The Dillon-Wolfe Function for Cryptography

Eric Cornet

August 29, 2012

Master thesis:
Mathematical Sciences

Under supervision of:
Gido Schmitz NBV
Jaap van Oosten UU
Second reader:
Gerard Tel UU

Abstract

Boolean functions, $f : \mathbb{F}_2^n \to \mathbb{F}_2$, have applications in cryptography. To encrypt and decrypt a message in symmetric cryptography, *substitution permutation networks* (SPN) are used. An example of such an SPN is the Advanced Encryption Standard (AES). An S-box is a component of an SPN which is essential for the security of that cipher. Such an S-box consists of vectorial Boolean functions, $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$, which satisfies certain cryptographic criteria at high levels. Two criteria are that an S-box is a permutation and that it is *almost perfect nonlinear* (APN). Recently the first APN permutation in even dimension was found by Dillon and Wolfe [13] and is therefore called a Dillon-Wolfe function. It is the purpose of this master thesis to explain in which context Boolean functions are used as an S-box in cryptography, to study Boolean functions in general and to use this to explain how to find a Dillon-Wolfe function. Further we give a general upper bound on the algorithm that Dillon and Wolfe used, we determine other cryptographic properties of a Dillon-Wolfe function and conclude that a Dillon-Wolfe function can be used as an S-box.

Contents

1 Introduction

An important component of a particular cryptographic algorithm, a substitution permutation network SPN, is called the S-box and consists of a vector of Boolean functions. Boolean functions are functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. They are selected for an S-box if they have certain cryptographic properties. Since there do not exist Boolean functions which achieve the maximum of all properties at once, trade-offs have to be made.

For one, two or three variables, Boolean functions are rather simple functions which can even be investigated by hand. But when the number of variables grows, the number of different Boolean functions grows exponentially, since there are 2^{2^n} different Boolean functions with n variables. Therefore finding functions with certain properties can not be done by a brute force search.

It was long an open question if there exists a vectorial Boolean function which is a permutation and is *almost perfect nonlinear* APN. In this master thesis we investigate a recent result of Dillon and Wolfe [13] of an APN permutation in dimension six. We give the theory which is needed to understand how they have found this Dillon-Wolfe function and how we can determine whether this Dillon-Wolfe function can be used in an SPN as an S-box. We also give a general upper bound on the algorithm which Dillon and Wolfe used to find this APN permutation.

Therefore we start with an introduction in symmetric cryptography and coding theory. Then we study Boolean functions and vectorial Boolean functions in general and with this theory we can deduce a Dillon-Wolfe function.

Notation Since cryptography and in particular Boolean functions is quite a new research field in mathematics, there are not many conventions in notations. I tried to follow the notation used by Carlet in [5] and [6] since this work gives a good overview of the theory developed in the last fifty years. Most notation will become clear in definitions. Carlet uses two different kinds of summation which need to be clarified. We use $+$ and $\sum$ for the usual summation, that is $+, \sum : \mathbb{Z} \to \mathbb{Z}$ (or $\mathbb{R} \to \mathbb{R}$) and $\oplus$ and $\bigoplus$ for summation modulo 2, i.e. $\oplus, \bigoplus : \mathbb{F}_2^n \to \mathbb{F}_2$. With this notation we define the inner product of two binary vectors as the function $\cdot : \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2$ with $x \cdot y = \bigoplus_{i=1}^n x_i y_i$. Since the vector space $\mathbb{F}_2^n$ can be identified with the field $\mathbb{F}_{2^n}$ in which normally the symbol $+$ is used for addition, we will use $+$ for the sum of two vectors in $\mathbb{F}_2^n$ when $n > 1$.

Structure of the thesis The thesis consists of five sections. The first

four sections contain basic results which are needed for the last section. Below, the content and the purpose of each section is given, together with the most important references. For important or recent results, the references are given within the text.

- Section 2. A general idea of symmetric cryptography is given, to give the reader an idea in which context Boolean functions are used. What is a cipher, what is a S-box and what kind of attacks are known? General theories on ciphers are from Buchmann [2]. For the section on cryptanalysis we used papers of Shannon [18], Biham and Shamir [1] and Matsui [17]. The figures are from Carlet [5].

- Section 3. An introduction in (linear) coding theory and error correcting codes. Enough to tread Hamming codes and double simplex codes which are needed for the construction of a Dillon-Wolfe function. The theory is mainly taken from van Lint [20], Hall [11] and Dillon et al. [13].

- Section 4. Almost all relevant basic properties for Boolean functions $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ for $m = 1$ are described. This section contains most of the theory on which I have concentrated in the first months. The theory comes from Carlet's overview on Boolean functions [5]. In the last subsection we give the meaning of properties of Boolean function for cryptographic applications.

- Section 5. Most notions that have been introduced in the previous section can be easily extended to vectorial Boolean functions, $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ for $m \geq 1$. The theory on Boolean functions needed for the construction of a Dillon-Wolfe function is completed. Most results are from Budaghyan [3], Carlet [4], [6] and Chabaud and Vaudenay [7].

- Section 6. Coding theory and the theory of vectorial Boolean functions come together to derive a Dillon Wolfe function. The construction was introduced in the paper of Dillon et al. [13].

  Further, two new results are given. In section 6.2.2 I state a general upper bound for the algorithm which Dillon and Wolfe have used and in section 6.4 the cryptographic properties of a Dillon-Wolfe function are derived and compared to a commonly used S-box, the inverse function. From this we conclude that the Dillon-Wolfe function can be used as an S-box.

2 Cryptography

The main goal of this section is to give an idea of the context in which Boolean functions are used in cryptography. In section 2.1 we start with some general notions of a cryptographic algorithm, up to *substitution permutation networks* SPN and the role of an S-box in an SPN. In section 2.2 we will say more on the possible attacks that can be made to break a cryptographic algorithm and reveal the secret message. This can be used to determine which properties of Boolean functions are desired and which are not.

2.1 Ciphers

Two thousand years ago the Romans already used cryptography when they sent military strategies to the generals who where fighting at the front. The encryption was done by a cyclic shift on the letters of the alphabet. It is known [19] that Julius Caesar always used three cyclic shifts, on the Roman alphabet in his case, to encrypt his confidential messages. Of course there are 26 cyclic shifts possible on our alphabet. For instance the sender could let the number of shifts depend on the first letter of the name of the receiver. If you want to send a secret message to general CORNET, the encryption is done by two cyclic shifts in such a way that the letters A,B,C,D,... become the letters C,D,E,F,... respectively. We get,

CORNET
EQTPGV

and the encrypted name of the general would be EQTPGV. The decryption is obviously done by reversed cyclic shifts. This algorithm is now known as the Caesar cipher.

2.1.1 Cipher, alphabet and XOR

In the book by Buchmann [2] a general definition of a cryptographic algorithm is given. A cryptographic algorithm is also called a cipher. A text which is not (yet) encrypted is called a plaintext and an encrypted plaintext is called a ciphertext.

Definition 2.1. A cipher is a tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ with the following properties.

i. The set $\mathcal{P}$ is the set of all possible plaintexts.

ii. The set $\mathcal{C}$ is the set of all possible ciphertexts.

iii. The set $\mathcal{K}$ is the set of all possible keys.

iv. $\mathcal{E} = \{E_k : k \in \mathcal{K}\}$ is the family of all encryption functions $E_k : \mathcal{P} \to \mathcal{C}$.

v. $\mathcal{D} = \{D_k : k \in \mathcal{K}\}$ is the family of all decryption functions $D_k : \mathcal{C} \to \mathcal{P}$.

vi. For every encryption key $e \in \mathcal{K}$ there exists a decryption key $d \in \mathcal{K}$ such that $D_d(E_e(p)) = p$ for all plaintexts $p \in \mathcal{P}$.

When using a cipher for the communication of a secret message, a minimal requirement is that the decryption key is kept secret. In some ciphers the decryption key can be computed from the encryption key. For these ciphers the encryption key must be kept secret as well and exchanged before the start of the communication. This kind of ciphers are called symmetric. The Caesar cipher is an example of a symmetric cipher. In asymmetric key cryptography the encryption key and the decryption key are distinct and can not be derived from each other, that is they can not be derived from each other in a reasonable amount of time. Therefore the encryption key can be made public. This is also known as public key cryptography. A famous example in asymmetric cryptography is the RSA cipher. We will concentrate in this section on symmetric cryptography.

Every message consists of symbols from an alphabet Σ. If an alphabet Σ is of size m we identify Σ with the additive group $\mathbb{Z}/m\mathbb{Z} = \{0, 1, \ldots, m-1\}$. For example we can take the usual alphabet A,B,C,...,Z with 26 symbols, or the binary alphabet $\mathbb{F}_2$. An often used alphabet is the set of (ANSI) ASCII symbols used in a personal computer. It consists of 128 symbols which represents all keys on the keyboard and more[1]. Each symbol has its unique string of eight bits, seven which determine the symbol and one extra bit to detect a possible transmission error[2]. Of course a message in ASCII symbols can also be viewed as a message in $\mathbb{F}_2$. We will only consider the binary alphabet $\mathbb{F}_2$, with exception of the following example.

