Master Thesis Computing Science

Radboud University Nijmegen

Side-Channel Analysis of Keccak and Ascon

Author:
Niels Samwel

Supervisor:
Prof. dr. Joan Daemen

Second Assessor:
Dr. Lejla Batina

Daily Supervisor:
MSc. Kostas Papagiannopoulos

August 31, 2016

Abstract

This thesis is about side-channel analysis of the SHA-3 competition winner Keccak and a similar algorithm Ascon. During the operation of such an algorithm on a device information will leak in many different ways. In this thesis we only look at the information that is leaked by the power consumption of a device. This leakage can be exploited with a technique called DPA. With DPA one tries to obtain a small part of secret data, usually the secret key using a many power traces with random input messages and the key in each trace. We verified a theoretical attack with experiments on an actual chip. We also compared an attack on MAC-Keccak implemented on an FPGA to an ASIC. For Ascon which was implemented on an FPGA we crafted a similar attack as Keccak. With DPA being a very powerful tool for an attacker hardware needs to be designed with countermeasures against this. Threshold implementations are such countermeasures, we tried to attack an threshold implementation on an FPGA of Ascon to see if it was possible with a feasible amount of power traces. Finally, we look a the noise components in a power trace and how much effect the effect of electrical noise is with Ascon.

Acknowledgments

I would like to thank my supervisors for their continuous support and feedback throughout the semester. Their supervision kept me going with new ideas and improvements. Next I would like to thank Benedikt Gierlichs and Ingrid Verbauwhede from COSIC from KU Leuven for supplying us with the board with the SHA-3 chip. Without this much of the research would not have been possible. Finally I would like to thank my parents for giving me the opportunity to move to Nijmegen and focus on my education. Without them I would not have been able to dedicate as much time to my study as I did.

Contents

List of Figures

Chapter 1

Introduction

DPA is a statistical attack on the power consumption of the implementation of a cipher on the data it processes. The statistical attack correlates the power consumption with intermediate values computed by the attacker. The register that is attacked is called a sensitive variable and can be part of the secret key. The intermediate value for that register depends on part of the secret key and part of the variable input. Because the power consumption depends on the values in the sensitive variable which depends on the input and the key an attacker can exploit this. The attacker makes an hypothesis for part of the key and computes the intermediate values for different hypotheses for the sensitive variable. To distinguish between the correct and incorrect hypotheses a distinguisher is used. One is the correlation coefficient and is explained in chapter 6 [22]. Another selection function is distance of means. Many power traces are often necessary for the selection function to be able to distinguish between the correct and incorrect sub key candidates. Keccak is a cryptographic sponge function [4] which among other things is capable of hashing and symmetric encryption. It has an internal state on which several rounds of the permutation function, also called a round function is applied to mix up the input combined with the internal state. For the security of the algorithm it is crucial that the internal state remains secret. Keccak is a flexible sponge of seven permutations that ranges from an internal state of 25 to 1600 bits. A state size up to 200 bits is lightweight and the 1600 bit state is used when high performance is required. As the size of the internal state can be changed from a lightweight cipher to a large internal state. The security of Keccak is proportional to the capacity which can easily be increased by increasing by increasing the size of the internal state.

Ascon is an authenticated encryption algorithm with a structure based on a duplex which is based on a cryptographic sponge function. Because of this it has similar properties as Keccak. Ascon is a candidate in the CAESAR competition for a new authenticated encryption algorithm. The algorithm also uses a variant of the S-box of Keccak which makes the leakage model very similar.
A threshold implementation can be used to protect against DPA, it splits a variable up in different shares in such a way that knowing bits from one share can not help an attacker to determine bits of the original variable.

DPA is done frequently especially because it is a very powerful attack. DPA of protected and unprotected Keccak has been studied theoretically and verified with simulations in [6]. However, it is interesting to see whether the theory hold when conducting the attack on an actual ASIC.

With Keccak as the SHA-3 standard and since the cryptographic primitive is capable of different applications like keyed modes for encryption, MAC computation and authenticated encryption. Research has been done on the resistance against side-channel analysis. Two authenticated encryption algorithms using Keccak are Keyak [16] and Ketje [7]. The previously mentioned simulation of unprotected Keccak resulted in a theoretical success rate to obtain the correct key candidate from as a function of the number of traces. This resulted in the following research question: Does DPA on Keccak hardware implementations follow the theoretical success rate?

The round function of Keccak consists of several steps which can be classified in a linear and a non-linear part. As both parts may leak information during operation of a device it may be interesting to determine how the success rate differs and which part is more vulnerable to side-channel analysis, in particular DPA. An attack on the linear part of Keccak was already done on an implementation on an FPGA by [20]. We would like to see the differences compared to an attack on an ASIC instead of an FPGA with a similar architecture where one round of Keccak-f is applied each clock cycle. This resulted in the following research question: How does the success rate of an attack on the linear and non-linear part of Keccak-f vary between the ASIC and FPGA implementation?

Since Ascon uses a variant of the S-box of Keccak DPA on Ascon should be similar to DPA on Keccak with an attack on the non-linear part of both

algorithms. This resulted in the following research question: Can the same techniques used on Keccak be used on Ascon for DPA on the non-linear part of both algorithms?

For Ascon there is no ASIC available which is why the we use an implementation on an FPGA. On any platform there will be electrical noise generated by the circuit. This electrical noise is assumed to be Gaussian white noise and can be reduced by averaging the power traces with an equal input. With this we hope to be able to determine how large the electrical noise component is from the total amount of noise and how it affect the success rate of the attack on Ascon. This resulted in the following research question: How does averaging affect the success rate of an attack on Ascon and what does this say about the contributions of the algorithmic noise to the total noise?

In [6] a simulation was done on a threshold implementation for Keccak on a fully parallel implementation also known as a high speed core which required a huge amount of traces compared to an attack on an unprotected implementation. We try to attack a threshold implementation of Ascon on an FPGA with a feasible amount of traces. This resulted in the following research question: How feasible is an attack on a threshold implementation of a scaled-down version of Ascon?

Chapter 2

Background Information

2.1 Encryption Algorithms

With today's usage of the Internet where it is common to communicate with the bank, governmental institutions, companies or to have communication between people there is a huge need to keep these communications secret. To accomplish this encryption algorithms are used. There are two types of cryptography, asymmetric and symmetric cryptography. Asymmetric cryptography is used e.g. in key establishment in symmetric cryptography and digital signatures, symmetric cryptography is used e.g. in encryption and authentication of data. In this thesis we only focus on symmetric cryptographic algorithms.

2.1.1 Symmetric Cryptography

Symmetric cryptographic algorithms are used to encrypt or authenticate data for communication over an insecure channel. Both parties use the same key, more information on key management can be found in [2]. An important property of symmetric encryption is that a message M is encrypted with a secret key K the decryption of that results in M, $M = \mathrm{dec}(\mathrm{enc}(M, K), K)$. Another property is when a message is encrypted with a key $C = \mathrm{enc}(M, K)$ the resulting cipher text C can not be linked with the original message M.

Over the years many different symmetric encryption algorithms have been published and used. But with the computational power increasing all the

time many algorithms like several instances of DES, RC2 and many others stopped being secure and should not be used anymore. The latest symmetric encryption standard is called AES, Advanced Encryption Standard [11] which is a block cipher. Some other used block ciphers are certain instances of DES, PRESENT because it is very compact, and Blowfish. Block ciphers need to be used in a mode to be able to encrypt data. Some commonly used modes for encryption are ECB, CBC and CTR. Stream ciphers are also used for encryption, some commonly used stream ciphers are AES in CTR mode and ChaCha.

2.2 Hashing

Hashing is used to verify the integrity of data by applying the hash function on the data and comparing the output of the function to a value which is the correct output value for that data. If the values match the data is equal, if there is no match the data is not equal and something may have gone wrong during the transmission of the data. A hash function is a deterministic one-way function meaning that if we compute a hash function for a certain value X the result $H(X)$ will always be the same. It maps data of arbitrary size to an output of a fixed size. Hash functions should be collision resistant, meaning $X \neq Y, H(X) = H(Y)$ and have high pre-image resistance, meaning the should be hard to compute X from $H(X)$.

2.2.1 Message Authentication Codes

A message authentication code is a hash of a message combined with a secret key which allows a verifier to verify the integrity of the message. The message can be of arbitrary size while the size of the key depends on what algorithm in used. Unlike a normal hash which everyone can verify the message only the parties who hold the key are able to do this.

2.3 SHA-3

SHA-3 (Secure Hashing Algorithm) is a new hashing standard by the National Institute of Standards and Technology (NIST) for which there was a competition which ran over several years. The purpose of the competition

was to establish a new secure standard for cryptographic hashing after SHA-0, SHA-1 and SHA-2. The competition consisted of three rounds and for each round there was a conference where the candidates were presented and discussed. After each round additional criteria was discussed to reduce the amount of candidates each round. Initially there were 64 candidates of of which 51 completed the minimal submission requirements for the first round of the selection process. After the second round five finalists were selected for the final round. The five finalists are BLAKE [3], Grøstl [15], JH [28], Skein [14] and the winner Keccak [5] which won on 2 October 2012.

The hashing algorithm BLAKE is built on previously built and studied components to provide security. The algorithm iterates through several rounds depending on the size to compress the input. The compression function is based on a stream cipher ChaCha which has a high performance and can be parallelized.
Like BLAKE Grøstl also is an iterated hash function with a compression function. This algorithm uses components very different from the previous SHA functions. It borrows components from AES so it has similar countermeasures to side-channel attacks as AES.
Similar to the previous candidates JH also uses previously built components in its design. For JH the designers generalized the design of AES to design large block ciphers. For hashing a constant key is used. Since is it is similar to AES, again it has similar countermeasures to side-channel attacks.
Skein uses the block cipher Threefish as its core. Using Threefish in a unique block iteration mode enables the algorithm to compress input of an arbitrary length to a fixed length output. Since Skein uses an already established block cipher it protects against similar side channel attacks as the block cipher.
The winner of the SHA-3 competition Keccak uses a cryptographic sponge function as its core which has an internal state where a round function is computed on. Keccak can be used for many applications like hashing, symmetric encryption and creating MAC's.

For the third and final round of the SHA-3 competition an ASIC was produced by Virginia Tech [17] with the five finalists on it. The chip was produced using standard cell CMOS technology. For the final round of the competition it was important that each candidate could thoroughly be tested for performance on an ASIC instead of an FPGA as a high performing candidate would score better during the evaluation by the NIST. The chip contains the five finalists but since the power is gated to the selected candidate there is no noise from the other candidates. Later in this thesis we describe an attack

using traces generated by this chip.

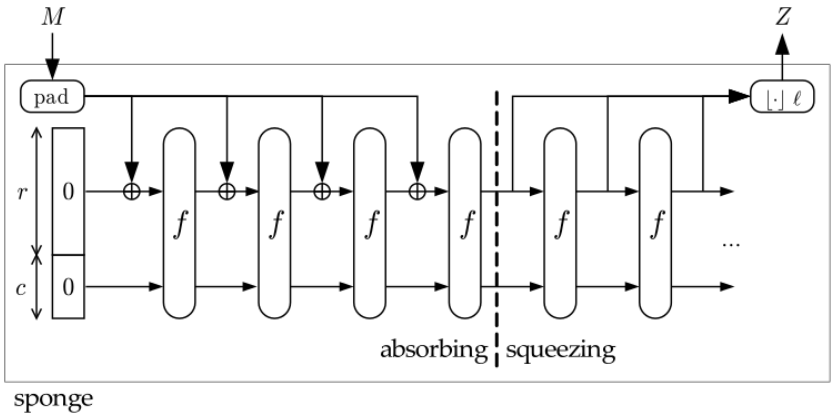
2.4 Cryptographic Sponge Functions

A cryptographic sponge function [4] is a construction that can be used for different applications. A sponge function has arbitrary input and output length. It uses a fixed amount of memory to store its internal state. And a permutation function that operates on a fixed amount of bits b called the width. The state consists of $b = r + c$ where r is the bit-rate and c is the capacity. A sponge function consist of two phases, an absorbing phase and a squeezing phase as can be seen in Figure 2.1a. Before the absorbing phase the input message M is padded and split into blocks of length r and the state is initialized with 0. During the absorbing phase each input block of length r is XOR'ed into the first r bits of the state. After a block is XOR'ed into the state a certain amount of rounds of the permutation function is computed on the state. During the squeezing phase the first r bits of the state are returned as a block of output. After each block again a certain amount of rounds of the permutation function is computed on the state. A user can choose an arbitrary amount of output blocks.
Figure 2.1b shows a duplex function which is a closely related function to the sponge function. Before the function starts the input is padded, split into blocks of length r and the state is initialized on 0. Each input block is XOR'ed into the state, a certain amount of rounds of the permutation function is computed on the state and the first r bits of the state are returned as output.
These two constructions can be used for different applications like hashing, symmetric encryption and creating MAC's. It is nice that it is possible to use a single permutation function for many different applications.

