Georg STÜTZ

Sharing of small vectorial Boolean functions: A side-channel countermeasure

MASTER THESIS

written to obtain the academic degree of a Master of Science (MSc)

Master programme Mathematical Computer Science

Graz University of Technology

Adviser:
Univ.-Prof. Dr. Vincent Rijmen

Institute of Applied Information Processing and Communications

March 2012

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

....

date (signature)

Acknowledgements

Firstly, I would like to thank my advisers Vincent Rijmen and Martin Schläffer for their guidance and support on this thesis. Special thanks go to Martin, for the endless time and the effort he spent on discussions and corrections of this thesis.

Secondly, I would like to thank my parents Franz and Isolde Stütz who supported me in every possible way throughout my whole studies. I also want to thank my girlfriend, Kristina, as well as my friends and colleagues who have always been there for me.

Abstract

The field of side-channel analysis poses a major threat to cryptographic devices. By the use of statistical analysis of measurements of physical side channel information, like instantaneous power consumption, hardware implementations become vulnerable to practical attacks. Several masking countermeasures have been proposed which start from idealized hardware models. Applied to real hardware, devices still leak information about the processed values. The reasons are hardware effects, like glitches or timing delays, together with the fact that intermediate values are not independent of the unmasked function inputs. Nikova et al. propose a countermeasure with low hardware requirements, based on secret sharing. The approach of Nikova et al. is to guarantee the independence of intermediate values by the use of a secret sharing scheme.

In this thesis we analyze secure implementations according to this secret sharing approach of affine equivalent S-boxes. We characterize three required properties *balancedness*, *correctness* and *non-completeness* in different representations of secure implementations of vectorial Boolean functions. We analyze the method of direct sharing for equivalence classes under affine transformation equivalence. We present an approach to construct a balanced, correct and non-complete implementation for a given vectorial Boolean function. We show that it is easy to fulfill only two of the three required properties, like correctness and non-completeness or correctness and balancedness. For simple S-boxes also all three properties can be fulfilled. However it remains open if there exist secure implementations which fulfill all three properties for cryptographically strong S-boxes.

Kurzfassung

Seitenkanalattacken stellen eine große Bedrohung für kryptografische Hardware dar. Hardware Implementationen werden durch die Verwendung statistischer Methoden zur Analyse von Messungen aus Seitenkanal-Informationen, wie dem Stromverbrauch, anfällig für praktische Attacken. Diverse Maskierungs-Gegenmaßnahmen gegen Seitenkanalattacken, welche von idealisierten Eigenschaften der Hardware ausgehen, wurden vorgestellt. Bei der Anwendung dieser Gegenmaßnahmen auf echte Implementationen, arbeiteten die Gegenmaßnahmen nicht wie behauptet. Ursachen dafür sind Eigenschaften der Hardware, wie Glitches oder Verzugszeiten, sowie der Umstand, dass die Zwischenergebnisse nicht unabhängig von den unmaskierten Eingabeparametern sind. Nikova et al. schlagen eine Gegenmaßnahme vor, welche niedrige Anforderungen an die Hardware stellt und auf Secret Sharing basiert. Der Ansatz von Nikova et al. ist es, die Unabhängigkeit der Zwischenresultate durch Verwendung eines Secret Sharing Schemas zu garantieren.

In dieser Arbeit werden sichere Implementierungen von affin äquivalenten S-Boxen mit dem Secret Sharing Schema von Nikova et al. betrachtet. Wir charakterisieren drei notwendige Bedingungen, *balancedness*, *correctness* und *non-completeness* in unterschiedlichen Repräsentationen von vektorwertigen Boolschen Funktionen. Ferner untersuchen wir die Methode des *direct sharing* für Klassen unter affiner Äquivalenz. Wir präsentieren Methoden um Realisierungen von vektorwertigen Boolschen Funktionen zu konstruieren, sodass die Eigenschaften *balancedness*, *correctness* und *non-completeness* erfüllt sind. Desweiteren zeigen wir, dass es einfach ist, nur zwei der drei notwendigen Eigenschaften zu erreichen, wie *balancedness* und *correctness* oder *correctness* und *non-completeness*. Für einfache S-Boxen können ebenfalls alle drei Eigenschaften erreicht werden. Es bleibt jedoch offen, ob sichere Realisierungen für kryptografisch starke S-Boxen existieren.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In 1996 Kocher presented *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems* [30] and founded with it the field of modern side-channel analysis. Up to that time the focus of attacking cryptographic hardware devices was to attack the underlying cryptographic principle. Kocher, on the other hand, highlighted in his work [30] the leakage of secret information through physical channels of hardware implementations of cryptographic principles. His work led to a change of awareness since secure cryptographic principles became breakable by their implementation in hardware.

Nowadays side-channel attacks pose a major threat to cryptographic devices. The reasons for this can be found in low requirements for attacks, e.g. low information about implementation details. On the other hand it is hard to construct devices that emit no information, since for example each device needs to consume some power.

However a major research for countermeasures to prevent side-channel leakage has been made in the last decades. The general idea of mostly all countermeasures is to introduce randomness into the calculation in order to distort the side-channel leakage. These countermeasures can be employed in software [43] or in hardware. Hardware countermeasures can be applied on the algorithmic level, like masking an AES S-box [36] or on the gate level as algorithmic independent approach, like dual rail pre-charge logic styles [70, 50]. But research showed that these methods do not always work as proposed [43, 36, 68].

These countermeasures do not work as claimed, since the masked intermediate values from the calculations are not independent of the unmasked inputs. Chari et al. [17] and Nikova et al. [44, 45] counter this circumstance by using secret sharing techniques. The idea of this approach is that if the input values of the cryptographic function are split up into new inputs in

such a way that subsets of the new inputs become statistically independent of the primary inputs. By using these new inputs, the intermediate values and hence the side-channel information, becomes independent of the primary inputs. This secret sharing technique has to be designed in such a way, that a output results, which is correct with respect to the primary inputs.

In this thesis we employ the sharing technique introduced by Chari et al. in [17] and extended by Nikova et al. in [44, 45] to S-boxes of small size. We therefore analyze certain properties of these S-boxes and present approaches to share small S-boxes.

1.1 Outline of this thesis

This thesis is organized as follows.

Chapter 2 will be dedicated to an introduction to Boolean functions. We will introduce a variant of the discrete Fourier transformation, called the Walsh-Hadamard transformation. We will further discuss balanced Boolean functions and vectorial Boolean functions, which both play an important role in the rest of the thesis. Finally we will discuss criteria of Boolean functions when used in cryptographic applications.

Chapter 3 will handle the topic of side-channel attacks. We start this chapter with a general introduction to side-channel leakage of hardware devices. We then introduce the Simple Power Analysis, a first side-channel analysis method presented by Kocher in [30]. We continue with the Differential Power Analysis, which is a more sophisticated analysis method to analyze huge number of power traces. We further discuss variants of the Differential Power Analysis method and close the chapter with a consideration of state of the art countermeasures against side-channel attacks.

Chapter 4 will be dedicated to a general introduction to secret sharing schemes. We will cover the field of multi-party computation protocols and common masking schemes. We will also discuss a common problem of masking schemes in the presence of certain hardware effects, called glitches. The rest of this chapter will be dedicated to the secret sharing scheme which will be used for the rest of this thesis. We will present the results from Nikova et al. [44, 45], who prove the security of this sharing method. Finally we will talk about secure sharing of affine equivalent S-boxes.

Chapter 5 will analyze certain properties of shared functions in truth tables and cover several approaches to build secure sharings for given S-boxes. We thereby only consider representatives of classes under the affine equivalence relation. We present known realizations of 3×3 S-boxes, introduce correction terms and search methods to find secure sharings.

Chapter 6 will handle affine equivalent S-boxes under the aspect of the Walsh transformation. We characterize certain properties which are important in order to obtain a secure sharing, by mean of the Walsh transformation. We further investigate on the Walsh-transformation of the shared and unshared classes of 3×3 S-boxes.

Chapter 2

Boolean functions

Boolean functions play an important role in game theory [74, 38], coding theory [35, 9], the design of combinational logic [76, 85], complexity theory [73, 79] and many more. In general, Boolean functions are mappings between Boolean algebras which were introduced by the mathematician George Boole in 1847. However Boolean functions also play an important role in the field of cryptography where cryptographic transformations like *(linear) feedback shift registers ((L)FSR)* in stream ciphers or S-boxes in block ciphers can be constructed by combining nonlinear Boolean functions.
In this chapter we will first give formal definitions of Boolean functions and related properties of them in Section 2.1. In Section 2.2 and Section 2.3 we will describe the *discrete Fourier transformation* and address the topic of balanced Boolean functions. In Section 2.4 we will assess the topic of vectorial Boolean functions. In Section 2.5 we will describe criteria for Boolean functions in cryptography.

2.1 Introduction to Boolean functions

2.1.1 Definitions

In this thesis we often associate the set $\{0, 1\}$ with the finite field $\mathbb{F}_2$, containing two elements. $\mathbb{F}_2^n$ denotes the set of all binary vectors of length n and is viewed as an $\mathbb{F}_2$-vector space. Since some additions of bits are considered in $\mathbb{Z}$ (with characteristic 0) and some are considered in $\mathbb{F}_2$ (with characteristic 2) we denote additions in $\mathbb{Z}$ with $+$ and additions in $\mathbb{F}_2$ with $\oplus$. Addition of vectors in $\mathbb{F}_2^n$ are denoted with $+$ since F_2^n can be identified with $\mathbb{F}_{2^n}$ the field with 2^n elements.

Definition 2.1.1: A Boolean function in n variables is a function

$$f : \mathbb{F}_2^n \to \mathbb{F}_2.$$

The set of all Boolean functions from $\mathbb{F}_2^n \to \mathbb{F}_2$ is denoted by $\mathcal{BF}_n$. The set $\mathcal{BF}_n$ has size 2^{2^n}.

Definition 2.1.2 ([13])**:** The *sign function* $\chi : \mathbb{F}_2 \to \mathbb{R}^\times$ is the only non-trivial group homomorphism, that is $\chi(0) = 1$, $\chi(1) = -1$ or in general $\chi(a) = (-1)^a = 1 - 2a$.
For a Boolean function f we define the *sign function* of f to be

$$f_\chi : \mathbb{F}_2^n \to \mathbb{R}^\times \quad f_\chi := (-1)^{f(x)} = 1 - 2f(x).$$

It is obvious that $(f \oplus g)_\chi = f_\chi g_\chi$ holds for all Boolean functions f and g. To deduce a formula for the product of two Boolean functions we take a look at the truth Table 2.1 and get the formula [19]

$$\chi(a \oplus b) + 2\chi(ab) = 1 + \chi(a) + \chi(b) \quad \text{for all } a, b \in \mathbb{F}_2.$$

Thus, the following formula [19] holds for all $f, g \in \mathcal{BF}_n$

$$2(fg)_\chi = 1 + f_\chi + g_\chi - (f \oplus g)_\chi$$

a	b	$a \oplus b$	ab	$\chi(a)$	$\chi(b)$	$\chi(a \oplus b)$	$\chi(ab)$
0	0	0	0	1	1	1	1
0	1	1	0	1	-1	-1	1
1	0	1	0	-1	1	-1	1
1	1	0	1	-1	-1	1	-1

Table 2.1: Correspondence between the truth table and the values of the sign function

Definition 2.1.3: For $x \in \mathbb{F}_2^n$ the *Hamming weight* $\mathrm{w}_H(x)$ of x is the number of nonzero coordinates of x. More formally, let $N = \{1, \ldots, n\}$ then the Hamming weight of x is

$$\mathrm{w}_H(x) = |\{i \in N : x_i \neq 0\}|.$$

The set $\{i \in N : x_i \neq 0\}$ is called the *support* of x.
For a Boolean function f the *Hamming weight* of f is the size of the *support of the function* f, that is

$$\mathrm{w}_H(f) = |\{x \in \mathbb{F}_2^n : f(x) \neq 0\}|$$

2.1.2 Affine mappings

Next we want to introduce an important class of transformations on Boolean functions, the so called *affine (coordinate) transformations*.

Definition 2.1.4 ([13])**:** Let $L : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a linear transformation and $t \in \mathbb{F}_2^n$ be a vector. Then the mapping

$$x \mapsto L(x) + t$$

is an *affine mapping*. The linear mapping L can also be represented as an element of $\mathbb{F}_2^{n\times n}$, that is the set of $n \times n$ matrices over $\mathbb{F}_2$. The mapping is called *invertible* if L is an invertible linear mapping. This is equivalent to

$$\exists L' \in \mathbb{F}_2^{n\times n} : L \cdot L' = L' \cdot L = I$$

where I denotes the identity matrix in $\mathbb{F}_2^{n\times n}$.

The set of invertible affine mappings forms an algebraic group $\mathcal{A}$ with the multiplication operation defined as:

$$(M, t) \cdot (N, u) = (MN, t + Mu),$$

where (M, t) and (N, u) are two invertible affine mappings with $M, N \in \mathbb{F}_2^{n\times n}$ and $t, u \in \mathbb{F}_2^n$. To calculate the order of the group $\mathcal{A}$ it suffices to calculate the number of invertible matrices over the field $\mathbb{F}_2$, which is the order of the general linear group $GL(n, \mathbb{F}_2)$.

$$|GL(n, \mathbb{F}_2)| = \prod_{i=0}^{n-1}(2^n - 2^i) = 2^{\frac{n(n-1)}{2}} \prod_{i=1}^{n}(2^i - 1) < 2^{n^2} \tag{2.1}$$

So by combining the number of invertible matrices with the number of vectors of length n in $\mathbb{F}_2$ we get:

$$|\mathcal{A}| = 2^n \cdot \prod_{i=0}^{n-1}(2^n - 2^i) = 2^{\frac{n(n+1)}{2}} \prod_{i=1}^{n}(2^i - 1) \leq 2^{n+n^2} \tag{2.2}$$

Equations (2.1) and (2.2) can be found in [34].

2.1.3 Representations of Boolean functions

We now want to take a look at the representation of Boolean functions which can be done in several ways. We only want to take two representations into account, namely the representation using its *truth table* and by the *algebraic*

normal form (ANF).
To represent a Boolean function f with its truth table, all values $x \in \mathbb{F}_2^n$ and all values $f(x)$ are simply written in a table. Of course this representation is not very practical, especially for large n since the size of the table grows exponential in n.
The representation with the algebraic normal form is the one which is most usually used in cryptography and is the representation of the Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ as n-variable polynomial, [13]:

$$f(x) = \bigoplus_{I \in \mathcal{P}(N)} a_I \left(\prod_{i \in I} x_i \right) = \bigoplus_{I \in \mathcal{P}(N)} a_I x^I, \tag{2.3}$$

where $N = \{1, \ldots, n\}$ and $\mathcal{P}(N)$ denotes the power set of N. Since every bit in $\mathbb{F}_2$ equals its own square the exponents for each coordinate x_i are at most 1. Before we consider the uniqueness of the ANF we want to take a look at a small example.

Example: Let f be the function with the following truth Table 2.2

x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.2: Truth table of the function f

This function can be split up into the following two nontrivial atomic functions f_1 and f_2, such that $f = f_1 + f_2$. The truth tables of f_1 and f_2 are listed in Table 2.3: The function f_1 takes value $1 \Leftrightarrow 1 \oplus x_1 = 1$ and $x_2 = 1$,

x_1	x_2	$f_1(x_1, x_2)$	$f_2(x_1, x_2)$
0	0	0	0
0	1	1	0
1	0	0	1
1	1	0	0

Table 2.3: Truth tables of the functions f_1 and f_2

that is $\Leftrightarrow (1 \oplus x_1)x_2 = 1$. It follows that the ANF of f_1 equals $x_2 \oplus x_1x_2$. For the function f_2 we get that f_2 takes value $1 \Leftrightarrow x_1 = 1$ and $1 \oplus x_2 = 1$. We obtain that the ANF of f_2 is $x_1 \oplus x_1x_2$. By adding both functions we finally get the ANF of $f = f_1 \oplus f_2 = x_1 \oplus x_2$.

We now want to take a look at the existence and the uniqueness of the ANF. What we did in the example above was, that we applied the Lagrange interpolation method [13] to the function f which yields a polynomial in $\mathbb{F}_2[x_1,\ldots,x_n]/(x_1^2\oplus x_1,\ldots,x_n^2\oplus x_n)$ which is the quotient ring $\mathbb{F}_2[x_1,\ldots,x_n]$ modulo $(x_1^2\oplus x_1,\ldots,x_n^2\oplus x_n)$ which is an ideal. This implies that the mapping from every polynomial $P\in\mathbb{F}_2[x_1,\ldots,x_n]/(x_1^2\oplus x_1,\ldots,x_n^2\oplus x_n)$ to the corresponding function $x\in\mathbb{F}_2^n\to P(x)$, is surjective on the set of functions $\mathcal{BF}_n$. Since the sets $\mathcal{BF}_n$ and $\mathbb{F}_2[x_1,\ldots,x_n]/(x_1^2\oplus x_1,\ldots,x_n^2\oplus x_n)$ have the same size this mapping is one to one.

Definition 2.1.5 ([35])**:** For a Boolean function f the *algebraic degree* of

$$f=\bigoplus_{I\in\mathcal{P}(N)} a_I x^I$$

is defined as

$$\deg(f):=\begin{cases}-\infty, & \text{if } f\equiv 0\\ \max_{I\in\mathcal{P}}\{|I|:a_I\neq 0\}, & \text{for all nonzero functions } f\end{cases}.$$

Definition 2.1.6: A Boolean function f is called *linear* if $\deg(f)=1$ and $f(0)=0$ which is equivalent to $a_I=0$ for $I=\emptyset$.
The function f is called *affine* if $\deg(f)=1$.
The function f is called *quadratic* if $\deg(f)=2$.

In the next section we will introduce a useful tool when it comes to analyze Boolean functions.This tool is the so called *discrete Fourier transform*.

2.2 Walsh Hadamard transformation

When analyzing Boolean functions, the framework of the *discrete Fourier transform* often turns out to be a very useful tool. The Fourier transform [13] also called *Hadamard* transform is the linear mapping which maps any Boolean function $f\in\mathcal{BF}_n$ to the function $\hat{f}$ defined on $\mathbb{F}_2^n$ by

$$\hat{f}(u)=\sum_{x\in\mathbb{F}_2^n} f(x)(-1)^{x\cdot u} \tag{2.4}$$

where $x\cdot u$ denotes the usual inner product, which is

$$x\cdot u=\bigoplus_{i=1}^{n} x_1\cdot u_1.$$

There exists a simple divide-and-conquer algorithm to compute $\hat{f}$, which is called the *Fast Fourier Transform* (FFT). We list this algorithm below.

Algorithm 1 ([13])**:** For every $a = (a_1, \ldots, a_{n-1}) \in \mathbb{F}_2^{n-1}$ and every $a_n \in \mathbb{F}_2^n$ the following holds

$$\hat{f}(a_1, \ldots, a_n) = \sum_{(x_1,\ldots,x_{n-1}) \in \mathbb{F}_2^{n-1}} (-1)^{a \cdot x} \left[f(x_1, \ldots, x_{n-1}, 0) + (-1)^{a_n} f(x_1, \ldots, x_{n-1}, 1) \right].$$

Let the vectors in the truth-table of f be ordered in lexicographic order with the bit of highest weight on the right. Further let $g_0(x) = f(x_1, \ldots, x_{n-1}, 0) + f(x_1, \ldots, x_{n-1}, 1)$ and $g_1(x) = f(x_1, \ldots, x_{n-1}, 0) - f(x_1, \ldots, x_{n-1}, 1)$. Then the table of $\hat{f}$ equals the concatenation of the tables of the discrete Fourier transformation of the functions g_0 and g_1.

1. Write the truth-table of f, in which the binary vectors of length n are in lexicographic order as above.

2. Let f_0 be the restriction to $\mathbb{F}_2^{n-1} \times \{0\}$ and f_1 be the restriction to $\mathbb{F}_2^{n-1} \times \{1\}$. Now replace the values of f_0 by those of $f_0 + f_1$ and those of f_1 by $f_0 - f_1$.

3. Apply step 2 recursively to the functions which are now obtained in the places of f_0 and f_1.

This algorithm terminates after $n2^n$ steps, when the recursion in step 3 arrives in functions in one variable each. After these $n2^n$ steps the global table gives the values of $\hat{f}$.

Remark: Note that the value of $\hat{f}(0)$ equals the Hamming weight $\mathrm{w}_H(f)$. By computing $\widehat{f \oplus g}(0)$ we get the Hamming weight of $f \oplus g$, which is the number of elements of the set $\{x \in \mathbb{F}_2^n : f(x) \oplus g(x) \neq 0\} = \{x \in \mathbb{F}_2^n : f(x) \neq g(x)\}$. This is also called the *Hamming distance* d_H of f and g denoted by $\mathrm{d}_H(f, g)$.

Applying the above described discrete Fourier transformation to the sign function χ_f of f yields

$$\hat{f}_\chi(u) = \sum_{x \in \mathbb{F}_2^n} f_\chi(x)(-1)^{x \cdot u} = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus x \cdot u}. \tag{2.5}$$

The discrete Fourier transformation of the sign function of f is called the Walsh-transformation (or Hadamard-Walsh-transformation) of f.

Remark: [13] Since $f_\chi(u)$ equals $1 - 2f(u)$ the following holds for all $u \in \mathbb{F}_2^n$

$$\hat{f_\chi}(u) = 2^n \delta_0(u) - 2\hat{f}(u) \tag{2.6}$$

where δ_0 denotes the indicator function for the 0-vector, i.e. $\delta_0(u) = 1$ if and only if u equals the 0-vector. δ_0 is also called the *Dirac* symbol.
If we evaluate equation 2.6 at value 0 and transform it we get

$$\mathrm{w}_H(f) = \hat{f}(0) = 2^{n-1} - \frac{\hat{f_\chi}(0)}{2}. \tag{2.7}$$

Applying $f \oplus l_a$ to relation 2.7 where $l_a = a \cdot x$ gives:

$$\mathrm{d}_H(f, l_a) = \mathrm{w}_H(f \oplus l_a) = 2^{n-1} - \frac{\hat{f_\chi}(a)}{2}. \tag{2.8}$$

Example: Let $f = x_1 \oplus x_3 \oplus x_1x_3 \oplus x_2x_3$. Then the truth table, the discrete Fourier transform and the Walsh transform of f are listed in Table 2.4.

x_1	x_2	x_3	x_1x_3	x_2x_3	$f(x)$	$f_\chi(x)$	$\hat{f}(x)$	$\hat{f_\chi}(x)$
0	0	0	0	0	0	1	4	0
0	0	1	0	0	1	-1	0	0
0	1	0	0	0	0	1	2	-4
0	1	1	0	1	0	1	-2	4
1	0	0	0	0	1	-1	-2	4
1	0	1	1	0	1	-1	-2	4
1	1	0	0	0	1	-1	0	0
1	1	1	1	1	0	1	0	0

Table 2.4: Discrete Fourier transform and Walsh transform of the function f

We now want to take a look at some useful properties of the Fourier transformation.

Lemma 2.2.1 ([13])**:** Let E be any subspace of the vector space $\mathbb{F}_2^n$ and let l be any linear form on E which is not null. Then

$$\sum_{x \in E} (-1)^{l(x)} = 0. \tag{2.9}$$

Proof: Since the linear form l is not null, the support of l is an affine hyperplane of E and has $2^{\dim E - 1} = \frac{|E|}{2}$ elements. Hence, $\sum_{x \in E}(-1)^{l(x)}$ is a sum over an equal amount of 1's and -1's and therefore it is null. □

Theorem 2.2.2 ([13])**:** Let E be any subspace of the vector space $\mathbb{F}_2^n$ and let $\mathbb{1}_E$ be its indicator function, i.e. $\mathbb{1}_E(x) = 1$ if and only if $x \in E$. Then:

$$\widehat{\mathbb{1}_E} = |E|\, \mathbb{1}_{E^\perp}, \tag{2.10}$$

where $E^\perp$ denotes the orthogonal complement of E and is defined as $E^\perp = \{x \in \mathbb{F}_2^n | \forall y \in E : x \cdot y = 0\}$. In particular let $E = \mathbb{F}_2^n$ then $\widehat{\mathbb{1}_E} = 2^n \delta_0$.

Proof: Let $u \in \mathbb{F}_2^n$ arbitrary, then we have $\widehat{\mathbb{1}_E}(u) = \sum_{x \in E} (-1)^{x \cdot u}$. If for $x \in E$ the linear form $x \mapsto x \cdot u$ is not null on E then $\widehat{\mathbb{1}_E}(u)$ is null, according to Lemma 2.2.1. For the case that $x \cdot u = 0$, i.e. $u \in E^\perp$, then $\widehat{\mathbb{1}_E}(u) = |E|$. □

We will dedicate the next section to balanced Boolean functions.

2.3 Balanced Boolean functions

As we see in Section 2.5, *balanced* Boolean functions play an important role in cryptography. We start with a definition of *balancedness*. Informally speaking, balancedness means that a Boolean function yields each value in the image set equally often. We want to give a formal definition of balancedness as next.

Definition 2.3.1 ([13])**:** A Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is called *balanced* if it takes every value 2^{n-1} times.

Remark: Note that a Boolean function f is balanced if and only if $\hat{f}_\chi(0) = 0$.

When it comes to the design of cryptographic functions or the properties of the composition of Boolean functions it is not always obvious which compositions lead to balanced Boolean functions.
In the following we denote for a Boolean function f by $|f|^1 = \mathrm{w}_H(f)$ the Hamming weight of f and by $|f|^0 = \mathrm{w}_H(f \oplus 1) = 2^n - |f|^1$. Thus, we get for a balanced function f that $|f|^1 = |f|^0 = 2^{\frac{n}{2}}$.

We start how the property of balancedness is effected by the elementary Boolean operations AND, OR and NOT which are denoted by $f \cdot g$, $f \oplus g \oplus f \cdot g$ and $1 \oplus f$. Other elementary Boolean operations like NAND or NOR can easily be constructed out of the above mentioned functions.

Theorem 2.3.1 ([16])**:** Let $f_1 : \mathbb{F}_2^m \to \mathbb{F}_2$ and $f_2 : \mathbb{F}_2^n \to \mathbb{F}_2$ be any non-constant Boolean function. Further let X_1 and X_2 be the set of input variables for the functions f_1 and f_2 with $X_1 \cap X_2 = \emptyset$. Then $F(X_1, X_2) = g(f_1(X_1), f_2(X_2))$ is not balanced, if g is one of the functions AND, OR, NAND or NOR.