[1]For a table of all (ANSI) ASCII symbols see [2]

[2]More about such extra bits will be explained in section 3.1.2 about error correcting codes.

Example 2.2. Let the alphabet be $\Sigma = \mathbb{Z}/26\mathbb{Z}$ corresponding to the usual alphabet A,B,C,...,Z. The Caesar cipher, used in the introduction to send a message to general CORNET, consists of $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathbb{Z}/26\mathbb{Z}$, the encryption function $E_e : \mathbb{Z}/26\mathbb{Z} \to \mathbb{Z}/26\mathbb{Z}$ with $E_e(x) = x + 2 \mod 26$ and the decryption function $D_e(x) = x - 2 \mod 26$. In general the encryption of the Caesar cipher is $x \mapsto x + d \mod 26$, for $1 \geq d \geq 25$, with decryption function $x \mapsto x - d \mod 26$.

2.1.2 Stream ciphers

A common used operation for encryption and decryption is exclusive or, abbreviated as XOR.

Definition 2.3. Define the XOR operation $\oplus$ as bitwise addition modulo 2. Let x, y be two binary vectors of length n, then $x \oplus y = (x_1 \oplus y_1, \ldots, x_n \oplus y_n)$ such that for $1 \leq i \leq n$ we have;

x_i	y_i	$x_i \oplus y_i$
0	0	0
1	0	1
0	1	1
1	1	0

A particular symmetric cipher which uses the XOR operation is the stream cipher. In a stream cipher the plaintext p is encrypted by XOR with the key k, therefore k must be of the same length as p. The ciphertext c can be decrypted similarly. Let n be the size of the message. The set of plaintexts equals the set of ciphertexts and the set of all keys, $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathbb{F}_2^n$. For $p, c, k \in \mathbb{F}_2^n$ we have $E_k, D_k : \mathbb{F}_2^n \to \mathbb{F}_2^n$ such that

$$E_k(p) = p \oplus k = c,$$

$$D_k(c) = c \oplus k = p.$$

This cipher ensures unconditional security if and only if the key is truly random and never used twice, this is known as the Vernam cipher[3]. For most

[3]The Vernam cipher was used for communication between the USA and the USSR during the cold war. The keys, of the same length as the message, where carried by diplomats, Carlet [5].

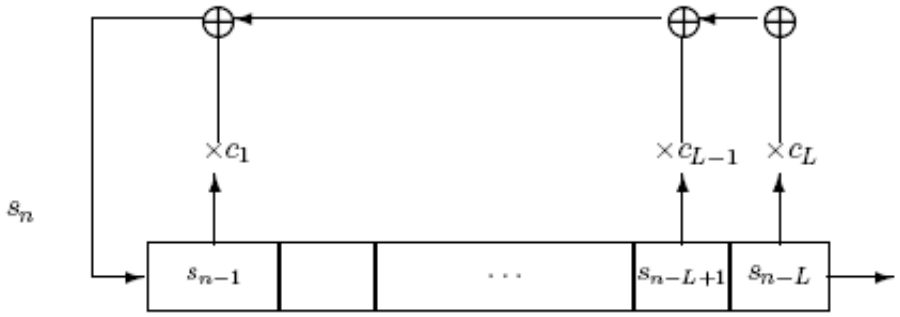

Figure 1: LFSR

applications generating a random key takes too much time and pseudo random keys are used. Starting with an initial key $K_0 = (k_1, \ldots, k_m)$ of length $m < n$ a *keystream* K is generated bit by bit, $K = k_1, \ldots, k_m, k_{m+1}, \ldots, k_n$. The keystream is pseudo random if it 'looks' random, that is it can not be distinguished from a random sequence in polynomial time. For more about pseudo random sequences, see [10].

A cipher containing a keystream is called a *stream cipher*. The initial key K_0 and the generation algorithm must be shared before the communication starts. If the keystream does not depend on the (encrypted) message, then it can be computed synchronously by the sender and receiver. Such ciphers are called synchronous stream ciphers and are traditionally found in constrained telecommunication solutions.

Example 2.4. A simple way to generate pseudo random keys is the *linear feedback shift register* LFSR. Let $K_0 = (s_1, \ldots, s_L)$ be the initial key and $(c_1, \ldots, c_L)$ be the feedback coefficients. New bits are generated by the recursion relation

$$s_n = \bigoplus_{i=1}^{L} c_i s_{n-i}.$$

This operation is represented in figure 1. The keystream is generated bit by bit. For each next state, s_n is computed, where $\times$ is the usual multiplication. The values $s_{n-1}, \ldots, s_{n-L}$ move to the right giving the output bit s_{n-L}. The keystream has a maximum length of $2^L - 1$ bits, since the LFSR has a period of at most $2^L - 1$. So the initial random key can be of reasonably smaller

size then the plaintext.

The LFSR is cryptographically weak since the Berlekamp-Massey algorithm can recover all secret bits if $2L$ coefficients of the keystream are known. There are other ways to generate pseudo random keys, for example the combiner model and filter model. In both models LFSR's are combined with Boolean functions, for more on this see Carlet [5].

2.1.3 Block ciphers and SPN.

Stream ciphers can encrypt arbitrarily long documents. Another cipher which can encrypt arbitrarily long documents is a block cipher. Block ciphers are symmetric ciphers which encrypt and decrypt a block of fixed length of bits. When a document must be encrypted it is partitioned into blocks of size n. If needed, the last block of the document is filled up with extra bits to make it into a block of length n. These extra bits have a certain structure such that they can be recognized after decryption. The block cipher encrypts each block separately, hence the set of all plaintexts equals $\mathcal{P} = \mathbb{F}_2^n$ and the set of all ciphertexts equals $\mathcal{C} = \mathbb{F}_2^n$. The encryption function E_k of a block cipher is then a permutation on $\mathbb{F}_2^n$ and the decryption function D_k is its inverse.

To increase security of the block cipher, the encryption is repeated several rounds r. Each round i, $1 \leq i \leq r$, consist of the same computations but with a different round key K_i, hence each round a new round ciphertext c_i is produced. This way more keys are needed, $K_1, \ldots, K_r$. These round keys can be produced in a similar way as the keystream.

A particular block cipher is the *substitution permutation network* SPN. In general, an SPN with r rounds starts with a block plaintext $p \in \mathbb{F}_2^n$ of n bits and uses a keystream $K_0, K_1, \cdots, K_r$ where each round key consists of n bits. First the initial key K_0 is added to p by XOR, producing the round ciphertext c_0. Then each round consists of three components:

- The first component is a highly nonlinear permutation and is called the Substitution-box, abbreviated as S-box. It actually consists of k parallel smaller S-boxes which are fixed vectorial Boolean functions $S_1, \ldots, S_k : \mathbb{F}_2^m \to \mathbb{F}_2^r$ with $n = k \cdot m$. An S-box $S = (S_1, \ldots, S_k) : \mathbb{F}_2^n \to \mathbb{F}_2^n$ acts on every round ciphertext $c_i \mapsto S(c_i)$.

- The second component is a fixed permutation on the indices of the bits $(x_1, \ldots, x_n) \in \mathbb{F}_2^n$, $P : \mathbb{F}_2^n \to \mathbb{F}_2^n$, which maps $S(c_i) \mapsto P \circ S(c_i)$.

- The third component of each round is addition of a round key K_i of length n by XOR.

Hence an SPN is of the form

$$\begin{array}{rll} & R_0(p) & = \; p \oplus K_0 = c_0, \\ 1 \leq i \leq r & R_i(c_{i-1}) & = \; P \circ S(c_{i-1}) \oplus K_i = c_i, \end{array}$$

and c_r is the ciphertext of the SPN.

The consequence of these components will be explained in the section on cryptanalysis. Note that the two functions S and P are fixed each round. They do not need to be secret and can be made public, hence S^{-1} and P^{-1} are fixed and public as well. Therefore decryption can be done if the initial key and the generation algorithm of the round keys is known.

The vectorial Boolean functions used as S-boxes are very interesting and of main importance of this thesis. Besides nonlinearity other properties must be satisfied, this will be explained in the sections about (vectorial) Boolean functions.

An example of an SPN is given in figure 2. The block length is $n = 16$ bits, each path represents a bit. The SPN consists of $r = 3$ rounds. It starts with adding (XOR) the initial key K_0 to the plaintext followed by round 1. A round consists of four fixed S-boxes $S_1, S_2, S_3, S_4 : \mathbb{F}_2^4 \to \mathbb{F}_2^4$ acting on blocks of 4 bits. Followed by a fixed linear permutation $P : \mathbb{F}_2^{16} \to \mathbb{F}_2^{16}$ and then an XOR addition of the round key. In the third round the linear permutation is omitted. The SPN produces after three rounds a 16 bit ciphertext.