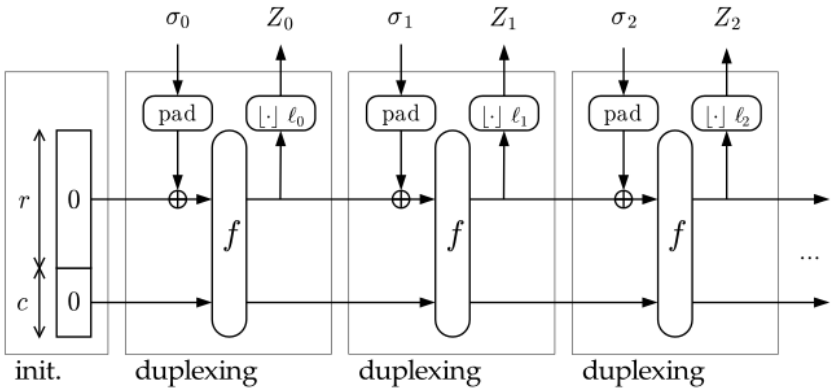
2.5 Keccak

The following text is directly from the Keccak reference document [5].

Keccak is a family of sponge functions [4] that use as a building block a permutation from a set of 7 permutations. In this chapter, we introduce our conventions and notation, specify the 7 permutations underlying Keccak



(a) Sponge



(b) Duplex

Figure 2.1: A visualization of both the sponge and duplex construction [4]

and the KECCAK sponge functions. We also give conventions for naming parts of the KECCAK state.

2.5.1 Conventions and notation

We denote the absolute value of a real number x is denoted by $|x|$.

2.5.1.1 Bitstrings

We denote the length in bits of a bitstring M by $|M|$. A bitstring M can be considered as a sequence of blocks of some fixed length x, where the last block may be shorter. The number of blocks of M is denoted by $|M|_x$. The blocks of M are denoted by M_i and the index ranges from 0 to $|M|_x - 1$.

We denote the set of all bitstrings including the empty string by $\mathbb{Z}_2^*$ and excluding the empty string by $\mathbb{Z}_2^+$. The set of infinite-length bitstrings is denoted by $\mathbb{Z}_2^\infty$.

2.5.2 Padding rules

For the padding rule we use the following notation: the padding of a message M to a sequence of x-bit blocks is denoted by $M||\mathsf{pad}[x](|M|)$. This notation highlights that we only consider padding rules that append a bitstring that is fully determined by the bitlength of M and the block length x. We may omit $[x]$, $(|M|)$ or both if their value is clear from the context.

KECCAK makes use of the *multi-rate* padding.

Definition 1. *Multi-rate padding, denoted by* $\mathsf{pad10^*1}$*, appends a single bit 1 followed by the minimum number of bits 0 followed by a single bit 1 such that the length of the result is a multiple of the block length.*

Multi-rate padding appends at least 2 bits and at most the number of bits in a block plus one.

2.5.3 The Keccak-f permutations

There are 7 KECCAK-f permutations, indicated by KECCAK-$f[b]$, where $b = 25 \times 2^\ell$ and ℓ ranges from 0 to 6. KECCAK-$f[b]$ is a permutation over

$\mathbb{Z}_2^b$, where the bits of s are numbered from 0 to $b-1$. We call b the width of the permutation.

The permutation KECCAK-$f[b]$ is described as a sequence of operations on a state a that is a three-dimensional array of elements of GF(2), namely $a(5,5,w)$, with $w = 2^\ell$. The expression $a(x,y,z)$ with $x,y \in \mathbb{Z}_5$ and $z \in \mathbb{Z}_w$, denotes the bit in position (x,y,z). It follows that indexing starts from zero. The mapping between the bits of s and those of a is $s(w(5y+x)+z) = a(x,y,z)$. Expressions in the x and y coordinates should be taken modulo 5 and expressions in the z coordinate modulo w. We may sometimes omit the (z) index, both the (y,z) indices or all three indices, implying that the statement is valid for all values of the omitted indices.

KECCAK-$f[b]$ is an iterated permutation, consisting of a sequence of n_r rounds R, indexed with i_r from 0 to $n_\mathrm{r}-1$. A round consists of five steps:

$$\mathrm{R} = \iota \circ \chi \circ \pi \circ \rho \circ \theta, \text{ with}$$

$$\begin{aligned}
\theta:\ & a(x,y,z) \leftarrow a(x,y,z) + \sum_{y'=0}^{4} a(x-1,y',z) + \sum_{y'=0}^{4} a(x+1,y',z-1),\\
\rho:\ & a(x,y,z) \leftarrow a(x,y,z-(t+1)(t+2)/2),\\
& \quad \text{with } t \text{ satisfying } 0 \le t < 24 \text{ and } \begin{pmatrix}0&1\\2&3\end{pmatrix}^t \begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}x\\y\end{pmatrix} \text{ in } \mathrm{GF}(5)^{2\times 2},\\
& \quad \text{or } t = -1 \text{ if } x = y = 0,\\
\pi:\ & a(x,y) \leftarrow a(x',y'), \text{ with } \begin{pmatrix}x\\y\end{pmatrix} = \begin{pmatrix}0&1\\2&3\end{pmatrix} \begin{pmatrix}x'\\y'\end{pmatrix},\\
\chi:\ & a(x) \leftarrow a(x) + (a(x+1)+1)a(x+2),\\
\iota:\ & a \leftarrow a + \mathrm{RC}(i_\mathrm{r}).
\end{aligned}$$

The additions and multiplications between the terms are in GF(2). With the exception of the value of the round constants $\mathrm{RC}(i_\mathrm{r})$, these rounds are identical. The round constants are given by (with the first index denoting the round number)

$$\mathrm{RC}[i_\mathrm{r}](0,0,2^j-1) = \mathrm{rc}[j+7i_\mathrm{r}] \text{ for all } 0 \le j \le \ell,$$

and all other values of $\mathrm{RC}[i_\mathrm{r}](x,y,z)$ are zero. The values $\mathrm{rc}[t] \in \mathrm{GF}(2)$ are defined as the output of a binary linear feedback shift register (LFSR):

$$\mathrm{rc}[t] = \left(x^t \bmod x^8 + x^6 + x^5 + x^4 + 1\right) \bmod x \text{ in } \mathrm{GF}(2)[x].$$

The number of rounds n_r is determined by the width of the permutation, namely,

$$n_r = 12 + 2\ell.$$

2.5.4 The sponge construction

The sponge construction [4] builds a function SPONGE$[f, \mathsf{pad}, r]$ with variable-length input and arbitrary output length using a fixed-length permutation (or transformation) f, a padding rule "pad" and a parameter *bitrate* r. The permutation f operates on a fixed number of bits, the *width* b. The value $c = b - r$ is called the *capacity*.

For the padding rule we use the following notation: the padding of a message M to a sequence of x-bit blocks is denoted by $M||\mathsf{pad}[x](|M|)$, where $|M|$ is the length of M in bits.

Algorithm 1 The sponge construction SPONGE$[f, \mathsf{pad}, r]$

Require: $r < b$

Interface: $Z = \mathsf{sponge}(M, \ell)$ with $M \in \mathbb{Z}_2^*$, integer $\ell > 0$ and $Z \in \mathbb{Z}_2^\ell$
$P = M||\mathsf{pad}[r](|M|)$
$s = 0^b$
for $i = 0$ to $|P|_r - 1$ **do**
  $s = s \oplus (P_i||0^{b-r})$
  $s = f(s)$
end for
$Z = \lfloor s \rfloor_r$
while $|Z|_r r < \ell$ **do**
  $s = f(s)$
  $Z = Z||\lfloor s \rfloor_r$
end while
return $\lfloor Z \rfloor_\ell$

Initially, the state has value 0^b, called the *root state*. The root state has a fixed value and shall never be considered as an input. This is crucial for the security of the sponge construction.

2.5.5 The Keccak sponge functions

We define the sponge function denoted by KECCAK$[r, c]$ by applying the sponge construction as specified in Algorithm 1 with KECCAK-$f[r + c]$, multi-rate padding and the bitrate r.

$$\text{KECCAK}[r, c] \triangleq \text{SPONGE}[\text{KECCAK-}f[r + c], \text{pad10}^*1, r].$$

This specifies KECCAK$[r, c]$ for any combination of $r > 0$ and c such that $r + c$ is a width supported by the KECCAK-f permutations.

The default value for r is $1600 - c$ and the default value for c is 576:

$$\begin{aligned} \text{KECCAK}[c] &\triangleq \text{KECCAK}[r = 1600 - c, c], \\ \text{KECCAK}[] &\triangleq \text{KECCAK}[c = 576]. \end{aligned}$$

2.5.6 Security claim for the Keccak sponge functions

For each of the supported parameter values, we make a *flat sponge claim* [4, Section "The flat sponge claim"].

Claim 1. *The expected success probability of any attack against* KECCAK$[r, c]$ *with a workload equivalent to N calls to* KECCAK-$f[r + c]$ *or its inverse shall be smaller than or equal to that for a random oracle plus*

$$1 - \exp\left(-N(N+1)2^{-(c+1)}\right).$$

We exclude here weaknesses due to the mere fact that KECCAK-$f[r + c]$ *can be described compactly and can be efficiently executed, e.g., the so-called random oracle implementation impossibility [4, Section "The impossibility of implementing a random oracle"].*

Note that the claimed capacity is equal to the capacity used by the sponge construction.

2.5.7 Parts of the state

In this subsection, we define names of parts of the KECCAK-f state, as illustrated in Figure 2.2. This naming convention may help use a common terminology when analyzing or describing properties of KECCAK-f.

The one-dimensional parts are:

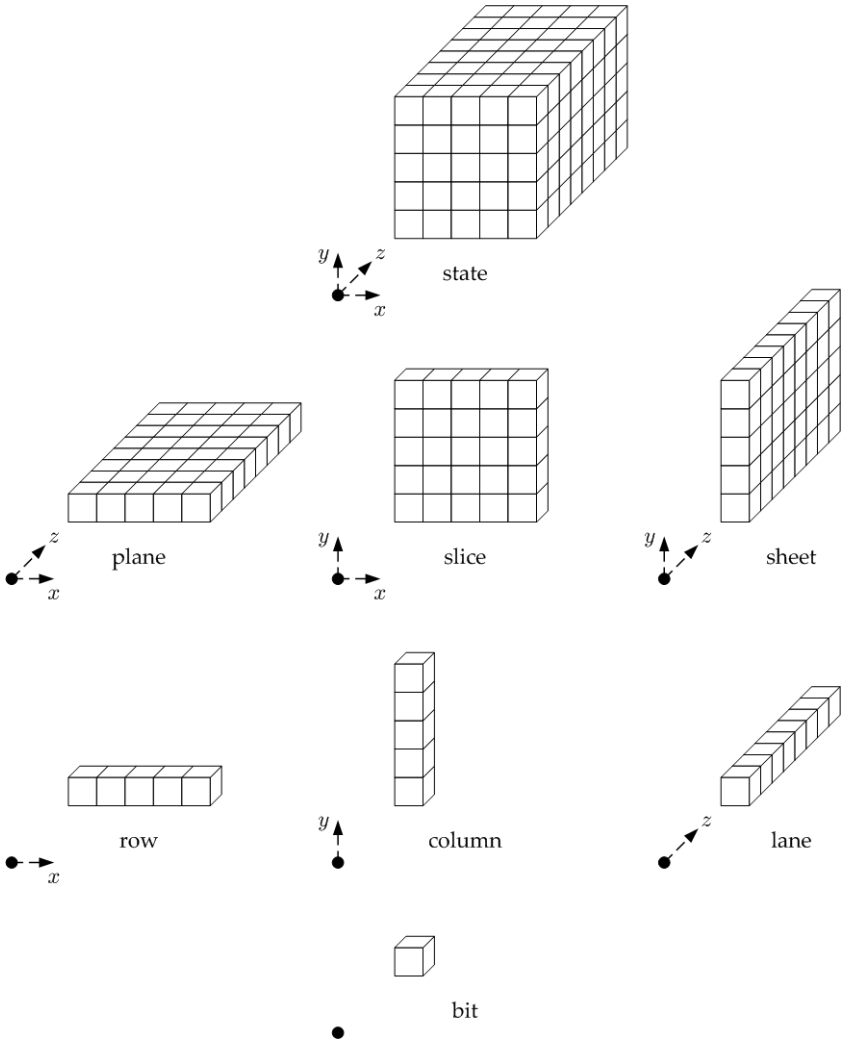


Figure 2.2: Naming conventions for parts of the KECCAK-f state

- A *row* is a set of 5 bits with constant y and z coordinates.
- A *column* is a set of 5 bits with constant x and z coordinates.
- A *lane* is a set of w bits with constant x and y coordinates.

