitwise operations. However, many symmetric encryption algorithms like Simon and SPECK [9] or Twofish [21] also use arithmetic operations like additions over $\mathbb{Z}/2^k\mathbb{Z}$. Those operations cannot be directly computed on boolean shares. However, *arithmetic masking* is another masking representation that can solve this problem. Here, a k -bit value V is stored as

$$V = A_V + M_V \mod 2^k$$

In this representation, arithmetic operations can be split up to operate only on the shares. For example, addition of two values becomes

$$\underbrace{X + Y}_{=V} = \underbrace{A_X + A_Y}_{=A_V} + \underbrace{M_X + M_Y}_{=M_V}$$

where both A_V and M_V were calculated without having X or Y as an intermediate value. In section 4.2.3, this construction on arithmetic shares is extended to more arithmetic operators.

Goubin's algorithm

Clearly, for ciphers that mix boolean and arithmetic operators, a method for converting between both masking types has to be found. First steps in this direction have been taken by Messerges in [22]. However, his conversion algorithm was again vulnerable to Differential Power Analysis. Subsequently, Goubin proposed in [23] an improved algorithm to convert from arithmetic to boolean masking and vice-versa. In his algorithm, no intermediate values correlate with the data to be masked. Hence, this algorithm is not vulnerable to first-order power analysis.

Both algorithms can be found in the Appendix **TODO** ref

Surprisingly, converting between the two types of masking is fairly efficient. The conversion from arithmetic to boolean masking on k -bit numbers needs only $5k+5$ elementary operations. The other direction is possible in constant time with 7 elementary operations.

Higher-Order Masking

If a masking is correctly used, the leakage at any single point in time does not correlate with the secret key. Thus, attacking only a single timepoint as in DPA, CPA, or Template attacks cannot allow for conclusions about the key. However, it still is possible to collect side-channel values at *multiple different* timepoints t_0, \dots, t_n . Then, the system may be broken by analyze the correlation between a predicted intermediate value I (without masking) and a combination of the leakage values $L(t_0), \dots, L(t_n)$. For example, Messerges shows in [24] that it can be successful to measure leakage at two timepoints and then maximizing the correlation between I and $|L(t_0) - L(t_1)|$.

This motivates *higher-order masking*. In n -th order masking, all values are split into n shares (in comparison to 2 shares). Hence, our masking equations become

$$\begin{aligned} V &= M_V^1 \oplus M_V^2 \oplus \dots \oplus M_V^n && \text{for boolean masking} \\ V &= M_V^1 + M_V^2 + \dots + M_V^n && \text{for arithmetic masking} \end{aligned}$$

Higher-order masking is currently less used in practice. However, [25] shows how to construct an efficient way to fully mask the AES algorithm for any order. However, note that any n -th order masking can, theoretically, be broken by a $n+1$ th-order attack. Experimentally, the authors of [25] have found out that breaking n th-order masking requires exponentially many collected traces in n . Hence, a very effective way of preventing higher-order attacks is to limit or throttle encryption requests.

Chapter 3

Background on Julia

3.1 Types

3.2 Multiple Dispatch

Chapter 4

Implementation

Operation	Resulting A_Z	Resulting M_Z
$Z = X \oplus Y$	$A_X \oplus A_Y$	$M_X \oplus M_Y$
$Z = X \& Y$	$(A_X \& A_Y) \oplus (A_X \& M_Y) \oplus (M_X \& A_Y)$	$M_X \& M_Y$
$Z = X \mid Y$	$(A_X \& A_Y) \oplus (A_X \& M_Y) \oplus (M_X \& A_Y)$	$M_X \& M_Y$

4.1 Ciphers

4.2 Custom Types

4.2.1 Integer-like types

TODO same as in Docu?

4.2.2 Logging

4.2.3 Masking

Boolean Masking

Include this table, or something similar.

Arithmetic Masking

Conversion

4.3 Attacks

4.4 Documentation

TODO link to docu.

Chapter 5

Evaluation

Timing would be nice. + we need space for the nice pictures rendered by Plot

Take <https://github.com/faf0/AES.jl/blob/master/src/aes-code.jl> and show how easy to port? Take <https://github.com/mkfryatt/BinaryECC> and show how easy to port?

Chapter 6

Summary and Conclusions

6.1 Summary

6.2 Future Work

SASCA = completely generic analysis on crypto algos? Infer constraints.
Possible but hard?

Bibliography

- [1] Northeastern University. Tescase dataset. <https://chest.coe.neu.edu/>.
- [2] Maryland National Security Agency, Fort George G. Meade. Nacsim 5000: Tempest fundamentals. Partially declassified transcript: <http://cryptome.org/nacsim-5000.htm>, 1982.
- [3] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
- [4] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Joye and Quisquater [26], pages 16–29.
- [5] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.
- [6] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.
- [7] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.
- [8] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag, 2002.

- [9] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. *IACR Cryptol. ePrint Arch.*, 2013:404, 2013.
- [10] David Brumley and Dan Boneh. Remote timing attacks are practical. In *Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003*. USENIX Association, 2003.
- [11] The OpenSSL Project. OpenSSL: The open source toolkit for SSL/TLS. www.openssl.org, April 2003.
- [12] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
- [13] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.
- [14] Stefan Mangard. A simple power-analysis (SPA) attack on implementations of the AES key expansion. In Pil Joong Lee and Chae Hoon Lim, editors, *Information Security and Cryptology - ICISC 2002, 5th International Conference Seoul, Korea, November 28-29, 2002, Revised Papers*, volume 2587 of *Lecture Notes in Computer Science*, pages 343–358. Springer, 2002.
- [15] Kai Schramm, Thomas J. Wollinger, and Christof Paar. A new class of collision attacks and its application to DES. In Thomas Johansson, editor, *Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003, Revised Papers*, volume 2887 of *Lecture Notes in Computer Science*, pages 206–222. Springer, 2003.
- [16] Kai Schramm, Gregor Leander, Patrick Felke, and Christof Paar. A collision-attack on AES: combining side channel- and differential-attack. In Joye and Quisquater [26], pages 163–175.

- [17] Andrey Bogdanov, Ilya Kizhvatov, and Andrei Pyshkin. Algebraic methods in side-channel collision attacks and practical collision detection. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *Progress in Cryptology - INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008. Proceedings*, volume 5365 of *Lecture Notes in Computer Science*, pages 251–265. Springer, 2008.
- [18] Mathieu Renauld and François-Xavier Standaert. Algebraic side-channel attacks. *IACR Cryptol. ePrint Arch.*, 2009:279, 2009.
- [19] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. *IACR Cryptol. ePrint Arch.*, 2014:410, 2014.
- [20] Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.
- [21] Bruce Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Twofish: A 128bit block cipher. 01 1998.
- [22] Thomas S. Messerges. Securing the AES finalists against power analysis attacks. In Bruce Schneier, editor, *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, volume 1978 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 2000.
- [23] Louis Goubin. A sound method for switching between boolean and arithmetic masking. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 3–15. Springer, 2001.
- [24] Thomas S. Messerges. Using second-order power analysis to attack DPA resistant software. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings*, volume 1965 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2000.

- [25] Kai Schramm and Christof Paar. Higher order masking of the AES. In David Pointcheval, editor, *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, volume 3860 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2006.
- [26] Marc Joye and Jean-Jacques Quisquater, editors. *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*. Springer, 2004.