Proof: Let $|f_1|^1 = n_1$ and $|f_2|^1 = n_2$. Without loss of generality let g be the OR function. If $F(X_1, X_2)$ is balanced, the following equation must hold for the hamming weight of $f_1(X_1) \oplus f_2(X_2) \oplus f_1(X_1)f_2(X_2)$:

$$\begin{aligned} n_1 \cdot 2^n + n_2 \cdot 2^m - n_1 \cdot n_2 &= 2^{m+n-1} \\ n_1 \cdot 2^n + ()2^m - n_1) \cdot n_2 &= 2^{m+n-1} \\ 2^{n-1} - \frac{2^{n-1}}{\frac{2^m}{n_1} - 1} &= n_2. \end{aligned}$$

Since n_2 is an integer, this implies that $\frac{2^m}{n_1}$ is an odd number. However, this is impossible, since otherwise 2^m would be divisible by an odd number. Hence $f(X_1, X_2)$ can not be balanced. □

Next we show a necessary and sufficient condition for a Boolean function to be balanced.

Theorem 2.3.2 ([16])**:** A function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is balanced if and only if there exists a bijection π on the set $\mathbb{F}_2^n$ such that for all $\alpha \in \mathbb{F}_2^n$ the following two conditions hold:

1. $\pi(\alpha) \neq \alpha$ and
2. $f(\alpha) \neq f(\pi(\alpha))$.

The proof of this theorem does not lead to much insights on the composition properties of balanced functions and therefore is omitted here.
Next we want to take a look at disjunctive compositions of functions.

Theorem 2.3.3 ([16])**:** Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function with input set X_1. Let further be $g : \mathbb{F}_2^{m+1} \to \mathbb{F}_2$ be a Boolean function with input set $X_2 \cup \{p\}$ (i.e. $(p, X_2) \mapsto g(p, X_2)$) on which we require that $X_1 \cap (X_2 \cup \{p\}) = \emptyset$. Let $F(X_1, X_2) = g(f(X_1), X_2)$. If $f(X_1)$ and $g(p, X_2)$ are balanced, then $F(X_1, X_2)$ is also balanced.

Proof: Let $|X_1| = m$ and $|X_2| = n$. Since $f(X_1)$ and $g(p, X_2)$ are balanced, $|f|^1 = 2^{m-1}$ and $|g|^1 = 2^{(n+1)-1} = 2^m$. We have to show that $|F|^1 = 2^{m+n-1}$. Let therefore $k^g_{p^d}$ be the Hamming weight of g with value $d \in \mathbb{F}_2$ assigned to the variable p, that is $k^g_{p^d} = \mathrm{w}_H(g(d, X_2))$. By using this notation the Hamming weight of $F(X_1, X_2)$ is given by

$$\begin{aligned} |F|^1 &= k^g_{p^0} \cdot |f|^1 + k^g_{p^1} \cdot |f|^1 = |f|^1 \cdot \left(k^g_{p^0} + k^g_{p^1}\right) \\ &= 2^{m-1} \cdot |g|^1 = 2^{m+n-1}. \end{aligned}$$

$\Rightarrow F(X_1, X_2)$ is a balanced function. □

This theorem and its proof can easily be extended for the case of k balanced functions $f_1, \dots f_k$ instead of the function f. The requirement $X_1 \cap X_2 = \emptyset$ has of course to be generalized to $X_i \cap X_j = \emptyset$ for all pairs $i \neq j$ of input sets of the functions.
We now want to bring up a small example which illustrates the theorem above

Example: Let $f(a,b) = a \oplus b$ and $g(p,c,d) = p \oplus p \cdot c \oplus d$. The sets X_1 and X_2 are $X_1 = \{a,b\}$ and $X_2 = \{c,d\}$. The composition of the functions $F(a,b,c,d) = g(f(a,b),c,d) = a \oplus b \oplus a \cdot c \oplus b \cdot c \oplus d$. By evaluating all 2^4 possible values for $\{a,b,c,d\}$ it can easily be verified that F is a balanced function.

We now want to take a look at disjunctive compositions in which one of the involved functions is not balanced.

Definition 2.3.2 ([16])**:** A Boolean function $f(x_1, \dots, x_n)$ is said to be balanced with respect to x_i if and only if

$$\begin{aligned} \mathrm{w}_H(f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)) &= \mathrm{w}_H(f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n)) \\ &= \frac{1}{2}\mathrm{w}_H(f(x_1, \dots, x_n)). \end{aligned}$$

Theorem 2.3.4 ([16])**:** Let $f(X_1)$ be an unbalanced Boolean function and $g(p, X_2)$ be a balanced Boolean function with $X_1 \cap X_2 = \emptyset$. Then $F(X_1, X_2) = g(f(X_1), X_2)$ is balanced if and only if $g(p, X_2)$ is balanced with respect to p.

Proof: We start the proof with direction $\Rightarrow$. As in the proof of Theorem 2.3.3 let k_p^g denote the Hamming weight of $g(1, X_2)$. Suppose $F(X_1, X_2)$ is balanced. Then $|F(X_1, X_2)|^1 = 2^{m+n-1}$ where $|X_1| = m$, $|X_2| = n$ and

$$k_p^g \, |f|^1 + (2^n - k_p^g) \cdot (2^m - |f|^1) = 2^{m+n-1}$$

which yields $k_p^g \cdot (2^m - 2\,|f|^1) = 2^{m+n-1} - 2^n \, |f|^1$. Since $f(X_1)$ is unbalanced $2^{m+n-1} - 2^n \, |f|^1 \neq 0$ and we can divide both sides by $2^m - 2\,|f|^1$ and get

$$k_p^g = \frac{2^{m+n-1} - 2\,|f|^1}{2^m - 2\,|f|^1} = 2^{n-1}.$$

Since the Hamming weight of $\mathrm{w}_H(g(p, X_2)) = 2^n$ we just showed that $g(p, X_2)$ is balanced with respect to p.
The direction $\Leftarrow$ remains to prove. Suppose $g(p, X_2)$ is balanced with respect

to p. Then $k_p^g = 2^{n-1}$ and the Hamming weight of $F(X_1, X_2)$ can be written as:

$$\begin{aligned} |F(X_1, X_2)| &= k_p^g \, |f|^1 + (2^n - k_p^g) \cdot (2^m - |f|^1) \\ &= 2k_p^g \, |f|^1 + 2^{m+n} - k_p^g 2^m - 2^n \, |f|^1 \\ &= 2^{m+n-1}. \end{aligned}$$

Therefore we get that $F(X_1, X_2)$ is balanced. □

A simple corollary out of Theorem 2.3.4 is the following one:

Corollary 2.3.5 ([16])**:** If $f(X)$ is an unbalanced Boolean function and $c \notin X$. Then the function $g(X, c) = f(X) \oplus c$ is balanced.

Theorem 2.3.6 ([16])**:** Let $F(X_1, X_2) = g(f(X_1), X_2)$ with $X_1 \cap X_2 = \emptyset$ and $|X_2| = n$. Let $g(p, X_2)$ be balanced with respect to p. Then $F(X_1, X_2)$ is balanced if and only if $g(p, X_2)$ is balanced.

Proof: It suffices to prove sufficiency, since the necessity follows from Theorems 2.3.3 and 2.3.4. Let $F(X_1, X_2)$ be balanced and let $|X_1| = m$. Then of course $|F(X_1, X_2)|^1 = 2^{m+n-1}$ and:

$$\begin{aligned} 2^{m+n-1} &= k_p^g \, |f|^1 + (2^m - |f|^1) \cdot (|g|^1 - k_p^g) \\ &= 2^n \, |f|^1 + 2^m \, |g|^1 - |f|^1 \, |g|^1 - 2^{m+n-1}. \end{aligned}$$

This gives us

$$|g|^1 \, (2^m - |f|^1) = 2^{m+n-1} - 2^n \, |f|^1 .$$

Since f is non-constant $|f|^1 \neq 2^m$ and therefore $(2^m - |f|^1) \neq 0$. So we can divide by $(2^m - |f|^1)$ and finally get $|g|^1 = 2^n$, which is equivalent to the balancedness of $g(p, X_2)$. □

As next we want to consider non-disjunctive compositions of Boolean functions.

Theorem 2.3.7 ([16])**:** Let $F(X_1 \cup X_2) = g(f(X_1), X_1 \cap X_2, X_2 \setminus X_1)$ be a Boolean function. Further let for all combinations λ of $X_1 \cap X_2$ let

$$|g(1, \lambda, X_2 \setminus X_1)|^1 = |g(0, \lambda, X_2 \setminus X_1)|^1 .$$

Then the function $F(X_1 \cup X_2)$ is balanced if and only if $g(p, X_2)$ is balanced with respect to p.

Proof: Let $X_1 \cap X_2 = \{x_1, \ldots, x_k\}$ and let $d_1, \ldots, d_k \in \mathbb{F}_2$. Further let $|X_1| = m$ and $|X_2| = n$. The Hamming weight of F can be expressed as

$$|F|^1 = \sum_{(d_1,\ldots,d_k) \in \mathbb{F}_2^k} \left(k^g_{p, x_1^{d_1}, \ldots, x_k^{d_k}} \cdot k^f_{x_1^{d_1}, \ldots, x_k^{d_k}} + k^g_{p \oplus 1, x_1^{d_1}, \ldots, x_k^{d_k}} \cdot k^{f \oplus 1}_{x_1^{d_1}, \ldots, x_k^{d_k}} \right) \quad (2.11)$$

Since for all combinations of $(d_1, \ldots, d_k) \in \mathbb{F}_2^k$ $|g(1, d_1, \ldots, d_k, X_2 \setminus X_1)| = |g(0, d_1, \ldots, d_k, X_2 \setminus X_1)|$ holds, the equation 2.11 simplifies to:

$$\begin{aligned} |F|^1 &= \sum_{(d_1,\ldots,d_k)\in\mathbb{F}_2^k} \left(k^g_{p,x_1^{d_1},\ldots,x_k^{d_k}} \left(k^f_{x_1^{d_1},\ldots,x_k^{d_k}} + k^{f\oplus 1}_{x_1^{d_1},\ldots,x_k^{d_k}} \right) \right) \\ &= 2^{m-k} \cdot \sum_{(d_1,\ldots,d_k)\in\mathbb{F}_2^k} k^g_{p,x_1^{d_1},\ldots,x_k^{d_k}} \\ &= 2^{m-k} k_p^g. \end{aligned}$$

So we can deduce that the function $F(X_1 \cup X_2)$ is balanced if and only if $2^{m-k}k_p^g = 2^{m+n-k-1}$ which is equivalent to the balancedness of $g(p, X_2)$ with respect to p. □

The next corollary follows directly from Theorem 2.3.7.

Corollary 2.3.8 ([16])**:** Let $f(X_1)$ be a Boolean function. Then $f(X_1) \oplus X_2$ is balanced if $X_2 \setminus X_1 \neq \emptyset$.

The following theorem makes a similar statement like Theorem 2.3.1, but for non-disjunctive compositions of functions.

Theorem 2.3.9 ([16])**:** Let $f_1(X_1)$ and $f_2(X_2)$ be any non-constant Boolean functions and let $X_1 \cap X_2 = x$. Then $g(f_1(X_1), f_2(X_2))$ is unbalanced if g is one of the functions AND, OR, NAND or NOR and $f_1, f_2 \notin \{x, 1 \oplus x\}$.

The proof of this theorem can be found in [16]. In the next Section we will give a short overview about vectorial Boolean functions

2.4 Vectorial Boolean functions

In this section we discuss *vectorial* Boolean functions. Vectorial Boolean functions are functions which map from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. For the case $m = 1$ Boolean functions, which only produce a single output, result and are of course included in the set of all vectorial Boolean functions.

Notation: Let F be a vectorial Boolean function mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. Then we call F an (n, m) function.

For an (n, m) function F, the Boolean functions $(f_1, \ldots, f_m)$ define the function F, i.e. $F(x) = (f_1(x), \ldots, f_m(x))$. The functions $(f_1, \ldots, f_m)$ are called the *coordinate* functions of F.

The representation of (n, m) can easily be extended from Boolean functions. For Boolean functions we listed the representation in *truth-table* and

in *algebraic normal form* (ANF). The representation via truth table extends to a table of size $2^n \times m$ and the ANF of an (n, m)-function is the ANF of its coordinate functions.
The *algebraic degree* of an (n, m)-function is by definition the global degree of its ANF and is therefore the maximum degree of its coordinate functions.

Definition 2.4.1 (VecCarlet10)**:** Let F be an (n, m) function and $v \in \mathbb{F}_2^{m*}$, where $\mathbb{F}_2^{m*}$ denotes the vector space $\mathbb{F}_2^m$ without the 0 vector, i.e. $\mathbb{F}_2^{m*} = \mathbb{F}_2^m \setminus \{0\}$. Then $v \cdot F$ is a component function of F. The set of all component functions of F is spanned by the coordinate functions of F, without the null function if the coordinate functions are $\mathbb{F}_2$-linearly independent.

As in the previous section we want to define the *Walsh transform* of vectorial Boolean functions.

Definition 2.4.2 ([14])**:** The *Walsh transform* of a vectorial Boolean function F is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. It maps any ordered pair $(u, v) \in \mathbb{F}_2^n \times \mathbb{F}_2^{m*}$ to the value of the Walsh transform of the component $v \cdot F$.

$$(u, v) \in \mathbb{F}_2^n \times \mathbb{F}_2^{m*} \mapsto \widehat{v \cdot F_\chi}(u) = \sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot F(x) \oplus x \cdot u} \tag{2.12}$$

The multiset of values of the Walsh transform of F, which can be seen as a matrix

$$\left(\widehat{v \cdot F_\chi}(u)\right)_{\substack{u \in \mathbb{F}_2^n \\ v \in \mathbb{F}_2^{m*}}},$$

is called the *Walsh spectrum* of F. The multiset of the absolute values of the Walsh spectrum of F is called the *extended Walsh spectrum* and the set of (u, v) such that $\widehat{v \cdot F_\chi}(u) \neq 0$ is called the *Walsh support* of F.

For Boolean functions we introduced in definition 2.1.4 the set of affine coordinate transformations. For (n, m)-functions we can transform both sides of the function, this means we can not only transform the input variables as in the case of Boolean functions, but also the output of the (n, m)-function. We call an affine transformation *left affine transformation* if we compose F with an affine automorphism on the left. The same notation holds for *right affine transformations*.

As for Boolean functions, we want to give the definition for a *balanced* vectorial Boolean function.

Definition 2.4.3 ([14])**:** Let F be an (n, m) function. Then F is *balanced*, if it takes every value in $\mathbb{F}_2^m$ the same number of times, that is 2^{n-m}.

Note that if $n = m$, which is the case for most S-boxes, balanced is equivalent to *bijective*.

Theorem 2.4.1 ([33])**:** An (n, m)-function is balanced if an only if its component functions are balanced. That is, if and only if, for every $v \neq 0$, $v \in \mathbb{F}_2^m$ the function $v \cdot F$ is balanced.

Proof: For $b \in \mathbb{F}_2^m$ let φ_b be the indicator function of $F^{-1}(b) = \{x \in \mathbb{F}_2^n : F(x) = b\}$. Then the following relation is true for all (n, m) functions, all $x \in \mathbb{F}_2^n$ and all $b \in \mathbb{F}_2^m$:

$$\sum_{v \in F_2^m} (-1)^{v \cdot (F(x)+b)} = \begin{cases} 2^m, & \text{if } F(x) = b \\ 0, & \text{otherwise} \end{cases} = 2^m \varphi_b(x). \tag{2.13}$$

This holds, since the function $v \mapsto v \cdot (f(x) + b)$ is linear and therefore it is either balanced or null. Thus

$$\sum_{x \in \mathbb{F}_2^n} \sum_{v \in F_2^m} (-1)^{v \cdot (F(x)+b)} = 2^m \left| F^{-1}(b) \right| = 2^m \mathrm{w}_H(\varphi_b). \tag{2.14}$$

Hence the discrete Fourier transform of the function $v \mapsto \sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot F(x)}$ equals the function $b \mapsto 2^m \left| F^{-1}(b) \right|$. We know from the previous section that a Boolean function has constant Fourier transform if and only if it is null at every nonzero vector. We therefore can deduce that F is balanced if and only if the function $v \mapsto \sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot F(x)}$ is null for all $v \in \mathbb{F}_2^m \setminus \{0\}$. □

In the next section we will discuss criteria for Boolean functions when used in cryptography.

2.5 Criteria for Boolean functions in cryptography

In this section we want to list some main cryptographic properties of Boolean functions. All these properties aim at the two fundamental principles in the design of cryptographic systems, which were introduced by Shannon [60] more than half a decade ago. These two principles are *confusion* and *diffusion*. Confusion scopes at masking any algebraic structure of the system and is closely related to the complexity of the involved Boolean function. The second principle, diffusion, aims at distributing any minor change in the input, like the data or the key, over all outputs. To quantify the resistance of a cryptographic system against the known attacks, fundamental characteristics were introduced [40, 53, 61]. We start with the *algebraic degree* in Section 2.5.1 and continue with the *nonlinearity* in Section 2.5.2. In Sections 2.5.3 and 2.5.4 we review the property of balancedness and list other criteria.

2.5.1 Algebraic degree

We start with the *algebraic degree* of Boolean functions. Since Boolean functions with low algebraic degree make the higher differential attack [29, 32] easier, functions with high algebraic degree are preferred.
It is known that when n, the number of variables of the Boolean function, tends to infinity, random Boolean functions almost surely have algebraic degree at least $n-1$. This holds because the number of Boolean functions of degree at most $n-2$ equals $\sum_{i=0}^{n-2} {}^{n}/_{i} = 2^{2^n-n-1}$ and is negligible compared to the number 2^{2^n} of all Boolean functions. It should also be noted, that the algebraic degree is invariant under affine transformations.

2.5.2 Nonlinearity

The next characteristic which is important to Boolean functions in cryptography is the so called *nonlinearity*. The nonlinearity is a measure of the distance from the Boolean function to all affine functions. We say that there is a correlation between a Boolean function f and a linear function l if $\mathrm{d}_H(f,l)$ is different from 2^{n-1}. It should be noted that any Boolean function has correlation with some linear function. This correlation should be small since the existence of affine approximations of the Boolean function which are involved in an cryptosystem allows to build attacks on this system. This field of research is called *linear cryptanalysis* [83, 39]. The *nonlinearity* of a Boolean function f is the Hamming distance between f and any affine function. To counter attacks the nonlinearity of a cryptographic function must be high and is surely one of the most important cryptographic criteria. It can easily be seen that the nonlinearity also is an affine invariant.
To compute the nonlinearity of a function f we investigate the Walsh-transform of f: let $l_a = a \cdot x$ be any linear function. According to relation 2.8 we have that $\mathrm{d}_H(f,l_a) = 2^{n-1} + \frac{1}{2}\hat{f}_a(a)$ and we deduce

$$\mathrm{d}_H(f, l_a \oplus 1) = 2^{n-1} + \frac{1}{2}\hat{f}_a(a).$$

So we get that the nonlinearity of f is equal to

$$\mathrm{nl}(f) = 2^{n-1} - \frac{1}{2}\max_{a\in\mathbb{F}_2^n}\left|\hat{f}_\chi(a)\right|. \tag{2.15}$$

We see that a function has high nonlinearity if all the values of the Walsh-transform have small absolute value. It can be shown that $\max_{a\in\mathbb{F}_2^n}\left|\hat{f}_\chi\right| \geq$

$2^{n/2}$ and this implies that

$$\mathrm{nl}(f) \leq 2^{n-1} - 2^{\frac{n}{2}-1}. \tag{2.16}$$

Nonlinearity of a vectorial Boolean function

We now want to take a look at the *nonlinearity* of vectorial Boolean functions. The notion of nonlinearity has been generalized by Nyberg [46].

Definition 2.5.1 ([46])**:** The *nonlinearity* $nl(F)$ of an (n, m)- function F is the minimum nonlinearity of all the component functions of F. More formally this is

$$nl(F) := \min_{v \in \mathbb{F}_2^{m*}} nl(v \cdot F).$$

So the nonlinearity of an (n, m) function F is the minimum Hamming distance between the component functions of F and all affine functions on n variables. The nonlinearity is often used as a quantification of the resistance of an S-box against a linear attack. The nonlinearity of a (n, m)-function is a left and right affine invariant. Another property of the nonlinearity is that in the case that F is a permutation, we have that $nl(F) = nl(F^{-1})$. By taking a look at the definition of the nonlinearity $nl(F)$ we see why this holds:

$$nl(F) = 2^{n-1} - \frac{1}{2} \max_{\substack{(u,v) \in \mathbb{F}_2^n \times \mathbb{F}_2^n \\ (u,v) \neq (0,0)}} \left| \sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot F(x) \oplus u \cdot x} \right|.$$

By exchanging x with F^{-1} we get the desired result.
The reason why a high nonlinearity is important for block ciphers lies in the *linear attack* introduced by Matsui [39]. The linear attack tries to find good linear approximations of the involved functions in the block cipher. It therefore uses triples (α, β, γ) of binary strings such that, a block m of plaintext and a key k being randomly chosen. The bit $\alpha \cdot m \oplus \beta \cdot c \oplus \gamma \cdot k$ is null with probability different from $1/2$. The attack is more efficient if the probability is more distant to $1/2$. These above illustrated linear approximations work less good, the higher the nonlinearity of the block cipher is.

2.5.3 Balancedness

Another important property of Boolean functions, when used in cryptography, is balancedness. We want to recall definition 2.3.1.

Definition 2.5.2 ([13])**:** A Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is called *balanced* if it takes every value 2^{n-1} times.

Though not so important for block ciphers, we want to state a generalization to balancedness, the so called *resiliency*.

Definition 2.5.3 ([13])**:** Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function and let further be $m \in \mathbb{N}$. The function f is called an $m-$*resilient* function if any of its restrictions obtained by fixing at most m of its input coordinates x_i is balanced.

We also want to state how to analyze the m-resiliency of a given Boolean function f.

Theorem 2.5.1 ([83])**:** Let f be an n variable Boolean function. Then f is m-resilient if and only if $\hat{f}_\chi(u) = 0$ for all $u \in \mathbb{F}_2^n$ such that $\mathrm{w}_H(u) \leq m$. Equivalently, f is m-resilient if and only if f is balanced and $\hat{f}(u) = 0$ for all $u \in \mathbb{F}_2^n$ suc that $0 < \mathrm{w}_H(u) \leq m$.

2.5.4 Other criteria

There exist many more cryptographic criteria like the *Strict Avalanche Criterion* [78] introduced by Webster and Tavares or the *algebraic immunity* of a Boolean function which is the minimum degree of a nonzero Boolean function such that $f \cdot g = 0$ (this means that g is an annihilator of f) or $(f \oplus 1)g = 0$ (this is equivalent to that g is a multiple of f). For further information on the topic of cryptographic criteria of Boolean functions we may refer to the Chapter about Boolean functions [13] in the Book [18].

Chapter 3

Side-Channel Analysis

Several measures to quantify the security of a cryptographic algorithm have been proposed in the literature [66]. Of course these criteria only cover the properties of the algorithm itself, but not its implementation. Alas, it turned out that even if a cryptographic algorithm is secure in a theoretical manner, it still can be attacked by a number of practical attacks on its implementation. One class of these attacks are so called side-channel attacks which we will discuss in this chapter.

The following chapter is organized as follows. In Section 3.1 we give a short overview on implementations of cryptographic algorithms and attacks on hardware devices. In Section 3.2 we present the simple power analysis, a method introduced by Kocher in [30]. In Section 3.3 we focus on the differential power analysis, also introduced by Kocher in [31]. Finally we give a short overview on countermeasures against side-channel attacks in Section 3.4.

3.1 Introduction

In this section we shortly discuss the topic of hardware implementations of cryptographic algorithms, give a common classification of attacks on hardware devices and regard the power consumption of a CMOS gate.

3.1.1 Hardware in cryptography

Security in modern systems needs to achieve three goals: confidentiality integrity and authenticity [41]. In modern systems this is achieved by using cryptographic algorithms like the Data Encryption Standard (DES) [57] or

more commonly the Advanced Encryption Standard (AES) [22]. A block cipher is a mathematical function which transforms some data, called plaintext, by the use of some secret information, called key, into random-looking text, called ciphertext. Due to Kerkhoff's principle the details about the cryptographic algorithm are assumed to be known. The only information which is kept secret is the key. For a good first overview about the topic of cryptography we refer to the book by Menezes et al. [41] or the book by Schneier [57].
To execute a cryptographic algorithm, usually a computer or specially designed hardware is used. We differentiate between two sorts of cryptographic devices. The first type is called dedicated cryptographic hardware, like a smart card, whose only purpose it is to perform cryptographic operations. The second type is called general purpose hardware, like a field programmable gate array (FPGA). The other method to implement a cryptographic algorithm on a computer is by the use of cryptographic software, for example the Java^{TM}cryptographic extension library (Java-JCE) [1].

3.1.2 Types of attacks on cryptographic hardware

In the following we give a short classification of attacks on cryptographic devices. We generally distinguish between the following two criteria. The first criterion is whether an attack is active or passive.

- **Active attacks:** In an active attack the attacker tries to reveal or extract the secret information of a cryptographic device by interfering the normal operation. This can either be done in an invasive or non-invasive way.

- **Passive attack:** In a passive attack the attacker only observes the behaviour of the cryptographic device. For example an attacker could measure the power consumption or execution time of the device during its operation.

The second criterion is the interface which is used on the cryptographic device to attack the device.

- **Invasive attack:** In an invasive attack, the attacker typically starts with decomposing the device under attack. By the use of a probing device, an attacker can get access to the different parts of the device, which is usually an integrated circuit. An invasive attack can be passive, if the probing station is used only to observe the signals. In contrary an invasive attack is active if the attacker tries to induce faults over the probes on the device.

Invasive attacks form the most powerful class which an attacker can mount on a cryptographic device. However only few publications exist on this topic, since they normally require expensive hardware. Important publications are for example from Anderson [5] and Skorobogatov [62].

- **Semi-invasive attack:** In a semi-invasive attack the device is also demounted, but in contrast to an invasive attack, no electrical contact to the device is made.

  In a passive semi-invasive attack the attacker usually tries to read out memory cells of the device without probing the device itself. An attack that has been performed in this way has been published by Samyde et al. in [56].
  In an active semi-invasive attack the attacker usually tries to induce faults in the device under attack. Common methods for provoking these faults are X-rays, electromagnetic fields or light. A successful attack using light to induce faults on the device has been published by Sundström and Alvandpour in [67].

- **Non-invasive attack:** In a non-invasive attack the attacker normally operates with the device as it is. This means that the attacker only uses directly accessible interfaces. Most importantly to note is that there is no evidence left behind after the attack has been performed. Clearly these attacks form a serious threat to cryptographic devices.
  In a passive non-invasive attack, usually called *side-channel* attack, the attacker tries to reveal the secret of a device by measuring the execution time, the power consumption or its electromagnetic field. These three different side-channels lead to timing attacks [30], power analysis attacks [31] and electromagnetic attacks [23, 54] respectively.
  In an active non-invasive attack the attacker usually tries to induct faults through the accessible interfaces. This can be done for example by clock glitches or power glitches. For a survey on this kind of attacks we refer to [6].