The two-dimensional parts are:

- A *sheet* is a set of $5w$ bits with constant x coordinate.
- A *plane* is a set of $5w$ bits with constant y coordinate.
- A *slice* is a set of 25 bits with constant z coordinate.

2.6 Ascon

Ascon [12] is a cipher which is a candidate in the Competition for Authenticated Encryption: Security, Applicability, and Robustness or in short CAESAR [1]. To be considered a candidate for the competition a cipher must follow certain functional requirements. It must have five inputs which consist of a variable length plain-text, a variable length associated-data, a fixed length secret message number which may be zero, a fixed length public message number and a fixed length key. The security requirements of those inputs are that they must all keep integrity. The plaintext and secret message number must keep confidentiality. If either message number is used more than once with a key the cipher may lose its security. A submission is required to a have list of recommended parameters like the key length. In this thesis when Ascon is mentioned, Ascon-128 is meant, where are 128-bit key and nonce are used and the blocksize of the input data is 64-bit
The authenticated encryption algorithm Ascon uses a cryptographic duplex construction for encryption and a sponge construction for the absorption of the associated data. Figure 2.3 shows the process of Ascon. For encryption it takes four inputs, plaintext P_i, associated data A_i, nonce N and a key K. The block cipher produces two outputs the ciphertext C_i and a tag T. p^* is the permutation function, in this case $a = 12$ and $b = 8$ which are the number of rounds given by the recommended parameters for the algorithm. The nonce is the public message and the secret message has length zero. The tag is used during decryption to authenticate the ciphertext. During decryption the algorithm uses the tag as an input and produces the plaintext as output and the result of the validation of the tag.

The internal state of the algorithm consists of 320 bits which is split up into

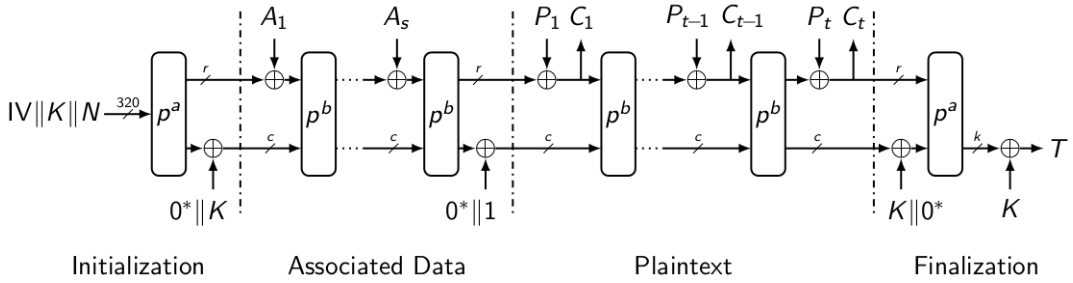


Figure 2.3: Ascon encryption [12]

five registers of 64 bits, named x_0 to x_4. During the initialization of the algorithm register x_0 of the internal state is initialized with a 64-bit initialization vector which is a constant listed in the recommended parameters. Register x_1 is initialized with the most significant 64-bits of the key, register x_2 with the least significant part of the key. And registers x_3 and x_4 are initialized with the most significant and the least significant half of the nonce respectively. Also the associated data and plaintext are padded to have a length of a multiple of 64 bits. When the internal state is initialized twelve rounds of the round function are applied. Next, the associated data is absorbed into the state. For each block six rounds of the round function are applied. After each block of plaintext is duplexed into the state, again six rounds of the round function are applied, except for the final block. During the finalization of the part of the algorithm another twelve rounds of the round function are applied and a tag is produced which is used to authenticate the data during decryption.

The round or permutation function used in Ascon consists of three parts. The first part is the addition of a round constant to register x_2 on the least significant bits of the register which can be computed as follows for each round i.

$$RC_i = \texttt{0xF} - i || \texttt{0x0} + i$$

Where $||$ means the concatenation of the two parts. The second part is a non-linear five-bit S-box with algebraic degree two, which takes one bit from each register shown in Figure 2.4 and replaces it with the output of the S-box. The S-box is a variant of the S-box used in Keccak and is shown in Figure 2.5. Table 2.1 shows the results of the 5-bit S-box.

The final part is the linear diffusion layer where each register is twice rotated and XOR'ed with itself which is called $\Sigma_i(x_i)$. Below are the expressions of

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S(x)$	4	11	31	20	26	21	9	2	27	5	8	18	29	3	6	28
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$S(x)$	30	19	7	14	0	13	17	24	16	12	1	25	22	10	15	23

Table 2.1: The 5-bit S-box of Ascon

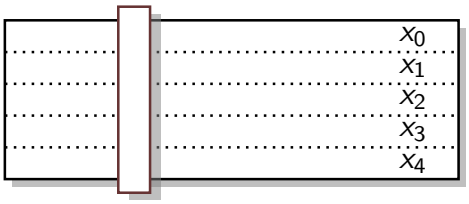


Figure 2.4: Input of the S-box of Ascon [12]

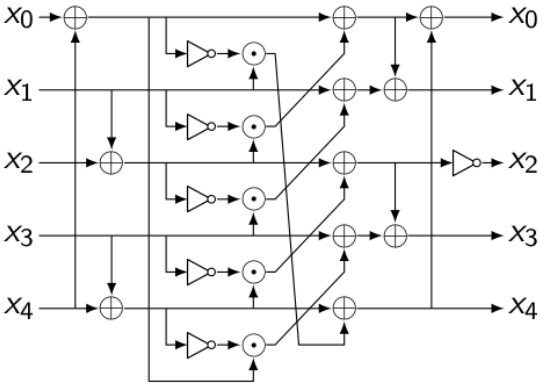


Figure 2.5: The S-box of Ascon [12]

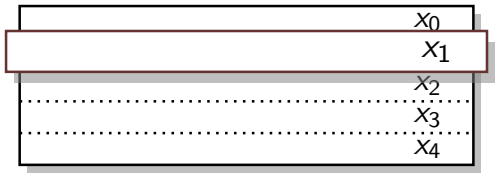


Figure 2.6: Input of the linear diffusion layer of Ascon [12]

the linear diffusion layer.

$$\begin{aligned}
\Sigma_0(x_0) &= x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28) \\
\Sigma_1(x_1) &= x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39) \\
\Sigma_2(x_2) &= x_2 \oplus (x_2 \ggg \ 1) \oplus (x_2 \ggg \ 6) \\
\Sigma_3(x_3) &= x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17) \\
\Sigma_4(x_4) &= x_4 \oplus (x_4 \ggg \ 7) \oplus (x_4 \ggg 41)
\end{aligned}$$

Chapter 3

Side-Channel Analysis

3.1 Introduction

When a processor computes a cryptographic function on some data it is unavoidable that information leaks. With side-channel analysis an attacker tries to exploit this leakage and whether that is possible depends on many different factors. Side-channels that can leak information are for instance time, electromagnetic fields, and power consumption. In this thesis we only look at the power consumption, electromagnetic fields is similar to power consumption but requires a whole different setup and models and time leakage requires completely different techniques to exploit the leakage which is why it is not in the scope if this thesis. To measure the power consumption of a device a small resistor is put in series with the circuit and connected to the ground. The voltage difference over the resistor is the power consumption. Oscilloscopes can sample this difference at very high sampling rates. An example of a power trace is shown in Figure 3.1.

To exploit the information leakage of the power consumption of a cryptographic function an attacker has two options called simple power analysis and differential power analysis. With SPA an attacker only requires a few power traces and looks for patterns in the trace that can be related to key specific actions in certain cryptographic function. If this function is not protected against SPA an attacker can often obtain the key that was used by just looking at a plot of the power trace. In-depth knowledge of the device is required for this attack as the attacker needs to know exactly what happens when in the power trace.

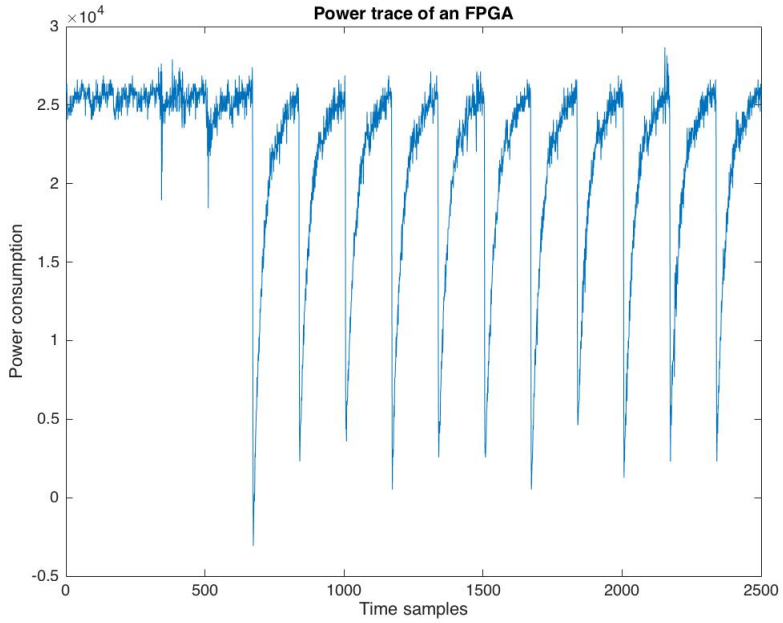


Figure 3.1: Example of a power trace

With DPA less in-depth knowledge of the device is required as it is more of a statistical approach which requires many power traces compared to SPA where the same key is used. The distinguisher requires a power consumption model to compute the hypothetical power consumption of the device for a certain input at a specific time during the operation. The computed hypothetical power consumption can then be related to the actual power consumption to obtain small parts of the key that was used during the operation of the device.

Since it is possible to obtain the key that is used during a cryptographic operation with few resources. A designer of a cryptographic device needs to take this account and apply countermeasures where required. In this thesis we only look at an implementation protected against DPA.

3.2 Simple Power Analysis

In [18, Section 2] and [22, Chapter 5] SPA is explained. With SPA an attacker tries to reveal the key from only a small amount of power traces, sometimes

even a single power trace. When there are more traces available with the same plaintext as input the attacker can try to reduce the noise by computing the mean of the traces. Whether an attacker has just one or multiple traces, the attack is the same. The attacker tries to find patterns in the recorded trace which are caused by key dependent operations. For SPA to work the key must have a significant impact on the power consumption. In-depth knowledge about the device is required to be able to distinguish between the key dependent operations in the power trace.

3.3 Differential Power Analysis

3.3.1 General Description

DPA [22, Chapter 6] is used to reveal the secret key of cryptographic devices based on a large number of power traces that have been recorded while the devices encrypt or decrypt different blocks of data. Compared to SPA DPA has the advantage that no in depth knowledge of the device is required. It is often sufficient to know what algorithm is executed on the device. Another difference between the two attacks is the way the traces are analyzed. In SPA an attacker tries to find patterns in the power consumption along the time axis of the trace. On the other hand in DPA attacks the time axis is not important. With DPA an attacker analyzes how power consumption at fixed moments of time depends on the processed data. DPA focuses on the data dependency of a large amount of power traces. More in detail, a DPA attack consists of five general steps described below.