Examples of block ciphers are the *Data Encryption Standard* DES and the Rijndael cipher also known as the *Advanced Encryption Standard* AES. The DES is a so called Feistel cipher, which is a different block cipher then an SPN[4]. It was the standard cipher for decryption of digital communication in the USA since the seventies. It works on blocks of 64 bits, uses a 64 bit key and consists of 16 rounds. In the nineties DES was no longer considered to be secure. The National Institute of Standards and Technology NIST of the USA organized a contest for the best alternative cipher. In 2001 the Rijndael cipher, from J. Daemen and V. Rijmen, was announced to be the next standard cipher, the Advanced Encryption Standard AES. AES works on blocks of size 128. There can be chosen between keys of length $128, 192$ or 256 bits together with $10, 12$ or 14 rounds respectively. For a detailed

[4]Feistel ciphers and in particular DES are explained by Buchmann in [2].

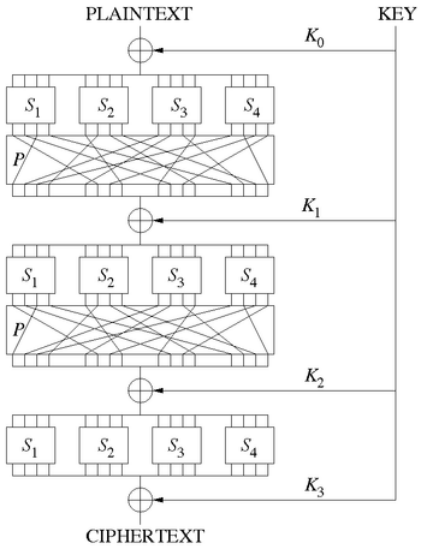

Figure 2: A substitution permutation network SPN cipher.

description of the AES algorithm see the announcement of the AES from the NIST in 2001 [9].

2.2 Cryptanalysis

In cryptanalysis the weaknesses of ciphers are analyzed on possible attacks. For example the Caesar cipher is very weak since the key space $\mathcal{K}$ is very small, $\mathcal{K} = \mathbb{Z}/26\mathbb{Z}$. With an exhaustive search on $\mathbb{Z}/26\mathbb{Z}$ we easily find a unique reasonable combination of plaintext and key. Another weakness is that the encryption takes over any statistical structure of the plaintext. If the language of the message is known, an enemy can use statistics of single letters, letter combinations and words which commonly occur, such as "the", "and", "-tion" and "that" in the English language.

Although statistics are an important tool in cryptanalysis, we will not go into it in much detail. We only give a global idea of a cryptographic attack and possible weaknesses of ciphers.

2.2.1 An attack

An attack consists of analyzing the probabilities of all elements of $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{K}$. Given one or more ciphertexts, are there elements in $\mathcal{K}$ which have a higher probability to occur? For a good cipher the spaces $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{K}$ are large and the probabilities are uniformly distributed. As Shannon said in [18]: "Indeed it is only the existence of these other possibilities that give the system any secrecy."

The general assumption in cryptanalysis is that the attacker knows:

1. One or more ciphertexts, encrypted with the same key.
2. The cipher that is used, i.e. the tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$.
3. The enemy has unlimited computer time and space capacity.

This are somewhat pessimistic assumptions, but therefore safe. Such an attack is also known as a "cipher-only attack". For some attacks additional assumption are made, e.g. pairs of plaintexts and ciphertexts are known (known-plain attack). Since the attacker knows the cipher, an attack comes down to recover the decryption key $d \in \mathcal{K}$.

Definition 2.5. A cryptographic attack is an attempt to recover the decryption key.

With this key the right decryption function $D_d \in \mathcal{D}$ can be found and the original plaintext can be computed. Recall that for a symmetric cipher the encryption key $e \in \mathcal{K}$ can be computed from the decryption key d and so their corresponding functions D_d and E_e. Therefore, by recovering the encryption key e, the cipher can be broken.

2.2.2 Confusion and diffusion

C.E. Shannon introduced in the paper [18] in 1949 two fundamental principles for secure ciphers, *confusion* and *diffusion*. These two principles are still the basis of every modern cryptographic algorithm.

Confusion Any algebraic structure of the cipher need to be concealed. Hence the relation between the plaintext and the ciphertext needs to be complex, that is: the attacker can not derive any information from the distribution of the ciphertext.

Diffusion Each bit of the plaintext and key has to have influence on all bits of the ciphertext.

Example 2.6. The S-box and linear permutation of an SPN are used to satisfy these two principles. We analyze the consequence of a small change in the input of an SPN. When one bit of a plaintext is changed, the S-box in the first round changes more then one bit in a nonlinear way. These bits are spread over the block by the linear permutation. The second round the S-box changes more bits and those will be spread out over the hole block again by the linear permutation. Repeating this in more rounds ensures diffusion and confusion, that is, small changes in the plaintext have large and "complex" consequences in the ciphertext.

2.2.3 Differential and linear attack

There are different types of attacks on block ciphers and stream ciphers known. The two main attacks are the differential attack and the linear attack. Both attacks consist of a certain idea, the actual technique depends on the cipher that needs to be broken. They both where developed to break the DES. So far, no reasonable attacks on the AES cipher are known. Only theoretical attacks which are faster then brute force, but still have unrealistic computation time.

The differential attack is first described by Biham and Shamir in 1991 [1]. It is a chosen plaintext attack. This means we assume that the attacker can choose a plaintext and gets the corresponding ciphertext, so he has an arbitrary amount of pairs of plain- and ciphertexts. The idea of a differential attack is to make small differences in the plaintext and look for non-random behavior in the ciphertext differences. Then use this to recover some of the key bits of the last round key K_r of the keystream $K = K_0, K_1, \dots K_r$. When enough bits are recovered an exhaustive can be made to find the hole decryption key. The differential attack was the first serious attack on the DES cipher.

Remark 2.7. The last round key is closely related to the ciphertext. Recall the description of a general SPN in section 2.1.3. Let r be the number of rounds, S the S-box and P the linear permutation. The last round R_r of the SPN is of the form:

$$R_r(c_{r-1}) = P \circ S(c_{r-1}) \oplus K_r = c.$$

Where $c_{r-1} = R_{r-1} \circ \cdots \circ R_0(p)$, p the plaintext and c is the ciphertext.

The linear attack is introduced by Matsui in 1993 [17]. It is a known plaintext attack. This means that the attacker is assumed to have pairs of

plaintexts and ciphertexts. Like the differential attack, this is also an attack on the last round key K_r. The idea is to make a linear approximation of the (nonlinear) S-box and use this to make a linear approximation of the whole SPN. Given a plaintext we compute the approximation for c'_{r-1} and the approximation for $P \circ S(c'_{r-1}) = c'$. This approximation has a certain probability to be right. This gives us a probability of the last round key K_r to be right by $c' \oplus c = K_r$. When enough bits are known with high probability they are assumed to be good. The other bits are found by an exhaustive search and the decryption key is recovered.

The linear attack was also introduced as an attack on DES. Matsui even proved that if the attacker can assume that the plaintext is in English and written in ASCII code, i.e. with a pc keyboard, the three general assumptions of an attack are sufficient to make a successful ciphertext-only attack.

3 Coding Theory

In this section we will introduce some basics of coding theory. In section 3.1 we talk about the three parameters $[n, k, d]$ of a code and basic properties such as the generator matrix and the parity check matrix of a code. In section 3.2 We will say more about the Hamming code and the corresponding simplex code since these codes have strong relations with Boolean functions.

3.1 Error Correcting Codes

We start with a famous example where codes are used. In 1972 the first black-and-white photograph was made from the planet Mars. These photos where sent from the satellite to the receiver station on the earth. The satellite put a grid on the photograph and measured the darkness of each square on the scale of 0 to 63. Each value of darkness can be represented by binary numbers and transformed into a string of six 0s and 1s to make all 64 different 'words'. The satellite could send these words to the earth. The received signal will be very weak and due to thermal noise it could happen that some 0s changed into 1s and vice versa. One can imagine that this will have major consequences for the received photograph. To prevent this they did not use six bit words, but words of a length of 32 bits. This made it possible to construct all 64 words in such a way that each two words differ on 16 places. If one 0 was chanced into a 1 it was still clear which word it should have been and the receiver could *correct* this *error*. We will see in this section that with the code used for the photograph of Mars it is even possible to correct 7 errors.

We compare this idea to the words in our language. We can clearly make many more words with the letters of the alphabet than we use, or are stated in the dictionary. If a word is not too small it is not a problem if one of the letters is changed into another one, in most cases we will detect the error and correct it easily, even without knowing the context. But from 'in most cases' the question arises; how can we order the space of all words efficiently? Coding theory studies this question.

3.1.1 Binary linear codes

There are different types of codes; here, we are only interested in linear codes. In general we take a group as an alphabet, but since we investigate codes that are related to Boolean functions we restrict ourself to the binary alphabet

$\mathbb{F}_2$. A word w of length n is then an element of the vector space $\mathbb{F}_2^n$, $w \in \mathbb{F}_2^n$. So we will treat only binary linear codes and we will just talk about them as 'codes'.