In this thesis we only consider passive non-invasive attacks on the cryptographic devices which implement the cryptographic algorithms, or mathematical functions on which we investigate.

3.1.3 Power consumption

In this section we want to take a closer look into the power consumption of digital circuits. These circuits like FPGAs or Application Specific Integrated

Circuits (ASICs) all contain basic modules called logic cells. These logic cells take one or more logic inputs and map these to logic values on the output, according to the function these cells represent. To implement logic cells in hardware, complementary metal-oxide semiconductor (CMOS) [77] transistors are commonly used.

For CMOS based integrated circuits it is known, that the total power consumption of the circuit is the sum of the power consumptions of all logic cells that make up the circuit. Hence, the total power consumption strongly depends on the number of logic cells and the connections between these cells. We illustrate below the model of an inverter as the simplest form of a logic cell. Figure 3.1 displays the model of a CMOS inverter. The circuit is powered by a constant power supply V_{DD}. The input of the circuit is denoted by a and the output is denoted by q.

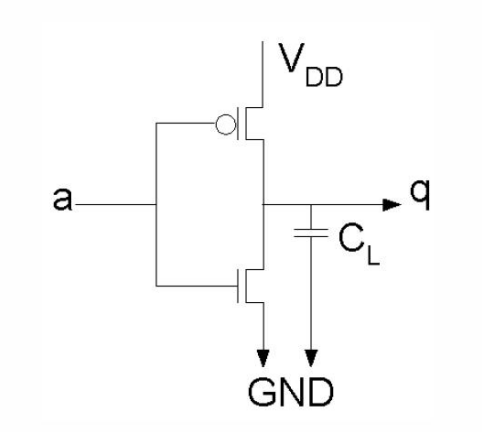


Figure 3.1: Schematic representation of a CMOS inverter with lumped capacitances. [36]. The circuit is provided by a constant supply voltage V_{DD} and has the input signal a and output signal q. The lumped capacitances are denoted by C_L.

The power consumption of such an inverter can be divided into two parts. The static power consumption P_{stat}, which is consumed by the circuit when there is no switching activity. The second part is the dynamic power consumption P_{dyn} which is consumed by the circuit when there is switching activity in the circuit. In Table 3.1 we list the four transitions that can occur at the node q. The node q has the value q_{t-1} at time $t-1$ and has the value q_t at time t. The power that is consumed to perform the transition is denoted by $P_{00} \dots P_{11}$. Further we denote by $p_{00} \dots p_{11}$ the probability that these transitions occur.

In standard CMOS logic the power consumptions P_{00} and P_{11} are nearly

q_{t-1}	q_t	Power	Probability
0	0	P_{00}	p_{00}
0	1	P_{01}	p_{01}
1	0	P_{10}	p_{10}
1	1	P_{11}	p_{11}

Table 3.1: Transitions of the value q in a CMOS gate at time t.

the same. Due to the characteristic of the circuit, the power consumptions P_{01} and P_{10} are not equal. Further it is known that

$$P_{00} \approx P_{11} \ll P_{10} \neq P_{01}. \tag{3.1}$$

Hence an attacker can use this knowledge (3.1) for an attack. For more details on how to use this information we refer to Section 3.2 and Section 3.3 where Simple Power Analysis and Differential Power Analysis are discussed. For a thorough analysis of the characteristic power consumption of the CMOS inverter we refer to the book of Mangard et al. [36].

3.2 Simple power analysis

Simple power analysis (SPA) has first been introduced by Kocher et al. in [31]. The idea of a simple power analysis attack is, to find a direct connection between the power trace of a cryptographic device and the processed values. Usually SPA works with very few power traces but therefore needs a detailed knowledge about the implementation of the cryptographic algorithm. We start with a description of SPA attacks.

3.2.1 Description

As already noted above, an SPA attack operates on power traces of a cryptographic device. In an SPA attack the following fact is exploited. When the cryptographic device encrypts some data d with the key k the power consumption of the device becomes a function $f(d, k)$ of the key and the processed data. If there is enough knowledge about the implementation, the attacker can investigate on the points of interest in the power trace, where the data gets processed.
Normally an SPA operates with very few or in the very extreme just one power trace. If there is only one power trace, the attack is called single-shot SPA. If there are a few traces available the attack is called multiple-shot SPA. On the multiple-shot SPA we can further distinguish between multiple

power traces with the same plaintext, which gets processed multiple times or multiple power traces with different plaintext.

To deduce the key or parts of the key in an SPA, the attacker usually has to derive the key from the power trace by a visual inspection. Examples of SPA attacks can for example be found in the book of Mangard et al. [36].

3.3 Differential power analysis

The differential power analysis (DPA) is the most common side-channel attack. The reason is, that no detailed information about the specific implementation or detailed knowledge about the hardware is needed. We conquer this lack of knowledge with statistical tools and, in contrast to SPA, with a large number of power traces.

We start in Section 3.3.1 with a general description of the method. In Section 3.3.2 we describe how we use the correlation coefficient in a DPA attack. We discuss alternatives to the correlation coefficient in a DPA attack in Section 3.3.3 and describe distribution based attacks in Section 3.3.4. Finally we describe higher order DPA attacks in Section 3.3.5.

3.3.1 Description

We start with a short description of the method. The idea behind a DPA attack is to find out how the power consumption at fixed moments of time depends on the processed data. To find this dependence we need a huge number of power traces. The advantage of a DPA attack, in contrast to an SPA attack, is that we do not need detailed information about the device. It suffices to know which cryptographic algorithm is used. The disadvantage is that the attacker needs to posses the device for a longer period of time to gain the large number of traces.

We now discuss the general 5 steps of a DPA attack [36].

- **Step 1 – Choose an intermediate result of the algorithm:** In the first step we choose an intermediate result of the algorithm which is used inside the device. The intermediate result needs to be a function $f(d, k)$ where d is some known data and k is the key or at least some part of the key. Note that we need to know which data d is processed in this step.

- **Step 2 – Measure the power consumption:** In this step we measure the power consumption of the device while it processes D different data blocks. The processed data d must be known for all D recorded power traces. These known data values get written in a vector $\mathbf{d} = (d_1, \ldots, d_D)$, where d_i is the processed data from the i-th run. We refer to the recorded power trace that corresponds to the processed data block d_i with $\mathbf{t_i} = (T_{i,1}, \ldots, T_{i,T})$ where T is the length of the trace. We refer to the set of vectors $\mathbf{t_i}$ with the $D \times T$ matrix $\mathbf{T}$. Note that it is mandatory that the power traces, that are written in the same column in the matrix $\mathbf{T}$ are caused by the same operation of the algorithm.

- **Step 3 – Calculate hypothetical intermediate values:** In this step we calculate for each possible choice of k a hypothetical intermediate value. We further write all possible values for k into the vector $\mathbf{k} = (k_1, \ldots, k_K)$. K denotes the total number of possible values for the key k. This vector $\mathbf{k}$ is normally called key hypotheses. Given the vector $\mathbf{d}$ of data values and the vector $\mathbf{k}$ of key hypotheses, we can easily calculate hypothetical intermediate values $v_{i,j} = f(d_i, k_j)$ for all D process runs and all K key hypotheses. This calculation results in a $D \times K$ matrix $\mathbf{V}$. Since we calculated the intermediate results for all key hypotheses k_j, one of the columns of $\mathbf{V}$ contains the intermediate values that were calculated by the real device during the D runs. We refer to this column with ck, hence k_{ck} is the key which is used inside the device. So the goal for the further steps is to determine which column of $\mathbf{V}$ has been processed by the device.

- **Step 4 – Map intermediate values to power consumption values:** In this step we map the hypothetical intermediate values $\mathbf{V}$ to a map of hypothetical power traces $\mathbf{H}$. To do so, we simulate the power consumption of the device when it processes the intermediate results $v_{i,j}$. Out of this simulation we obtain the hypothetical power values $h_{i,j}$, which we write into a $D \times K$ matrix $\mathbf{H}$.

- **Step 5 – Comparison of hypothetical power values with the real power consumption:** The final step of a DPA attack consists of a comparison between the columns $\mathbf{h}_i$ of the matrix $\mathbf{H}$ with each column $\mathbf{t}_j$ of the matrix $\mathbf{T}$. This means, that we compare the hypothetical power traces for each possible key k with the recorded power trace. We store the result of this comparison in a matrix $\mathbf{R}$ which has size $K \times T$. The matrix $\mathbf{R}$ consists of entries $r_{i,j}$ which are the result of the comparison between $\mathbf{h}_i$ and $\mathbf{t}_j$. The comparison is based on methods which are discussed later in this section. The only important detail to

note about these methods is, that the value $r_{i,j}$ is higher, the better the compared vectors match.

The intermediate value that has been chosen in step 1 needs to be processed by the device at some time. We refer to to this position of the power traces with *ct*. Hence column $\mathbf{t}_{ct}$ contains the power consumption values that correspond to the intermediate values $\mathbf{v}_{ck}$. Based on these values $\mathbf{v}_{ck}$ the hypothetical power consumption values $\mathbf{h}_{ck}$ have been simulated. Hence the columns $\mathbf{h}_{ck}$ and $\mathbf{t}_{ct}$ are strongly related. Accordingly the value $r_{ck,ct}$ is the highest value in the matrix $\mathbf{R}$.
Note that in practice it is possible that all values of $\mathbf{R}$ are low. In this case the number D of power traces is usually just too low. If there are more elements in the matrices $\mathbf{H}$ and $\mathbf{T}$, the relationship between the matrices can be determined more precisely.

In the next sections we describe how to determine the relationship between the columns $\mathbf{h}_i$ and $\mathbf{t}_j$.

3.3.2 Attacks based on the correlation coefficient

In this section we discuss the most common way to measure the linear relation between $\mathbf{h}_i$ and $\mathbf{t}_j$, the so called correlation coefficient. There exists a well established theory for the correlation coefficient which can be used to model statistical properties of a DPA attack. We start with a definition of the correlation coefficient.

Definition 3.3.1 ([36])**:** Let x and y be two random variables. Then the correlation ρ between x and y is

$$\rho(x,y) = \frac{\mathrm{cov}(x,y)}{\sqrt{\mathrm{Var}(x)\mathrm{Var}(y)}} = \frac{\mathbb{E}\left[(x-\mathbb{E}[x])\cdot(y-\mathbb{E}[y])\right]}{\sqrt{\mathrm{Var}(x)\mathrm{Var}(y)}} \tag{3.2}$$

To use the correlation coefficient in a DPA attack, we calculate the correlation coefficients between the vectors $\mathbf{h}_i$ and $\mathbf{t}_j$ for $i \in \{1,\ldots,K\}$ and $j \in \{1,\ldots,T\}$ in step 5. This gives the linear dependence between the vectors $\mathbf{h}_i$ and $\mathbf{t}_j$. We now give a formula to calculate the values $r_{i,j}$ of the matrix $\mathbf{R}$.

$$r_{i,j} = \frac{\sum_{d=1}^{D}(h_{d,i}-\mathbb{E}[\mathbf{h}_i])\cdot(t_{d,j}-\mathbb{E}[\mathbf{t}_j])}{\sqrt{\sum_{d=1}^{D}(h_{d,i}-\mathbb{E}[\mathbf{h}_i])^2\cdot\sum_{d=1}^{D}(t_{d,j}-\mathbb{E}[\mathbf{t}_j])^2}} \tag{3.3}$$

Since we do not know the true distribution of $\mathbf{h_i}$ and $\mathbf{t}_j$ we replace the values $\mathbb{E}[\mathbf{h}_i]$ and $\mathbb{E}[\mathbf{t}_j]$ with the estimator which is the mean value of the sample $h_{d,i}$

and $t_{d,j}$. We denote the mean values of $\mathbf{h}_i$ and $\mathbf{t}_j$ with $\bar{h}_i$ and $\bar{t}_j$. Hence we get the following formula for the values $r_{i,j}$:

$$r_{i,j} = \frac{\sum_{d=1}^{D}(h_{d,i} - \bar{h}_i) \cdot (t_{d,j} - \bar{t}_j)}{\sqrt{\sum_{d=1}^{D}(h_{d,i} - \bar{h}_i)^2 \cdot \sum_{d=1}^{D}(t_{d,j} - \bar{t}_j)^2}} \tag{3.4}$$

There has been numerous research on DPA attacks, for example by Örs et al. [47] who attack ASIC AES implementations. In [63] Standaert et al. attack standard FPGA implementations of the DES. Tillich and Herbst attack software countermeasures against side-channel analysis in [69].

In the following section we describe alternatives to the correlation coefficient.

3.3.3 Attacks based on alternatives

We now present some alternatives to the correlation coefficient to determine the relation between the vectors $\mathbf{h}_i$ and $\mathbf{t}_j$. For example Kocher et al. [31] used the *difference-of-means* method before using the correlation coefficient as a distinguisher. For all the methods we list below, the attack runs for the steps 1,2,3 unchanged. The first difference is in step 4, when we map the matrix $\mathbf{V}$ to the matrix $\mathbf{H}$. In step 4 we need to change the power model since the methods below only allow a binary power model. Binary means in this case that there is either power consumed, which results in $h_{i,j} = 1$, or no power consumed which results in $h_{i,j} = 0$.
The next difference is of course in step 5 when we need to calculate the matrix $\mathbf{R}$. We describe below how this is done for the alternative methods.

Difference of means

We start with the difference of means method to determine the relation between $\mathbf{h}_i$ and $\mathbf{t}_j$. The basic idea behind this method is based on the following observation. When creating the matrix $\mathbf{H}$ we assume that the power consumption values are different for some intermediate values than for all the other intermediate values. To check if a key hypothesis k_i is correct, we split the matrix $\mathbf{T}$ into two sets of rows according to the vector $\mathbf{h}_i$. We do this in the following way. We put the rows of $\mathbf{T}$ in the first set which correspond to the indices of the zeros of $\mathbf{h}_i$. The second set of rows of $\mathbf{T}$ contains the rows which correspond to the indices of the ones of $\mathbf{h}_i$. We denote with $\mathbf{m}_{0i}$ the vector of mean values of the rows from the first set of the matrix $\mathbf{T}$. Analogously we denote with $\mathbf{m_{1i}}$ the vector of mean values of the rows from the second set. The key hypothesis k_i turns out to be correct,

if there is a significant difference between $\mathbf{m}_{0i}$ and $\mathbf{m}_{1i}$ at some point in time. The result of a DPA attack with the difference of means method is a matrix $\mathbf{R}$ where each row of $\mathbf{R}$ corresponds to the difference between $\mathbf{m}_{0i}$ and $\mathbf{m}_{1i}$ for one key hypothesis k_i.

Distance of means

Next we present the distance of means method to determine the correct key hypothesis. The distance of means method is an improvement of the difference of means test. The reason is that the distance of means test also uses the standard deviation of the values. We describe the method below.
We proceed as in the difference of means test and split the matrix $\mathbf{T}$ as before. The next step is to compare the means of the two sets with the distance of means test. We therefore calculate the values $r_{i,j}$ as follows [36]:

$$r_{i,j} = \frac{m_{1i,j} - m_{0i,j}}{s_{i,j}} \tag{3.5}$$

where $s_{i,j}$ denotes the standard deviation of the difference distribution of the two sets.

Generalized maximum-likelihood testing

Last we describe a method which was proposed by Agrawal et al. [2]. The method is based on the generalized maximum likelihood testing [28].

For using this method in a DPA attack, we proceed as in the difference of means test and split the matrix $\mathbf{T}$ as before into a 0-bin and 1-bin. We then extend the matrix $\mathbf{H}$ by an additional column $\mathbf{h}_{K+1}$ which contains a random sequence of zeros and ones. This column is called the *null hypothesis*.
In step 5 of the DPA attack we now compare the K key hypotheses to the null hypothesis $\mathbf{h}_{K+1}$. The idea of this comparison is that under the wrong key hypothesis, the signals in the 0-bin and 1-bin have similar distributions due to a random separation in the two bins. Hence, we cannot detect much difference, compared to the null hypothesis. For the correct key hypothesis, the distribution of signals in the 0-bin differs from the distribution in the 1-bin. Hence there is a difference compared to the null hypothesis. The formula to calculate the values $r_{i,j}$ can be found with a detailed explanation of it in [2]. Agrawal et al. state in [2] that their method needs fewer power traces compared to the difference of mean method, in order to derive the correct key. Note, that the due to the random generation of the null hypothesis, the generalized maximum-likelihood testing is not deterministic.

3.3.4 Attacks based on the distribution

In this section we present DPA attacks that use distinguishers which compare the distribution of the real power traces and the hypothetical power traces. We present the mutual information attack proposed by Gierlichs et al. in [25] and the Kolmogorov-Smirnov distinguisher proposed by [80] below. Note that a DPA attack that uses one of these distinguishers runs unchanged up to step 5. The difference in step 5 is that we compare the distribution of the vectors $\mathbf{h}_i$ and $\mathbf{t}_j$ with the distinguishers presented below.

Mutual information attack

Mutual information measures the total information shared between two random variables X and Y. This measure is expressed in bits. The mutual information [60], expressed via the entropies of X and Y, is

$$I(X;Y) = H(X) - H(X|Y), \tag{3.6}$$

where $H(X)$ is the entropy of X and $H(X|Y)$ the entropy of $X|Y$. The mutual information satisfies $0 \leq I(X;Y) \leq H(X)$, where the lower bound is reached if and only if X and Y are independent. The upper bound is achieved if Y completely determines X.

When we use the mutual information as a distinguisher for the key hypotheses, we calculate the mutual information between the power traces and the hypothetical power traces under a key hypothesis. Hence we calculate values $r_{i,j}$ such that

$$r_{i,j} = I(\mathbf{t}_i; \mathbf{h}_j)$$

If the key hypothesis is wrong, the mutual information will be 0 or very low, due to the independence of the variables. If the key hypothesis is correct, and the point in time for the power trace $\mathbf{t}$ is wrong, the mutual information will also be zero or very low again. If the point in time and the key hypothesis are correct the mutual information $I(\mathbf{t}_i; \mathbf{h}_j)$ is greater than zero. Hence, the two vectors $\mathbf{t}_i$ and $\mathbf{h}_j$ are dependent by definition.

The mutual information is a functional of probability distributions. When we use the mutual information in a DPA attack, the true distributions of the samples (power traces) are not known. Hence we have to estimate the distribution of the samples. This is explained in detail in [25]. Gierlichs et al. report in [25] that the MIA works as a "generic" distinguisher, which means that a key recovery is possible even in the absence of a good power model. The MIA is adaptable to higher order attacks, since the mutual information extends well to multi-variate distributions. Whitnall et al. report in [80] that the MIA is sensitive to noise in the power measurements.

Kolmogorov-Smirnov Test

The Kolmogorov-Smirnov (KS) test [65] is a non-parametric statistical test for distinguishing between distributions. The distance between the empirical cumulative distribution functions (CDF) of two samples $\mathbf{A} = \{A_i | 1 \leq 1 \leq n\}$ and $\mathbf{B} = \{B_j | 1 \leq j \leq m\}$ is measured by the KS test statistic. It tests whether the two sets of samples have been drawn from the same distribution. The KS test statistic is defined as $K(A \| B) = \sup_{x \in \mathbf{A} \cup \mathbf{B}} |F_A(x) - F_B(x)|$ where F_A and F_B are the empirical CDFs, i.e.

$$F_A(x) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{A_i \leq x}$$

where $\mathbb{1}_{A_i \leq x}$ is the indicator function taking the value 1 if and only if $A_1 \leq x$.

Informally, a KS distinguisher measures the maximum distance between the distribution of the global traces $\mathbf{T}$ and the distribution of the conditional traces $\mathbf{T}|\mathbf{h}_i$. The formula to calculate the values from a KS distinguisher can be found in [80]. However the values are high for a correct key hypothesis and low for the remaining key hypotheses.

Whitnall et al. report in [80] that the KSA is more robust against noise than the MIA. Another advantage of the KSA compared to the MIA is, that there is no use of estimating the underlying probability density functions. Since the KS test extends for bivariate distributions, the KSA can be extended to second-order DPA.

3.3.5 Higher order attacks

Higher order DPA attacks were defined by Kocher et al. in [31], as a DPA attack that combines one or more samples within one single power trace. This allows us to distinguish DPA attacks of different orders.

Definition 3.3.2 ([43])**:** An nth-order DPA attack makes use of n different samples in the power trace, that correspond to n different intermediate values of the cryptographic algorithm.

For example in a second order DPA attack the event under investigation typically is the fact that two intermediate values which occur at different time, are equal (or different).
In practice higher order attacks are more difficult to mount than first-order attacks. Daemen et al. describe in [20] that second-order DPA attacks need a more complex layout, increased memory and processing requirements, as well as an increased number of power consumption traces. Though higher

order attacks are more complex, several publications on higher order attacks have been made [43, 64, 48, 24].

3.4 Selection of countermeasures

In the following section we analyze countermeasures against DPA attacks. We first discuss countermeasures on the software level. We then analyze countermeasures on the algorithmic level and the gate level. Both are hardware countermeasures. We start with software countermeasures below.

3.4.1 Software countermeasures

When a cryptographic algorithm is implemented on a general purpose hardware, like a generic 8051-based microcontroller, the reduction of side-channel leakage on the hardware side is not possible. The only freedom that remains is to modify the implementation of the cryptographic algorithm in software. A standard procedure to reduce side-channel leakage is to introduce random masks in the software. Consider for example the following code which takes some data as input and modifies the data with the use of the secret key. For simplicity we assume that the data is combined with the secret key by an xor operation.

```
1: Function1(data)
2: {
3:   result = data ^ SecretKey;
4:   . . .
5:   other operations . . .
6: }
```

As long as we have control over the data which is processed by $Function1$, this implementation is vulnerable to a DPA attack in line 3. As a common countermeasures, we modify the routine as follows.

```
1: Function2(data)
2: {
3:   RandomMask = rand();
4:   Temp = data ^ RandomMask
5:   result = Temp ^ SecretKey;
6:   . . .
7:   other operations . . .
8: }
```

This function *Function*2 is safe from first-order DPA attacks [43], since we do not know the values which get processed in the operation where the secret key is involved. However Messerges showed in [43] that *Function*2 is vulnerable to second-order DPA attacks on the lines 3 and 5. The author further points out that for a second-order DPA attack a more detailed knowledge about the implementation is needed, since an attacker must know which points in the power consumption trace are important.

3.4.2 Hardware countermeasures

The general goal of hardware countermeasures against side-channel attacks is, to try to make the power consumption of the circuit independent of the processed values. Usually this is done by adding randomness to the circuit.

Algorithmic countermeasures

We start by presenting a countermeasure on the algorithmic level. The idea of this approach is to add a side-channel countermeasure, for example a masking scheme, to the algorithm. We present an approach from Mangard et al. [36] for a masked AES [22] implementation below.

Since all AES operations are linear except for the *SubBytes* step, we only focus on masking the S-box. The AES S-box consists of 256 Bytes and is generated by determining the multiplicative inverse for a given element in $\mathbb{F}_{256}^*$. The masking scheme, by Mangard et al. [36], is based on the S-box architecture described by Wolkerstorfer et al. in [82]. Wolkerstorfer et al. use a composite field arithmetic to represent the field $\mathbb{F}_{256}$. The authors of [82] represent elements of $\mathbb{F}_{256}$ as linear polynomials $v_h x + v_l$ with $v_h, v_l \in \mathbb{F}_{16}$. Hence, $\mathbb{F}_{256}$ is a quadratic extension of $\mathbb{F}_{16}$.

To compute the inverse of an element in $\mathbb{F}_{256}$ we can use the following formulas, which can be computed in $\mathbb{F}_{16}$ [36]:

$$\begin{aligned}(v_h x \oplus v_l)^{-1} &= v_h' x + v_l' \\ v_h' &= v_h \cdot w' \\ v_l' &= (v_h + v_l) \cdot w' \\ w' &= w^{-1} \\ w &= (v_h^2 \cdot p_0) + (v_h \cdot v_l) + v_l^2\end{aligned} \tag{3.7}$$

These operations are calculated modulo a polynomial that is fixed when the quadratic extension is defined. p_0 is defined in compliance with this polynomial.

When we want to calculate the inverse of a masked input in $\mathbb{F}_{256}$, we first map the input as well as the mask to the composite field representation. A linear mapping for this has been described by Wolkerstorfer et al. in [82]. Since the mapping is linear, it can be masked easily. The input of the inversion in $\mathbb{F}_{16}$ becomes than $(v_h + m_h)x + (v_l + m_l)$. The goal of Mangard et al. [36] is that all inputs and outputs of the inversion are masked. Thus the authors of [36] want to achieve the following representation.

$$((v_h + m_h)x + (v_l + m_l))^{-1} = (v'_h + m'_h)x + (v'_l + m'_l)$$

This can be achieved by the following formulas [36]:

$$\begin{aligned} v'_h + m'_h &= v_h \cdot w' + m'_h \\ v'_l + m'_l &= (v_h + v_l) \cdot w' + m'_l \\ w' + m'_w &= w^{-1} + m'_w \\ w + m_w &= (v_h^2 \cdot p_0) + (v_h \cdot v_l) + v_l^2 + m_w \end{aligned} \tag{3.8}$$

To calculate the inversion in $\mathbb{F}_{16}$ of the element w, Mangard et al. represent $\mathbb{F}_{16}$ as quadratic extension of $\mathbb{F}_4$. The formulas (3.8) can be used again to calculate the inversion of the masked input in $\mathbb{F}_4$. In the field $\mathbb{F}_4$ inversion of elements is equal to squaring, hence $x^{-1} = x^2$ for all $x \in \mathbb{F}_4$.

Pramstaller et al. have reported in [52] that an implementation of this masked S-box is two to three times larger and slower than an implementation of an unmasked AES S-box.

Mangard et al. describe in [36] the vulnerability of this masked S-box to second order DPA attacks. The authors describe an attack in which they use the correlation coefficient as a distinguisher. The power consumption measurements of the implementation were taken during the first AES encryption round. The issue for the side-channel leakage in the circuit are certain hardware effects, called *glitches*. We refer to Section 4.2 for a description of these hardware effects.

Gate based countermeasures

We now present countermeasures which are based to counter side-channel leakage on the gate level. Special logic design styles have been proposed for this purpose like Wave Dynamic Differential Logic (WDDL) proposed by Tiri et. al. in [70] or Masked Dual-Rail Pre-charge Logic (MDPL) introduced by Popp and Mangard in [50]. We will discuss the MDPL logic style below.