In step 1 an attacker choses an intermediate result of the cryptographic algorithm that is executed by the device. This result needs to be a function $f(d, k)$ where d is a known non-constant data value and k is a small part of the key. These intermediate results can be used to reveal k. Usually d is the plaintext or the ciphertext.

In step 2 an attacker measures the power consumption of the cryptographic device while is encrypts or decrypts D different data blocks. For each of these runs the attacker needs to store the corresponding value d which is required in the calculation of the intermediate results from step 1. These known input values are stored as a vector $\boldsymbol{d} = (d_1, ..., d_D)'$, where d_i is the data values in the i_{th} encryption or decryption run.

During each run the power trace is recorded. The power trace that corresponds to data block d_i as $\boldsymbol{t}'_i = (t_{i,1}, ..., t_{i,T})$, where T is the length of the trace. Since the attacker stores a traces for each of the D data blocks, the traces can be written as a matrix $\boldsymbol{T}$ with with size $D \times T$. For the attack to work it is important that the traces are aligned, meaning that each column $\boldsymbol{t}_j$ of the matrix $\boldsymbol{T}$ needs to correspond to the same operation. In order to obtain aligned power traces, the oscilloscope needs to be triggered by a trigger signal so the oscilloscope starts recording at the exact same time for each run. In this thesis a trigger signal is always available from the attacked device.

In step 3 an attacker calculates hypothetical intermediate values for every possible choice of k. These possible choices are written as a vector $\boldsymbol{k} = (k_1, ..., k_K)$, where K is the total number of choices for k. With DPA we refer to the elements of this vector as key hypotheses or key candidates. Given data vector $\boldsymbol{d}$ and the key hyptheses $\boldsymbol{k}$ an attacker can calculate hypothetical intermediate values $f(d, k)$ for all D run and for all K key hypotheses. This results in a matrix $\boldsymbol{V}$ of size $D \times K$.

$$v_{i,j} = f(d_i, k_j) \;\; i = i, ..., D \;\; j = 1, ..., K$$

Column j of $\boldsymbol{V}$ contains the intermediate results based on key hypothesis k_j. One column of $\boldsymbol{V}$ contains all intermediate values for D runs of the device. Vector $\boldsymbol{k}$ contains all possible values for k so the key that was used by the device is in $\boldsymbol{k}$. We refer to the used key in the device as he correct key candidate or correct sub key candidate. The goal of the attack is to find which column of $\boldsymbol{V}$ has been processed by the device during the D runs. As soon as we know which column of $\boldsymbol{V}$ has been processed in the device we know the correct key hypothesis.

In step 4 an attacker maps the hypothetical intermediate values $\boldsymbol{V}$ to hypothetical power consumption values in matrix $\boldsymbol{H}$. This is done using a model. The quality of the model strongly depends on the knowledge of the device by the attacker. Two frequently used models are the Hamming weight and the Hamming distance model [10]. The Hamming weight is the number of bits that are one of the intermediate value. The Hamming distance is the number of bits that have flipped from the old state to the new state of the intermediate value.The result of the model and the intermediate value is stored in matrix $\boldsymbol{H}$.

In step 5 an attacker compares the hypothetical power consumption values

with the power traces. In this step each column h_i of matrix $\boldsymbol{H}$ is compared with each column t_j of matrix $\boldsymbol{T}$. This means that the attacker compares the hypothetical power consumption values of each key hypothesis with the recorded traces at every position. The result is stored in a matrix $\boldsymbol{R}$ of size $K \times T$, where each element $r_{i,j}$ contains the result of the comparison between columns h_i and t_j. The comparison is done based on distinguishers discussed later. All distinguishers have the property that a higher value for $r_{i,j}$ means a better match between columns h_i and t_j. To determine the correct key candidate an attacker simply looks at the highest value in $\boldsymbol{R}$. It happens in practice that all values in $\boldsymbol{R}$. are almost equal. In this case an attacker has usually not measured enough power traces. To resolve this an attacker simple acquires more traces and goes through the steps again.

3.3.2 Difference of Means

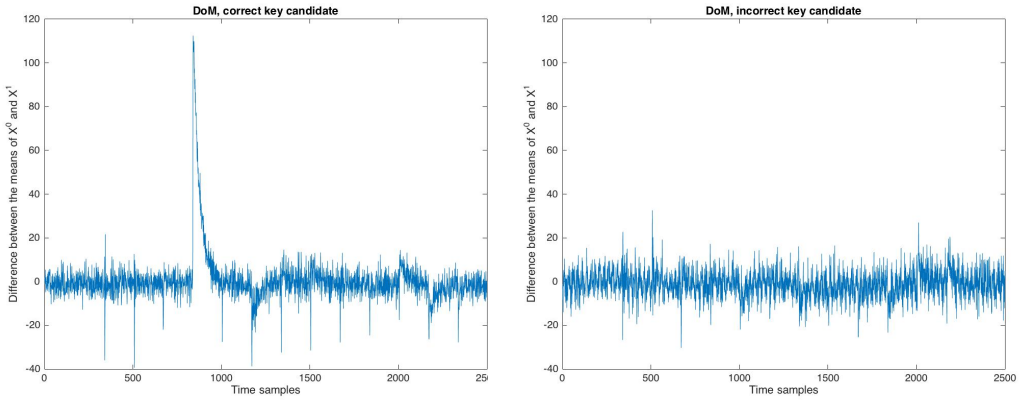
A commonly used distinguisher in DPA is difference of means or DoM. With this distinguisher an attacker splits the set of traces $\boldsymbol{T}$ up based on the hypothetical power consumption values in $\boldsymbol{H}$. Next the difference of means of those sets is subtracted and highest difference results in the correct key candidate.
For each key candidate the set of traces is split up based on the hypothetical power consumption values meaning that if $h_{i,j}$ is 0 trace t_i goes into one set and if $h_{i,j}$ is 1 trace t_i goes into another set. When this is done for all D values in $\boldsymbol{H}$ the two sets are subtracted from each other and stored in row r_i, all T time samples included. This is repeated for each key candidate. The row in $\boldsymbol{R}$ with the highest value is the correct key candidate. An example plot two rows of $\boldsymbol{R}$ containing a correct and incorrect candidate is shown in Figure 3.2.

3.3.3 Correlation Power Analysis

The correlation coefficient is the most common way to determine linear relationships between data. Therefore it is an excellent choice for DPA attacks. The correlation coefficient is defined as follows.

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$$

In DPA attacks, the correlation coefficient is used to determine the linear relationship between columns h_i and t_j for $i = 1, ..., K$ and $j = 1, ..., T$. This



(a) Correct key candidate (b) Incorrect key candidate

Figure 3.2: Example results for DoM

results in a matrix $\boldsymbol{R}$ of estimated correlation coefficients. We estimate each value $r_{i,j}$ based on the D elements of the columns h_i and t_j. We use $\bar{h_i}$ and $\bar{t_j}$ to denote the mean values of the columns h_i and t_j.

$$r_{i,j} = \frac{\sum_{d=1}^{D} (h_{d,i} - \bar{h_i}) \cdot (t_{d,j} - \bar{t_j})}{\sqrt{\sum_{d=1}^{D} (h_{d,i} - \bar{h_i})^2 \cdot (t_{d,j} - \bar{t_j})^2}}$$

The highest value $r_{i,j}$ denotes the correct key hypothesis.

3.3.4 Combining partitions of the correlation coefficient

When using many traces with many samples each the set of power traces may become very large. For instance, storing five hundred thousand traces requires approximately 7.5 GB of storage and RAM. If five million traces are required to successfully attack the implementation that would require 75 GB of memory. There is usually plenty of hard disk storage available but large amounts of RAM are not available in comparable amounts. Even if only the first cycle of each trace is stored which will reduce the required memory to approximately one tenth, but as more traces are required it still is unpractical. If the set is split up into several smaller partitions which can be stored in RAM for which the correlation coefficient is computed and the results can

be combined as if the correlation coefficient was computed on the whole set this could help.
Another case is for parallel computing as each partition can be used in parallel and the combination of the results can be done in $\log_2 n$ steps.

A method for this was described by Dunlap [13] and used by [9] for side-channel analysis. The formulas for this method are summarized below. For this the total set of power traces with their hypothetical power consumption values is split up in N partitions with each n elements.

$$R_{i,j} = \frac{\sum_{k=1}^{N} n_k \left(\sigma_{H_k}\sigma_{T_k}\rho(H_k, T_k) + \delta_{H_k}\delta_{T_k}\right)}{\sqrt{\sum_{k=1}^{N} n_k \left(\sigma_k^2 + \delta_{H_k}^2\right)} \cdot \sqrt{\sum_{k=1}^{N} n_k \left(\sigma_{T_k}^2 + \delta_{T_k}^2\right)}}$$
$$\delta_{X_i} = M_{X_i} - M_X$$

Where M denotes the mean of a set. If we use these formulas to incrementally add each set, $N = 2$ and the size of that partition increases each step by n. We also need to compute the new variance and mean of the combined set as follows for two sets with each m and n elements.

$$\sigma^2 = \frac{m(\sigma_m^2 + \delta_m^2) + n(\sigma_n^2 + \delta_n^2)}{m+n}$$
$$M = \frac{mM_m + nM_n}{m+n}$$

With these formulas it is possible to compute the correlation coefficient for a very large set of power traces without using a lot of memory.

3.3.5 Higher-order attacks

With DPA attacks that are widely used designers of implementations for cryptographic operations must protect their designs against such attacks. This can be accomplished on different levels in the algorithm, for instance in the protocol where the usage of a single key can be limited or in the in actual design of the implementation of the algorithm. There are several techniques for protecting the design on an implementation level. One is masking [27] where the data is masked and the algorithm is computed separately on the masked data and the mask. At the end of the computation the data is unmasked and can be read. The computations on the data and the mask can

be done in parallel. A specific type of masking is a threshold implementation where several shares are used to mask the data. This technique makes it harder to correlate the intermediate value with the power consumption as the power consumption is not directly related to that data. A more detailed explanation about threshold implementations is given in section 3.4.1.

To attack masked implementation previous attacks are not going to work. These attacks fall into the category of first-order attacks and to attack implementations with multiple shares higher-order attacks must be used [26]. If an implementation has two shares on which the computations on each share are computed in parallel a second order attack must be used. A second-order attack is similar to a difference of means attack, except for the distinguisher. Instead of the mean the variance is used which is the average distance to the mean of all the samples in a distribution. The variance of a distribution X is defined as follows.

$$Var(X) = E(X - \mu)^2$$

Where $E()$ is the expected value and μ is the mean of distribution X. Other names for the mean and the variance are respectively the first and second central moment.

When an implementation uses three shares a third-order attack must be used, this number increases with each share but since implementations with no more then three shares are studied in the thesis higher then third-order attacks are not explained. A third-order attack again is similar to difference of means except where the difference of the mean is used, the difference of the skewness is used. The skewness determines if a distribution is symmetrical on the mean or leans more to the left or the the right. The skewness of distribution X is defined as follows.

$$Skew(X) = \frac{E(X - \mu)^3}{\sigma^3}$$

From the previous formula the $E(X - \mu)^3$ is also called the third central moment and the skewness is the third standardized moment. The pattern that emerges can be used to compute any arbitrary statistical moment.

3.3.6 Computing the third standardized moment

Attacking a TI is not possible anymore with a first order attack like CPA or difference of means. To attack a TI with three shares we need to look at the

difference of the skewness. A problem with the formula of the skewness is that all the variables must be stored in memory. When only a small amount of traces is used that is fine but when attacking TI's it usually takes millions of traces which may not fit.