Definition 3.1. Let n, k be two integers, $0 \leq k \leq n$. A (binary linear) code C is a linear subspace of $\mathbb{F}_2^n$ of dimension k, i.e. $\mathbb{F}_2^k \simeq C \subseteq \mathbb{F}_2^n$.

The elements of a code C are called codewords and are of length n. The number of codewords of C is 2^k. If a code C consists of only one codeword, then C is called trivial, i.e. $C = \{(0, \ldots, 0)\}$. We define the Hamming distance between two words of $\mathbb{F}_2^n$.

Definition 3.2. Let $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ be two words in $\mathbb{F}_2^n$. The Hamming distance between x and y is defined as

$$d_H(x, y) := |\{i : \ x_i \neq y_i\}|, \ 1 \leq i \leq n.$$

Note that this is a metric on $\mathbb{F}_2^n$. There are two special words in $\mathbb{F}_2^n$. The all zero word, which is written as $\mathbf{0} := (0, \ldots, 0)$ and the all one word which is written as $\mathbf{1} := (1, \ldots, 1)$. With this definitions we can define the Hamming weight of a word x as its Hamming distance to $\mathbf{0}$.

Definition 3.3. Let $x \in \mathbb{F}_2^n$, the Hamming weight of x is

$$w_H(x) := d_H(x, \mathbf{0})$$

Note that this is equal to sum of all coordinates, $w_H(x) = \sum_{i=1}^n x_i$. An important property of a code is the minimal distance between all codewords.

Definition 3.4. The minimum distance d of a code C is defined as

$$d := min\{d_H(x, y) : x, y \in C, \ x \neq y\}$$

If the minimal distance of a code C is 1, then it is possible that a change of one bit will lead to another codeword. This way an error can not be detected and certainly not corrected. Clearly, for a code C to detect or correct errors a higher minimal distance is needed. The minimal distance can be easily derived from a code C with the following lemma.

Lemma 3.5. *Let C be a code, the minimum distance d of C is equal to the minimum weight of all nonzero codewords, that is*

$$d = \min_{c \in C \setminus \{\boldsymbol{0}\}} w_H(c).$$

Proof. Since C is linear we have for any two codewords $c, c' \in C$ that $c + c' \in C$. Note that we have $c - c' = c + c'$ since $C \subseteq \mathbb{F}_2^n$. We get that $d_H(c, c') = d_H(c + c', \mathbf{0}) = w_H(c + c')$ and therefore $\min_{c,c' \in C,\ c \neq c'} d_H(c, c') = \min_{c \in C \setminus \{\mathbf{0}\}} w_H(c)$. □

We now have seen the three most important parameters of a code. Let C be a code with codewords of length n, dimension k and with minimum distance d, then C is called an $[n, k, d]$ code.[5] If d is not known we call C a $[n, k]$ code.

Example 3.6. Recall the code in the beginning of this section, used for sending photos of Mars. It is a binary linear code. The length of the codewords are 32, the dimension of the code is 6 since there are $2^6 = 64$ codewords and the minimal distance between each two codewords is 16. Therefore this is a $[32, 6, 16]$ code.

Since we have a metric on $\mathbb{F}_2^n$ we can define a ball (or sphere) around each codeword $c \in C$ of radius $\rho \in \mathbb{N}$ by

$$B_\rho(c) := \{x \in \mathbb{F}_2^n : d_H(c, x) \leq \rho\}.$$

For small ρ it is possible that all balls $B_\rho(c)$, of all codewords $c \in C$ are disjoint. We have special interest in the maximum radius e for which this is true.

$$e := \max\{e \in \mathbb{N} : B_e(c) \cap B_e(c') = \emptyset, \text{ for all } c, c' \in C,\ c \neq c'\}$$

For a code C with minimal distance d we have that $d = 2e + 1$ or $d = 2e + 2$. If a word $w \in \mathbb{F}_2^n$ is received it contains possible errors. The receiver needs to check which ball $B_e(c)$ contains the word w and correct w to the codeword c. We see that e equals the number of errors that can be corrected. It can be that there are words $w \in \mathbb{F}_2^n$ that are not contained in any ball $B_e(c)$, these can still be corrected if there is a unique closest codeword c. It can happen (and it mostly will) that there are words $w \in \mathbb{F}_2^n$ with no unique closest codeword c. For these words errors can only be detected but not corrected. A code C with radius e in which every word $w \in \mathbb{F}_2^n$ is contained in a ball $B_e(c)$ for some codeword $c \in C$ is called *perfect*.

Example 3.7. The code C of odd length n that only consists of the vectors $\mathbf{0}$ and $\mathbf{1}$ is perfect. It has minimal distance $d = n$ and the radius $e = (n-1)/2$.

[5] In some literature they write $(n, 2^k, d)$ code instead of $[n, k, d]$ code.

All words $w \in \mathbb{F}_2^n$ such that $w_H(w) \leq e$ will be corrected as $\mathbf{0}$ and words such that $w_H(w) \geq (n-e) = e+1$ will be corrected as $\mathbf{1}$. Note further that C is of dimension 1, therefore C is an $[n, 1, n]$ code.

A perfect code is desirable since it can correct every word in our space, $w \in \mathbb{F}_2^n$, but most codes we will see below are not perfect. Other properties are desirable as well and trade-offs must be made. For instance for a fixed dimension k of the code, i.e. the numbers of codewords equals 2^k, and a fixed minimal distance d we want n to be as small as possible since the use of smaller words has an high advantage for communication (time, power) and storage. And of course, when fewer bits are used fewer errors can occur. We will not go into details about the error probability and the information rate of a code, for more on this see [20].

3.1.2 Structure of a code

For each code there exists an so called generator matrix.

Definition 3.8. Let C be a code of length n and dimension k, the generator matrix is an $k \times n$ matrix G such that the rows of G form a basis of C.

Since the rows of G form a basis of C and by the fact that C is linear it follows that

$$C = \{a \cdot G : a \in \mathbb{F}_2^k\}.$$

Note that, since the basis of C is not unique, the generator matrix is not unique either. For every code C there exists a generator matrix which is written in a standard form $G = (I_k \ P)$, where I_k is the $k \times k$ identity matrix and P an $k \times (n-k)$ matrix. We will see below that these codes are indeed equivalent. The standard form $(I_k \ P)$ is also called the reduced echelon form since it is equal to the reduced echelon form of any generator matrix G of the code C. If the generator matrix is written in the standard form, the first k bits of a codeword are called information bits. When the information bits are determined, the last $n-k$ bits are fixed. Otherwise the dimension of the code would be greater than k. These $n-k$ bits are called redundancy bits or *parity check* bits.

Remark 3.9. The name parity check bits is used for historic reason. The first codes where made by adding one extra bit to a string of information bits. This extra bit depends on the previous bits and ensures that the codeword is of even weight, hence it is called the parity check bit. An example of such

code is the ASCII-code used in the keyboard of a computer. A codeword $c = (c_1, \ldots, c_8)$ has length $n = 8$ of which are $k = 7$ information bits. The last bit c_8 is the parity bit and equals $\bigoplus_{i=1}^{7} c_i = c_8$. It follows that $c_8 \oplus \bigoplus_{i=1}^{7} c_i = 0$, hence c is of even weight. The ASCII-code is an $[8, 7, 2]$ code. For such code, an error can be detected. In such case, the receiver could ask to send the codeword again.

Example 3.10. Let C be a $[6, 3, 3]$ code. There are $2^3 = 8$ codewords of length 6 which can be represented by the first three bits of the 6-tuple $c = (c_1, \ldots, c_6)$. These are called the information bits. The three 'extra' bits are called parity check bits and are determined as follows, $c_4 = c_2 + c_3$, $c_5 = c_1 + c_3$, $c_6 = c_1 + c_2$. The generator matrix G of the code C, becomes

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

Note that this generator matrix is in the standard form. Multiply each element of $a \in \mathbb{F}_2^3$ with G we get all codewords c, i.e. $aG = c$,

$$\begin{aligned} C &= \{\mathbf{0}, (100011), (010101), (110110), \\ &\quad (001110), (101101), (011011), (111000)\}. \end{aligned}$$

The minimum weight of all nonzero codewords is 3, so by lemma 3.5 the minimum distance d is indeed 3.

We define an equivalence relation for $[n, k]$-codes.

Definition 3.11. Let C_1, C_2 be two $[n, k]$-codes and G_1, G_2 the $k \times n$ generator matrices respectively. The two codes C_1 and C_2 are equivalent if there exists a $k \times k$ invertible matrix M and an $n \times n$ permutation matrix P such that

$$MG_1 = G_2P.$$

Observe that M induces a change of the basis of C_1 and P induces a permutation on the columns of G_2, that is it induces a fixed permutation on the bits of each codewords of C_2. It is easy to show with linear algebra that this defines an equivalence relation since both matrices M and P have an inverse M^{-1}, P^{-1} respectively.