The concept of both logic styles is to balance the power consumption of the gate transitions from $0 \mapsto 1$ and $1 \mapsto 0$, see equation (3.1). Mangard

and Popp try to achieve this by implementing the following idea. To balance the power consumption they use a Dual-Rail Pre-charged (DRP) logic style. DRP logic styles have the property that the transitions need the same amount of power, if all pairs of complementary wires are perfectly balanced. This means that they have the exact same capacity load, which in practice is very hard or even impossible to guarantee. To handle this problem, they further use a masking scheme for the logical values. This means that a logical value d is represented as $d_m = d \oplus m$ where m is a mask value which gets updated in every clock cycle. In Figure 3.2 an AND gate and an OR gate in MDPL style are shown.

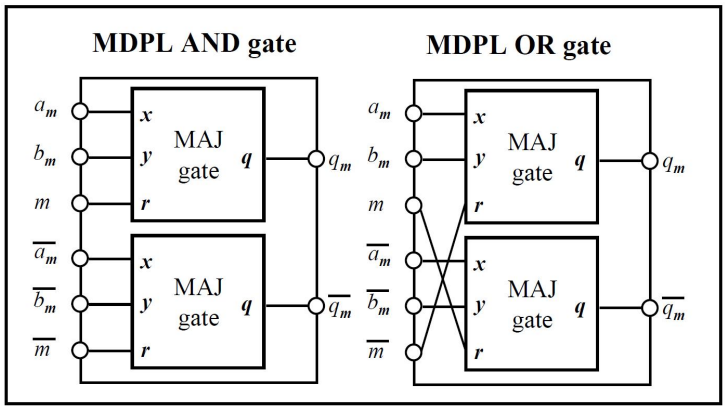


Figure 3.2: AND and OR gate in MDPL style [68], with inputs $(a_m, \bar{a}_m, b_m, \bar{b}_m, m, \bar{m})$ and outputs $(q_m, \bar{q}_m)$. MAJ represents a majority gate.

Popp and Mangard state in [50] that their design can improve side channel leakage caused by the difference of loading capacitances between to complementary logic gates. Suzuki and Saeki describe however in [68] that the circuit is vulnerable to side-channel leakage caused by the difference of delay time between the input signals of the MDPL gates.

3.4.3 Summary

The problem why these countermeasures do not work in practice is, that the processed values are not independent of the input values like data and key. Hence, information about the input values is leaked through the side-channels. In the next chapter we will present a different approach against side-channel leakage. We therefore try to achieve that the input values of

logical circuits are independent of the input values of the cryptographic algorithm.

Chapter 4

Secret sharing

In this section we introduce the concept of *secret sharing* or *multi-party computation* (MPC). The goal is, to enable multiple parties a mutual computation over their inputs, by keeping these inputs private. For example consider the following problem. Given a set of millionaires who want to determine which of them is richest, without revealing their net worth. This problem is called the *millionaires problem* and was suggested by Yao in 1982 in [84].
Another approach to MPC is the so called *secret sharing*. The idea of *secret sharing* is to start with a secret and divide it into pieces. These pieces are called shares and are distributed amongst the parties. The collection of specific subsets of the parties information allows the reconstruction of the original secret. If two or more shares are required, for example to trigger an critical action, a secret sharing scheme may serve as a *shared control scheme*.
When we talk about secret sharing schemes in this chapter, we want to keep in mind the following idea. We first split the input x of a cryptographic function in a certain way to inputs $x_1, \ldots, x_n$ such that each x_i with $i \in \{1, \ldots, n\}$ is independent of x. When we then use these x_i as inputs, the intermediate results will be independent of the original input x. Hence no side-channel information which is leaked from these intermediate values gives information about the value x.

The chapter is organized as follows. We start in Section 4.1 with an introduction to sharing schemes. In Section 4.2 we will take a closer look on masking schemes. In Section 4.3 we will introduce the secret sharing we use in this thesis and prove its security against side-channel attacks. Section 4.4 will cover the sharing of affine equivalent S-boxes.

4.1 Introduction to secret sharing

In this section we give a short introduction to secrete sharing methods. We start in Section 4.1.1 with simple shared control schemes and continue in Section 4.1.2 with threshold schemes.

4.1.1 Simple shared control schemes

We start with a simple shared control scheme for two parties. The goal is, to share a secret S, $0 \leq S \leq m-1$ for $m \in \mathbb{N}$, such that no single party has knowledge about the secret S (except a trusted third party). We use the following method to share the secret S between the two parties. A trusted third party T generates a random number S_1 such that $1 \leq S_1 \leq m-1$ and gives the numbers S_1 and $S_2 = S - S_1 \mod m$ to two parties A and B respectively. To employ the secret value S, both A and B have to supply their secrets S_1 and S_2. By addition modulo m the secret S can be recovered. Assume that A and B are trusted not to collude, then neither A nor B has any information about the secret S, since their information is a random number between 0 and $m-1$.
We can generalize this scheme to share the secret S between t parties. The procedure works as follows: The trusted third party T generates $t-1$ random numbers S_i, such that $1 \leq S_1 \leq m-1$, for all $i \in \{1, \ldots, t-1$. The parties $P_1, \ldots, P_{t-1}$ get the random S_i, where the party P_t gets $S_t = S - \sum_{i=1}^{t-1} S_i$. To recover the original secret S, the secretes S_i, for $i \in \{1, \ldots, t\}$ are added modulo m, i.e. $S = \sum_{i=1}^{t} S_i \mod m$.

4.1.2 Threshold schemes

We now want to introduce so called *threshold schemes*.

Definition 4.1.1 ([41])**:** Let S be a secret number and $t, n \in \mathbb{N}$ with $t \leq n$. Let further T be a trusted third party. A (t, n) *threshold scheme* is a method by which T calculates secret shares S_i, $1 \leq i \leq n$ from S, and distributes them securely to the parties P_i, such that the following is true:

- Any t or more parties who join their secrets may easily recover S.
- Any $t-1$ or fewer parties who join their secrets my not recover the. secret S

A *perfect* threshold scheme is a threshold scheme in which the knowledge of $t-1$ or fewer shares gives no advantage about the secret S, to an opponent over knowing no shares.

Before we present an example of a (t, n) threshold scheme, we want to generalize the definition of a (t, n) threshold scheme.

Definition 4.1.2 ([11])**:** A (c, t, n) *ramp scheme*, where $1 \leq c < t \leq n$, is a sharing of secrets among n parties, such that

1. Any set of at least size t parties can reconstruct the secret.

2. Any set of at most c parties has absolutely no information on the secret.

So a (t, n) threshold scheme is a $(t-1, t, n)$ ramp scheme. As an example of a (t, n) threshold scheme, we present Shamir's threshold scheme [59]. This threshold scheme is based on polynomial interpolation of univariate polynomials. The essential idea in this scheme is, that a univariate polynomial of degree $t-1$ is uniquely defined by t points (x_i, y_i) such that $x_i \neq x_j$ for $i \neq j$.

Algorithm 2 (Shamir's (t, n) threshold scheme – Setup [41])**:**
Input: A secret $S \geq 0$ which should be distributed among n parties
Output: A distribution of n shares to n parties.

1. The trusted party T chooses a prime $p \geq \max(S, n)$ and defines $a_0 = S$.

2. T chooses randomly $t-1$ independent coefficients $a_1, \ldots, a_{t-1}$, such that $0 \leq a_i \leq p-1$. This defines the random polynomial

$$f(x) = \sum_{i=0}^{t-1} a_i x^i \in \mathbb{Z}_p[x].$$

3. T computes the secrets $S_i = f(i) \mod p$ for $1 \leq i \leq n$.

4. T distributes the tuple (i, S_i) to the party P_i for $1 \leq i \leq n$.

Algorithm 3 (Shamir's (t, n) threshold scheme – Pooling of shares [41])**:**
Input: Any group of t parties P_i with shares (i, S_i)
Output: The secret S Calculate the coefficients of the polynomial $f(x)$ as follows

$$f(x) = \sum_{i=1}^{t} S_i \prod_{\substack{1 \leq j \leq t \\ j \neq i}} \frac{x-j}{i-j}. \tag{4.1}$$

Recover the secret S by calculating $f(0) = a_0 = S$.

The method (4.1) to obtain the coefficients a_j for $0 \leq j \leq t-1$ is called *Lagrange interpolation* method [8].

Before we go on, we shortly want to discuss some properties of Shamir's secret sharing scheme. The scheme does not depend on *unproven assumptions* (e.g., the difficulty of number-theoretic problems), unlike many cryptographic schemes and can easily be extended for new parties without impact on existing shares. It is a *perfect* sharing scheme, since given the knowledge of $t-1$ or fewer shares, all values $0 \leq S \leq p-1$ remain equally probable.

We give two more definitions of properties related to secret sharing protocols.

Definition 4.1.3 ([7])**:** A secret sharing protocol is t-private if any set of at most t parties cannot disrupt the communication or compute additional information than they could solely from their set of private inputs and outputs.
A secret sharing protocol is t-resilient if no set of at most t parties can interfere on the correctness of the outputs of the remaining parties.

To demonstrate the power of secret sharing protocols we state the following theorem from Ben-Or et al.

Theorem 4.1.1 ([7])**:** For every function f and $t < n/2$ there exists a t-private protocol.

The proof of this theorem can be found in [7].

4.2 Masking schemes

In this section we discuss a special case of secret sharing, called *masking*. The idea of masking is to try to randomize the intermediate values of a cryptographic algorithm [42]. If this randomization succeeds, the power consumption does not correlate with the intermediate values anymore. We present the most common masking scheme below and analyze its behavior under the influence of certain hardware effects, called *glitches* [37].

Glitches are the transitions which occur at the output of a gate, before the gate switches to the correct output. More information about glitches can be found in literature about VLSI design, for example in [55].

We now illustrate the most common masking scheme, which is Boolean or linear masking. In Boolean masking a mask is added to the input variable by an XOR operation. More formally, let $x \in \mathbb{F}_2$ be an input of a combinational circuit and $m \in \mathbb{F}_2$ a random value. Then we get the masked input $x_m \in \mathbb{F}_2$ by adding the value m to x, i.e $x_m = x \oplus m$.

4.2.1 Glitches in a masked AND gate

Mangard et al. have showed in [37] how glitches can affect a traditionally masked AND gate. We present their approach below.
We therefore use the implementation of Trichina et al. for the masked AND gate [71], which is illustrated in Figure 4.1. As illustrated in Figure 4.1, the masked AND gate has the inputs $x_m = x \oplus m_x$, $y_m = y \oplus m_y$, m_x, m_y and m_z. The outputs are the random mask m_z and the masked result z_m calculated as follows:

$$z_m = x_m y_m \oplus (m_y x_m \oplus (m_x y_m \oplus (m_x m_y \oplus m_x))) \tag{4.2}$$

The order in which the XOR sums are evaluated is not arbitrary, since consider for example the sum $m_x y_m \oplus m_x m_y = y m_x$ which leaks information about y.

Assume that a glitch occurs on the input y_m. Since the propagation of this glitch depends on the values m_x and x_m, the power consumption of the circuit depends on the number of gates, that are affected by the propagation of that glitch. Table 4.1 shows, that the power consumption depends on the values of m_x and x_m. We deduce that the mean power consumption is different for $x = 0$ and $x = 1$, hence the power consumption leaks information about the value of x. Glitches on any of the other inputs cause a similar leakage of information.

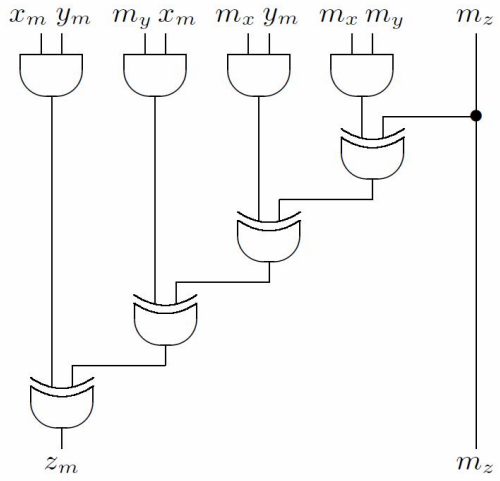


Figure 4.1: Schematic representation of a masked AND gate [44]

Though many different masking schemes have been proposed, like [4], [10] and [72], all of them have been broken again [3], [26] and [75]. The reason

x	m_x	x_m	AND	XOR
0	0	0	0	0
1	0	1	1	1
1	1	0	1	2
0	1	1	2	2

Table 4.1: Number of affected gates in the circuit shown in Figure 4.1, when a glitch occurs on input y_m

why these approaches do not work is, that there is no independence between the processed values and the inputs of the calculation.

4.3 Secret sharing used in this thesis

Below we introduce a sharing scheme which is used for the rest of this thesis. We start by giving a short overview of the sharing approach and linking it to MPC protocols.

4.3.1 Overview

In this thesis we use secret sharing schemes to share the input variables which are processed by a given function, like an S-box, or more generally a given combinational circuit. We therefore split each input-variable into $s \in \mathbb{N}$ additive shares. This approach has been proposed by Chari et al. in [17]. They extensively describe their approach for the case $s = 2$ but do not investigate how nonlinear operations should be realized. Nikova et al. examine how this sharing scheme can be extended to nonlinear functions in [44]. Before we start with the terminology, we want to draw a connection to MPC protocols.

We started this section with an introduction to MPC. The approach by Nikova et al. [44] is a $(1, s, s)$ ramp scheme, since firstly the output of all parties is needed to compute the output of the circuit. Secondly each input x_i is used in several functions and each two functions together possibly use all inputs.

4.3.2 Terminology

We now start with an introduction on the terminology which is used for the rest of the work.

We denote stochastic variables by small characters $x, y, \ldots$ and samples of these variables by capital characters $X, Y, \ldots$. We further denote by $\mathbb{P}(x = X)$ the probability that the stochastic variable x takes the value X, where *probability* is the number of times a variable x takes the value X, divided by the number of values that the input x can take.

Notation: We denote for a variable x the vector of $s \in \mathbb{N}$ additive shares x_i by $\bar{x} = (x_1, \ldots, x_s)$ and split x up into s shares by $x = \bigoplus_{i=1}^{s} x_i$.

Notation: For a function with $n \in \mathbb{N}$ input variables $x_1, \ldots, x_n$ which are all split up into s shares we write:

$$(\bar{x}_1, \ldots, \bar{x}_n) = (x_{1,1}, \ldots, x_{1,s}, \ldots, x_{n,1}, \ldots, x_{n,s})$$

such that $x_i = \bigoplus_{j=1}^{s} x_{i,j}$ for all $i \in \{1, \ldots, n\}$.

Let f be a function with $n \in \mathbb{N}$ inputs. Let further denote Q the number of different values the input vector $(X_1, \ldots, X_n)$ of f can take. Then Q^s is the number of different values that the vector of input shares $(\bar{X}_1, \ldots, \bar{X}_n)$ can take. We limit our work to secret sharing schemes where the following holds

$$\mathbb{P}(\bar{x}_1 = \bar{X}_1, \ldots, \bar{x}_n = \bar{X}_n) = Q^{1-s}\mathbb{P}(x_1 = \bigoplus_{j=1}^{s} X_{1,j}, \ldots, x_n = \bigoplus_{j=1}^{s} X_{n,j}). \quad (4.3)$$

This means that any bias, which is present in the joint distribution of the shares $(\bar{x}_1, \ldots, \bar{x}_n)$ is based on a bias in the distribution of the unshared variables $(x_1, \ldots, x_n)$. Hence we have the following: Let $(\bar{X}_1, \ldots, \bar{X}_n)$ and $(\bar{Y}_1, \ldots, \bar{Y}_n)$ be two shares of the vector $(X_1, \ldots, X_n)$. (That means $X_i = \bigoplus_{j_1}^{s} X_{i,j} = \bigoplus_{j_1}^{s} Y_{i,j}$ for all $i \in \{1, \ldots, n\}$.) Then

$$\mathbb{P}(\bar{x}_1 = \bar{X}_1, \ldots, \bar{x}_n = \bar{X}_n) = \mathbb{P}(\bar{x}_1 = \bar{Y}_1, \ldots, \bar{x}_n = \bar{Y}_n).$$

4.3.3 Requirements

In this section we present the requirements which are necessary in order to achieve a secure secret sharing scheme. We start with the property of *correctness*. When we split a function f into the sum of s functions $f_1, \ldots, f_s$ we naturally require that the sum of these s functions gives the correct output for all input combinations of the function f and $f_1, \ldots, f_s$ respectively. We formalize this in the following definition.

Definition 4.3.1 (Correctness[44])**:** Let $(z_1, \ldots, z_m) = F(x_1, \ldots, x_n)$ be a vectorial Boolean function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. Then the set of functions

$$F_i(\bar{x}_1, \ldots, \bar{x}_n) \quad \text{for } i \in \{1, \ldots, s\}$$

is a *sharing* (or *realization*) of F if and only if

$$(z_1, \ldots, z_m) = F(x_1, \ldots, x_n) = \bigoplus_{i=1}^{s} F_i(\bar{x}_1, \ldots, \bar{x}_n) \quad \forall (x_1, \ldots, x_n) \in \mathbb{F}_2^n \quad (4.4)$$

and all shares $(\bar{x}_1, \ldots, \bar{x}_n)$ satisfying $\bigoplus_{j=1}^{s} x_{i,j} = x_i$ for all $i \in \{1, \ldots, n\}$.

The next property we need is the so called *non-completeness*. Before we go into the definition, we need to define the following notation.

Notation: We write $\bar{x}^i$ for the vector which is independent of share i, i.e:

$$\bar{x}^i = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_s).$$

The second property treats the independence of the inputs of the functions $f_1, \ldots, f_s$ of the original input of the function f. To achieve this we require for a function f_i that each vector of input shares is missing at least one component. This means that we require for the function f_i that only inputs with share index unequal to i. More formally only $\bar{x}_1^i, \ldots, \bar{x}_n^i$ serve as input of the function f_i. This property is formalized for all s share functions of f in the following definition.

Definition 4.3.2 (Non-completeness[44])**:** Let $z_j = F_j(\bar{x}_1, \ldots, \bar{x}_n)$ be a sharing of the function $z = F(x_1, \ldots, x_n)$ with s shares. Then the set of functions F_j for $j \in \{1, \ldots, s\}$ is said to fulfill the *non-completeness* property if and only if every function F_j for $j \in \{1, \ldots, s\}$ is independent of at least one share of each input variable x_i for $i \in \{1, \ldots, n\}$. Without loss of generality, we require that F_j is independent of $x_{i,j}$ for all $i \in \{1, \ldots, n\}$ and for all $j \in \{1, \ldots, s\}$:

$$
\begin{aligned}
z_1 =& F_1(x_{1,2}, x_{1,3}, \ldots, x_{1,s}, \ldots, x_{n,2}, x_{n,3}, \ldots, x_{n,s}) = F_1(\bar{x_1}^1, \ldots, \bar{x_n}^1) \\
z_2 =& F_2(x_{1,1}, x_{1,3}, \ldots, x_{1,s}, \ldots, x_{n,1}, x_{n,3}, \ldots, x_{n,s}) = F_2(\bar{x_1}^2, \ldots, \bar{x_n}^2) \\
&\vdots \\
z_s =& F_s(x_{1,1}, x_{1,2}, \ldots, x_{1,s-1}, \ldots, x_{n,1}, x_{n,2}, \ldots, x_{n,s-1}) = F_s(\bar{x_1}^s, \ldots, \bar{x_n}^s)
\end{aligned}
$$

Let $\bar{F} : \mathbb{F}_2^{n \cdot s} \mapsto \mathbb{F}_2^{m \cdot s}$ be a sharing of the vectorial Boolean function $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ with s shares. Then $\bar{F}$ fulfills the *non-completeness property* if and only if all sharings of each coordinate function of F fulfill the non-completeness property.

Next we want to list the so called *uniformity* property. When we share a set of outputs $z_1, \ldots, z_m$ to $\bar{z}_1, \ldots, \bar{z}_m$ we require that $\oplus_{j=1}^s z_{i,j}$ and z_i are identically distributed. In other words, we require that every set of shared output values $\{(\bar{z}_1, \ldots, \bar{z}_m)\}$, that share the output value $(z_1, \ldots, z_m)$, occur equally likely as the output value $(z_1, \ldots, z_m)$. We formalize this in the following definition.

Definition 4.3.3 (Uniformity[44])**:** A sharing $(\bar{z}_1, \ldots, \bar{z}_m) = \bar{F}(\bar{x}_1, \ldots, \bar{x}_n)$ of $(z_1, \ldots, z_m) = F(x_1, \ldots, x_n)$ is uniform, if the distribution of the shares of the output satisfies

$$
\mathbb{P}(\bar{z}_1 = \bar{Z}_1, \ldots, \bar{z}_m = \bar{Z}_m) = Q^{1-s}\mathbb{P}\left(z_1 = \bigoplus_{j=1}^{s} Z_{1,j}, \ldots, z_m = \bigoplus_{j=1}^{s} Z_{m,j}\right) \tag{4.5}
$$

provided that the distribution of the shares of the input satisfies (4.3).

We give an example of this requirement below.

Example: Assume we have a function F mapping from $\mathbb{F}_2^4$ to $\mathbb{F}_2^2$. Let (a, b, c, d) be the inputs of the function F and (e, f) be the output of the function. When we share this function F with three shares, we expand the input to $(a_1, a_2, a_3, b_1, \ldots, d_3)$ and the output to $(e_1, \ldots, f_3)$. Assume further that the input $(a, b, c, d) = 1011$ gives the output $(e, f) = 10$. Then we want that every set of combinations of $(e_1, e_2, e_3, f_1, f_2, f_3)$ with $e_1 + e_2 + e_3 = 1$ and $f_1 + f_2 + f_3 = 0$ occurs equally likely.

If F is an invertible function, for example an S-box, then property (4.5) is satisfied by invertible sharings. This is equal to the condition that every vector $(\bar{Z}_1, \ldots, \bar{Z}_m)$ is reached for exactly one input vector $(\bar{X}_1, \ldots, \bar{X}_n)$. In the case of S-boxes *uniformity* is equivalent to *balancedness*.

4.3.4 Sharing linear transformations

Let $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ be a linear function. Then we can share this function F, with $s \in \mathbb{N}$ shares, by using the linearity of the function F. Let $(z_1, \ldots, z_m) = F(x_1, \ldots, x_n)$ then we get the sharing of F by setting

$$\begin{aligned}(z_{1,i}, \ldots, z_{m,i}) &= F(x_{1,i+1}, \ldots, x_{n,i+1}), \quad 1 \leq i < s \\ (z_{1,s}, \ldots, z_{m,s}) &= F(x_{1,1}, \ldots, x_{n,1}).\end{aligned}$$

Since the function F is linear we have that :

$$\begin{aligned}(z_1, \ldots, z_m) &= \sum_{j=1}^{s} (z_{1,j}, \ldots, z_{m,j}) = \sum_{j=1}^{s} F\left(x_{1,j}, \ldots, x_{n,j}\right) \\ &= F\left(\sum_{j=1}^{s} x_{1,j}, \ldots, x_{n,j}\right) = F(x_1, \ldots, x_n)\end{aligned}$$

It is clear that such an implementation of a linear function does not leak any information about the values $(x_1, \ldots, x_n)$ and consequently no information about the values $(z_1, \ldots, z_m)$, even in the presence of glitches. In the next section we will present the theorem by Nikova et al. and investigate on the implementation of nonlinear functions.

4.3.5 Implementing nonlinear functions

We start with the main theorem of this section. The theorem makes a statement about the following idea. If the share $z_{i,j}$ does not depend on the input shares $x_{1,j}, \ldots, x_{n,j}$, then $z_{i,j}$ cannot be correlated to $x_1, \ldots, x_n$. Hence if we only use $\bar{x}_1^j, \ldots, \bar{x}_n^j$ as inputs of the functions $f_{i,j}$ the results $z_{1,j}, \ldots, z_{m,j}$ and all intermediate values cannot be correlated to $x_1, \ldots, x_n$.

Theorem 4.3.1 ([44])**:** Let $(z_1, \ldots, z_m) = F(x_1, \ldots, x_n)$ be a function and $(\bar{z}_1, \ldots, \bar{z}_m) = \bar{F}(\bar{x}_1, \ldots, \bar{x}_n)$ be a sharing of F. If $\bar{F}$ satisfies properties 4.3.1, 4.3.2 and (4.3), then each of the output shares $z_{i,j}$ is statistically independent of the input variables x_i. Furthermore the same holds for all intermediate values that are computed during the computation of the output shares. Particularly this holds for physical quantities like power consumption, electromagnetic radiation etc., since these are functions of the intermediate values.

Proof: Without loss of generality we prove this theorem for the output variable z_1. To prove the theorem, we show that an arbitrary function, which only depends on a subset of the shared inputs is statistically independent of the input variables $x_1, \ldots, x_n$.
Let $\phi(\bar{x}_1^1, \ldots, \bar{x}_n^1)$ be an arbitrary function of the $n \times (s-1)$ input variables $x_{i,j}$ $1 \leq i \leq n$ and $2 \leq j \leq s$.

$$\begin{aligned}\mathbb{P}(\phi = \Phi) &= \sum_{\substack{\bar{X}_1,\ldots,\bar{X}_n \\ \phi(\bar{X}_1^1,\ldots,\bar{X}_n^1)=\Phi}} \mathbb{P}(\bar{x}_1 = \bar{X}_1, \ldots, \bar{x}_n = \bar{X}_n) \\ &= \sum_{\substack{\bar{X}_1^1,\ldots,\bar{X}_n^1 \\ \phi(\bar{X}_1^1,\ldots,\bar{X}_n^1)=\Phi}} \sum_{X_{1,1},\ldots,X_{n,1}} \mathbb{P}(\bar{x}_1 = \bar{X}_1, \ldots, \bar{x}_n = \bar{X}_n)\end{aligned}$$

Since $X_i = \bigoplus_{j=1}^{s} X_{i,j}$, we can change variables and replace $X_{1,1}, \ldots, X_{n,1}$ by $X_1, \ldots, X_n$. By the use of (4.3) we obtain:

$$\begin{aligned}\mathbb{P}(\phi = \Phi) &= \sum_{\substack{\bar{X}_1^1,\ldots,\bar{X}_n^1 \\ \phi(\bar{X}_1^1,\ldots,\bar{X}_n^1)=\Phi}} Q^{1-s} \underbrace{\sum_{X_1,\ldots,X_n} \mathbb{P}(x_1 = X_1, \ldots, x_n = X_n)}_{=1} \\ &= \sum_{\substack{\bar{X}_1^1,\ldots,\bar{X}_n^1 \\ \phi(\bar{X}_1^1,\ldots,\bar{X}_n^1)=\Phi}} Q^{1-s}\end{aligned}$$

Next we compute the conditional probability

$$\begin{aligned}&\mathbb{P}(\phi = \Phi | x_1 = X_1, \ldots, x_n = X_n) \\ &= \sum_{\substack{\bar{X}_1,\ldots,\bar{X}_n \\ \phi(\bar{X}_1^1,\ldots,\bar{X}_n^1)=\Phi}} \mathbb{P}(\bar{x}_1 = \bar{X}_1, \ldots, \bar{x}_n = \bar{X}_n | x_1 = X_1, \ldots, x_n = X_n) \\ &= \sum_{\substack{\bar{X}_1^1,\ldots,\bar{X}_n^1 \\ \phi(\bar{X}_1^1,\ldots,\bar{X}_n^1)=\Phi}} \underbrace{\sum_{X_{1,1},\ldots,X_{n,1}} \mathbb{P}(\bar{x}_1 = \bar{X}_1, \ldots, \bar{x}_n = \bar{X}_n | x_1 = X_1, \ldots, x_n = X_n)}_{\mathcal{S}}\end{aligned}$$

The terms in the sum $\mathcal{S}$ are zero except for exactly one combination of $X_{1,1}, \ldots, X_{n,1}$ which satisfies

$$X_i = \bigoplus_{j=1}^{s} X_{i,j}.$$

We can conclude from (4.3) that in this case $\mathbb{P}(\bar{x}_1 = \bar{X}_1, \ldots, \bar{x}_n = \bar{X}_n | x_1 = X_1, \ldots, x_n = X_n) = Q^{1-s}$. Therefore we can conclude that ϕ and $x_1, \ldots, x_n$ are statistically independent. □

It is important to note that there are no assumptions made in theorem 4.3.1 about the behavior of the circuit or the used hardware technology. This means that even in the presence of glitches the theorem holds.