Using an algorithm to iteratively compute the skewness makes it possible to compute the skewness of a very large set while using a small and constant amount of RAM. To do this we first need to compute the third central moment to compute the third standardized moment or the skewness. It is possible to do this if a set is partitioned into two sets [25]. If the set is partitioned in two sets A and B, these formula's apply.

$$\begin{aligned}
\delta &= \mu_B - \mu_A \\
\mu &= \mu_A + \delta\frac{n_B}{n} \\
M_2 &= M_{2A} + M_{2B} + \delta^2\frac{n_A n_B}{n} \\
M_3 &= M_{3A} + M_{3B} + \delta^3\frac{n_A n_B(n_A - n_B)}{n^2} - 3\delta\frac{n_A M_{2B} - n_B M_{2A}}{n}
\end{aligned}$$

And finally the skewness.

$$\mathrm{S} = \sqrt{n}\frac{M_3}{M_2^{3/2}}$$

If we simplify those formulas where one set contains only a single element the formula's reduce to the following, allowing us to compute the third central moment and skewness iteratively.

$$\begin{aligned}
\delta &= x - \mu \\
\mu &= \mu + \frac{\delta}{n} \\
M_3 &= M_3 + \delta^3\frac{(n-1)(n-2)}{n^2} - 3\delta\frac{M_2}{n} \\
M_2 &= M_2 + \delta^2\frac{n-1}{n}
\end{aligned}$$

The four formulas must be computed such that each iteration a variable is added and the actual skewness is only computed in the end. Using these formula's makes it possible to compute the skewness of a large amount of traces stored on a hard disk without using a large amount of RAM. While collecting the traces it is also possible to compute the skewness without storing the traces on the hard disk.

3.4 Countermeasures

3.4.1 Threshold Implementations

The previous implementations can be attacked by using first-order DPA. A technique to protect against first-order DPA is a threshold implementation [23]. A threshold implementation or TI has three requirements: correctness, non-completeness and uniformity. Correctness means that the output of a function on each share must be equal to the output of that function used on the XOR of the shares. Non-completeness means that a function can not combine all shares, at least one must not be used.
All linear parts of a round function in a TI can be applied to each share separately and will be conform the requirements. If there is an addition of a round constant it should be added to a single share. The non-linear part is a different story. To satisfy the non-completeness requirement the non-linear function can only use two shares in case of a three share implementation. If there are three shares A, B and C the non-linear part will look like this.

$$A' = f(B, C)$$
$$B' = f(A, C)$$
$$C' = f(A, B)$$

Not all non-linear functions keep the uniformity of their input shares, this is a problem because it has been proven that if the shares are uniformly distributed, correct and non-complete the implementation is secure against first-order DPA [24]. It is possible to preserve the uniformity with by adding new random bits since both Ascon and Keccak do not a have uniform round function this is required. It is possible to reduce the amount of new random bits that are required each round [8] for Keccak. Since Ascon uses a variation of the S-box of Keccak this works for Ascon as well.
The used threshold implementations from Ascon and Keccak both have three shares and one round of the round function is computed in parallel each clock cycle.

Chapter 4

Related Work

4.1 MAC-Keccak

Research has been done on a specific version of Keccak, called MAC-Keccak [20] where Keccak-f[1600] was used. MAC stands for message authentication code and is used to verify the integrity of a message where both parties use the same key. In this case features of the sponge construction are not used as the key and the message are put in the initial state on which 24 rounds of the permutation function are performed. In this paper they attacked the θ part of the round function. To do this they attacked 32-bits of a sheet which contained information about 16 key bits. With a success rate of almost 100% using thirty thousand traces they were able to extract 8 key bits of information out of this since part of those bits are XOR'ed. When the attack is performed on the whole state the key could be extracted by solving the system of linear equations. In the paper they used the attack described below the obtain the whole key from the previous result.
Another attack that was performed was on part of the θ function where the parity between the planes is computed. This time they attacked 8 bits from the result which contained information about 8 key bits. With a success rate of 90% using five hundred thousand traces they were able to extract all 8 key bits. The result of this attack shows that it is possible to attack the linear part of the round function of Keccak for the implementation that was used.

Chapter 5

Differential Power Analysis on Ascon

5.1 Introduction

The experiment is done using VHDL code which runs on a Sakura-G with a Spartan-6 FPGA. The oscilloscope that is used during the experiment is a Teledyne Lecroy WaveRunner 610Zi. The implementation uses a finite state machine to control the behavior of the algorithm. A start signal is implemented so the oscilloscope can be triggered to show the power consumption from that point. In the experiment the oscilloscope is set to a sampling rate of 250 million samples per second which results in 2502 samples for each power trace.
In Figure 5.1 the setup is shown with the oscilloscope and the Sakura-G board to capture power traces. On the oscilloscope we see the yellow block wave that triggers the oscilloscope to capture the signal. The gray device to the left of the Sakura-G is to program the FPGA with the required implementation.

5.2 Selection Function

As mentioned before the internal state of Ascon consists of five 64-bit registers. Since we know the contents of the state at initialization except for the key part and we can vary the nonce each run we pick the end of the first round as our initial point of attack. As sensitive variable we pick the MSB

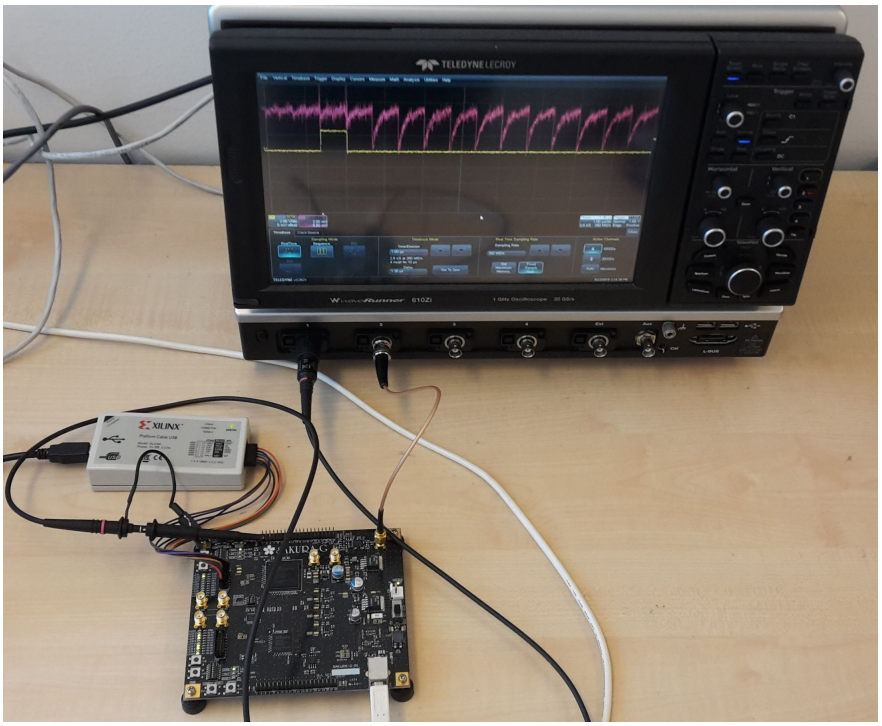


Figure 5.1: A photo of the setup for capturing traces with the Sakura-G

of x_0, since the contents of that register is known before the first round two power models can be used, the Hamming distance and the Hamming weight model. In this attack we use the Hamming distance model.
For the attack intermediate values have to be computed for each different key guess on the nonce. The linear step for x_0 is the following:

$$\Sigma_0(x_0) = x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28) \tag{5.1}$$

The output of the non-linear S-box for x_0 can be expressed in the following way:

$$y_0 = x_4x_1 + x_3 + x_2x_1 + x_2 + x_1x_0 + x_1 + x_0$$

This can be rewritten to one quadratic term.

$$y_0 = x_1(x_4 + x_2 + x_0 + 1) + x_3 + x_2 + x_0$$

We use the Hamming distance model, even though we know the contents of the register before the first we still add this term a_0 to the equation similar as in [6, Section 3].

$$y_0 = x_1(x_4 + x_2 + x_0 + 1) + x_3 + x_2 + x_0 + a_0$$

The equation now determines the activity of the register after the first round. If we look at how the state is initialized we see that x_1 and x_2 are constant key bits, x_3 and x_4 are variable bits of the nonce and x_0 is a constant bit of the IV. All the bits that add a constant amount to the activity of the register can be removed. Doing so results in:

$$y_0 = x_1(x_4 + 1) + x_3 \tag{5.2}$$

As a result the intermediate value now only depends on one bit from one register of the key and two bits from two registers of the nonce. If we combine equations (5.1) and (5.2) we get the following selection function.

$$S_i(N, K^*) = \kappa_0^*(v_{i+64} + 1) + v_i + \kappa_1^*(v_{i+109} + 1) + v_{i+45} + \kappa_2^*(v_{i+100} + 1) + v_{i+36} \tag{5.3}$$

Where the N is the 128-bit nonce and κ_i^* is a bit from a key guess. Since there are three key bits in the selection function there are eight key guesses. In Figure 5.2 we can see that using fifty thousand traces results in a success rate of almost 1, so with this attack on register x_0 it is possible to obtain the most significant 64-bits of the key.

Since the remaining number of unknown key bits is too large to enumerate the least significant half of the key must be attacked as well. In the S-box x_1 is the only register in which has a result with a quadratic term containing x_2 which contains the least significant half of the key before the first round. Starting again with the linear step:

$$\Sigma_1(x_1) = x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39) \tag{5.4}$$

And the output of the S-box for x_1:

$$\begin{aligned} y_1 &= x_4 + x_3x_2 + x_3x_1 + x_3 + x_2x_1 + x_2 + x_1 + x_0 \\ y_1 &= x_3(x_2 + x_1 + 1) + x_4 + x_2x_1 + x_2 + x_1 + x_0 \\ y_1 &= x_3(x_2 + x_1 + 1) + x_4 \\ y_1 &= x_3(x_{12} + 1) + x_4 \end{aligned} \tag{5.5}$$

Rewriting the expression with the least amount of quadratic terms and removing the terms that contribute a constant amount to the activity in the register results in equation (5.5). In the attack it will not be possible to distinguish the XOR between x_1 and x_2 so their result will be regarded as one term x_{12}. To create a selection function for the bits in register x_1 equations (5.4) and (5.5) can be combined.

$$S_i(N, K^*) = v_i(\kappa_0^* + 1) + v_{i+64} + v_{i+3}(\kappa_1^* + 1) + v_{i+67} + v_{i+25}(\kappa_2^* + 1) + v_{i+89} \tag{5.6}$$

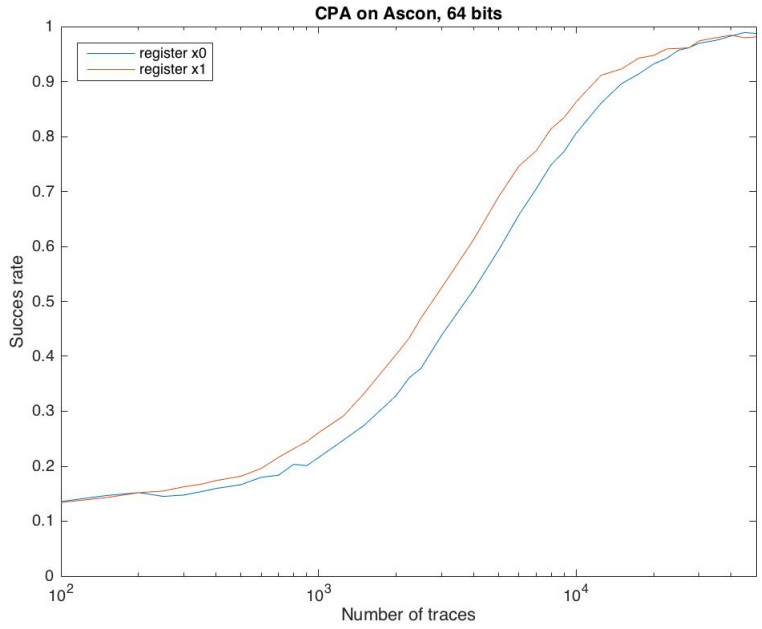


Figure 5.2: Success rate of attack on Ascon bit by bit

Figure 5.2 shows that the success rate of the attack on register x_1 approaches 1 at fifty thousand traces, with this attack it is possible to obtain the XOR x_{12} between bits from register x_1 and x_2 which contain the key before the first round. Since the first attack resulted in the key bits stored in x_1 this can be XOR'ed with x_{12} to obtain x_2. With high success rate it is possible to obtain the key from Ascon using DPA.