For each code C there exist a dual code $C^{\perp}$.

Definition 3.12. Let C be a code, the dual code $C^\perp$ is defined as

$$C^\perp = \{x \in \mathbb{F}_2^n : \forall c \in C,\ x \cdot c = 0\}.$$

Recall that the inner product is defined as $x \cdot c = \bigoplus_{i=1}^n x_i c_i$. If C is a $[n,k]$ code, then its dual $C^\perp$ is a $[n, n-k]$ code. If C consists of even words only, i.e. $\forall c \in C\ w_H(c)$ is even, then the intersection $C \cap C^\perp$ is larger than $\{\mathbf{0}\}$. When $C = C^\perp$ the code C is called self dual. The dual code $C^\perp$ has again a $(n-k) \times n$ generator matrix H. By the definition of the dual code it follows that for $c \in C$ and $c' \in C^\perp$ we have $c \cdot c' = 0$. With linear algebra we get that $GH^T = 0$. In particular we have

$$c \in C \Leftrightarrow cH^T = 0. \tag{1}$$

The matrix H is called the *parity check matrix* of code C since it can be used by the receiver to check for a word $w \in \mathbb{F}_2^n$ whether all the parity check bits are correct, i.e. whether $wH^T = \mathbf{0}$. With relation (1) it is possible to define a code C by its parity check matrix H. Note that this matrix is not unique either. If the generator matrix G is written in the standard form $G = (I_k\ P)$, then H can be written as $H = (P^T\ I_{n-k})$.[6]

Example 3.13. The dual $C^\perp$ of the $[6,3,3]$ code C from example 3.10 can be obtained as follows. Recall the construction of the parity check bits of C, $c_4 = c_2 + c_3$, $c_5 = c_1 + c_3$, $c_6 = c_1 + c_2$. We want for the parity check matrix that $c_2 + c_3 + c_4 = 0$, $c_1 + c_3 + c_5 = 0$, $c_1 + c_2 + c_6 = 0$. We get

$$H = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

This is the 3×6 generator matrix for the dual code $C^\perp$. Note that H is of the form $(P^T\ I_{n-k})$, where G was written in the standard form $G = (I_k\ P)$. We have $C^\perp = \{aH : a \in \mathbb{F}_2^3\}$. We can derive all codewords of $C^\perp$,

$$\begin{aligned} C^\perp \ = \ & \{\mathbf{0}, (011100), (101010), (110110), \\ & (110001), (101101), (011011), (000111)\}. \end{aligned}$$

So $C^\perp$ is also a $[6,3,3]$ code and $C \cap C^\perp = \mathbf{0}$. Further we have that $c \in C$ if and only if $cH^T = 0$, H is the parity check matrix of code C.

[6] This holds since we restrict ourselves to binary alphabet.

From the construction of H we make the following observation. Let $h_1, \ldots, h_n$ be all the columns of H and $a = (a_1, \ldots, a_n)$ some vector in $\mathbb{F}_2^n$, then aH^T equals $a_1h_1 + \cdots + a_nh_n$. If $a_1h_1 \oplus \cdots \oplus a_nh_n = 0$ then the columns h_i, corresponding to every ith bit with $a_i = 1$, are linearly dependent. Using this notation, the minimal number of dependent columns of H is equal to

$$\min_{a \in \mathbb{F}_2^n} \{w_H(a) : a_1h_1 \oplus \cdots \oplus a_nh_n = 0\}.$$

Lemma 3.14. *Let H be the parity check matrix of a $[n, k, d]$ code C. Then the minimal distance d is equal to the minimal number of dependent columns of H.*

Proof. We have that $c \in C$ if and only if $cH^T = 0$. It follows that each codeword c with weight l, corresponds to l linear dependent columns of H. By lemma 3.5 the minimum weight of all nonzero codewords equals d. Hence d equals the minimal number of linear dependent columns of H. □

There exists a natural way to extend a code C with one extra bit.

Definition 3.15. Let C be a code and $(c_1, \ldots, c_n) \in C$ a codeword. The extended code $\overline{C}$ consists of the codewords $(c_1, \ldots, c_{n+1})$, where $c_{n+1} = \bigoplus_{i=1}^{n} c_i$. That is

$$\overline{C} = \{(c_1, \ldots, c_{n+1}) \in \mathbb{F}_2^{n+1} : (c_1, \ldots, c_n) \in C \text{ and } c_{n+1} = \bigoplus_{i=1}^{n} c_i\}.$$

For all codewords $c \in \overline{C}$ we get that $\bigoplus_{i=1}^{n+1} c_i = 0$, so all codewords are of even weight. Recall that the minimal distance d is equal to the minimum weight of all nonzero codewords, lemma 3.5. It follows that if a code C has an odd minimal distance d, then the extended code $\overline{C}$ has minimum distance $d + 1$. If d is even, then the minimal distance of $\overline{C}$ equals the minimum distance of C. The generator matrix of the extended code $\overline{C}$ can be obtained from the generator matrix G of C by adding a new column which makes each row of even weight. Let $(g_{i,j})$ be the elements of G, then

$$\overline{G} = \begin{pmatrix} g_{1,1} & \cdots & g_{1,n} & \bigoplus_{i=1}^{n} g_{1,i} \\ \vdots & & \vdots & \vdots \\ g_{k,1} & \cdots & g_{k,n} & \bigoplus_{i=1}^{n} g_{k,i} \end{pmatrix}.$$

Note that $\overline{G}$ is indeed a $k \times (n+1)$ matrix. To obtain the parity check matrix $\overline{\overline{H}}$ a row must be added which corresponds to $c_1 \oplus \ldots \oplus c_{n+1} = 0$, it follows that

$$\overline{\overline{H}} = \begin{pmatrix} 1 & \cdots & 1 & 1 \\ & & & 0 \\ & H & & \vdots \\ & & & 0 \end{pmatrix}.$$

Observe that with this notation $\overline{H}$ is the $(n-k) \times (n+1)$ generator matrix of the extended dual code $\overline{C^{\perp}}$.

3.2 Simplex Codes

A colleague of C.E. Shannon at Bell Laboratories was R. Hamming. He invented the first error correcting code. This code is called the Hamming code and is seen as the beginning of coding theory, see [11]. Simplex codes are closely related to Hamming codes and have a connection to Boolean functions.

3.2.1 Hamming codes

Hamming codes exist only for codes of length $2^m - 1$, for some integer m.

Definition 3.16. Let m be a positive integer and H be an $m \times (2^m - 1)$ matrix such that the columns $x \in \mathbb{F}_2^n$ are nonzero and pairwise distinct. A code which is defined by such a parity check matrix H is called a Hamming code.

Observe that every nonzero vector of $\mathbb{F}_2^m$ is a column of H. Since two nonzero vectors in $\mathbb{F}_2^m$ add up to another vector the minimal number of dependent columns is 3. It follows from lemma 3.14 that the minimal distance $d = 3$. We conclude that a Hamming code is a $[2^m - 1, 2^m - m - 1, 3]$ code.

Example 3.17. We take $m = 3$, the 3×7 parity check matrix H

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

This defines a $[7, 4, 3]$ Hamming code. All codewords can be found by as follows.

Firstly, all words of weight three can be found by looking at all combinations of three linear dependent columns. We get the codewords:

$$(1,1,1,0,0,0,0),(1,0,0,1,1,0,0),(1,0,0,0,0,1,1),$$

$$(0,1,0,1,0,1,0),(0,0,1,1,0,0,1),(0,0,1,0,1,1,0).$$

Secondly, we take all linear combinations of these codewords and get:

$$\mathbf{0},(0,1,1,1,1,0,0),(0,1,1,0,0,1,1),(1,0,1,1,0,1,0),(1,1,0,1,0,0,1)$$

$$(1,1,0,0,1,1,0),(0,1,1,0,0,1,1),(0,0,0,1,1,1,1),(1,0,1,0,1,0,1),\mathbf{1}.$$

These are all $2^4 = 16$ codewords. Since the minimum distance $d = 3 = 2e+1$, this code can correct $e = 1$ error. If we make a ball $B_1(c)$ of radius 1 around every codeword c, then each $B_1(c)$ will consist of $n + 1 = 2^3$ words. Since we have 2^4 codewords the union of all balls covers the whole space, $2^4 \cdot 2^3 = 2^7 = |\mathbb{F}_2^7|$. Therefore this $[7, 4, 3]$ Hamming code is a perfect code; in fact, all Hamming codes are perfect codes.