Theorem 4.3.2 ([44])**:** The minimum number s of shares required to implement a product of D variables with a sharing satisfying property 4.3.1 and 4.3.2 is given by

$$s \geq D + 1 \ .$$

Proof: We prove this theorem by listing a method to assign the terms of a product of D factors with s shares each to the output shares such that the properties 4.3.1 and 4.3.2 are fulfilled. Collect in the first output share all terms that do not contain the first share of any of the inputs. Collect in the second output share all terms that contain the first share of any of the inputs, but not the second share of any of the inputs. By continuing this way, collect in output share j all the terms containing the input shares $1, \ldots, i-1$, but not input share i. Only if $s - 1 \geq D$, there are no terms left after the step s.

We can conclude that the minimum number of shares to implement a nonlinear function is ≥ 3. Nikova et al. report in [44] that the number of gates increases with a factor of s^2, when changing over from 1 share to s shares.

4.3.6 Pipelining

When we share a cryptographic algorithm like the AES [22], which uses a number of transformation rounds that transform the input into the final output, it suffices to focus on the sharing of one round. Since the output of a round is the input of the next round we need to ensure that (4.3) is fulfilled. To ensure (4.3) we require that the sharing fulfills (4.5). Before we state a theorem which uses this requirement, we start a more general approach below.

Pipelining is a hardware implementation technique where a logical circuit with l levels gets split up into two circuits with $l/2$ levels. These levels get

separated by a register which stores the intermediate result of the first stage until the active phase of the next clock cycle. Though pipelining introduces latency, the additional clock cycles can be made up by increasing the clock frequency [82].

The separation of a logical circuit can also reduce the number of shares and the number of gates required to implement a function that has to be protected against side-channel analysis in the presence of glitches. Another advantage of this approach is, that registers that store the results at the end of a stage bound the propagation of glitches and delays.

We now give a result by Nikova et al. [44] about the security of pipelined implementations.

Theorem 4.3.3 ([44])**:** Consider a pipelined sharing consisting of r combinational layers and r registers. 1 register for the final output and $r-1$ registers for the intermediate values. The function that computes the intermediate value $y_{i,j,t}$ at stage t is denoted by $f_{i,j,t}$. The power consumption in the circuit that implements $f_{i,j,t}$ is denoted by $P_{i,j,t}$. We further assume that the distribution of the shares of the input satisfies (4.3). Since this condition needs to be fulfilled at the input of each pipelining stage which is formed by the output of the previous stage, we require that the functions $f_{i,j,t}$ satisfy property 4.3.3.
Under these conditions the following holds: No linear combination of the power consumptions $P_{i,j,k}$ is statistically correlated to any of the input variables x_i nor to any of the output variables y_i.

Proof: We prove that for an arbitrary but fixed choice of the linear coefficients $c_{i,j,t}$, the covariance

$$\operatorname{cov}\left(\sum_{i,j,t} c_{i,j,t}P_{i,j,t}, y_1\right) = 0.$$

We start this proof with the definition of the covariance

$$\begin{aligned}\operatorname{cov}\left(\sum_{i,j,t} c_{i,j,t}P_{i,j,t}, y_1\right) &= \mathbb{E}\left[y_1 \sum_{i,j,t} c_{i,j,t}P_{i,j,t}\right] - \mathbb{E}[y_1]\mathbb{E}\left[\sum_{i,j,t} c_{i,j,t}P_{i,j,t}\right] \\ &= \sum_{i,j,t} c_{i,j,t}\left(\mathbb{E}[y_1 P_{i,j,t}] - \mathbb{E}[y_1]\mathbb{E}[P_{i,j,t}]\right)\end{aligned}$$

Since by our assumptions each of the functions $f_{i,j,t}$ satisfies properties 4.3.1 and 4.3.2, we know that y_1 and $P_{i,j,t}$ are statistically independent. Therefore we can conclude that $\mathbb{E}[y_1 P_{i,j,t}] = \mathbb{E}[y_1]\mathbb{E}[P_{i,j,t}]$. □

4.3.7 Limitations

Theorem 4.3.3 states that if the properties correctness 4.3.1, non-completeness 4.3.2 and uniformity 4.3.3 are fulfilled, the proposed sharing method is secure against a linear combination of side-channel leakage. Hence the method from Nikova et al. [44] is secure against a higher order DPA attack which uses the correlation coefficient. In [45] the authors simulate such a DPA attack on the shared Noekeon S-box [21]. Their results show, that the shared S-box is secure against a DPA attack which uses the correlation coefficient as measure.
However the proposed sharing method is not designed to be secure against an attack which uses nonlinear combinations of intermediate values. An example for such an attack is a *mutual information analysis* (MIA) [25]. However a MIA attack is more sensible to noise in the power traces, compared to an attack with the correlation coefficient. The authors of [45] performed a MIA attack on the simulated shared Noekeon S-box without adding noise to the simulation. The result was that they did not succeed to perform a successful MIA attack for the shared implementation due to the noise characteristic of the combinational circuit.

4.3.8 Decomposition of functions

As we have seen in theorem 4.3.2, the number of shares to realize a function rises with the algebraic degree of the function. An idea to realize the sharing of a function f with higher algebraic degree is, to decompose the function f into functions g and h such that $f(x) = h(g(x))$ for all x in the domain of f. The goal in this decomposition is of course to lower the number of shares needed to realize a sharing for g and h. In Figure 4.2 we illustrate a decomposition of a function f into functions g and h such that $f(x) = g(h(x))$. On the right hand side the shared decomposition is illustrated.

To apply theorem 4.3.3 we need to ensure in such a decomposition that the output of the function h fulfills the uniformity property (4.5). To ensure the resistance against glitches and timing delays in this approach we have to install a glitches and timing delays resistant layer between the functions h and g.

Nikova et al. decompose in [44] the NOEKEON S-box [21]. The S-box of the cipher NOEKEON is a function with algebraic degree 3. Hence a sharing with three shares is not possible. To share the S-box with three shares, the authors of [44] decompose the function into two functions with algebraic degree 2. Due to this decomposition the authors of [44] manage to

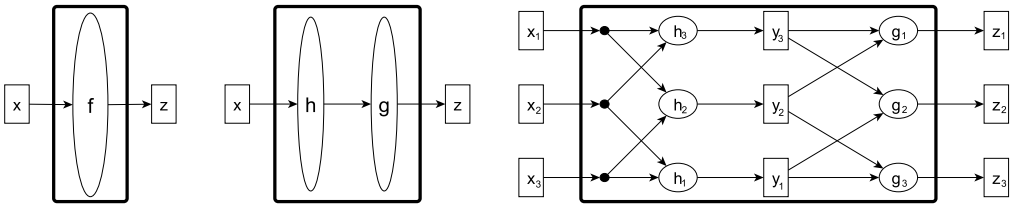


Figure 4.2: The two diagrams on the left show a schematic representation of a decomposition of a function f into functions g and h. The illustration on the right hand side shows the shared decomposition with inputs x_1, x_2, x_3, intermediate values y_1, y_2, y_3 and outputs z_1, z_2, z_3.

share this S-box with three shares.

Decomposition of Present

We now present a result from Poschmann et al. [51] for the block cipher PRESENT [12]. Present is a lightweight block cipher designed for constrained hardware. Each round of this block cipher consists of an XOR with the roundkey, a substitution layer and a permutation layer. The only nonlinear transformation is the substitution layer which consists of 16 applications of the same four bit S-box S. The S-box can be described with inputs $x, y, z, w \in \mathbb{F}_2$ and outputs $S(x, y, z, w) = (s_3, s_2, s_1, s_0)$ as follows:

$$
\begin{aligned}
s_3 &= xyw + xzw + yzw + xz + yz + yw + zw + x + z + w + 1 \\
s_2 &= xyw + xzw + xz + yw + zw + x + y + w + 1 \\
s_1 &= xzw + xyw + yzw + xy + xz + yw + zw + x + z + w \\
s_0 &= yz + yw + zw + x + y
\end{aligned}
$$

The algebraic degree of this S-box is three. To share the S-box with three shares, Poschmann et al. [51] decompose S into $S(x) = G(H(x))$, with quadratic functions G and H. To construct these functions G and H the authors describe in [51] a search method. We give the functions $H(x, y, z, w) = (h_3, h_2, h_1, h_0)$ and $G(x, y, z, w) = (g_3, g_2, g_1, g_0)$ in ANF below [51]:

$$
\begin{aligned}
h_3 &= y + z + w & \qquad g_3 &= xw + y + z + w \\
h_2 &= y + z + 1 & \qquad g_2 &= zw + x \\
h_1 &= zw + yw + z + x + 1 & \qquad g_1 &= xw + y + z \\
h_0 &= xy + xz + yz + 1 & \qquad g_0 &= yw + z
\end{aligned}
$$

In the next section we want to list the shared multiplication in $\mathbb{F}_4$ as an example for the sharing of a nonlinear non-balanced function.

4.3.9 Shared multiplication in GF(4)

In this section we present the shared multiplication in the field with four elements denoted by $\mathbb{F}_4$ which is a result form Nikova et al. [44]. For an easier representation of the field $\mathbb{F}_4$ the authors of [44] use so called normal bases. This means that the field $\mathbb{F}_4$ is considered as a vector space over $\mathbb{F}_2$ with basis elements $\{v = 01, v^2 = 10\}$. Hence we get that every element $x \in \mathbb{F}_4$ has a representation as $x = av + bv^2$ with $a, b \in \mathbb{F}_2$. (a, b) are then called the coordinates of x. To represent the multiplication of two elements $x \in \mathbb{F}_4$ and $y \in \mathbb{F}_4$ with coordinates $(a, b) \in \mathbb{F}_2^2$ and $(c, d)\mathbb{F}_2^2$ respectively we make use of the following formula [44] where we denote by (e, f) the coordinates of the product of x and y.

$$(e, f) = (a, b) \times (c, d) \Leftrightarrow \begin{cases} e = (a + b)(c + d) + ac \\ f = (a + b)(c + d) + bd \end{cases} \tag{4.6}$$

The authors of [44] share this multiplication with three shares. In other words the multiplication is represented as

$$(e_1+e_2+e_3, f_1+f_2+f_3) = (a_1+a_2+a_3, b_1+b_2+b_3) \times (c_1+c_2+c_3, d_1+d2+d_3)$$

To obtain a uniform sharing, the authors use the concept of *correction terms*, which will be introduced in Section 5.3. We list the functions for a shared multiplication in $\mathbb{F}_4$ below [44]:

$$\begin{aligned}
e_1 =& (a_2d_2 + a_2d_3 + a_3d_2 + b_2c_2 + b_2c_2 + b_3c_2 + b_2d_2 + b_2d_3 + b_3d_2) \\
&+ (a_3 + b_2c_2 + b_3c_3 + a_2c_2) \\
e_2 =& (a_1d_3 + a_3d_1 + a_3d_3 + b_1c_3 + b_3c_1 + b_3c_3 + b_1d_3 + b_3d_1 + b_3d_3) \\
&+ (a_1 + a_3 + d_1 + b_1c_1 + b_3 + c_3 + a_1d_1) \\
e_3 =& (a_1d_1 + a_1d_2 + a_2d_1 + b_1c_1 + b_1c_2 + b_2c_1 + b_1d_1 + b_1d_2 + b_2d_1) \\
&+ (a_1 + d_1 + b_1c_1 + b_2c_2 + a_2c_2 + a_1d_1) \\
f_1 =& (a_2c_2 + a_2c_3 + a_3c_3 + a_2d_2 + a_2d_3 + a_3d_2 + b_2c_2 + b_2c_3 + b_3c_2) \\
&+ (c_3 + d_3 + a_2c_2 + a_3c_3 + b_2d_2 + b_3d_3) \\
f_2 =& (a_1c_3 + a_3c_1 + a_3c_3 + a_1d_3 + a_3d_1 + a_3d_3 + b_1c_3 + b_3c_1 + b_3c_3) \\
&+ (c_3 + d_1 + d_3 + a_3c_3 + b_1d_1 + b_3d_3) \\
f_3 =& (a_1c_1 + a_1c_2 + a_2c_1 + a_1d_1 + a_1d_2 + a_2d_1 + b_1c_1 + b_1c_2 + b_2c_1) \\
&+ (d_1a_2c_2 + b_1d_1 + b_2d_2)
\end{aligned}$$

In the next section we will analyze how to share affine equivalent S-boxes.

4.4 Sharing of affine equivalent S-boxes

In this section we are introducing *affine equivalent* S-boxes. When analyzing properties for all possible S-boxes of a fixed size, it is useful to classify the set of S-boxes under a proper equivalence relation. The reason is, that the number of possible invertible functions from n inputs to n outputs is $2^n!$. We start this section with an introduction to affine equivalent S-boxes. We then present, as a result of this thesis, a theorem about sharing of affine equivalent S-boxes.

4.4.1 Introduction to affine equivalent S-boxes

In this section we define *affine equivalent* S-boxes. We start with a recall of the definition of affine equivalence from Section 2.1.

Definition 4.4.1 (affine equivalent S-box[34])**:** Two S-boxes $S_1, S_2 : \mathbb{F}_2^n \to \mathbb{F}_2^n$ are called *affine equivalent* if and only if there exist two invertible affine mappings A_1, A_2 such that $S_2 = A_2^{-1} \circ S_1 \circ A_1$. Where an affine mapping A can be written as $A(x) = L(x) + c$ for $L \in \mathbb{F}_2^{n \times n}$ and $c \in \mathbb{F}_2^n$. The affine mapping A is called invertible if and only if $\det L \neq 0$.

In Table 4.2 we give the numbers of invertible Boolean functions from $\mathbb{F}_2^n \mapsto \mathbb{F}_2^n$ and the numbers of invertible Boolean functions under affine equivalence.

n	1	2	3	4	5
Number of invertible Boolean functions	2	24	40320	2092278988800	263130836933693530167218012160000000
Number of invertible Boolean functions under the equivalence of affine transformation	1	1	4	302	2569966041123963092

Table 4.2: Number of invertible Boolean functions from $\mathbb{F}_2^n \mapsto \mathbb{F}_2^n$ and number of invertible Boolean functions under affine equivalence for $n = 1 \ldots 5$.

As we see, the number of equivalence classes under this equivalence relation is relatively small compared to the number of all possible invertible

Boolean functions.

4.4.2 Sharing of affine equivalent S-boxes

In the following we analyze if it suffices to find a sharing for one representative of a class under affine equivalence to obtain a sharing for all S-boxes in this class. We therefore are of course interested in sharings which fulfill the correctness 4.3.1, non-completeness 4.3.2 and balancedness 4.3.3 properties.

Let S_1 and S_2 be two affine equivalent n bit S-boxes. That means that there exist two invertible affine mappings $A_1 = L_1 + b_1$ and $A_2 = L_2 + b_2$ with $L_1, L_2 \in \mathrm{GL}_n(\mathbb{F}_2)$ and $b_1, b_2 \in \mathbb{F}_2^n$ such that we can write the S-boxes in the following way

$$S_2(\mathbf{x}) = L_2^{-1} S_1 \left(L_1 \cdot \mathbf{x} + b_1\right) + b_2 \; \forall \mathbf{x} \in \mathbb{F}_2^n$$

For the following we split this transformation into two parts

$$\tilde{S}_1(\mathbf{x}) = S_1 \left(L_1 \cdot \mathbf{x} + b_1\right) \tag{4.7}$$

$$S_2(\mathbf{x}) = L_2^{-1} \tilde{S}_1 \left(\mathbf{x}\right) + b_2 \tag{4.8}$$

We start by analyzing the transformation 4.7. Since $\mathbf{x} \mapsto L_1 \cdot \mathbf{x} + b_1$ is a bijective mapping on $\mathbb{F}_2^n$, for invertible linear mappings L_1, we can construct a bijective mapping on $\mathbb{F}_2^{n \cdot s}$, where s denotes the number of shares, as follows. For $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{F}_2^n$ we denote by $\bar{\mathbf{x}} \in \mathbb{F}_2^{n \cdot s}$ the shared vector of $\mathbf{x}$, with shares $x_{i,j},\ j \in \{1, \ldots, s\}$ for all coordinates $i \in \{1, \ldots, n\}$. We order the shares in the shared vector $\bar{\mathbf{x}}$ in the following way

$$\bar{\mathbf{x}} = (x_{1,1}, x_{2,1}, \ldots, x_{n,1}, x_{1,2}, \ldots, x_{n,2}, \ldots, x_{1,s}, \ldots, x_{n,s}) \in \mathbb{F}_2^{n \cdot s}.$$

We further require that

$$x_i = \bigoplus_{k=1}^{s} x_{i,k} \; \forall i \in \{1, \ldots, n\}.$$

We now construct an affine bijective mapping on $\mathbb{F}_2^{n \cdot s}$ by setting

$$\bar{L_1} = \begin{pmatrix} L_1 & 0 & \cdots & 0 \\ 0 & L_1 & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & L_1 \end{pmatrix} \in \mathbb{F}_2^{n \cdot s \times n \cdot s}, \quad \bar{b_1} = \begin{pmatrix} b_1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \in \mathbb{F}_2^{n \cdot s}.$$

By defining the mapping on $F_2^{n \cdot s}$ by setting

$$\bar{\mathbf{x}} \mapsto \bar{L}_1 \cdot \bar{\mathbf{x}} + \bar{b}_1, \tag{4.9}$$

we get a bijective mapping on $\mathbb{F}_2^{n \cdot s}$ since $\det \bar{L}_1 \neq 0 \Leftrightarrow \det L_1 \neq 0$.

We now focus on equation 4.7:
For the S-box $\tilde{S}_1 : \mathbb{F}_2^n \mapsto \mathbb{F}_2^n$, with coordinate functions $F_i : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ for $i \in \{1, \ldots, n\}$, we write

$$\tilde{S}_1(\mathbf{x}) = \begin{pmatrix} F_1(\mathbf{x}) \\ \vdots \\ F_n(\mathbf{x}) \end{pmatrix} \in \mathbb{F}_2^n.$$

For the shared S-box $\tilde{S}_1^s$, with coordinate functions $F_{i,j}$ for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, s\}$ we write

$$\tilde{S}_1^s(\bar{\mathbf{x}}) = \begin{pmatrix} F_{11}(\bar{\mathbf{x}}) \\ F_{21}(\bar{\mathbf{x}}) \\ \vdots \\ F_{n1}(\bar{\mathbf{x}}) \\ \vdots \\ F_{1s}(\bar{\mathbf{x}}) \\ \vdots \\ F_{ns}(\bar{\mathbf{x}}) \end{pmatrix} \in \mathbb{F}_2^{n \cdot s}.$$

To obtain the correctness property 4.3.1, we require that

$$F_i = \bigoplus_{k=1}^{s} F_{i,k} \ \forall i \in \{1, \ldots, n\}.$$

We now again construct a mapping on $F_2^{n \cdot s}$

$$\bar{\mathbf{x}} \mapsto \bar{L}_2^{-1} \cdot \bar{\mathbf{x}} + \bar{b}_2 \tag{4.10}$$

by setting $\bar{L}_2^{-1}$ and $\bar{b}_2$ as denoted as below.

$$\bar{L}_2^{-1} = \begin{pmatrix} L_2^{-1} & 0 & \cdots & 0 \\ 0 & L_2^{-1} & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & L_2^{-1} \end{pmatrix} \in \mathbb{F}_2^{n \cdot s \times n \cdot s}, \quad \bar{b}_2 = \begin{pmatrix} b_2 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \in \mathbb{F}_2^{n \cdot s}.$$

As for the mapping (4.9), the mapping $\bar{\mathbf{x}} \mapsto \bar{L}_2^{-1} \cdot \bar{\mathbf{x}} + \bar{b}_2$ is invertible if and only if $\det L_2 \neq 0$.

Theorem 4.4.1: Let S_1 and S_2 be two affine equivalent S-boxes. If there exists a sharing with s shares for the S-box S_1 that fulfills the correctness 4.3.1, balancedness 4.3.3 and the non-completeness property 4.3.2, then there also exists a sharing with s shares for the S-box S_2 which fulfills the correctness, balancedness and non-completeness property.

Proof: In the following we prove, that the balancedness, correctness and non-completeness property are invariant under the transformations (4.9) and (4.10).
We begin with the property of balancedness. Since the above described mappings (4.9) and (4.10) are bijective on $\mathbb{F}_2^{n \cdot s}$, the S-box S_2 fulfills the uniformity property if we apply these mappings.
Next we show, that the mappings (4.9) and (4.10) preserve the correctness property. Since the mapping (4.9) only operates on the input variables of S-box S_1, we assume without loss of generality, that $S_2 = L_2^{-1} S_1(\mathbf{x}) + b_2$. Since there exists a correct sharing for the S-box S_1 we can write the S-box S_2 as follows:

$$
\begin{aligned}
L_2^{-1} S_1(\mathbf{x}) + b_2 &= L_2^{-1} \begin{pmatrix} F_1(\mathbf{x}) \\ \vdots \\ F_n(\mathbf{x}) \end{pmatrix} + b_2 \\
&= L_2^{-1} \begin{pmatrix} F_{11}(\bar{\mathbf{x}}) + \cdots + F_{1s}(\bar{\mathbf{x}} \\ \vdots \\ F_{n1}(\bar{\mathbf{x}}) + \cdots + F_{ns}(\bar{\mathbf{x}} \end{pmatrix} + b_2 \\
&= L_2^{-1} \begin{pmatrix} F_{11}(\bar{\mathbf{x}}) \\ \vdots \\ F_{n1}(\bar{\mathbf{x}}) \end{pmatrix} + \cdots + L_2^{-1} \begin{pmatrix} F_{1s}(\bar{\mathbf{x}}) \\ \vdots \\ F_{ns}(\bar{\mathbf{x}}) \end{pmatrix} + b_2 \\
&= \begin{pmatrix} \tilde{F}_1(\mathbf{x}) \\ \vdots \\ \tilde{F}_n(\mathbf{x}) \end{pmatrix} + b_2 = S_2(\mathbf{x})
\end{aligned}
$$

What remains to proof is the property of non-completeness. Therefore we again consider without loss of generality only the case where $S_2(\mathbf{x}) = S_1(L_1\mathbf{x} + b_1)$ and assume further that $b_1 = 0 \in F_2^n$ since addition of constant terms does not affect the non-completeness property. We now take a closer look at the transformation from S-box $\bar{S}_1^s$ into the S-box $\bar{S}_2^s$:

$$
\begin{aligned}
S_1^s(\bar{L}_1 \cdot \bar{\mathbf{x}}) &= \begin{pmatrix} F_{11}(\bar{L}_1 \cdot \bar{\mathbf{x}}) \\ \vdots \\ F_{ns}(\bar{L}_1 \cdot \bar{\mathbf{x}}) \end{pmatrix} = \begin{pmatrix} F_{11}(\bar{L}_1 \cdot \bar{\mathbf{x}}^1) \\ \vdots \\ F_{ns}(\bar{L}_1 \cdot \bar{\mathbf{x}}^s) \end{pmatrix} \\
&= \begin{pmatrix} F_{11}\left(\bar{L}_1 \cdot (0, \ldots, 0, x_{1,2}, \ldots, x_{n,2}, \ldots, x_{1,s}, \ldots, x_{n,s})\right) \\ \vdots \\ F_{ns}\left(\bar{L}_1 \cdot (x_{1,1}, \ldots, x_{n,1}, \ldots, x_{1,s-1}, \ldots, x_{n,s-1}, 0, \ldots, 0)\right) \end{pmatrix} \\
&= \begin{pmatrix} F_{11}\left(0, \ldots, 0, \tilde{x}_{1,2}, \ldots, \tilde{x}_{n,2}, \ldots, \tilde{x}_{1,s}, \ldots, \tilde{x}_{n,s}\right) \\ \vdots \\ F_{ns}\left(\tilde{x}_{1,1}, \ldots, \tilde{x}_{n,1}, \ldots, \tilde{x}_{1,s-1}, \ldots, \tilde{x}_{n,s-1}, 0, \ldots, 0\right) \end{pmatrix} \\
&= \begin{pmatrix} F_{11}(\tilde{\mathbf{x}}^1) \\ \vdots \\ F_{ns}(\tilde{\mathbf{x}}^s) \end{pmatrix} = \begin{pmatrix} F_{11}(\tilde{\mathbf{x}}) \\ \vdots \\ F_{ns}(\tilde{\mathbf{x}}) \end{pmatrix} = S_2^s(\tilde{\mathbf{x}}).
\end{aligned}
$$

□

As we have shown in theorem 4.4.1 it suffices to concentrate on representatives of classes under affine equivalence, to obtain a sharing for that class.

Assume we have a sharing $\bar{S} : \mathbb{F}_2^{n \cdot s} \mapsto \mathbb{F}_2^{n \cdot s}$ for a given S-box S. Then we can construct more sharings of the S-box S out of $\bar{S}$ by adding vectors $\mathbf{a} \in \mathbb{F}_2^{n \cdot s}$ such that these vectors $\mathbf{a}$ do not violate the correctness property. The addition of vectors $\mathbf{a} \in \mathbb{F}_2^{n \cdot s}$ clearly does not affect the balancedness property, since the transformation $\bar{S} \mapsto \bar{S} + \mathbf{a}$ is bijective for all $\mathbf{a} \in \mathbb{F}_2^{n \cdot s}$. We now want to formalize the set of vectors which do not affect the correctness property of the S-box $\bar{S}$. We therefore define the set $\mathcal{C}_{n\cdots}$ as follows

$$
\mathcal{C}_{n\cdots} := \left\{ \mathbf{x} \in \mathbb{F}_2^{n \cdot s} : \mathbf{x} = \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{pmatrix}, \text{ with } \mathbf{x}_i \in \mathbb{F}_2^s \text{ and } \mathrm{w}_H(\mathbf{x}_i) \text{ is even.} \right\}
$$

The cardinality of the set $\mathcal{C}_{n\cdots}$ is $2^{n \cdot (s-1)}$. Hence we can construct $2^{n \cdot (s-1)}$ sharings out of the sharing $\bar{S}$ which of course fulfill the balancedness, correctness and non-completeness property.