Looking at the remaining output registers of the S-box.

$$
\begin{aligned}
y_2 &= x_4x_3 + x_4 + x_2 + 1 \\
y_3 &= x_4x_0 + x_4 + x_3x_0 + x_3 + x_2 + x_1 + x_0 \\
y_4 &= x_4x_1 + x_4 + x_3 + x_1x_0 + x_1
\end{aligned}
$$

We can see that y_2 and y_3 have no non-linear terms with a key and a nonce register so they can not be attacked. Register y_4 does have a non-linear term with a key and a nonce register and can be attack. The expression can be

rewritten in a similar way.

$$\begin{aligned} y_4 &= x_4x_1 + x_4 + x_3 + x_1x_0 + x_1 \\ y_4 &= x_4(x_1 + 1) + x_3 + x_1x_0 + x_1 \\ y_4 &= x_4(x_1 + 1) + x_3 \end{aligned}$$

Since it was already possible to obtain the whole key by attacking register y_0 and y_1 register y_4 was not attacked. It may be interesting to look at this later as the leakage is not exactly the same for registers y_0 and y_1 so this might also be the case for register y_4.

5.3 Minimal required amount of traces

A good way to describe how well a device is protected against DPA is by looking at how many traces are required to successfully obtain the key. A way to do this theoretically was proposed in [21]. To calculate a lower bound on the required number of samples they used the distance between the distributions where the correlation $\rho = 0$ and $\rho = \rho_{max}$. The following equation was derived:

$$S = 3 + 8\left(\frac{Z_\alpha}{\ln(\frac{1+\rho_{max}}{1-\rho_{max}})}\right)^2$$

In the equation Z_α is a quantile which determines the distributions of $\rho = 0$ and $\rho = \rho_{max}$. The higher α the bigger the distance between the distributions which results in a higher peak in the correlation. A higher α also means a higher number of required traces. Next we need a way to compute ρ_{max}. In chapter 6.3 of [22] a way to compute an estimate of an upper bound of the correlation is derived.

$$\rho_{max} = \sqrt{\frac{a}{a+n}}$$

Where a is the number of wires that process the attacked intermediate value and n is the number of wires that process the other statistically independent bits. To get this simplified equation some strict assumptions were made but since an attacker usually does not have more information the result will be a good estimate of the upper bound of the correlation coefficient.

In the case of Ascon one bit is attacked and the state has a total of 320 bits this results in $\rho_{max} = 0.056$. Setting $\alpha = 0.9999$ should result in a lower

bound of the number of traces that are used where we can determine the correct key with high probability. If we compute the lower bound on the number of traces we get $S \approx 9000$. Figure 5.2 shows that the success rate starts to increase rapidly after approximately 10000 traces. This means that the calculated lower bound is a bit low but a good estimate to start with.

5.4 Algorithmic and other noise

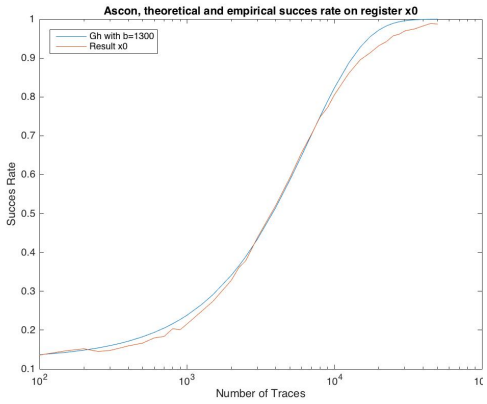
In the previous section we saw that the estimated lower bound on the required amount of traces to be able to distinguish between the distributions of correct and incorrect key guesses was lower than the empirical results from Figure 5.2. This could be because the noise levels are a lot higher in the FPGA that was used than the noise of the model which may be more applicable to an actual ASIC.
In [6] they show how to compute the theoretical success probability from the total number of bits in the state, the number of key guesses and the amount of traces that are used. This is done by modeling the algorithmic noise and the power consumption of a chip. To compute the success probability of this model the difference of means of the modeled power consumption of the correct and incorrect key hypotheses is computed.

$$G_h(\sigma^2) = \int_0^\infty \left(erf\left(\frac{t}{\sqrt{2}\sigma} \right) \right)^{h-1} \left(\mathcal{N}_{(-1;\sigma^2)}(t) + \mathcal{N}_{(1;\sigma^2)}(t) \right) dt$$

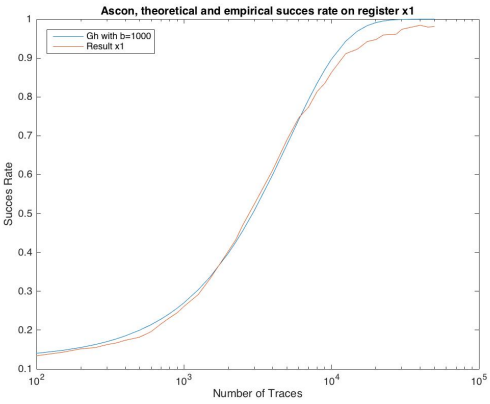
$$P_{\text{success}} = G_h\left(\frac{b}{|M|} \right)$$

Where h is the number of key hypotheses, b is the number of bits in the state and $|M|$ is the number of traces that are used. This is based on a difference of means attack but correlation power analysis on one bit is equivalent.

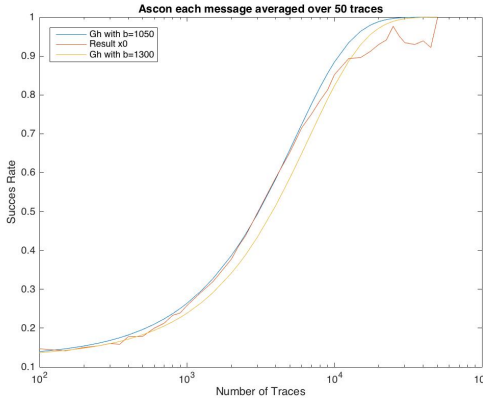
In Figure 5.3a we see that the theoretical success probability of a state with 1300 bits follows the success rate from the attack oon register x_0 very closely. This means that the algorithmic noise $320/1300 \approx 0.25$ is approximately 25% and the rest of the noise is 75%. For register x_1 this is the case with $b = 1000$ and this results in algorithmic noise of $320/1000 = 0.32$ is 32% and the rest of the noise is 68%. In both results the amount of algorithmic noise is low. The total amount of flip-flops in the implementation is lower than 1000 which means the FPGA generates a lot of noise other than algorithmic noise.



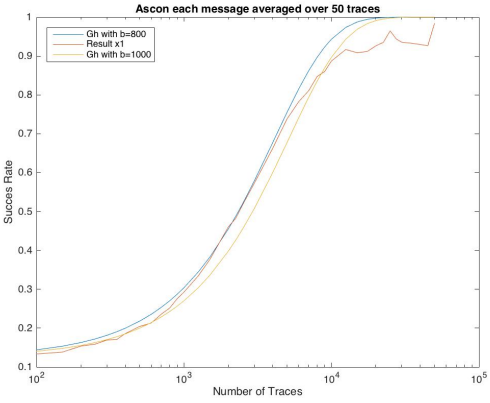
(a) Noise for register x0



(b) Noise for register x1



(c) Noise for register x0 with averaging



(d) Noise for register x1 with averaging

Figure 5.3: Success rate with the algorithmic noise

We saw that a lot of the generated noise is noise other than algorithmic noise. Other noise that is present is electrical noise. We assume the electrical noise is Gaussian which makes it possible to reduce it by averaging over a set amount of traces with an equal input.
In this case the traces were averaged over fifty equal inputs and the result can be observed in Figure 5.3c and 5.3d. Both figures show that the success rate is a bit higher for the same amount of traces. The result is still not close to the theoretical curve where the state contains 320 bits. This means that other than the theoretical algorithmic noise and electrical noise there is a lot of other noise still present in the power trace. The causes of this unknown noise require more research to be determined.

Chapter 6

Differential Power Analysis on TI of Ascon

6.1 Introduction

The threshold implementation used in the experiment is a VHDL implementation which runs on the Sakura-G board which contains a Spartan-6 FPGA. The oscilloscope that is used during the experiment is a Teledyne Lecroy WaveRunner 610Zi. The TI uses three shares which are initialized by a python script and send to the FPGA. The used sampling rate is 250 million samples per second which resulted in 2502 samples per trace.

6.2 Selection Function

An internal state of 320 bits will have a lot of algorithmic noise. When using a state of this size the theoretical amount of traces required to successfully obtain the correct key is infeasible [6] as it might require an amount of traces in the order of a billion. To be able to obtain the correct key with a feasible amount of traces it was required to reduce the amount of algorithmic noise by decreasing the size of the state. In Ascon the state consists of five 64-bit registers, to reduce the amount of algorithmic noise the size of the registers are decreased to 4-bit registers each. In total the state consists of 20 bits and with three shares there are 60 bits.

This change does not affect the substitution layer of the round function from the algorithm. The round constant is changed to the following where i is the round number starting from 0.

$$x_2 = x_2 \oplus (\texttt{0xF} - i * \texttt{0x1})$$

The linear layer is also affected, the rotations are computed modulo the size of the registers and result in the following expressions.

$$\begin{aligned}
\Sigma_0(x_0) &= x_0 \oplus (x_0 \ggg 3) \oplus (x_0 \ggg 0) \\
\Sigma_1(x_1) &= x_1 \oplus (x_1 \ggg 1) \oplus (x_1 \ggg 3) \\
\Sigma_2(x_2) &= x_2 \oplus (x_2 \ggg 1) \oplus (x_2 \ggg 2) \\
\Sigma_3(x_3) &= x_3 \oplus (x_3 \ggg 2) \oplus (x_3 \ggg 1) \\
\Sigma_4(x_4) &= x_4 \oplus (x_4 \ggg 3) \oplus (x_4 \ggg 1)
\end{aligned}$$

Since $28 \mod 4 = 0$ the two XOR's cancel each other out and result in a shorter linear expression.

$$\Sigma_0(x_0) = (x_0 \ggg 3)$$

To attack the x_0 register we can combine Equation (5.2) from the attack on the unprotected implementation and $\Sigma_0(x_0)$ to obtain a selection function.

$$S_i(N, K^*) = \kappa_0^*(v_{i+5} + 1) + v_{i+1}$$

The equation contains only one key bit, as a result there are two key candidates in the attack.

To attack register x_1 we can combine Equation (5.5) again from the previous attack on the unprotected implementation and $\Sigma_1(x_1)$. This results in the following selection function.

$$S_i(N, K^*) = v_i(\kappa_0^* + 1) + v_{i+4} + v_{i+3}(\kappa_1^* + 1) + v_{i+7} + v_{i+1}(\kappa_2^* + 1) + v_{i+5}$$

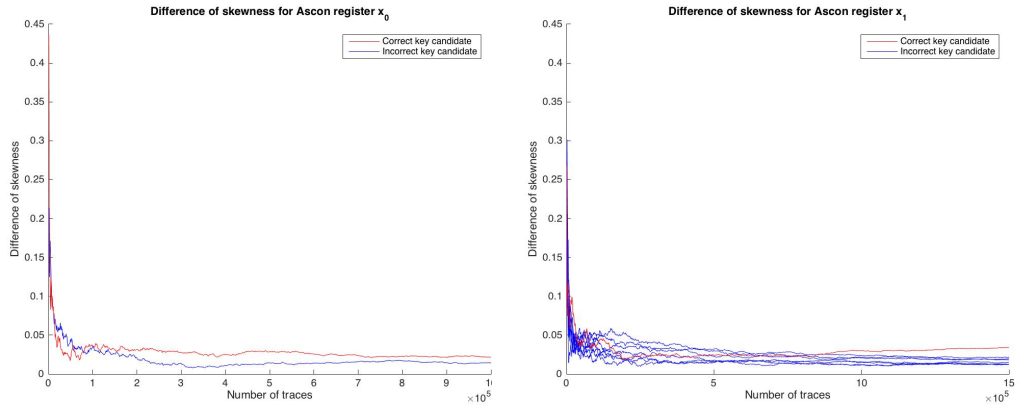
This equations contains three key bits and as a result there are eight key candidates to be considered in the attack.