We can identify a vector space $\mathbb{F}_2^m$ with the field extension $V = \mathbb{F}_{2^m}$. Let ω be a primitive element of V with minimal polynomial $f_{min}^{\omega}(x)$ over $\mathbb{F}_2$ of degree m. We have that ω generates the multiplicative group V^* of V, $\langle\omega\rangle = \{\omega, \omega^2, \ldots, \omega^{2^m-1} = 1\} = V^*$ and $\{1, \omega, \omega^2, \ldots, \omega^{m-1}\}$ is a basis of V. Then $(a_1, \ldots, a_m) \in \mathbb{F}_2^m$ corresponds to the element $a_1 + a_2\omega + a_3\omega^2 + \cdots + a_m\omega^{m-1} \in V$.

The $m \times 2^m - 1$ matrix H consisting of columns $x \in V^*$ defines a Hamming code and is denoted as

$$H = [x] := [\omega^t : t \in \{0, 1, \ldots, 2^m - 2\}].$$

More generally we have the following lemma.

Lemma 3.18. *Let $f : V \to V$ be a function, then*

$$H = [f(x)] := [f(\omega^t) : t \in \{0, 1, \ldots, 2^m - 2\}], \tag{2}$$

defines a Hamming code if and only if the function f is a permutation on V which leaves zero fixed, $f(0) = 0$.

Proof. Note that we can view f as $f|_{V^*} : V^* \to V^*$ a permutation on V^*. By definition of a Hamming code, every column of $[f(x)]$ will be nonzero and distinct. The lemma follows immediately. □

3.2.2 Simplex and double simplex codes

Simplex codes are constructed from Hamming codes.

Definition 3.19. Let C be a Hamming code. The dual of a Hamming code $C^\perp$ is called a simplex code.

A simplex code $C^\perp$ is generated by the parity check matrix H of the corresponding Hamming code. By lemma 3.18, every simplex code corresponds to a permutation f on V which leaves zero fixed, by the generating matrix $H = [f(x)]$.

Corollary 3.20. *Let $f : V \to V$ be a function and H the $m \times 2^m - 1$ matrix*

$$H = [f(x)] = [f(\omega^t) : t \in \{0, 1, \ldots, 2^m - 2\}].$$

Then H generates a simplex code if and only if f is a permutation and $f(0) = 0$.

The simplex code is then defined as $C^\perp = \{aH : a \in V\}$. Using the notation from (2) we get for each $a \in V$ that a codeword is of the form $[a \cdot f(\omega^t) : t \in \{0, 1, \ldots, 2^m - 2\}]$, where a and $f(\omega^t)$ are vectors in V written in the basis $\{1, \omega, \ldots, \omega^{m-1}\}$. We denote such a simplex code $C^\perp$ as

$$C^\perp := \langle f(x) \rangle.$$

Example 3.21. Let ω be a primitive element of V and $f^\omega_{min}(x) = x^3 + x + 1$ it's minimal polynomial over $\mathbb{F}_2$. Then $V \simeq \mathbb{F}_2[x]/f^\omega_{min}(x)$, ω generates the multiplicative group of V, i.e. $\langle \omega \rangle = V^*$, and $\{1, \omega, \omega^2\}$ form a basis of V together with the relation $w^3 + w + 1 = 0$. We can define the simplex code $C^\perp$ by the 3×7 generator matrix

$$H = [x] = (1, \omega, \ldots, \omega^6) = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

The simplex code $C^\perp$ is the set $\{aH : a \in V\}$. We get the codewords

$$\{\mathbf{0}, (1,0,1,1,1,0,0), (0,1,0,1,1,1,0), (0,0,1,0,1,1,1),$$

$$(1,0,0,1,0,1,1), (1,1,0,0,1,0,1), (1,1,1,0,0,1,0), (0,1,1,1,0,0,1)\}$$

Observe that the codewords are ordered in such way that each next codeword is a cyclic shift to the right. Codes for which this is possible are called cyclic codes. There exist simplex codes and Hamming code which are cyclic. An example of a noncyclic Hamming code is example 3.17, the dual of this Hamming code is a noncyclic simplex code.

For two codes C_1, C_2 we can make a new code C by taking the direct sum of all codewords, that is

$$C = \{c \oplus d : c \in C_1,\ d \in C_2\}.$$

Definition 3.22. The direct sum of two distinct simplex codes is called a double simplex code.

Let f_1 and f_2 two distinct functions as in corollary 3.20 which generate two simplex codes $C_1^\perp$ and $C_2^\perp$ respectively. We write a double simplex code $C^\perp$ as

$$C^\perp := \langle f_1(x) \rangle \oplus \langle f_2(x) \rangle.$$

Let H_1 and H_2 be the generator matrices for $C_1^\perp$ and $C_2^\perp$ respectively. The $2m \times 2^m - 1$ generator matrix H of the double simplex code $C^\perp$ is of the from

$$H = \begin{bmatrix} H_1 \\ H_2 \end{bmatrix} = \begin{bmatrix} f_1(x) \\ f_2(x) \end{bmatrix}.$$

We are interested in a particular double simplex code.

Definition 3.23. Let f be a nontrivial permutation on V which vanishes at 0. Define the double simplex code $C_f^\perp$ generated by f as the double simplex code where $f_1(x) = x$ and $f_2(x) = f(x)$, that is

$$C_f^\perp := \langle x \rangle \oplus \langle f(x) \rangle.$$

The generator matrix of this simplex code is then defined as

$$H_f := \begin{bmatrix} x \\ f(x) \end{bmatrix}.$$

Proposition 3.24. *Let $f_1, f_2 : V \to V$ be two permutations such that $C^\perp = \langle f_1(x) \rangle \oplus \langle f_2(x) \rangle$ is a double simplex code. Then $C^\perp$ is equivalent to the double simplex code $C_{f'}^\perp = \langle x \rangle \oplus \langle f'(x) \rangle$ with $f' = f_2 \circ f_1^{-1}$.*

Proof. Since f_1, f_2 are permutations on V with $f_1(0) = f_2(0) = 0$, f' is also a permutation on V and $f'(0) = f_2 \circ f_1^{-1}(0) = 0$. Let H and $H_{f'}$ be the $2m \times (2^m - 1)$ generator matrices of $C^\perp$ and $C_{f'}^\perp$ respectively. Then H and $H_{f'}$ are equal up to a permutation of the columns, that is for $f_1(x) = y$ we have

$$H = \begin{bmatrix} f_1(x) \\ f_2(x) \end{bmatrix} = \begin{bmatrix} y \\ f_2 \circ f_1^{-1}(y) \end{bmatrix} = \begin{bmatrix} x \\ f'(x) \end{bmatrix} P = H_{f'} P,$$

for some $(2^m - 1) \times (2^m - 1)$ permutation matrix P. Hence the two codes $C^\perp$ and $C_{f'}^\perp$ are equivalent. □

More can be said of these double simplex code using vector Boolean functions $f : \mathbb{F}_2^m \to \mathbb{F}_2^m$. This will be done in section 6 where coding theory and Boolean functions will be combined to derive the Dillon-Wolfe function.

4 Boolean Functions

With the work of George Boole on logical reasoning with the variables 0 and 1, he created a basis on which others could build and which eventually became the concept of the digital computer. Functions which have as input and output only zero's and one's are named after Boole: Boolean functions. These functions are often used in cryptography.

In this section we introduce Boolean functions and study their properties. We start in section 4.1 with basic definitions such as the Hamming weight, the algebraic normal form and the notion of the algebraic degree of a function. In section 4.2 we define the Fourier transform and the Walsh transform and we will derive several propositions for these transforms. In section 4.3 we define an equivalence relation for Boolean functions and deduce some invariant properties of Boolean functions. In section 4.4 we analyze the properties of Boolean functions for their use on cryptographic ciphers.

4.1 Definitions and Properties

We distinguish two kinds of Boolean functions.

Definition 4.1. For $n, m \in \mathbb{N}$, $m \neq 0$, a function $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ is called a Boolean function if $m = 1$. For $m > 1$, f is called a vectorial Boolean function.

In this section we look only at Boolean functions, this gives already quite a rich theory. Vectorial Boolean functions will be studied in chapter 5. When we talk about a Boolean function $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ we say, f is a Boolean function on $\mathbb{F}_2^n$ and write $f \in \mathcal{BF}_n$.

4.1.1 Basics

We start with some basic definitions for Boolean functions and derive some simple properties from them.

Definition 4.2. A Boolean function f on $\mathbb{F}_2^n$ is called linear if it is a sum of single variables, for $(a_1, \ldots, a_n) \in \mathbb{F}_2^n$ that is

$$f(x) = \bigoplus_{i=1}^{n} a_i x_i.$$

We will just write x for the vector $(x_1, \ldots, x_n)$ if it is clear from the context what we mean. Linear functions can also be denoted as an inner product $a \cdot x = \bigoplus_{i=1}^{n} a_i x_i$ with $a = (a_1, \ldots, a_n) \in \mathbb{F}_2^n$. Combining these functions with the two constant functions, 0 and 1, we get an affine function.

Definition 4.3. Boolean functions on $\mathbb{F}_2^n$ which are the sum of a constant and a linear Boolean function are called affine functions. That is, for $a_0 \in \mathbb{F}_2$ and $a = (a_1, \ldots, a_n) \in \mathbb{F}_2^n$ we have the affine functions $(a \cdot x) \oplus a_0 = a_0 \bigoplus_{i=1}^{n} a_i x_i$ where '$\cdot$' is the inner product.