4.5 Summary

In this chapter we have discussed multiple techniques of secret sharing. We have seen that it is, in order to obtain a secure sharing scheme, important to fulfill certain criteria. Namely these are, the non-completeness property to gain independent intermediate results and in consequence gain resistance against hardware effects like glitches. The uniformity property to resist statistical attacks, especially when considering pipelined implementations or cryptographic functions which contain multiple identic round transformations. Of course it is mandatory that the obtained sharing fulfills the correctness property. We have also seen that it suffices to focus on representatives of classes under affine equivalence in order to share the entire classes.

Chapter 5

Sharing of affine equivalent S-boxes

In the following chapter we are examining two different approaches for constructing realizations for given vectorial Boolean functions, such that these realizations are balanced, correct and non-complete. The first approach uses the truth table to evaluate the requirements. The second approach is to work with the ANF. We will see, that in both approaches there are properties which are easy to evaluate, and properties which are difficult to evaluate. The main focus of this chapter are 3×3 S-boxes with algebraic degree at most two. We only investigate representatives of the equivalence classes under affine equivalence presented in Section 4.4.

In Section 5.1 we will characterize the properties balancedness 4.3.3, correctness 4.3.1 and non-completeness 4.3.2 using the truth table of a shared function. In Section 5.2 we will analyze the equivalence classes of 3×3 S-boxes under affine equivalence. Sections 5.3, 5.4 and 5.5 will describe three more methods to find realizations which fulfill the balancedness, correctness and non-completeness properties. Finally we will describe in Section 5.6 the extension of 3×3 S-boxes to 4×4 S-boxes.

5.1 Characterizing properties in the truth table

In this section we characterize the properties balancedness 4.3.3, non-completeness 4.3.2 and correctness 4.3.1 in the truth table of a shared function. We start with the property of balancedness in Section 5.1.1 and describe the property of non-completeness in the truth table in Section 5.1.2. In Section

5.1.3 we analyze the property of correctness in the truth table of a shared function and describe a method to directly construct a balanced and correct truth table for a shared function. In Section 5.1.5 we describe a search based method to construct a balanced, correct and non-complete realization for a given S-box.

5.1.1 Balancedness in the truth table

We start with a recall of the definition of balancedness for a vectorial Boolean function F.

Definition 5.1.1 ([14])**:** Let F be an (n, m) function. Then F is *balanced*, if it takes every value in $\mathbb{F}_2^m$ the same number of times, that is 2^{n-m}.

Hence, we see that a function F is balanced by means of its truth table, if we see each output value exactly $n - m$ times. In the case of an S-box this is equivalent to, that we see each output value exactly once.

5.1.2 Non-completeness in the truth table

Let F be a function mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ and $\bar{F}$ be its realization with s shares mapping from $\mathbb{F}_2^{n \cdot s}$ to $\mathbb{F}_2^{m \cdot s}$. Let further be $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$ the inputs and outputs of F respectively and $\bar{x}_1, \ldots, \bar{x}_n$ and $\bar{y}_1, \ldots, \bar{y}_m$ be the inputs and outputs of $\bar{F}$ respectively. Since $\bar{F}$ fulfills the non-completeness property, we know that the output shares $y_{i,j}$ for all $i \in \{1, \ldots, m\}$ are independent of the input shares $x_{k,j}$ for all $k \in \{1, \ldots, n\}$. We now characterize the non-completeness property in the truth table of $\bar{F}$. Assume we reorder the columns of input values in the truth table of $\bar{F}$ such that the input columns become $x_{1,1}, x_{2,1}, \ldots, x_{n,1}, x_{1,2}, \ldots, x_{n,s}$. We arrange the columns of the output values analogously such that the columns become $y_{1,1}, y_{2,1}, \ldots, y_{m,1}, y_{1,2}, \ldots, y_{m,s}$. Hence the first m columns of the output values are now independent of the first n columns of input values. If the input values are ordered lexicographically, we can see that the first $2^{n \cdot (s-1)}$ rows of the output values $y_{1,1}, y_{2,1}, \ldots, y_{m,1}$ recur 2^n times. A similar observation can be made for the output shares with index j if we reorder the columns of the input values according to the input shares with index j.

5.1.3 Correctness in the truth table

Let F be a function mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ and $\bar{F}$ be its realization with s shares mapping from $\mathbb{F}_2^{n \cdot s}$ to $\mathbb{F}_2^{m \cdot s}$. Let further be $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$ the inputs and outputs of F respectively and $\bar{x}_1, \ldots, \bar{x}_n$ and $\bar{y}_1, \ldots, \bar{y}_m$ be

the shared inputs and shared outputs of $\bar{F}$ respectively. We can characterize the correctness property in the truth table of $\bar{F}$ by performing the following comparison. We first sum the columns $\oplus_{j=1}^{s} x_{i,j} = \hat{x}_i$ for all $i \in \{1, \ldots, n\}$ and $\oplus_{j=1}^{s} y_{i,j} = \hat{y}_i$ for all $i \in \{1, \ldots, m\}$. We then check if the function output $F(\hat{X}_1, \ldots, \hat{X}_n)$ equals the values stated in $\hat{y}_1, \ldots, \hat{y}_m$ for all $(\hat{X}_1, \ldots, \hat{X}_n) \in (\hat{x}_1, \ldots, \hat{x}_n)$. Thus, we do not observe the correctness of a function $\bar{F}$ with respect to a function F directly in the truth table.

5.1.4 Balancedness and correctness

In this section we show that it is possible to construct a truth table for the function $\bar{F}$ which fulfills the properties balancedness 4.3.3 and correctness 4.3.1 with respect to F. We describe this construction below.

Algorithm 4: Input: A vectorial Boolean function $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$
Output: A correct and balanced truth table of a function $\bar{F} : \mathbb{F}_2^{n \cdot s} \mapsto \mathbb{F}_2^{m \cdot s}$ with respect to F
For each row i in the truth table of F perform the following steps:

1. The entries in that row i correspond to an assignment $(X_1, \ldots, X_n)$ of the input variables $x_1, \ldots, x_n$ and the function output $(Y_1, \ldots, Y_m) = F(X_1, \ldots, X_n)$.

2. Construct the set $\mathcal{X}_{X_1,\ldots,X_n}$ of all possible assignments $(\bar{X}_1, \ldots, \bar{X}_n)$ of the variables $\bar{x}_1, \ldots, \bar{x}_n$ such that $\oplus_{j=1}^{s} X_{i,j} = X_i$ for all $i \in \{1, \ldots, n\}$.

3. Construct the set $\mathcal{Y}_{Y_1,\ldots,Y_m}$ of all possible assignments $(\bar{Y}_1, \ldots, \bar{Y}_m)$ of the outputs $\bar{y}_1, \ldots, \bar{y}_m$ such that $\oplus_{j=1}^{s} Y_{i,j} = Y_i$ for all $i \in \{1, \ldots, m\}$. Note that the size of the set $\mathcal{X}_{X_1,\ldots,X_n}$ is $2^{n \cdot (s-1)}$ and the size of the set $\mathcal{Y}_{Y_1,\ldots,Y_m}$ is $2^{m \cdot (s-1)}$.

4. Write the elements of the set $\mathcal{X}_{X_1,\ldots,X_n}$ in the columns $\bar{x}_1, \ldots, \bar{x}_n$ of the truth table of $\bar{F}$ and write the elements of the set $\mathcal{Y}_{Y_1,\ldots,Y_m}$ 2^{n-m} times in the columns $\bar{y}_1, \ldots, \bar{y}_m$.

The steps 1 – 4 are performed for all possible 2^n assignments $(X_1, \ldots, X_n)$ of the inputs $x_1, \ldots, x_n$ and corresponding outputs $(y_1, \ldots, y_m)$. The Algorithm 4 constructs a balanced and correct truth table for the function $\bar{F}$ with respect to the function F. Generally this construction does not result in a function $\bar{F}$ which fulfills the non-completeness property.

5.1.5 A truth table search

We now present a first approach to construct a shared S-box which fulfills all three properties. We operate on the truth table of the shared S-box. The approach is basically a search method which tries to find the truth table of the shared S-box, such that the shared S-box fulfills the balancedness 4.3.3, correctness 4.3.1 and non-completeness 4.3.2 properties. We describe this search for quadratic 3×3 S-boxes below.

The approach works as follows. We first consider the property non-completeness and balancedness for each output share in the truth table. We then employ the correctness property to the method. Finally we add the algebraic degree of the resulting functions into the search.

Let S be a 3×3 S-box with inputs $(u, v, w) \in \mathbb{F}_2^3$ and outputs $(x, y, z) \in \mathbb{F}_2^3$, i.e. $S(u, v, w) = (x, y, z)$. We define the shared inputs and outputs as $(u_1, u_2, u_3, v_1, v_2, v_3, w_1, w_2, w_3) \in \mathbb{F}_2^9$ and $(x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3) \in \mathbb{F}_2^9$ respectively.

The non-completeness property demands, that every output must be independent of at least one share of each input. Without loss of generality we require that the outputs x_1, y_1 and z_1 are independent of the inputs u_1, v_1 and w_1. This means that the outputs x_1, y_1 and z_1 can be seen as functions with only 6 inputs. The fact that the output x_1 is independent of the input u_1 means, that for all choices of $(U_2, U_3, V_1, V_2, V_3, W_1, W_2, W_3) \in \mathbb{F}_2^8$ the following holds

$$x_1(0, U_2, U_3, V_1, V_2, V_3, W_1, W_2, W_3) = x_1(1, U_2, U_3, V_1, V_2, V_3, W_1, W_2, W_3).$$

The same holds for the inputs v_1 and w_1. Hence, we get that the output function of x_1 only depends on the inputs $(u_2, u_3, v_2, v_3, w_2, w_3)$. We call the sequence of outputs of x_1 which only depends on the inputs $(u_2, u_3, v_2, v_3, w_2, w_3)$ as $\mathcal{X}_1 \in \mathbb{F}_2^6$. The same holds for the outputs y_1 and z_1, we name these sequences $\mathcal{Y}_1$ and $\mathcal{Z}_1$ respectively. Since $\mathcal{X}_1, \mathcal{Y}_1$ and $\mathcal{Z}_1$ are independent of (u_1, v_1, w_1), we get that these sequences repeat $2^3 = 8$ times in the truth table. We illustrate this in Table 5.1.

Since we want that the resulting truth table is balanced, we deduce that the sequences $\mathcal{X}_1, \mathcal{Y}_1$ and $\mathcal{Z}_1$ must contain the same number of $1's$ as $0's$. This reduces the possible choices for $\mathcal{X}_1$ to $\binom{2^6}{2^5} \leq 2^{61}$. By combining all possible sequences $\mathcal{X}_1, \mathcal{Y}_1, \mathcal{Z}_1, \mathcal{X}_2, \mathcal{Y}_2, \mathcal{Z}_2, \mathcal{X}_3, \mathcal{Y}_3, \mathcal{Z}_3$ (that is, we combine all possible choices of $\mathcal{X}_1$ with all possible choices of $\mathcal{Y}_1$ with all ... possible choices of $\mathcal{Z}_3$) we deduce, that the search space for three shared bits, to obtain a balanced and non-complete truth table is smaller or equal to $2^{61 \cdot 9} = 2^{549}$

u_1	v_1	w_1	u_2	v_1	w_1	u_1	v_1	w_1	x_1	y_1	z_1	x_2	y_2	z_2	x_3	y_3	z_3
0	0	0	0	0	0	0	0	0				$\divideontimes$	$*$	$\boxtimes$	$\star$	$\circledast$	$\boxplus$
$\vdots$	$\vdots$	$\vdots$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\mathcal{X}_1$	$\mathcal{Y}_1$	$\mathcal{Z}_1$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	0	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$				$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	1	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$				$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\mathcal{X}_1$	$\mathcal{Y}_1$	$\mathcal{Z}_1$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	0	1	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$				$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	1	0	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$				$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\mathcal{X}_1$	$\mathcal{Y}_1$	$\mathcal{Z}_1$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	0	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$				$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	1	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$				$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\mathcal{X}_1$	$\mathcal{Y}_1$	$\mathcal{Z}_1$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
0	1	1	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$				$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	0	0	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$				$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\mathcal{X}_1$	$\mathcal{Y}_1$	$\mathcal{Z}_1$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	0	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$				$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	1	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$				$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\mathcal{X}_1$	$\mathcal{Y}_1$	$\mathcal{Z}_1$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	0	1	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$				$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	1	0	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$				$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\mathcal{X}_1$	$\mathcal{Y}_1$	$\mathcal{Z}_1$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	0	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$				$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	1	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$				$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$	$\mathcal{X}_1$	$\mathcal{Y}_1$	$\mathcal{Z}_1$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	1	1	1	1	1	1	1	1				$\divideontimes$	$*$	$\boxtimes$	$\star$	$\circledast$	$\boxplus$

Table 5.1: Schematic representation of a truth table with valid non-completeness property for the outputs x_1, y_1 and z_1. $\mathcal{X}_1, \mathcal{Y}_1$ and $\mathcal{Z}_1$ are the sequences which are described in Section 5.1.5.
The various symbols $(\divideontimes, *, \boxtimes, \star, \circledast, \boxplus)$ in the columns x_2, x_3, y_2, y_3, z_2 and z_3 refer to the output of the output shares x_2, x_3, y_2, y_3, z_2 and z_3.

We now employ the correctness property to reduce the size of the search space. We therefore construct the third share of each output bit via the correctness property. Assume we already have the outputs of x_1 and x_2 fixed by the choice of sequences $\mathcal{X}_1$ and $\mathcal{X}_2$, then we can simply deduce the output x_3 via the correctness property 4.3.1. This gives

$$x_3 = x - x_1 - x_2 = x + x_1 + x_2.$$

Hence we can limit a theoretical search to all combinations of all possible balance sequences $\mathcal{X}_1, \mathcal{Y}_1, \mathcal{Z}_1, \mathcal{X}_2, \mathcal{Y}_2, \mathcal{Z}_2$. We deduce that the size of the search space is smaller or equal to $2^{61 \cdot 6} = 2^{366}$.

If we assume that the desired share functions have algebraic degree 2, we can reduce the search space for each shared output bit to $2^{\binom{6}{2}+\binom{6}{1}+1} = 2^{22}$. Hence we get that the size of the search space is smaller or equal to $2^{22 \cdot 6} = 2^{132}$.

We can enhance the effort of this search by using the following technique. We start by presenting the idea of this method. Instead of combining all possible choices for the outputs $(x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3)$ in one major loop, we combine subsets of these outputs separately.

The method works as follows. We store all possible combinations of outputs of x_1, x_2 and x_3 in a table such that (x_1, x_2, x_3) form a correct and balanced output for the first output. We do the same processing for the second output bit, with the outputs (y_1, y_2, y_3). We now reduce both tables in the following way. We combine each possible combination of (x_1, x_2, x_3) with each possible combination of (y_1, y_2, y_3) and keep only combinations of (x_1, x_2, x_3) and (y_1, y_2, y_3) which form a balanced output (i.e. as a function from $\mathbb{F}_2^9 \mapsto \mathbb{F}_2^6$). For the third output bit we also store all possible combinations of outputs of z_1, z_2 and z_3 in a table such that (z_1, z_2, z_3) form a correct and balanced output. We can now reduce the tables of outputs (x_1, x_2, x_3) and (y_1, y_2, y_3) further by combining both with the outputs (z_1, z_2, z_3). After this step only combinations in (x_1, x_2, x_3), (y_1, y_2, y_3) and (z_1, z_2, z_3) remain such that the combination of any two output vectors for example $(x_1, x_2, x_3, z_1, z_2, z_3)$ forms correct and balanced functions. Finally we check all combinations of the remaining table entries in (x_1, x_2, x_3), (y_1, y_2, y_3) and (z_1, z_2, z_3) for balancedness.

Since we do not know how many combinations remain after each of the steps, we can not estimate its true efforts. By estimating the worst case we get, that the size of the search space is 2^{366}, or alternatively for quadratic share functions 2^{132}, as before.

5.2 Sharing of 3×3 S-boxes with algebraic degree ≤ 2.

In this section we use the ANF, which was introduced in Section 2.1 and Section 2.4, as representation of Boolean functions. We start with an introduction to sharing with the ANF. We further determine which of the four classes of affine equivalent 3×3 S-boxes can be shared with three shares by only varying the assignments of the input variables.

5.2.1 Introduction

Assume we have a realization $\bar{F}(\bar{x}_1, \ldots, \bar{x}_n)$ for a given function $F(x_1, \ldots, x_n)$. Then we can easily check if the properties non-completeness and correctness are fulfilled, if we examine the ANF of both functions. We simply check the non-completeness property by controlling the input set of each coordinate function of $\bar{F}$. To check the correctness of the function $\bar{F}$ with respect to F we proceed as follows. For each coordinate function F_i of F evaluate the equality of the polynomials $F_i(\oplus_{j=1}^{s} x_{1,j}, \ldots, \oplus_{j=1}^{s} x_{n,j})$ and $\oplus_{j=1}^{s} F_{i,j}$. Thereby $F_{i,j}$ denotes the coordinate functions of $\bar{F}$ and s the number of shares.

On the other hand, if we have the ANF of a function F, we cannot check the balancedness of F without evaluating all possible inputs of F. Thus, given a function $\bar{F}$ in ANF, we can easily detect if te function fulfills the non-completeness and correctness property, but we cannot directly verify the balancedness of the function.

Direct sharing

We now describe how we build a function $\bar{F} : \mathbb{F}_2^9 \mapsto \mathbb{F}_2^9$ which fulfills the non-completeness and correctness properties for a given function $F : \mathbb{F}_2^3 \mapsto \mathbb{F}_2^3$. Let S be an arbitrary S-box mapping from $\mathbb{F}_2^3 \mapsto \mathbb{F}_2^3$ with algebraic degree at most two. Then S can be written as

$$S(u,v,w) = \begin{pmatrix} F_1(u,v,w) \\ F_2(u,v,w) \\ F_3(u,v,w) \end{pmatrix} = \begin{array}{l} \alpha_1 uv + \beta_1 uw + \gamma_1 vw + \delta_1 u + \epsilon_1 v + \zeta_1 w + \eta_1 \\ \alpha_2 uv + \beta_2 uw + \gamma_2 vw + \delta_2 u + \epsilon_2 v + \zeta_2 w + \eta_2 \\ \alpha_3 uv + \beta_3 uw + \gamma_3 vw + \delta_3 u + \epsilon_3 v + \zeta_3 w + \eta_3 \end{array}$$

with coordinate functions F_1, F_2, F_3 inputs $(u, v, w) \in \mathbb{F}_2^3$ and coefficients $\alpha_i, \beta_i, \gamma_i, \delta_i, \epsilon_i, \zeta_i, \eta_i \in \mathbb{F}_2$ for $i \in \{1, 2, 3\}$. To share this S-box we replace u, v, w with $\bar{u}, \bar{v}, \bar{w}$, where $\bar{u} = (u_1 + u_2 + u_3)$, $\bar{v} = (v_1 + v_2 + v_3)$ and $\bar{w} = (w_1 + w_2 + w_3)$. We further expand the set of coordinate functions. We

therefore replace F_i with (F_{i1}, F_{i2}, F_{i3}) for all $i \in \{1, 2, 3\}$. What remains is, to assign the shared terms of the functions F_1, F_2, F_3 to the functions $F_{11}, F_{12}, \ldots, F_{32}, F_{33}$. We therefore proceed as follows for every function F_i:

- assign linear terms with share index j to function $F_{i,j-1}$.
- assign quadratic terms with indices j and $j+1$ to function $F_{i,j-1}$.
- assign quadratic terms with only index j to $F_{i,j-1}$

At the end of this assignment the functions $F_{11}, F_{12}, \ldots, F_{32}, F_{33}$ fulfill the non-completeness and correctness property such that

$$\begin{aligned} F_{11} + F_{12} + F_{13} &= F_1(\bar{u}, \bar{v}, \bar{w}) \\ F_{21} + F_{22} + F_{23} &= F_2(\bar{u}, \bar{v}, \bar{w}) \\ F_{31} + F_{32} + F_{33} &= F_3(\bar{u}, \bar{v}, \bar{w}). \end{aligned}$$

The above described assignment is called *direct sharing*, if the above described assignment gives a balanced function and was introduced by Nikova et al. in [44]. To share the decompositions of the block ciphers Noekeon [44] and Present [51] (see Section 4.3.8) direct sharing has been successfully applied.

Notation: We call the terms $u_i \cdot v_j$ with $i \neq j$ *fixed terms*, since they have to be assigned to the share function F_k with $k = \{1, 2, 3\} \setminus \{i, j\}$. On the other hand we call terms which depend only on one share index *free terms*.

Assignments of free terms

We now describe a set of assignments of free terms, which is used later on in this section. The idea is, to not fix the assignment of a free term, but rather verify all assignments of the free term for balancedness, that fulfill the correctness and non-completeness property. However, we only consider assignments such that the number of free terms which are assigned to each share function of one bit is equal.

Let S be a 3×3 S-box and $\mathcal{T}_S$ be the set of all free terms, when we share S with three shares. Let further $\mathcal{F}_S := \{F_{11}, \ldots, F_{33}\}$ be the set of coordinate functions. The mapping

$$\tau : \mathcal{T}_S \to \mathcal{F}_S \quad \text{defined by} \quad t \in \mathcal{T}_S \mapsto \tau(t) \in \mathcal{F}_S$$

is called assignment. We call the assignment τ correct, if and only if the assignment of all free terms in $\mathcal{T}_S$ yields correct functions $F_{11}, \ldots, F_{33}$ with

respect to S. The assignment τ is called non-complete, if and only if the functions $F_{11}, \ldots, F_{33}$ fulfill the non-completeness property 4.3.2. The assignment τ is called equated, if and only if all share functions F_{ij} of each output bit have an equal number of free terms assigned. That is $|\tau^{-1}(F_{i,1})| = |\tau^{-1}(F_{i,2})| = |\tau^{-1}(F_{i,3})|$ for all $i \in \{1, 2, 3\}$.

We define the set of assignments $\mathcal{A}_S$, which we use for the rest of this section, to be

$$\mathcal{A}_S := \{\tau : \mathcal{T}_S \to \mathcal{F}_S | \tau \text{ is correct, non-complete and equated}\}.$$

Note that all assignments in $\mathcal{A}_S$ fulfill the correctness 4.3.1 and non-completeness property 4.3.2. What remains to be evaluated is the property of balancedness 4.3.3. We start with the analysis of the four equivalence classes below.

5.2.2 Class $\mathcal{S}_0^3$

We start with class $\mathcal{S}_0^3$ and state one representative S_0 of this class in ANF below:

$$\begin{aligned} F_1(u, v, w) &= u \\ F_2(u, v, w) &= v \\ F_3(u, v, w) &= w \end{aligned}$$

As we see, this S-box has algebraic degree one and is therefore a linear S-box. Applying affine transformations to this S-box S_0, preserves the algebraic degree. Hence, the class $\mathcal{S}_0^3$ contains all affine S-boxes. Sharing this S-box S_0 with the method of direct sharing provides a shared S-box $\bar{S}_0$ which fulfills all required properties. This S-box $\bar{S}_0$ stays correct, non-complete and balanced if we apply permutations to the shared output bits (i.e. $((F_{11}, F_{12}, F_{13}), (F_{21}, F_{22}, F_{23}), (F_{31}, F_{32}, F_{33}))$ that do not violate the non-completeness property. Consequently we get 8 S-boxes. We list one of them below.

$$\begin{aligned} F_{11}(\bar{u}, \bar{v}, \bar{w}) &= u_2, & F_{12}(\bar{u}, \bar{v}, \bar{w}) &= u_3, & F_{13}(\bar{u}, \bar{v}, \bar{w}) &= u_1 \\ F_{21}(\bar{u}, \bar{v}, \bar{w}) &= v_2, & F_{22}(\bar{u}, \bar{v}, \bar{w}) &= v_3, & F_{23}(\bar{u}, \bar{v}, \bar{w}) &= v_1 \\ F_{31}(\bar{u}, \bar{v}, \bar{w}) &= w_2, & F_{32}(\bar{u}, \bar{v}, \bar{w}) &= w_3, & F_{33}(\bar{u}, \bar{v}, \bar{w}) &= w_1 \end{aligned}$$

5.2.3 Class $\mathcal{S}_1^3$

The next equivalence class $\mathcal{S}_1^3$ has algebraic degree 2. We state the S-box S_1 as a representative of this class in ANF below:

$$\begin{aligned} F_1(u,v,w) &= u + wv \\ F_2(u,v,w) &= v \\ F_3(u,v,w) &= w \end{aligned}$$

To evaluate the number of assignments that yield a correct, balanced and non-complete realization in $\mathcal{A}_{S_1}$ it suffices to check different assignments of the free terms for the functions F_{11}, F_{12}, F_{13} with a fixed assignment of the functions $F_{21}, F_{22}, F_{23}, F_{31}, F_{32}, F_{33}$. The reason is, that the functions F_2 and F_3 are linear. If one assignment of the functions $F_{21}, F_{22}, F_{23}, F_{31}, F_{32}, F_{33}$ gives a balanced shared S-box, then all assignments in $\mathcal{A}_{S_1}$ for the functions $F_{21}, \ldots, F_{33}$ give a balanced S-box, since different assignments of $F_{21}, \ldots, F_{33}$ correspond to a permutation of output columns in the truth table and therefore do not affect the balancedness.