Since the implementation is protected with a threshold implementation the mean of the power consumption of the two sets for each key candidate will be equal and a first-order attack with distinguisher like difference of means or the Pearson correlation will not work. As a consequence we need a higher

order distinguisher, since the implementation uses three shares a third order distinguisher is required. In this attack we will use the difference of skewness for this, Section 3.3.6 describes the definition and how it is computed for a large data set.

First we ran the attack for register x_0, after 3.4 million traces this resulted in a peak on the first clock cycle for the wrong key candidate. This was strange but as there are only two candidates so this is also the complement of the bit-value of the key candidate, this may also leak so we decided to continue to register x_1 and see if there was similar behavior. After over 33 million traces there was still no distinctive peak on the correct clock cycle for the correct key candidate or it's complement. Next we decided to collect new traces but instead of initialized the shares randomly we initialize them with zeros. Since there was no protection anymore a first order attack should suffice to obtain the correct key candidate. This was not the case, on the correct clock cycle for the correct key candidate there was a peak but there were also higher peaks on different clock cycles for incorrect key candidates. To verify that the attack was working we did a simulation of the first twelve rounds of Ascon which is the initialization part.

We modeled the power consumption similar as in [6] but instead of the Hamming distance model the Hamming weight model was used to model the contribution of the values in the registers to the power consumption. If a bit is 0 it contributes +1 to the power consumption and if a bit is 1 it contributes -1. A bit in the state of Ascon can be represented by three bits in the threshold implementation, for 0 there are 000, 011, 101 or 110, for 1 there are 111, 001, 010 or 100. Since the threshold implementation is initialized randomly the occurrence of each triplet is also random so for each bit in the state we compute the power consumption each round in the following way, if a bit in the state is 0 either all three registers are 0 contributing +3 to the power consumption or 1 bit is 0 and 2 bits are 1 which contributes -1 to the power consumption. The contribution of +3 has probability 1/4 and -1 a probability of 3/4. If the bit is 1 either all three registers are 1 which contributes -3 to the power consumption or 1 register is 1 and 2 are 0 which contributes +1. The contribution of -3 has a probability of 1/4 and the contribution of +1 has a probability of 3/4. This results in a mean of 0 and a variance of 3. The resulting power trace contains 12 samples, one for each clock cycle.



(a) Simulation of Ascon for register x_0 (b) Simulation of Ascon for register x_1

Figure 6.1: Simulation results for Ascon TI with 20 bit state

Figure 6.1 shows the result of the attack on the simulated traces for register x_0 and x_1 where the difference of skewness was used as a distinguisher. From both figures can be observed that it is possible to obtain the correct key candidate with the attack. For register x_0 150.000 traces were required and for register x_1 900.000. This result shows that the attack on this smaller threshold implementation of Ascon is not the issue with the traces from the FPGA. Why it does not work on traces collected with an FPGA is not clear, but during the synthesis of the VHDL code for the FPGA many things can happen so to find out exactly why the attack did not work more research should be done on the implementation of the FPGA.

Chapter 7

Differential Power Analysis on MAC-Keccak

7.1 Introduction

MAC-Keccak is an instance of Keccak where a message authentication code is computed using the Keccak function. In this case the key and the message are appended and then hashed. This is done using Keccak-f[1600] with a capacity of 576 and a rate of 1024. The key is 320 bits and is initially stored in the plane where $(y = 0)$. The remaining 704 bits are used the for message. If the key and the message is larger then the block size more blocks can be inserted but in this case we use a message size which when appended to the key results in exactly one block.

In section 4.1 we saw it is possible to attack the linear part θ of the Keccak round function with an implementation on an FPGA. Since the chip containing the Keccak function is available on the chip from the SHA-3 competition we execute the same attack as [20]. During the collection of the traces the SASEBO-R board is used with the SHA-3 chip. The oscilloscope that is used during the experiment is a Teledyne Lecroy WaveRunner 610Zi. The chip produces a block wave to display that the core is operating which is used as a trigger for the oscilloscope. A sampling rate of 1 billion samples per second is used which results in 2002 samples per trace.

7.2 Selection Function

In this attack part of a sheet shown in Figure 2.2 is used as a sensitive variable. Since in plane $(y = 0)$ the key is stored we look at the four lanes where that do not contain key material and eight columns in the lane of a sheet. Because of the parity computation this leads to key 16 bits in the selection function. As a leakage model the hamming weight model is used.

$$\theta(x,y,z) = A(x,y,z) + \sum_{y'=0}^{4} A(x-1,y',z) + \sum_{y'=0}^{4} A(x+1,y',z-1) \quad (7.1)$$

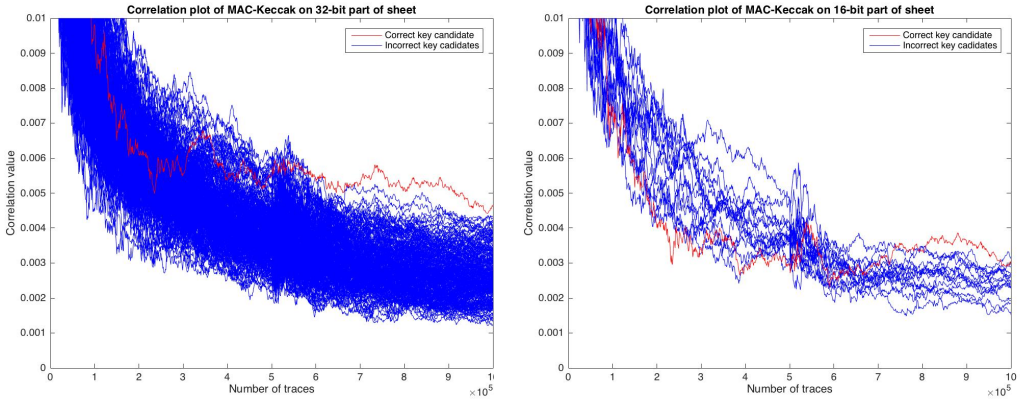
From equation (7.1) we can see that if $y > 0$ there are two key bits in the result, key bit $A(x-1,0,z)$ and $A(x+1,0,z-1)$. Since this is an XOR between the two bits it is not possible to distinguish between them during the attack, as a result only their combination can be found for each result of theta from a different column. In the attack we use the Hamming weight of four lanes and eight columns of a sheet, this results in the following selection function.

$$\sum_{z=i}^{i+l-1} \sum_{y=1}^{4} HW(\theta(x,y,z))$$

Where $i \in \{0,..,63\}, l \in \{1,..,64\}$ and $x \in \{0,..,4\}$. In this experiment we use $l = \{8,4,2,1\}$ to increase the correlation but still have a manageable amount of key candidates of 256, 16, 4 and 2 respectively. We attack different sizes of the intermediate value to try to show that the correlation value increases if a larger size of intermediate value is used.

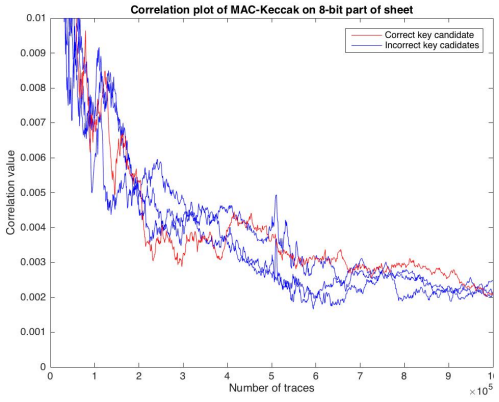
7.3 Results

Figure 7.1 shows the result of the attack on MAC-Keccak with different sizes as intermediate value. The graph is zoomed in on the y-axis because the differences in the correlation values are very small. In Figure 7.1a we can see that the correlation values of the key candidates converge, even though the correct key candidate does not have a much higher correlation compared to the incorrect key candidates it shows that after after 600.000 traces the correct key candidate always has the highest correlation value. To obtain a similar convergence for smaller sizes of the intermediate value it is likely that more traces are required. As we can see in Figures 7.1b, 7.1c and 7.1d the correct key candidate does not yet converge to a higher correlation value

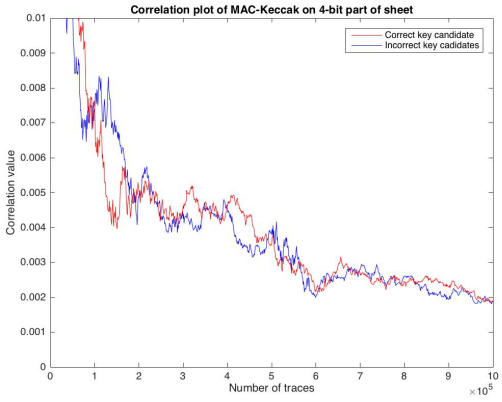


(a)

(b)



(c)



(d)

Figure 7.1: Correlation plot of MAC-Keccak for different sizes of the intermediate value

compared to the incorrect key candidates.

This shows that a larger intermediate value is more efficient when we look at the amount of traces that are required to obtain the correct key candidate. A larger intermediate value influences the amount of unknown key bits in the intermediate value and increases the search space and computational time to obtain the correct key candidate. Finding the right size for the intermediate value is a trade-off between efficiency in power traces and computational time. Which is more important is different for each situation. As computers are fast and collecting many traces is easy it is often not necessary to find the perfect trade-off as a good trade-off will also result in a successful attack.

Chapter 8

Differential Power Analysis on Keccak

8.1 Introduction

In this experiment we attack the non-linear part of Keccak called χ. The traces are generated by an ASIC containing the five finalists of the SHA-3 competition which is mounted on a SASEBO-R board and controlled by an FPGA. The ASIC has Keccak-f[1600] implemented with a rate of 1024 and a capacity of 576. The state is initialized with zeros, next the key is absorbed and 24 rounds of Keccak-f are applied as specified in Section 2.5. Then the message is duplexed into the state and again 24 rounds of Keccak-f are applied on the state. The traces are collected using a Teledyne Lecroy WaveRunner 610Zi. The oscilloscope is set a sampling rate of 1 billion samples per second which results in 2002 samples per trace.

In Figure 8.1 we see the oscilloscope with the SASEBO-R board. The black square on the top left corner of the SASEBO-R board is the mounted SHA-3 ASIC. In the bottom right corner we see power supply for the ASIC and the board.

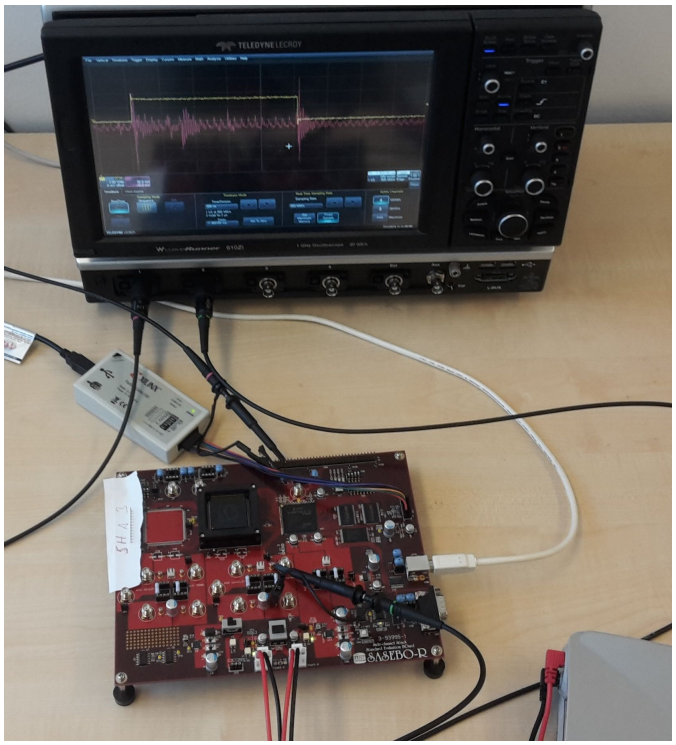


Figure 8.1: The setup for capturing traces with the Sasebo-R and the SHA-3 chip

8.2 Selection Function

In this attack we focus on a single bit of the 1600 bit state of Keccak and a row of 5 bits as shown in Figure 2.2. The attack on a single bit is equal to the attack on the simulation in [6] where the non-linear part χ is attacked. The first round after the data is absorbed is our point of attack. At this point it is possible to compute an intermediate value containing parts of the data and parts of the state which contains the key. To compute the hypothetical power consumption we use the Hamming distance model.