Definition 4.4. Let $x = (x_1, \ldots, x_n)$ be a vector of $\mathbb{F}_2^n$, then the support of x is the subset I of $N = \{1, 2, \ldots, n\}$ corresponding to all $x_i \neq 0$, i.e.

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

For Boolean vectors we get the following. Let $x, y \in \mathbb{F}_2^n$ and $I, J \subseteq N$ their support respectively. Then y is said to be in the support of x if $J \subseteq I$, we write $\mathrm{supp}(y) \subseteq \mathrm{supp}(x)$. The support of a function f is the subset of $\mathbb{F}_2^n$ such that $f(x) \neq 0$, that is $\mathrm{supp}(f) = \{x \in \mathbb{F}_2^n : f(x) \neq 0\}$.

Recall from chapter 3 that the Hamming weight of a vector $x \in \mathbb{F}_2^n$ is defined by the number of nonzero elements and therefore equal to the cardinality of the support, $w_H(x) = |\{x_i \in x : x_i \neq 0\}|$. The Hamming weight of a Boolean function and the Hamming distance between two Boolean functions is defined similar.

Definition 4.5. The Hamming weight of a Boolean function f on $\mathbb{F}_2^n$ is equal to the cardinality of the support, that is

$$w_H(f) = |\{x \in \mathbb{F}_2^n : f(x) \neq 0\}|.$$

Definition 4.6. The Hamming distance of two Boolean functions f, g on $\mathbb{F}_2^n$ is defined as

$$d_H(f, g) := w_H(f \oplus g).$$

Note that the Hamming weight of a Boolean function $f \in \mathcal{BF}_n$ is equal to $\sum_{x \in \mathbb{F}_2^n} f(x)$. There is a class of functions with a particular kind of support.

Definition 4.7. A Boolean function f on $\mathbb{F}_2^n$ is called balanced if the output is zero as many times as it is one, that is

$$|\{x \in \mathbb{F}_2^n : f(x) = 0\}| = |\{x \in \mathbb{F}_2^n : f(x) = 1\}|.$$

So for a balanced function f we have that $2|\text{supp}(f)| = |\mathbb{F}_2^n|$. Therefore we have for the Hamming weight of a balanced function that $w_H(f) = 2^{n-1}$. There are $\binom{2^n}{2^{n-1}}$ different Boolean functions on $\mathbb{F}_2^n$ which are balanced. We give the simplest family of balanced functions.

Lemma 4.8. *All nonzero linear functions l on $\mathbb{F}_2^n$ are balanced.*

Proof. Let $l(x) \in \mathcal{BF}_n$ be a nonzero linear function and $b \in \mathbb{F}_2^n$ such that $l(b) = 1$. Note that such b always exists. Then the map $x \mapsto x + b$ which maps from $l^{-1}(0)$ to $l^{-1}(1)$ is one to one since $l(x + b) = l(x) \oplus 1$. Hence $|l^{-1}(0)| = |l^{-1}(1)|$ and l is balanced. □

More general, all non constant affine functions are balanced since the function $l(x) \oplus 1$ is balanced if $l(x)$ is balanced.

Before we derive more results we need representations for Boolean functions.

4.1.2 Representations ANF and TT

There are different ways to represent a Boolean formula in n variables. We give two of them. A Boolean function $f \in \mathcal{BF}_n$ can be represented by its *truth table* TT. That is a table of values, for each $x \in \mathbb{F}_2^n$ the value of $f(x)$ is given, see table 1 for an example.

x_1	x_2	x_3	$f(x)$
0	0	0	1
1	0	0	0
0	1	0	1
1	1	0	1
0	0	1	1
1	0	1	0
0	1	1	1
1	1	1	0

Table 1: Truth table of a Boolean function $f \in \mathcal{BF}_3$.

The second representation is called the *algebraic normal form* ANF and is a polynomial representation of a Boolean function.

Definition 4.9. A Boolean function f on $\mathbb{F}_2^n$ can be written in the algebraic normal form, ANF. For $N = \{1, 2, \ldots, n\}$ we write

$$f(x_1, \ldots, x_n) = \bigoplus_{I \in \mathcal{P}(N)} a_I (\prod_{i \in I} x_i), \quad a_I \in \mathbb{F}_2,$$

or just

$$f(x) = \bigoplus_{I \in \mathcal{P}(N)} a_I x^I, \quad a_I \in \mathbb{F}_2.$$

Different values of a_I define different Boolean functions and since $|\mathcal{P}(N)| = 2^n$ we have that every Boolean function can be represented uniquely. We give an example

Example 4.10. The Boolean function f from the TT in table 1 can be written in its ANF, $f(x) = 1 \oplus x_1 \oplus x_1x_2 \oplus x_1x_2x_3$.

There exist several ways to go from the ANF to the TT representation and vice versa. From ANF to TT we can just compute the values of the Boolean function $f(x)$ for all vectors $x \in \mathbb{F}_2^n$ by brute force, of course there exist algorithms which are more efficient. From TT to ANF can also be easily computed. We first give the definition of an atomic formula.

Definition 4.11. For a vector $u \in \mathbb{F}_2^n$ the atomic formula f_u is the Boolean function with support $\{u\}$, i.e. $\text{supp}(f_u) = \{u\}$.

Let $u = (u_1, \ldots, u_n)$ and $\overline{u_i}$ the complement of the coordinate u_i, i.e. $\overline{u_i} = 1 \oplus u_i$. Then the atomic formula equals $f_u(x) = (\overline{u_1} \oplus x_1) \cdot \ldots \cdot (\overline{u_n} \oplus x_n)$, since for such f_u we have that $f_u(x) = 1$ if and only if $x = u$.

Observe that from a TT, the support of a Boolean function is easily derived. The Boolean function f is equal to the sum of all atomic formulas f_u such that u is in the support of f. Hence the ANF is obtained by

$$f(x) = \bigoplus_{u \in \text{supp}(f)} f_u(x).$$

Example 4.12. We compute the ANF of the Boolean function from the TT of table 1. The support of f is $\{(0,0,0), (0,1,0), (1,1,0), (0,0,1), (0,1,1)\}$.

We get the following atomic formulas;

$$\begin{array}{rcll} f_{000}(x) & = & (1\oplus x_1)(1\oplus x_2)(1\oplus x_3) & = & 1\oplus x_1\oplus x_2\oplus x_3\oplus x_1x_2\oplus x_1x_3 \\ & & & & \oplus x_2x_3\oplus x_1x_2x_3 \\ f_{010}(x) & = & (1\oplus x_1)x_2(1\oplus x_3) & = & x_2\oplus x_1x_2\oplus x_2x_3\oplus x_1x_2x_3 \\ f_{110}(x) & = & x_1x_2(1\oplus x_3) & = & x_1x_2\oplus x_1x_2x_3 \\ f_{001}(x) & = & (1\oplus x_1)(1\oplus x_2)x_3 & = & x_3\oplus x_1x_3\oplus x_2x_3\oplus x_1x_2x_3 \\ f_{011}(x) & = & (1\oplus x_1)x_2x_3 & = & x_2x_3\oplus x_1x_2x_3 \end{array}$$

Now we take the sum of all atomic formulas and get $f(x) = 1 \oplus x_1 \oplus x_1x_2 \oplus x_1x_2x_3$.

Remark 4.13. Each atomic f_u contains the monomial of n variables, $x^N = x_1x_2\cdots x_n$. Since we take the sum over all atomic functions modulo 2, $f(x)$ contains this monomial x^N if and only if the number of elements in the support of $f(x)$ is odd. We will say more on this in the section about the algebraic degree.

Remark 4.14. There exist other representations of Boolean functions, for example the Disjunctive Normal Form DNF. Each atomic function corresponds to a conjunctive clause. For instance the atomic formula $f_{010}(x) = (1\oplus x_1)x_2(1\oplus x_3)$ corresponds to $\phi_{010}(x) = (\neg x_1 \wedge x_2 \wedge \neg x_3)$. The ANF equals the sum of the atomic functions f_u and the DNF equals the disjunction of the corresponding conjunctive clauses ϕ_u, i.e. $f(x) \simeq \phi(x) = \bigvee_{u\in \mathrm{supp}(f)} \phi_u(x)$.

Let f be a Boolean function and $\bigoplus_{I\in\mathcal{P}(N)} a_I x^I$ its ANF. To compute the value $f(x)$ for an $x \in \mathbb{F}_2^n$, we have that all monomials x^I with I outside the support of x will be equal to zero, i.e. $\forall I \in \mathcal{P}(N)$ such that $I \not\subseteq \mathrm{supp}(x)$ we have $x^I = 0$. Therefore we can compute $f(x)$ by taking the sum of all the a_I such that I is in the support of x, that is

$$f(x) = \bigoplus_{I\subseteq \mathrm{supp}(x)} a_I. \tag{3}$$

Conversely we have the following lemma.