It further suffices to only check on those assignments of the free terms in the functions F_{11}, F_{12}, F_{13} that cannot be transformed into one another, by interchanging variables. For the functions F_{11}, F_{12}, F_{13} there are exactly two such different assignments. We list them both below:

$$\begin{aligned} F_{11}(\bar{u},\bar{v},\bar{w}) &= v_2w_3 + v_3w_2 + v_2w_2 + u_2 \\ F_{12}(\bar{u},\bar{v},\bar{w}) &= v_1w_3 + v_3w_1 + v_3w_3 + u_3 \\ F_{13}(\bar{u},\bar{v},\bar{w}) &= v_1w_2 + v_2w_1 + v_1w_1 + u_1 \end{aligned} \tag{5.1}$$

$$\begin{aligned} F_{11}(\bar{u},\bar{v},\bar{w}) &= v_2w_3 + v_3w_2 + v_2w_2 + u_3 \\ F_{12}(\bar{u},\bar{v},\bar{w}) &= v_1w_3 + v_3w_1 + v_3w_3 + u_1 \\ F_{13}(\bar{u},\bar{v},\bar{w}) &= v_1w_2 + v_2w_1 + v_1w_1 + u_2 \end{aligned} \tag{5.2}$$

Both assignments (5.1), (5.2) of the functions F_{11}, F_{12}, F_{13} yield a correct, balanced and non-complete S-box. Hence we get 16 realizations that fulfill the correctness, balancedness and non-completeness property out of all 16 assignments in $\mathcal{A}_{S_1}$. For the sake of completeness we list one of this realizations

of the S-box S_1 below.

$$
\begin{aligned}
F_{11}(\bar{u},\bar{v},\bar{w}) &= v_2w_3 + v_3w_2 + v_2w_2 + u_2\\
F_{12}(\bar{u},\bar{v},\bar{w}) &= v_1w_3 + v_3w_1 + v_3w_3 + u_3\\
F_{13}(\bar{u},\bar{v},\bar{w}) &= v_1w_2 + v_2w_1 + v_1w_1 + u_1\\
F_{21}(\bar{u},\bar{v},\bar{w}) &= v_2\\
F_{22}(\bar{u},\bar{v},\bar{w}) &= v_3\\
F_{23}(\bar{u},\bar{v},\bar{w}) &= v_1\\
F_{31}(\bar{u},\bar{v},\bar{w}) &= w_2\\
F_{32}(\bar{u},\bar{v},\bar{w}) &= w_3\\
F_{33}(\bar{u},\bar{v},\bar{w}) &= w_1
\end{aligned}
$$

5.2.4 Class $\mathcal{S}_2^3$

Below we state the S-box S_2 as representative of the third equivalence class $\mathcal{S}_2^3$ under affine equivalence:

$$
\begin{aligned}
F_1(u,v,w) &= u + wu + wv\\
F_2(u,v,w) &= v + wu\\
F_3(u,v,w) &= w
\end{aligned}
$$

As for the class $\mathcal{S}_1^3$ we only check for balancedness on different assignments of the free terms of the functions F_{11}, F_{12}, F_{13} and F_{21}, F_{22}, F_{23} for a fixed assignment of the functions F_{31}, F_{32}, F_{33}. For this S-box S_2 this means that we have to check 8 different assignments for the property of balancedness. Out of these 8 possible shared S-boxes, one S-box is balanced. Hence we get that 8 out of the 64 possible assignments from the set $\mathcal{A}_{S_2}$ yield a realization of the S-box S_2. We list one realization of the S-box S_2 below.

$$
\begin{aligned}
F_{11}(\bar{u},\bar{v},\bar{w}) &= v_2w_3 + u_2w_3 + v_3w_2 + u_3w_2 + v_2w_2 + u_2w_2 + u_3\\
F_{12}(\bar{u},\bar{v},\bar{w}) &= v_1w_3 + u_1w_3 + v_3w_1 + u_3w_1 + v_3w_3 + u_3w_3 + u_1\\
F_{13}(\bar{u},\bar{v},\bar{w}) &= v_1w_2 + u_1w_2 + u_2w_1 + v_2w_1 + v_1w_1 + u_1w_1 + u_2\\
F_{21}(\bar{u},\bar{v},\bar{w}) &= u_2w_3 + u_3w_2 + u_2w_2 + v_3\\
F_{22}(\bar{u},\bar{v},\bar{w}) &= u_1w_3 + u_3w_1 + u_3w_3 + v_1\\
F_{23}(\bar{u},\bar{v},\bar{w}) &= u_1w_2 + u_2w_1 + u_1w_1 + v_2\\
F_{31}(\bar{u},\bar{v},\bar{w}) &= w_2\\
F_{32}(\bar{u},\bar{v},\bar{w}) &= w_3\\
F_{33}(\bar{u},\bar{v},\bar{w}) &= w_1
\end{aligned}
$$

We observe in the share functions of the first output bit, that the free terms u_1w_1, u_2w_2, u_3w_3 and u_1, u_2, u_3 are not assigned to the same output share. Thus, u_2w_2 and u_2 need to be assigned to different shares, contrary to the method of direct sharing.

5.2.5 Class $\mathcal{S}_3^3$

We now show S-box S_3 as a representative of the fourth equivalence class $\mathcal{S}_3^3$ in ANF below:

$$\begin{aligned} F_1(u,v,w) &= u + vu + w \\ F_2(u,v,w) &= v + vu + w + wv \\ F_3(u,v,w) &= vu + wu + wv \end{aligned}$$

As for the previously discussed classes we only check on different assignments for the free terms of the functions $F_{11}, F_{12}, F_{13},\ F_{21}, F_{22}, F_{23}$ and F_{31}, F_{32}, F_{33}. There are 4 different assignments in each case for the functions F_{11}, F_{12}, F_{13} and F_{31}, F_{32}, F_{33} and 8 different assignments for the functions $F_{21}, F_{22},\ F_{23}$. Thus we get 128 different assignments of free terms on which we have to check the property of balancedness. Unfortunately none of them fulfills the balancedness property. Below we list one unbalanced sharing of the S-box S_3.

$$\begin{aligned} F_{11} &= u_3v_2 + u_2v_3 + u_3v_3 + u_2 + w_3 \\ F_{12} &= u_3v_1 + u_1v_3 + u_1v_1 + u_3 + w_1 \\ F_{13} &= u_2v_1 + u_1v_2 + u_2v_2 + u_1 + w_2 \\ F_{21} &= v_2w_3 + v_3w_2 + u_2v_3 + u_3v_2 + v_3w_3 + u_2v_2 + v_3 + w_2 \\ F_{22} &= v_1w_3 + v_3w_1 + u_1v_3 + u_3v_1 + v_1w_1 + u_3v_3 + v_1 + w_3 \\ F_{23} &= v_1w_2 + v_2w_1 + u_1v_2 + u_2v_1 + v_2w_2 + u_1v_1 + v_2 + w_1 \\ F_{31} &= v_2w_3 + v_3w_2 + u_2v_3 + u_3v_2 + u_2w_3 + u_3w_2 + v_2w_2 + u_3w_3 + u_2v_2 \\ F_{32} &= v_1w_3 + v_3w_1 + u_1v_3 + u_3v_1 + u_1w_3 + u_3w_1 + v_3w_3 + u_1w_1 + u_3v_3 \\ F_{33} &= v_1w_2 + v_2w_1 + u_1v_2 + u_2v_1 + u_1w_2 + u_2w_1 + v_1w_1 + u_2w_2 + u_1v_1 \end{aligned}$$

In the next section we will describe an approach to search for a balanced sharing for this S-box, and accordingly all members of the class $\mathcal{S}_3^3$, by adding additional terms.

constant:	1
linear:	u_1, u_2, u_3, v_1, v_2, v_3, w_1, w_2, w_3
quadratic:	u_1v_1, u_2v_2, u_3v_3, u_1w_1, u_2w_2, u_3w_3, v_1w_1, v_2w_2, v_3w_3
cubic:	$u_1v_1w_1$, $u_2v_2w_2$, $u_3v_3w_3$

Table 5.2: Correction terms up to degree three for 3×3 S-boxes in variables u_1, u_2, u_3, v_1, v_2, v_3, w_1, w_2, w_3.

5.3 Add correction terms

In this section we describe how we may find a balanced 4.3.3 realization, if we start with a function $\bar{F} : \mathbb{F}_2^{n \cdot s} \mapsto \mathbb{F}_2^{m \cdot s}$ which fulfills the correctness 4.3.1 and non-completeness 4.3.2 property for a function $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$. Since we do not know how to turn this function $\bar{F}$ directly into a balanced, correct and non-complete function, we perform a search which we describe in detail below for the case $n = m = s = 3$.

5.3.1 Overview

The search we describe in this section is based on the following idea. If we add certain terms to the shares of one output bit, such that the non-completeness 4.3.2 and correctness 4.3.1 property stay valid, the output of the involved share functions change. Accordingly the distribution of the output changes. Thus, if we add the right terms to the share functions, we may get a balanced 4.3.3 function.

We call terms that are applicable for this procedure *correction terms*. In order to keep the functions correct, we have to add a correction term to an even number of shares of one output bit, or not at all. Hence, we can add any linear term, or quadratic and cubic terms that have the same share index. For a better understanding we list the set of possible correction terms up to algebraic degree three in Table 5.2.

When working with three shares, adding a correction term an even number of times to the shares of one output bit means that we add a term to exactly two shares of one output bit. This preserves the correctness property, since by adding a correction term twice, it cancels out when summing over all share functions for one bit.

Since we do not know which correction terms, or which combinations of correction terms yield a balanced realization, we describe a search based approach below. To reduce the search space, any knowledge about which

combinations of correction terms are more applicable, or which combinations of correction terms are inapplicable at all, would be useful.

In the following sections 5.3.2 – 5.3.5 we describe a procedure, which searches for a balanced, correct and non-complete realization with three shares for a given 3×3 S-box, by adding combinations of correction terms. This procedure is divided into four steps, which are described in sections 5.3.2 – 5.3.5. In Section 5.3.2 we search for correction terms for each output share function. In Section 5.3.3 we search for correction terms for each shared output bit, which fulfill the balancedness 4.3.3 and correctness 4.3.1. In Section 5.3.4 we only keep correction terms which give a balanced function when we combine any two shared output bits. Finally in Section 5.3.5 we search for correction terms which give a balanced realization.

5.3.2 Search potential correction terms for each share function F_{ij}

In the first step we deterministically add possible sums of correction terms to the function F_{ij}. We then evaluate, if the resulting Boolean function is balanced (i.e. the number of 1's is 256). If F_{ij} is balanced, the correction terms are stored for the further steps. This step is performed for all functions F_{ij} for $i, j \in \{1, 2, 3\}$.

5.3.3 Find potential correction terms for each bit

To preserve correctness, the list of correction terms from Section 5.3.2 is examined for each output bit in the following way. We only keep sums of correction terms, such that each occurring correction term is added at most twice to the shared output bit.

For example we could add the following terms to the functions F_{11}, F_{12} and F_{13} of the unbalanced realization of the S-box S_3 from class $\mathcal{S}_3^3$. We add the terms $w_3 + w_2$, $w_1 + w_3$ and $w_2 + w_1$ to F_{11}, F_{12} and F_{13} respectively. The addition of these terms corresponds to a reassignment of the free terms w_1, w_2 and w_3. This leads to the following functions:

$$
\begin{aligned}
F_{11} &= u_3v_2 + u_2v_3 + u_3v_3 + u_2 + w_3 + (w_2 + w_3) \\
F_{12} &= u_3v_1 + u_1v_3 + u_1v_1 + u_3 + w_1 + (w_3 + w_1) \\
F_{13} &= u_2v_1 + u_1v_2 + u_2v_2 + u_1 + w_2 + (w_1 + w_2)
\end{aligned}
$$

We of course only keep such sums of correction terms in this step, which lead to a balanced function from $\mathbb{F}_2^9$ to $\mathbb{F}_2^3$. This step is performed for every output bit of the S-box.

5.3.4 Combine 2 S-box bits and check for balancedness

An arbitrary combination of the correction terms from Section 5.3.3 does in general not lead to a balanced output for the shared S-box. Because of this, we now combine the correction terms of two output bits from Section 5.3.3, such that their combination leads to a balanced function from $\mathbb{F}_2^9$ to $\mathbb{F}_2^6$.

For example we could add the following correction terms to the functions $F_{11}, F_{12}, F_{13}, F_{21}, F_{22}$ and F_{23} of the unbalanced realization of the S-box S_3 from class $\mathcal{S}_3^3$. We add the correction terms $w_3 + w_2$, $w_1 + w_3$ and $w_2 + w_1$ to the functions F_{11}, F_{12} and F_{13} respectively, as above. And add $v_2 + v_3$, $v_1 + v_3$ and $v_1 + v_2$ to the functions F_{21}, F_{22} and F_{23} respectively. This leads to the following functions:

$$
\begin{aligned}
F_{11} &= u_3v_2 + u_2v_3 + u_3v_3 + u_2 + w_3 + (w_2 + w_3)\\
F_{12} &= u_3v_1 + u_1v_3 + u_1v_1 + u_3 + w_1 + (w_3 + w_1)\\
F_{13} &= u_2v_1 + u_1v_2 + u_2v_2 + u_1 + w_2 + (w_1 + w_2)\\
F_{21} &= v_2w_3 + v_3w_2 + u_2v_3 + u_3v_2 + v_3w_3 + u_2v_2 + w_2 + v_3 + (v_2 + v_3)\\
F_{22} &= v_1w_3 + v_3w_1 + u_1v_3 + u_3v_1 + v_1w_1 + u_3v_3 + w_3 + v_1 + (v_1 + v_3)\\
F_{23} &= v_1w_2 + v_2w_1 + u_1v_2 + u_2v_1 + v_2w_2 + u_1v_1 + w_1 + v_2 + (v_1 + v_2)
\end{aligned}
$$

This step is performed for all pairs of output bits, which are $(1, 2)$, $(1, 3)$ and $(2, 3)$.

5.3.5 Combine all bits and check for balancedness

In the last step we check for all combinations of remaining correction terms, if the resulting function is balanced. For this evaluation we only take correction terms into account, which lasted after Section 5.3.4. We check the balancedness property for the following combinations of correction terms in the three output bits. We test all remaining correction terms of the first output bit with all remaining correction terms of the second output bit with all remaining correction terms of the third output bit. Since the remaining correction terms of all three output bits are combined with each other, the run-time of this step is cubic in the number of remaining terms after Section 5.3.4 from the first, second and third output bit. To check if any of the above described combinations results in a balanced function, we have to evaluate the resulting function $\bar{F}$ for all 2^9 combinations of the input variables and check if every output value occurs equally likely. In the case of an S-box equally likely means exactly once.

5.3.6 Results

In this section we discuss the limits of the above described approach and present our results.
Enumeration over all possible choices of the 22 correction terms, listed in 5.2, for each output bit leads to a search space size of 2^{66}. Hence we limit our search in the following way. We only use correction terms with algebraic degree at most 2. This limits the number of correction terms for each bit to 18 and would lead to an overall search space size of 2^{54}. Since this is still not practical, we further limit the number of correction terms, which are added to the functions F_{ij}, to 5.
We now list the results for Sections 5.3.2 –5.3.5.

In Section 5.3.2, where we search for all possible sums of correction terms, which meet the above described limitations, we find about 800 possible correction terms for each function F_{ij}. In Section 5.3.3 we find between 1000 and 9000 potential correction terms for each bit. At the end of Section 5.3.4 there remain between 0 and 500 potential correction for each of the three output bits. At the end of Section 5.3.5 never any corrections terms remained. Increasing the maximum algebraic degree of the correction terms or the maximum number of correction terms for each function F_{ij} did not lead to better results due to the running time of the procedure.

5.4 A randomized approach

In this section we randomize the approach from Chapter 5.3. We randomize the presented search method, by choosing random sums of correction terms in the first steps of the method. We describe the details of this approach in Section 5.4.1. In Section 5.4.2 we present the obtained results.

5.4.1 Description

As already noted above, we alter the deterministic search method from Section 5.3 by randomizing this search. The idea of this modification is to randomly choose sums of correction terms, instead of picking them in a deterministic way. Of course we only choose sums of correction terms, which lead to a balanced function $F_{ij} : \mathbb{F}_2^9 \mapsto \mathbb{F}_2$.
We therefore alter the selection procedure of correction terms from Section 5.3.2. To achieve comparable results and maximize the success rate of the method, we fix the number of sums of correction terms which get chosen in this step. The progressive sections 5.3.3 – 5.3.5 remain unaltered.

5.4.2 Results

Tables 5.3 and 5.4 show the average numbers of remaining terms after Sections 5.4.1, 5.3.3 and 5.3.4 which were obtained out of a control sample with size 25. The numbers of remaining functions after Section 5.3.5 is omitted. The reason is that there never remained any combinations of sums of correction terms which led to a correct, balanced and non-complete realization. We therefore start in section 5.4.1 with 512 balanced sums of correction terms for each of the nine functions F_{ij}. The values which are listed for Section 5.4.1 refer to the average numbers of random choices of correction terms which were needed to obtain 512 balanced sums of correction terms. It can be observed, that the randomized search behaves better, when the algebraic degree is lower. It also behaves better when the maximum number of correction terms, which are added to the functions F_{ij}, is lower.

maximum Hamming weight:		4	6	8	10	12
Section 5.4.1	bit 1	2111	788	579	564	561
	bit 2	2592	864	592	553	554
	bit 3	2977	899	594	558	541
Section 5.3.3	bit 1	1901	591	489	510	461
	bit 2	2294	657	450	498	481
	bit 3	2419	695	462	466	463
Section 5.3.4	bit 1	9	0	0	3	0
	bit 2	44	6	3	5	4
	bit 3	15	0	0	2	0

Table 5.3: The table shows the average number of remaining terms after each Section 5.4.1, 5.3.3 and 5.3.4 for correction terms with algebraic degree at most two. Maximum Hamming weight refers to the maximum number of correction terms which are added to each function $\mathbb{F}_{ij}$.

maximum Hamming weight:		4	6	8	10	12	14
Section 5.4.1	bit 1	5293	1548	917	820	805	797
	bit 2	6413	1685	921	823	817	805
	bit 3	7187	1726	959	833	813	814
Section 5.3.3	bit 1	872	186	106	101	89	83
	bit 2	1116	184	111	96	84	84
	bit 3	989	205	104	80	83	92
Section 5.3.4	bit 1	0	0	0	0	0	0
	bit 2	10	1	0	0	0	0
	bit 3	1	0	0	0	0	0

Table 5.4: The table shows the average number of remaining terms after each Section 5.4.1, 5.3.3 and 5.3.4 for correction terms with algebraic degree at most three. Maximum Hamming weight refers to the maximum number of correction terms which are added to each function $\mathbb{F}_{ij}$.

5.5 Fix two bits and construct the third bit

In this section we describe another approach on how to find a balanced sharing for a given S-box.

5.5.1 Overview

We start by presenting the general idea of this approach. If we choose functions for certain output shares, we might be able to construct the remaining output shares, such that the resulting function fulfills the correctness 4.3.1, balancedness 4.3.3 and non-completeness 4.3.2 property. More precisely we fix all share functions for certain output bits. Clearly, this choice of fixed functions has to fulfill the correctness, balancedness and non-completeness properties. We then try to construct the share functions of the remaining output bits. We do this by using an indefinite polynomial and the use of the constraints for the desired functions. The constraints in this case are the correctness, non-completeness and balancedness property. More precisely we use a search for the remaining output shares, since we can only provide correctness and non-completeness, when constructing the share functions in ANF.

We describe this approach below for 3×3 S-boxes which we share with three shares. We further fix 6 share functions which belong to two output bits and try to find the remaining three share functions.

5.5.2 Constructing the third bit

As already mentioned, we start with a balanced function from $\mathbb{F}_2^9 \mapsto \mathbb{F}_2^6$. These 6 functions are the functions which define the correct output of two shared bits. To find the functions for the third bit, we use the approach of an indefinite polynomial over $\mathbb{F}_2[x_1, \ldots, x_9]$.
We start to analyze this approach by listing the possible terms for one share function. We get 1 constant, 6 linear, 15 quadratic, 16 cubic and 19 terms of higher order which sums up to 56 indefinites for one share. This gives a total of 168 indefinites for all three shares of one bit.
To limit the number of indefinites we make use of the following two facts:

1. The algebraic normal form is unique. (see Section 2.1)
2. The function we search for must fulfill the correctness property.

This fixes all the indefinites for the terms which have different indices (fixed terms). What remains are the free terms:

- If a term is part of the original function, that means that the term must occur because of the correctness property, then we need one indefinite for each share where the term could fit.
- If the term is not part of the original function then, since the term must occur twice or not at all, we only need one indefinite for both occurrences.

We apply this method to the first bit of class $\mathcal{S}_3^3$. This means we start with two balanced output bits $(F_2, F_3) : \mathbb{F}_2^9 \mapsto \mathbb{F}_2^6$ and try to find a function F_1 that fulfills our required properties 4.3.3, 4.3.1 and 4.3.2.
We list in Table 5.5 the free terms which must be part of the final function.

$F_{11} =$	$u_3v_2 + u_2v_3 +$		u_2v_2	u_3v_3		u_2	u_3		w_2	w_3
$F_{12} =$	$u_3v_1 + u_1v_3 +$	u_1v_1		u_3v_3	u_1		u_3	w_1		w_3
$F_{13} =$	$u_2v_1 + u_1v_2 +$	u_1v_1	u_2v_2		u_1	u_2		w_1	w_2	

Table 5.5: List of mandatory free terms for the first bit in class $\mathcal{S}_3^3$

The following terms, listed in Table 5.6 are the terms which have to occur twice or not at all.

By summing up over all indefinites, we get 21 indefinites for the first bit.

$F_{11}+=$		v_2	v_3		u_2w_2	u_3w_3
$F_{12}+=$	v_1		v_3	u_1w_1		u_3w_3
$F_{13}+=$	v_1	v_2	v_3	u_1w_1	u_2w_2	
		v_2w_2	v_3w_3		$v_2u_2w_2$	$v_3u_3w_3$
	v_1w_1		v_3w_3	$v_1u_1w_1$		$v_3u_3w_3$
	v_1w_1	v_2w_2		$v_1u_1w_1$	$v_2u_2w_2$	

Table 5.6: List of correction terms for the first bit in class $\mathcal{S}_3^3$

5.5.3 Results

We tried several sharings in $\mathcal{A}_{S_3}$ for the fixed functions, as well as correction terms on this 6 fixed functions. None of these calculations for the third bit led to a balanced S-box for class $\mathcal{S}_3^3$.
The reason, why we did not get any results with this approach might be, that it is not possible to adjust the balancedness of the S-box with only three functions.

5.6 Extending 3×3 S-boxes to 4×4 S-boxes with algebraic degree ≤ 2.

In this section we shortly discuss the relation between some classes of 3×3 S-boxes with algebraic degree ≤ 2 and 4×4 S-boxes. Namely we can expand the classes $\mathcal{S}_1^3$, $\mathcal{S}_2^3$ and $\mathcal{S}_3^3$. Let therefore $\mathcal{S}$ be a representative of one of the classes $\mathcal{S}_1^3$, $\mathcal{S}_2^3$ or $\mathcal{S}_3^3$. Then we can transform the S-box $\mathcal{S}(u, v, w) = (y_1, y_2, y_3)$ into an 4×4 bit S-box $\tilde{\mathcal{S}}$ by setting $\tilde{\mathcal{S}}(u, v, w, x) = (y_1, y_2, y_3, x)$.
Consequently we can share the corresponding 4×4 if we can share the classes $\mathcal{S}_1^3$, $\mathcal{S}_2^3$ and $\mathcal{S}_3^3$. Recall that we were only able so far to share the classes $\mathcal{S}_1^3$ and $\mathcal{S}_2^3$. The class $\mathcal{S}_3^3$ and its corresponding 4×4 class remain unshared.

5.7 Summary

In this chapter we have analyzed realizations of affine equivalent S-boxes. We started with a characterization of the properties balancedness, correctness and non-completeness in the truth table of a realization. We analyzed the method of direct sharing for the equivalence classes under affine equivalence. We presented an approach whose idea it is to add terms to the function without violating the correctness property. We analyzed this method for the deterministic version and presented run-time results for the randomized

version. We can summarize that it is easy to fulfill only two of the three required properties, like correctness and non-completeness or correctness and balancedness. However it remains open if there exist realizations which fulfill all three properties for certain types of S-boxes like the class $\mathcal{S}_3^3$.

Chapter 6

Walsh transformation of affine equivalent S-boxes

In this chapter we will use the Walsh-transformation to analyze S-boxes. We will therefore point out how the the properties balancedness 4.3.3, non-completeness 4.3.2 and correctness 4.3.1 can be found in the Walsh spectrum. We will further analyze representatives of affine equivalent S-boxes by means of the Walsh transformation.

We start in Section 6.1 with the representation of the above named properties in the Walsh transformation of a vectorial Boolean function. In Section 6.2 we will list the Walsh transformations for the four representatives of 3×3 S-boxes presented in Chapter 5.2. In Section 6.3 we study the Walsh transformation of balanced and unbalanced sharings of a representative of class $\mathcal{S}_2^3$.

6.1 Finding properties in the Walsh transformation

In the following we draw a connection between the properties which are needed to obtain a secure sharing for a given S-box and the Walsh transformation. We start with the property of balancedness below.

6.1.1 The balancedness property

We recall from Chapter 2 that a Boolean function f is balanced if and only if $\hat{f}_\chi(0) = 0$. We further recall that a vectorial Boolean function is balanced if every component function is balanced. Hence we get that an (n, m)-function

F is balanced if for every $\mathbf{v} \neq 0$, $\mathbf{v} \in \mathbb{F}_2^m$

$$\widehat{\mathbf{v} \cdot F_\chi}(0) = 0.$$

Hence we deduce that a vectorial Boolean function F is balanced if the first row of the matrix

$$\left(\widehat{\mathbf{v} \cdot F_\chi}(\mathbf{u})\right)_{\substack{\mathbf{u}\in\mathbb{F}_2^n \\ \mathbf{v}\in\mathbb{F}_2^{m*}}}$$

is zero.

6.1.2 The non-completeness property

Let F be a vectorial Boolean function mapping from $\mathbb{F}_2^n \mapsto \mathbb{F}_2^m$. Assume further that the first coordinate function $F_1 : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ is independent of the first input, i.e. $F_1(0, \mathbf{x}) = F_1(1, \mathbf{x})$ for all $\mathbf{x} \in \mathbb{F}_2^{n-1}$. Then $\hat{F}_{1\chi}(1, \mathbf{u}) = 0$ for all $\mathbf{u} \in \mathbb{F}_2^{n-1}$. To see this we go into the definition of the Walsh-transformation.

$$\begin{aligned}
\widehat{F_{1\chi}}(1, \mathbf{u}) &= \sum_{\mathbf{x}\in\mathbb{F}_2^n} (-1)^{F_1(\mathbf{x})\oplus\mathbf{x}\cdot(1,\mathbf{u})} \\
&= \sum_{\mathbf{x}\in\mathbb{F}_2^{n-1}} \left((-1)^{F_1(0,\mathbf{x})\oplus(0,\mathbf{x})\cdot(1,\mathbf{u})} + (-1)^{F_1(1,\mathbf{x})\oplus(1,\mathbf{x})\cdot(1,\mathbf{u})}\right) \\
&= \sum_{\mathbf{x}\in\mathbb{F}_2^{n-1}} (-1)^{F_1(0,\mathbf{x})} \left((-1)^{(0,\mathbf{x})\cdot(1,\mathbf{u})} + (-1)^{(1,\mathbf{x})\cdot(1,\mathbf{u})}\right) \\
&= \sum_{\mathbf{x}\in\mathbb{F}_2^{n-1}} (-1)^{F_1(0,\mathbf{x})} \left((-1)^{\mathbf{x}\cdot\mathbf{u}} - (-1)^{\mathbf{x}\cdot\mathbf{u}}\right) = 0
\end{aligned}$$

Consider now a shared coordinate function $F_{i,j}$ of some vectorial Boolean function. Then, due to the non-completeness property $F_{i,j}$ is independent of all shares with index j. Hence we get that $\widehat{F_{i,j}}_\chi(\mathbf{u}) = 0$ for all vectors $\mathbf{u}$ which have at least one coordinate $u_{n+i} = 1$ for some $n \in \mathbb{N}_0$.
Consequently we get that $\widehat{\mathbf{v} \cdot F_\chi}(\mathbf{u}) = 0$ for all vectors $\mathbf{v}$ that combine only functions with the same share index j and all vectors $\mathbf{u}$ as described above.