We know Keccak-f consists of five steps ι, χ, π, ρ and θ. Let's call the linear part of the round function $\lambda = \pi \circ \rho \circ \theta$. From Section 2.5 we know χ is defined as follows.

$$\chi(a_{(x,y,z)}) \leftarrow a_{(x,y,z)} + (a_{(x+1,y,z)} + 1)a_{(x+2,y,z)}$$

The linear part can be computed separately for different data like the input and the key state after the absorption of the key and XOR'ed later to combine them before χ. This way the input of χ can be split up into bits from the key state and bits from the input message.

$$\chi(a_0) \leftarrow k_0 + m_0 + (k_1 + m_1 + 1)(k_2 + m_2)$$

Where m_* are the message bits of the output of λ and k_* are the bits from the key state after λ. We are interested in the activity d of the register where the bit is stored.

$$\begin{aligned} d &= a_0 + k_0 + m_0 + (k_1 + m_1 + 1)(k_2 + m_2) \\ d &= a_0 + k_0 + m_0 + k_1k_2 + m_1k_2 + k_2 + m_2k_1 + m_2m_1 + m_2 \\ d &= a_0 + k_0 + (k_1 + 1)k_2 + m_0 + (m_1 + 1)m_2 + m_2k_1 + m_1k_2 \end{aligned}$$

The result of $a_0 + k_0 + (k_1 + 1)k_2$ is equal for each trace and contributes a constant amount to the activity so it can be removed. This results in the following selection function.

$$S(M, K^*) = m_0 + (m_1 + 1)m_2 + m_2k_1^* + m_1k_2^*$$

The selection function contains two key bits which means there are four key candidates.

In the previous attack we needed to guess two key bits for one bit of the state, if we extend this attack to a full row of five bits we also need to guess

five key bits which makes the attack more efficient. Below we can see the resulting selection function.

$$\begin{aligned} S(M, K^*) = & \; m_0 \oplus (m_1 + 1)m_2 \oplus m_2 k_1^* \oplus m_1 k_2^* \\ & + m_1 \oplus (m_2 + 1)m_3 \oplus m_3 k_2^* \oplus m_2 k_3^* \\ & + m_2 \oplus (m_3 + 1)m_4 \oplus m_4 k_3^* \oplus m_3 k_4^* \\ & + m_3 \oplus (m_4 + 1)m_0 \oplus m_0 k_4^* \oplus m_4 k_0^* \\ & + m_4 \oplus (m_0 + 1)m_1 \oplus m_1 k_0^* \oplus m_0 k_1^* \end{aligned}$$

This selection function uses five key bits which results in 32 key guesses and uses five bits from $\lambda(M)$.

While this selection function seems to be correct at first glance it is not as the model for the power consumption is the Hamming distance and for each register this computes if the register flips or not, but we do not know which of the two. While for 1 bit this is fine, for 5 bits this does not work as the set is not split up properly since flipping of the register and not flipping of the register gets mixed up.

8.3 Results

In Figure 8.2 we see the results for the attack on a single bit of the Keccak state after the first round the message is absorbed. After approximately 1.3 million traces the correct key candidate is the one with the highest correlation coefficient. The differences values between the correct and incorrect key candidates is very small but the correct key candidate is clearly on higher then the rest. Compared to the results in the theoretical model on of the same attack, the attack on the ASIC requires many more traces. Where the theoretical model required around 10 thousand traces the ASIC required over a million.
The result for the attack on row of 5 bits of the state is omitted as the current selection function is not correct so the results would have no meaning.

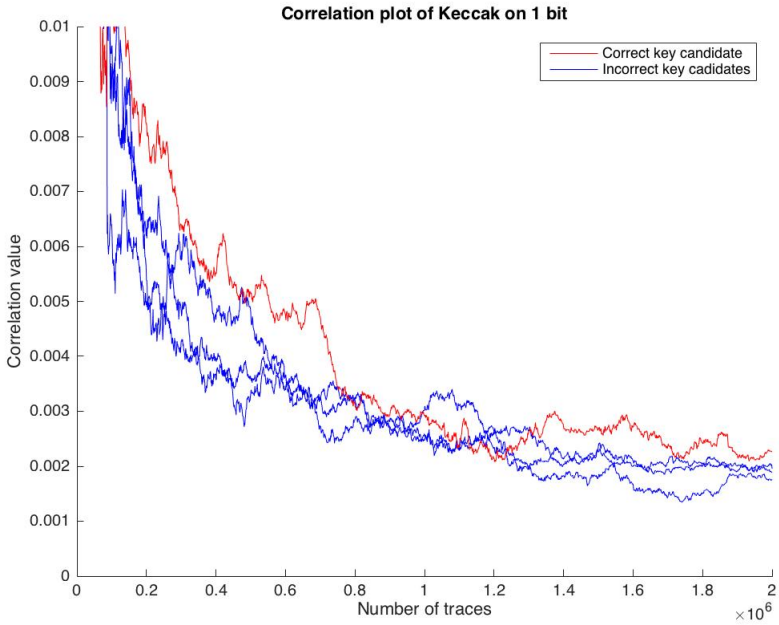


Figure 8.2: Correlation plot for CPA on 1 bit of the Keccak state

Chapter 9

Future Work

While working on this thesis different things were researched and unfortunately not everything could be covered in the amount of time there was.

During our research on Ascon we tried to find out how much the electrical noise affects the success rate of the attack. We were able to quantify the algorithmic noise using a model and we tried to reduce the electrical noise to see if the success rate would be according to the model which it was not. This means there is a lot of other noise in the signal which we could not explain according to the model. To determine where this noise comes from more research is required.

Our attack on the threshold implementation of Ascon did not work while the attack on the simulated traces did work which means the attack works. Since the implementation if very small and we do not really know what happens inside the FPGA cause by the synthesizer we could not explain why the attack did not work but we believe it was caused by this. To fully explain why it did not work more research in required.

While the attack on 1 bit of the state of Keccak was successful the attack on 5 five unfortunately was not as the attack on 1 bit could not easily be extended to more bits. An attack on would be more efficient compared to an attack on 1 bit, this makes it interesting to get a working attack but to accomplish this more research in required.

Even though the attack on 1 bit of the state of Keccak was successful, the

amount of traces that were required were much higher compared to the attack on the model. The model [6] only takes algorithmic noise into account and the results show that there is a lot of other noise as well. More research is required to create a model closer to the reality.

For the results for Ascon we were able to compute the success rate as it did not require too many traces. With the attack on the ASIC with Keccak the positive result required many traces and to compute the success rate it would require many more. For this reason we decide to plot the correlation values instead of the success rate. With more time it would still be possible to compute the success rates for these attacks.

Chapter 10

Conclusion

With our attack on Keccak we were unfortunately not able to compute the success rate due to the high amount of traces that were required. The immediately answers our first research question: Does DPA on Keccak hardware implementations follow the theoretical success rate of obtaining the correct key? It does not, the model which was used only takes algorithmic noise into account and this shows there is a lot of noise in the signal as well.

In [6] a generalization was shown for quadratic functions, since Ascon also has quadratic component in its round function. We applied the generalization on the round function of Ascon and we were able to successfully craft an attack. So it is possible to use the same techniques for Keccak on Ascon for their non-linear parts. The reason it works is likely because the non-linear part in Ascon is a variation of the non-linear part in Keccak.

During our research on Ascon we tried to reduce the electrical noise by computing the average over traces with an equal input. This resulted in a slight increase of the success rate but much lower then expected. So the reduction of the electrical noise does not affect the success rate of the attack on Ascon implemented on an FPGA very much.

With the attack on an unprotected implementation of Ascon being successful we also tried to attack an implementation protected with a threshold implementation. Looking at the model in [6] we decided to reduce the size of the state of Ascon which reduces the algorithmic noise as the complete state of 320 will likely take billions of traces to attack. With the smaller implementa-

tion with a 20 bit we were not able to successfully craft an attack. There were strange peaks on different clock cycles for different key candidates which we could not explain. For this reason we simulated traces and attacked those, we were able to attack the simulated traces which shows the attack works. We can't fully explain why it did not work but it is likely that the VHDL code is not the as what happens in the FPGA.

In this thesis we attack two parts of Keccak, the linear and the non-linear part. As expected the linear part required more traces to obtain the correct key candidate. Unfortunately we were not able to compute the success rate due to high amount of traces that would be required for this.

Bibliography

[1] Farzaneh Abed, Christian Forler, and Stefan Lucks. General overview of the authenticated schemes for the first round of the caesar competition. Cryptology ePrint Archive, Report 2014/792, 2014.

[2] Ross Anderson. *Security engineering*. John Wiley & Sons, 2008.

[3] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C-W Phan. Sha-3 proposal blake. *Submission to NIST*, 2008.

[4] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Cryptographic sponge functions. Submission to NIST (Round 3), 2011.

[5] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The Keccak reference, January 2011. `http://keccak.noekeon.org/`.

[6] Guido Bertoni, Joan Daemen, Nicolas Debande, Thanh-Ha Le, Michael Peeters, and Gilles Van Assche. Power analysis of hardware implementations protected with secret sharing. Cryptology ePrint Archive, Report 2013/067, 2013.

[7] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Caesar submission: Ketje v1. *CAESAR First Round Submission, March*, 2014.

[8] Begül Bilgin, Joan Daemen, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Gilles Van Assche. Efficient and first-order DPA resistant implementations of keccak. In *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, pages 187–199, 2013.

[9] Paul Bottinelli and Joppe W. Bos. Computational aspects of correlation power analysis. Cryptology ePrint Archive, Report 2015/260, 2015. `http://eprint.iacr.org/2015/260`.

[10] Eric Brier, Christophe Clavier, and Francis Olivier. *Correlation Power Analysis with a Leakage Model*, pages 16–29. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[11] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[12] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer. Ascon v1.1 submission to caesar, 2015.

[13] Jack W Dunlap. Combinative properties of correlation coefficients. *The Journal of Experimental Education*, 5(3):286–288, 1937.

[14] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The skein hash function family. *Submission to NIST (round 3)*, 7(7.5):3, 2010.

[15] Praveen Gauravaram, Lars R Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S Thomsen. Grøstl-a sha-3 candidate. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2009.

[16] B Guido, D Joan, P Michaël, VA Gilles, and VK Ronny. Caesar submission: Keyak v2. 2015.

[17] Xu Guo, Meeta Srivastav, Sinan Huang, Dinesh Ganta, Michael B Henry, Leyla Nazhandali, and Patrick Schaumont. Asic implementations of five sha-3 finalists. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*, pages 1006–1011. IEEE, 2012.

[18] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual International Cryptology Conference*, pages 388–397. Springer, 1999.

[19] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), February 1997. Updated by RFC 6151.

[20] Pei Luo, Yunsi Fei, Xin Fang, A. Adam Ding, Miriam Leeser, and David R. Kaeli. Power analysis attack on hardware implementation of mac-keccak on fpgas. In *ReConFig*, pages 1–7. IEEE, 2014.

[21] Stefan Mangard. *Topics in Cryptology – CT-RSA 2004: The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23-27, 2004, Proceedings*, chapter Hardware Countermeasures

against DPA – A Statistical Analysis of Their Effectiveness, pages 222–235. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[22] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[23] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. *Information and Communications Security: 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006. Proceedings*, chapter Threshold Implementations Against Side-Channel Attacks and Glitches, pages 529–545. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[24] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *Journal of Cryptology*, 24(2):292–321, 2011.

[25] Philippe Pierre Pebay. Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments. Technical report, Sandia National Laboratories, Sep 2008.

[26] Eric Peeters, François-Xavier Standaert, Nicolas Donckers, and Jean-Jacques Quisquater. Improved higher-order side-channel attacks with fpga experiments. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 309–323. Springer, 2005.

[27] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 142–159. Springer, 2013.

[28] Hongjun Wu. The hash function jh. *Submission to NIST (round 3)*, page 6, 2011.