6.1.3 The correctness property

Finally we want to take a look at the property of correctness. Let F be an (n, m) function which we share with s shares with the realization $\bar{F}$. When we look at the Walsh transformation of the function $\bar{F}$ we only consider vectors $\mathbf{v} \in \mathcal{V}_{m\cdot s}$ and $\mathbf{u} \in \mathcal{V}_{n\cdot s}$ with

$$\mathcal{V}_{k\cdot s} = \left\{\mathbf{x} \in \mathbb{F}_2^{k\cdot s} | x_{n+j} = 1 \text{ for } j \in \{1, \ldots, s\}, n \in \mathcal{I}, \text{ for } \mathcal{I} \in \mathcal{P}(\{1, \ldots, k\})\right\}.$$

The set $\mathcal{V}_{m\cdot s}$ is the set of vectors, which sum up all share functions of subsets of the m output bits. The set $\mathcal{V}_{n\cdot s}$ is the set of vectors, which sum up all shares of subsets of the n inputs. The correctness 4.3.1 property requires that the sum of all share functions $F_{i,j}(\bar{x}_1, \ldots, \bar{x}_n)$ is equal to $F_i(\oplus_{j=1}^{s} x_{1,j}, \ldots, \oplus_{j=1}^{s} x_{n,j})$ for all $(\bar{x}_1, \ldots, \bar{x}_n)$ and for all coordinate functions $i \in \{1, \ldots, m\}$. Hence, we only consider vectors $\mathbf{v} \in \mathcal{V}_{m\cdot s}$ and $\mathbf{u} \in \mathcal{V}_{n\cdot s}$. Thus we get in the Walsh transformation the following equation:

$$\widehat{\mathbf{v} \cdot \bar{F}_\chi}(\mathbf{u}) = 2^{n\cdot(s-1)} \cdot \widehat{\tilde{\mathbf{v}} \cdot F_\chi}(\tilde{\mathbf{u}})$$

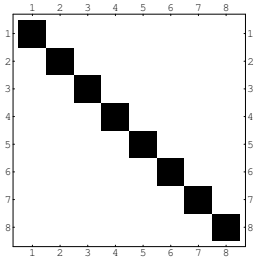
with $\tilde{\mathbf{v}} = (v_m, v_{2m}, \ldots, v_{m\cdot s})$ and $\tilde{\mathbf{u}} = (u_n, u_{2n}, \ldots, u_{n\cdot s})$.

6.2 Walsh transformation for the unshared S-boxes

We now list the Walsh transformations for representatives of the four classes of 3×3 S-boxes. For each of the four classes we give the Walsh transformation on the left hand side and in a graphical form on the right hand side. Black squares indicate thereby nonzero values in the corresponding matrix.

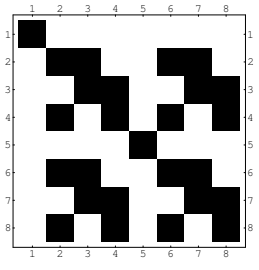
Class $\mathcal{S}_0^3$:

$$\begin{pmatrix} 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 \end{pmatrix}$$



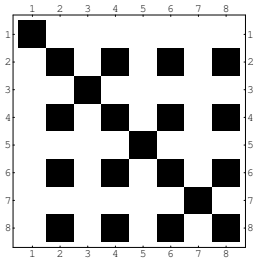
Class $\mathcal{S}_1^3$:

$$\begin{pmatrix} 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 4 & 0 & 0 & 4 & -4 & 0 \\ 0 & 0 & 4 & 4 & 0 & 0 & 4 & -4 \\ 0 & 4 & 0 & 4 & 0 & -4 & 0 & 4 \\ 0 & 0 & 0 & 0 & 8 & 0 & 0 & 0 \\ 0 & 4 & -4 & 0 & 0 & 4 & 4 & 0 \\ 0 & 0 & 4 & -4 & 0 & 0 & 4 & 4 \\ 0 & -4 & 0 & 4 & 0 & 4 & 0 & 4 \end{pmatrix}$$



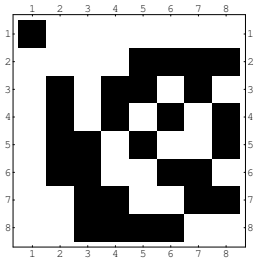
Class $\mathcal{S}_2^3$:

$$\begin{pmatrix} 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 4 & 0 & 4 & 0 & -4 \\ 0 & 0 & 8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 4 & 0 & -4 & 0 & 4 \\ 0 & 0 & 0 & 0 & 8 & 0 & 0 & 0 \\ 0 & 4 & 0 & -4 & 0 & 4 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 \\ 0 & -4 & 0 & 4 & 0 & 4 & 0 & 4 \end{pmatrix}$$



Class $\mathcal{S}_3^3$:

$$\begin{pmatrix} 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 4 & -4 & 4 \\ 0 & -4 & 0 & 4 & 4 & 0 & 4 & 0 \\ 0 & 4 & 0 & 4 & 0 & -4 & 0 & 4 \\ 0 & 4 & 4 & 0 & 4 & 0 & 0 & -4 \\ 0 & 4 & -4 & 0 & 0 & 4 & 4 & 0 \\ 0 & 0 & 4 & -4 & 0 & 0 & 4 & 4 \\ 0 & 0 & 4 & 4 & -4 & 4 & 0 & 0 \end{pmatrix}$$



6.3 Walsh transformation of Class $\mathcal{S}_2^3$

In this section we investigate on multiple sharing variants of the representative S_2 of class $\mathcal{S}_2^3$. We further take a look at the Walsh transformation of balanced and unbalanced S-box candidates in class $\mathcal{S}_2^3$.

6.3.1 Observations on S-box candidates in class $\mathcal{S}_2^3$

We start with a look at the Walsh transformation of the 64 S-box candidates S_2^i of the set $\mathcal{A}_{S_2}$. As described in Chapter 2 this results in matrices ($M_i \in \mathbb{Z}^{512\times512}$) for $i \in \{1,\ldots,64\}$. We thereby observe, that the rank of the matrix M_i equals the number of different values we get from the S-box candidate S_2^i. We describe this more formally below. Let therefore S_2^i be one of the 64 S-boxes from the set $\mathcal{A}_{S_2}$ and M_i the matrix

$$M_i = \left(\widehat{\mathbf{v}\cdot S_2^i(\mathbf{u})}\right)_{\substack{\mathbf{v}\in\mathbb{F}_2^9 \\ \mathbf{u}\in\mathbb{F}_2^9}}$$

then it can be observed that

$$\operatorname{rank}(M_i) = \left|\{x \in \mathbb{F}_2^9 | \exists u \in \mathbb{F}_2^9 : S_2^i(u) = x\}\right|.$$

This means that for a balanced shared S-box we obtain that $\operatorname{rank}(M) = 512$.

We now want to take a closer look at one of the non-balanced shared S-boxes from the set $\mathcal{A}_{S_2}$ in class $\mathcal{S}_2^3$. We state one of these correct and non-complete and non-balanced shares of the S-Box S_2 below.

$$\begin{aligned}
F_{11}(\bar{u},\bar{v},\bar{w}) &= v_2w_3 + u_2w_3 + v_3w_2 + u_3w_2 + v_2w_2 + u_2w_2 + u_3 \\
F_{12}(\bar{u},\bar{v},\bar{w}) &= v_1w_3 + u_1w_3 + v_3w_1 + u_3w_1 + v_3w_3 + u_3w_3 + u_1 \\
F_{13}(\bar{u},\bar{v},\bar{w}) &= v_1w_2 + u_1w_2 + u_2w_1 + v_2w_1 + v_1w_1 + u_1w_1 + u_2 \\
F_{21}(\bar{u},\bar{v},\bar{w}) &= u_2w_3 + u_3w_2 + u_2w_2 + v_2 \\
F_{22}(\bar{u},\bar{v},\bar{w}) &= u_1w_3 + u_3w_1 + u_3w_3 + v_3 \\
F_{23}(\bar{u},\bar{v},\bar{w}) &= u_1w_2 + u_2w_1 + u_1w_1 + v_1
\end{aligned} \tag{6.1}$$

We now take a closer look on the first row of the Walsh transformation matrix M of this vectorial Boolean function (6.1). This is the row where $\mathbf{u} = 0$. We investigate this row for the following function (with functions F_{31}, F_{32}, F_{33} as in Section 5.2.4).

Note, that the only difference between function (6.1) and the balanced function listed in Section 5.2.4 is the different assignment of the variables v_i in the second bit. By investigating the first row for unbalanced component functions we are of course interested in the smallest choice of coordinate functions. Since every further choice leads to an unbalanced component function as well. The minimum choices are: (F_{11}, F_{21}, F_{22}), (F_{12}, F_{22}, F_{23}) and (F_{13}, F_{21}, F_{23}).

6.4 Summary

In this chapter we have characterized the properties balancedness 4.3.3, non-completeness 4.3.2 and correctness 4.3.1 by means of the Walsh transformation. We investigated on the Walsh transformation of a non-balanced sharing of an S-box in class $\mathcal{S}_2^3$. We further gave the Walsh transformations of representatives of the classes $\mathcal{S}_0^3$, $\mathcal{S}_1^3$, $\mathcal{S}_2^3$ and $\mathcal{S}_3^3$.

Chapter 7

Conclusions

In this thesis we focused on countermeasures against side channel attacks on cryptographic devices. In particular we focused on a sharing scheme which is provable secure against first order DPA attacks.

We concentrated on the characterization of the properties balancedness, correctness and non-completeness in different representations of vectorial Boolean functions. We illustrated that the properties balancedness and non-completeness can be verified easy by using the truth table. On the other hand we pointed out that the ANF is beneficial in verifying the properties correctness and non-completeness. We further characterized the required properties by means of the Walsh transformation and demonstrated that combinations of two properties can be achieved by construction in both representations. We also presented multiple methods to construct realizations that fulfill all three properties.

Secure sharing of n bit S-boxes for large n is a hard problem especially when the algebraic degree of the function rises. We have seen that it was not even possible to construct a secure realization for an S-box in class $\mathcal{S}_3^3$ with algebraic degree 2. If we relinquish one of the properties, construction of such realizations becomes much easier but are insecure instead. We leave the existence of a secure realization for class $\mathcal{S}_3^3$ and the related 4×4 bit S-box class as open problem for future work.

We see several possibilities to extend our work. Firstly, it might be reasonable to explore properties of correction terms, in order to deduce more sophisticated methods when constructing balanced, correct and non-complete realizations.

Secondly, it might be possible to restudy the desired properties balancedness, correctness and non-completeness by exploring connections to different research fields like Coding theory.

Thirdly, one might investigate on the open question if it suffices to focus

on Boolean functions for realizations with algebraic degree d, if the function for which the realization is constructed has algebraic degree d.

Bibliography

[1] Java™cryptography extension (IAIK-JCE). `http://http://jce.iaik.tugraz.at`.

[2] Agrawal, D., Rao, J. R., and Rohatgi, P. Multi-channel attacks. In *CHES* (2003), C. D. Walter, Çetin Kaya Koç, and C. Paar, Eds., vol. 2779 of *Lecture Notes in Computer Science*, Springer, pp. 2–16.

[3] Akkar, M.-L., Bevan, R., and Goubin, L. Two power analysis attacks against one-mask methods. In *FSE* (2004), B. K. Roy and W. Meier, Eds., vol. 3017 of *Lecture Notes in Computer Science*, Springer, pp. 332–347.

[4] Akkar, M.-L., and Giraud, C. An implementation of des and aes, secure against some attacks. In Çetin Kaya Koç et al. [15], pp. 309–318.

[5] Anderson, R. J. *Security engineering - a guide to building dependable distributed systems (2. ed.)*. Wiley, 2008.

[6] Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., and Whelan, C. The sorcerers apprentice guide to fault attacks. *IACR Cryptology ePrint Archive 2004* (2004), 100.

[7] Ben-Or, M., Goldwasser, S., and Wigderson, A. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC* (1988), J. Simon, Ed., ACM, pp. 1–10.

[8] Berrut, J.-P., and Trefethen, L. N. Barycentric lagrange interpolation. *SIAM Review 46*, 3 (2004), 501–517.

[9] Blahut, R. E. *Algebraic Codes for Data Transmission*, 1 ed. Cambridge University Press, July 2002.

[10] Blömer, J., Guajardo, J., and Krummel, V. Provably secure masking of aes. In *Selected Areas in Cryptography* (2004), H. Handschuh

and M. A. Hasan, Eds., vol. 3357 of *Lecture Notes in Computer Science*, Springer, pp. 69–83.

[11] Blundo, C., Santis, A. D., and Vaccaro, U. Efficient sharing of many secrets. In *STACS* (1993), P. Enjalbert, A. Finkel, and K. W. Wagner, Eds., vol. 665 of *Lecture Notes in Computer Science*, Springer, pp. 692–703.

[12] Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J. B., Seurin, Y., and Vikkelsoe, C. Present: An ultra-lightweight block cipher. In *CHES* (2007), P. Paillier and I. Verbauwhede, Eds., vol. 4727 of *Lecture Notes in Computer Science*, Springer, pp. 450 – 466.

[13] Carlet, C. *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*. Cambridge University Press, 2010, ch. Boolean Functions for Cryptography and Error Correcting Codes, pp. 257 – 397.

[14] Carlet, C. *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*. Cambridge University Press, 2010, ch. Vectorial Boolean Functions for Cryptography, pp. 398 – 469.

[15] Çetin Kaya Koç, Naccache, D., and Paar, C., Eds. *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings* (2001), vol. 2162 of *Lecture Notes in Computer Science*, Springer.

[16] Chakrabarty, K., and Hayes, J. Balanced boolean functions. *Computers and Digital Techniques, IEE Proceedings - 145*, 1 (1998), 52 – 62.

[17] Chari, S., Jutla, C. S., Rao, J. R., and Rohatgi, P. Towards sound approaches to counteract power-analysis attacks. In Wiener [81], pp. 398–412.

[18] Crama, Y., and Hammer, P. *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2010.

[19] Cusick, T., and Stănică, P. *Cryptographic Boolean functions and applications*. Academic Press/Elsevier, 2009.

[20] Daemen, J., Peeters, M., and Assche, G. V. Bitslice ciphers and power analysis attacks. In Schneier [58], pp. 134–149.

[21] DAEMEN, J., PEETERS, M., ASSCHE, G. V., AND RIJMEN, V. Nessie proposal: NOEKEON. `http://www.cryptnessie.org`, 2000.

[22] DAEMEN, J., AND RIJMEN, V. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[23] GANDOLFI, K., MOURTEL, C., AND OLIVIER, F. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç et al. [15], pp. 251–261.

[24] GIERLICHS, B., BATINA, L., PRENEEL, B., AND VERBAUWHEDE, I. Revisiting higher-order dpa attacks:. In *CT-RSA* (2010), J. Pieprzyk, Ed., vol. 5985 of *Lecture Notes in Computer Science*, Springer, pp. 221–234.

[25] GIERLICHS, B., BATINA, L., TUYLS, P., AND PRENEEL, B. Mutual information analysis. In Oswald and Rohatgi [49], pp. 426–442.

[26] GOLIC, J. D., AND TYMEN, C. Multiplicative masking and power analysis of aes. In Jr. et al. [27], pp. 198–212.

[27] JR., B. S. K., ÇETIN KAYA KOÇ, AND PAAR, C., Eds. *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers* (2003), vol. 2523 of *Lecture Notes in Computer Science*, Springer.

[28] KAY, S. M. *Fundamentals of statistical signal processing: estimation theory*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[29] KNUDSEN, L. R. Truncated and higher order differentials. In *FSE* (1994), B. Preneel, Ed., vol. 1008 of *Lecture Notes in Computer Science*, Springer, pp. 196–211.

[30] KOCHER, P. C. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO* (1996), N. Koblitz, Ed., vol. 1109 of *Lecture Notes in Computer Science*, Springer, pp. 104–113.

[31] KOCHER, P. C., JAFFE, J., AND JUN, B. Differential power analysis. In Wiener [81], pp. 388 – 397.

[32] LAI, X. *Higher order derivatives and differential cryptanalysis*. Kluwer Academic Publishers, 1994, p. 227.

[33] LIDL, R., AND NIEDERREITER, H. *Finite Fields*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2008.

[34] Lorens, C. S. Invertible boolean functions. *IEEE Transactions on Electronic Computers*, 13 (1964).

[35] MacWilliams, F. J., and Sloane, N. J. A. *The Theory of Error-Correcting Codes (North-Holland Mathematical Library)*. North Holland, June 1988.

[36] Mangard, S., Oswald, E., and Popp, T. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

[37] Mangard, S., Popp, T., and Gammel, B. M. Side-channel leakage of masked cmos gates. In *CT-RSA* (2005), A. Menezes, Ed., vol. 3376 of *Lecture Notes in Computer Science*, Springer, pp. 351–365.

[38] Marichal, J.-L. The influence of variables on pseudo-boolean functions with applications to game theory and multicriteria decision making. *Discrete Applied Mathematics 107*, 1-3 (2000), 139–164.

[39] Matsui, M. Linear cryptoanalysis method for des cipher. In *EUROCRYPT* (1993), pp. 386–397.

[40] Meier, W., and Staffelbach, O. Nonlinearity criteria for cryptographic functions. In *EUROCRYPT* (1989), pp. 549–562.

[41] Menezes, A., van Oorschot, P. C., and Vanstone, S. A. *Handbook of Applied Cryptography*. CRC Press, 1996.

[42] Messerges, T. S. Securing the aes finalists against power analysis attacks. In Schneier [58], pp. 150–164.

[43] Messerges, T. S. Using second-order power analysis to attack dpa resistant software. In *CHES* (2000), Çetin Kaya Koç and C. Paar, Eds., vol. 1965 of *Lecture Notes in Computer Science*, Springer, pp. 238–251.

[44] Nikova, S., Rijmen, V., and Schläffer, M. Secure hardware implementation of non-linear functions in the presence of glitches. In *ICISC* (2008), P. J. Lee and J. H. Cheon, Eds., vol. 5461 of *Lecture Notes in Computer Science*, Springer, pp. 218 – 234.

[45] Nikova, S., Rijmen, V., and Schläffer, M. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptology 24*, 2 (2011), 292–321.

[46] Nyberg, K. Perfect nonlinear s-boxes. In *EUROCRYPT* (1991), pp. 378 – 386.

[47] Örs, S. B., Gürkaynak, F. K., Oswald, E., and Preneel, B. Power-analysis attack on an asic aes implementation. In *ITCC (2)* (2004), IEEE Computer Society, pp. 546–552.

[48] Oswald, E., Mangard, S., Herbst, C., and Tillich, S. Practical second-order dpa attacks for masked smart card implementations of block ciphers. In *CT-RSA* (2006), D. Pointcheval, Ed., vol. 3860 of *Lecture Notes in Computer Science*, Springer, pp. 192–207.

[49] Oswald, E., and Rohatgi, P., Eds. *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings* (2008), vol. 5154 of *Lecture Notes in Computer Science*, Springer.

[50] Popp, T., and Mangard, S. Masked dual-rail pre-charge logic: Dpa-resistance without routing constraints. In *CHES* (2005), J. R. Rao and B. Sunar, Eds., vol. 3659 of *Lecture Notes in Computer Science*, Springer, pp. 172–186.

[51] Poschmann, A., Moradi, A., Khoo, K., Lim, C.-W., Wang, H., and Ling, S. Side-channel resistant crypto for less than 2,300 ge. *J. Cryptol. 24* (April 2011), 322 – 345.

[52] Pramstaller, N., Oswald, M. E., Mangard, S., Gürkaynak, F. K., and Häne, S. A masked aes asic implementation. In *Proceedings of Austrochip 2004* (2004), M. L. Erwin Ofner, Ed., pp. 77 – 81.

[53] Preneel, B., Leekwijck, W. V., Linden, L. V., Govaerts, R., and Vandewalle, J. Propagation characteristics of boolean functions. In *EUROCRYPT* (1990), pp. 161–173.

[54] Quisquater, J.-J., and Samyde, D. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *E-smart* (2001), I. Attali and T. P. Jensen, Eds., vol. 2140 of *Lecture Notes in Computer Science*, Springer, pp. 200–210.

[55] Rabaey, J. M., Chandrakasan, A., and Nikolic, B. *Digital integrated circuits: A design perspective*, 2ed ed. Prentice Hall, 2004.

[56] Samyde, D., Skorobogatov, S. P., Anderson, R. J., and Quisquater, J.-J. On a new way to read data from memory. In *IEEE Security in Storage Workshop* (2002), pp. 65–69.

[57] Schneier, B. *Applied cryptography - protocols, algorithms, and source code in C (2. ed.)*. Wiley, 1996.

[58] Schneier, B., Ed. *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings* (2001), vol. 1978 of *Lecture Notes in Computer Science*, Springer.

[59] Shamir, A. How to share a secret. *Commun. ACM 22*, 11 (1979), 612–613.

[60] Shannon, C. E. Communication Theory of Secrecy Systems. *Bell System Technical Journal 28*, 4 (1949), 656 – 715.

[61] Siegenthaler, T. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Transactions on Information Theory 30*, 5 (1984), 776–780.

[62] Skorobogatov, S. P. Semi-invasive attacks – A new approach to hardware security analysis. Tech. Rep. UCAM-CL-TR-630, University of Cambridge, Computer Laboratory, Apr. 2005.

[63] Standaert, F.-X., Örs, S. B., Quisquater, J.-J., and Preneel, B. Power analysis attacks against fpga implementations of the des. In *FPL* (2004), J. Becker, M. Platzner, and S. Vernalde, Eds., vol. 3203 of *Lecture Notes in Computer Science*, Springer, pp. 84–94.

[64] Standaert, F.-X., Veyrat-Charvillon, N., Oswald, E., Gierlichs, B., Medwed, M., Kasper, M., and Mangard, S. The world is not enough: Another look on second-order dpa. In *ASIACRYPT* (2010), M. Abe, Ed., vol. 6477 of *Lecture Notes in Computer Science*, Springer, pp. 112–129.

[65] Stephens, M. A. Edf statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association 69*, 347 (1974), 730–737.

[66] Stinson, D. R. *Cryptography - theory and practice*. Discrete mathematics and its applications series. CRC Press, 1995.

[67] Sundström, T., and Alvandpour, A. A comparative analysis of logic styles for secure ic's against dpa attacks. *2005 Norchip* (2005), 297–300.

[68] Suzuki, D., and Saeki, M. Security evaluation of dpa countermeasures using dual-rail pre-charge logic style. In *CHES* (2006), L. Goubin and M. Matsui, Eds., vol. 4249 of *Lecture Notes in Computer Science*, Springer, pp. 255–269.

[69] Tillich, S., and Herbst, C. Attacking state-of-the-art software countermeasures-a case study for aes. In Oswald and Rohatgi [49], pp. 228–243.

[70] Tiri, K., and Verbauwhede, I. Place and route for secure standard cell design. In *CARDIS* (2004), J.-J. Quisquater, P. Paradinas, Y. Deswarte, and A. A. E. Kalam, Eds., Kluwer, pp. 143–158.

[71] Trichina, E., Korkishko, T., and Lee, K.-H. Small size, low power, side channel-immune aes coprocessor: Design and synthesis results. In *AES Conference* (2004), H. Dobbertin, V. Rijmen, and A. Sowa, Eds., vol. 3373 of *Lecture Notes in Computer Science*, Springer, pp. 113–127.

[72] Trichina, E., Seta, D. D., and Germani, L. Simplified adaptive multiplicative masking for aes. In Jr. et al. [27], pp. 187–197.

[73] Vollmer, H. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[74] Von Neumann, J., and Morgenstern, O. *Theory of games and economic behavior / by John Von Neumann and Oskar Morgenstern*. Princeton University Press, Princeton :, 1944.

[75] Waddle, J., and Wagner, D. Towards efficient second-order power analysis. In *CHES* (2004), M. Joye and J.-J. Quisquater, Eds., vol. 3156 of *Lecture Notes in Computer Science*, Springer, pp. 1–15.

[76] Wakerly, J. F. *Digital design - principles and practices*. Prentice Hall Series in computer engineering. Prentice Hall, 1990.

[77] Wanlass, F., and Sah, C. *Nanowatt logic using field-effect metal-oxide semiconductor triodes*. Institute of Electrical and Electronics Engineers, 1963, pp. 32–33.

[78] Webster, A. F., and Tavares, S. E. On the design of s-boxes. In *CRYPTO* (1985), H. C. Williams, Ed., vol. 218 of *Lecture Notes in Computer Science*, Springer, pp. 523–534.

[79] Wegener, I. *The complexity of Boolean functions.* Wiley-Teubner, 1987.

[80] Whitnall, C., Oswald, E., and Mather, L. An exploration of the kolmogorov-smirnov test as competitor to mutual information analysis. *IACR Cryptology ePrint Archive 2011* (2011), 380.

[81] Wiener, M. J., Ed. *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings* (1999), vol. 1666 of *Lecture Notes in Computer Science*, Springer.

[82] Wolkerstorfer, J., Oswald, E., and Lamberger, M. An asic implementation of the aes sboxes. In *CT-RSA* (2002), B. Preneel, Ed., vol. 2271 of *Lecture Notes in Computer Science*, Springer, pp. 67–78.

[83] Xiao, G.-Z., and Massey, J. L. A spectral characterization of correlation-immune combining functions. *IEEE Transactions on Information Theory 34*, 3 (1988), 569–571.

[84] Yao, A. C.-C. Protocols for secure computations (extended abstract). In *FOCS* (1982), IEEE Computer Society, pp. 160–164.

[85] Yeap, G. K. *Practical Low Power Digital VLSI Design.* Springer, Aug. 1997.