Lemma 4.15. *Let f be a Boolean function on $\mathbb{F}_2^n$ and $f(x) = \bigoplus_{I\in\mathcal{P}(N)} a_I x^I$ the ANF. For all $I \in \mathcal{P}(N)$ we have*

$$a_I = \bigoplus_{x\in\mathbb{F}_2^n,\ supp(x)\subseteq I} f(x)$$

Proof. We write $b_I = \bigoplus_{x \in \mathbb{F}_2^n,\ \mathrm{supp}(x) \subseteq I} f(x)$ and get the function $g(x) = \bigoplus_{I \in \mathcal{P}(N)} b_i x^I$. With relation (3) we obtain

$$\begin{aligned} g(u) &= \bigoplus_{I \subseteq \mathrm{supp}(u)} b_I = \bigoplus_{I \subseteq \mathrm{supp}(u)} \Big(\bigoplus_{x \in \mathbb{F}_2^n,\ \mathrm{supp}(x) \subseteq I} f(x) \Big) \\ &= \bigoplus_{x \in \mathbb{F}_2^n,\ \mathrm{supp}(x) \subseteq \mathrm{supp}(u)} 2^{w_H(u) - w_H(x)} f(x). \end{aligned}$$

If $x \neq u$, this sum is 0 since it vanishes modulo 2 and if $x = u$ we get that $g(u) = f(x)$. By the uniqueness of the ANF we have for all I, $a_I = b_I$. □

With this result we obtain another way to compute the ANF from the TT of a function, simply by computing each a_I, $I \subseteq \mathcal{P}(N)$. This way, several additions are done multiple times. There exist a divide and conquer butterfly algorithm which is more efficient. It is called the Fast Möbius transform and has a computational complexity of $n2^n$ XORs, instead of the 2^{2n} XORs of a brute force computation. The idea of the algorithm is to use the following property. We define the function $f_i(x_1, \ldots, x_{n-1}) = f(x_1, \ldots, x_{n-1}, i)$ with $i \in \mathbb{F}_2$, then for every $u = (u_1, \ldots, u_n) \in \mathbb{F}_2^n$ we have that $a_{\mathrm{supp}(u)}$ equals

$$\begin{cases} \displaystyle\bigoplus_{\mathrm{supp}(x_1,\ldots,x_{n-1}) \subseteq \mathrm{supp}(u_1,\ldots,u_{n-1})} f_0(x_1, \ldots, x_{n-1}) & \text{if } u_n = 0, \\ \displaystyle\bigoplus_{\mathrm{supp}(x_1,\ldots,x_{n-1}) \subseteq \mathrm{supp}(u_1,\ldots,u_{n-1})} f_0(x_1, \ldots, x_{n-1}) \oplus f_1(x_1, \ldots, x_{n-1}) & \text{if } u_n = 1. \end{cases}$$

The algorithm works almost similar as the Fast Fourier Transform algorithm which will be explained in detail in section 4.2. For details see Carlet [5].

4.1.3 The algebraic degree

An important property of a Boolean function f is the highest degree of a monomial in the ANF of f.

Definition 4.16. Let f be a Boolean function on $\mathbb{F}_2^n$ and $f(x) = \bigoplus_{I \in \mathcal{P}(N)} a_I x^I$ its ANF. The algebraic degree of f is defined as the maximum number $|I|$ such that a_I is non zero, i.e.

$$d^\circ(f) := \max_{I \in \mathcal{P}(N)} \{|I| : a_I \neq 0\}.$$

Example 4.17. We give some examples.

1. The function $f(x) = 1 \oplus x_1 \oplus x_4 \oplus x_1x_2 \oplus x_1x_2x_3 \oplus x_1x_3x_4$ has algebraic degree $d^\circ(f) = 3$.

2. All constant functions have degree 0.

3. If f is an affine function, then $d^\circ(f) \leq 1$.

We derive some properties of the algebraic degree. For the first result we use lemma 4.15.

Proposition 4.18. *Let f be a Boolean function on $\mathbb{F}_2^n$. The degree $d^\circ(f)$ equals the maximum dimension of the subspace $\{x \in \mathbb{F}_2^n |$ supp$(x) \subseteq I\}$ on which f takes the value 1 an odd number of times.*

Proof. Let f be a Boolean function on $\mathbb{F}_2^n$ of degree d and $\bigoplus_{I \in \mathcal{P}(N)} a_I x^I$ its ANF. From lemma 4.15 we know that $a_I = \bigoplus_{x \in \mathbb{F}_2^n, \text{supp}(x) \subseteq I} f(x)$. It follows that $a_I = 0$ if f takes the value 1 an even number of times on the subspace $\{x \in \mathbb{F}_2^n : \text{supp}(x) \subseteq I\}$. By definition we have $d^\circ(f) = \max_{I \in \mathcal{P}(N)} \{|I| : a_I \neq 0\}$. So the algebraic degree is equal to the maximal dimension of the subspace $\{x \in \mathbb{F}_2^n | \text{ supp}(x) \subseteq I\}$ on which f takes the value 1 an odd number of times. □

From this proposition we deduce a relation between the weight and the algebraic degree of a Boolean function.

Corollary 4.19. *Let f be a Boolean function on $\mathbb{F}_2^n$. The function f has maximal degree if and only if the weight of f is odd, i.e.*

$$d^\circ(f) = n \quad \Leftrightarrow \quad w_H(f) \text{ is odd}$$

Proof. This follows from proposition 4.18 for the subspace $I = \mathbb{F}_2^n$. It also follows from remark 4.13. □

Proposition 4.20. *Let f be a Boolean function on $\mathbb{F}_2^n$. If the algebraic degree of f is at most d, then the weight of f is at least 2^{n-d}, i.e.*

$$d^\circ(f) \leq d \quad \Rightarrow \quad w_H(f) \geq 2^{n-d}.$$

Proof. Let $f \in \mathcal{BF}_n$, $d^\circ(f) = d$ and let x^I be a monomial in the ANF of f such that $|I| = d$. Consider all 2^{n-d} restrictions of f which keep all coordinates outside I fixed. We write f^J for such a restriction, where $J \subseteq N \backslash I$. Each restriction f^J can be viewed as a function on $\mathbb{F}_2^d$ and is of algebraic degree d since it contains the monomial x^I. Note that by construction of f^J all other monomials are of degree strict less then d.
Viewing f^J as a function on $\mathbb{F}_2^d$ of maximal algebraic degree it follows from corollary 4.19 that f^J has odd weight. In particular we have $w_H(f^J) \geq 1$. Since the weight of f is equal to the sum of the weights of its restrictions, $w_H(f) = \sum_{J \subseteq N \backslash I} w_H(f^J)$ and since there are 2^{n-d} restrictions f^J of f. It follows that $w_H(f) \geq 2^{n-d}$ as desired. □

Corollary 4.21. *Let f, g be two distinct Boolean functions on $\mathbb{F}_2^n$. If the algebraic degree of f and g is at most d, then their Hamming distance is at least 2^{n-d}, i.e.*

$$d^\circ(f), d^\circ(g) \leq d \quad \Rightarrow \quad d_H(f, g) \geq 2^{n-d}.$$

Proof. Observe that for two distinct Boolean functions $f, g \in \mathcal{BF}_n$ such that $d^\circ(f) \leq d$ and $d^\circ(g) \leq d$, we have that $d^\circ(f \oplus g) \leq d$. Further we have that $d_H(f, g) = w_H(f \oplus g)$. Now apply proposition 4.20 to the function $f \oplus g$. □

Definition 4.22. Let f be a Boolean function on $\mathbb{F}_2^n$. The derivative in the direction of $a \in \mathbb{F}_2^n$ is defined as

$$D_a f(x) = f(x) \oplus f(x \oplus a).$$

The derivative has two simple non surprising properties.

Lemma 4.23. *Let $f, g \in \mathcal{BF}_n$, $e \in \mathbb{F}_2^n$ and $r \leq n$, then the following holds;*

i. $D_e(f \oplus g) = D_e f \oplus D_e g$.

ii. If $e \neq 0$, then $d^\circ(f) = r \;\Rightarrow\; d^\circ(D_e f) \leq r - 1$.

Proof. To prove i we just write out the definition.

$$\begin{aligned} D_e(f \oplus g)(x) &= (f \oplus g)(x) \oplus (f \oplus g)(x \oplus e) \\ &= f(x) \oplus g(x) \oplus f(x \oplus e) \oplus g(x \oplus e) \\ &= D_e f(x) \oplus D_e g(x). \end{aligned}$$

To prove ii write f in its ANF, $\bigoplus_{I\in\mathcal{P}(N)} a_I x^I$. Since f is the sum of monomials we have by i that it is sufficient to prove the statement for a single monomial. Let x^I be a monomial of degree r and $e = (e_1, \ldots, e_n) \in \mathbb{F}_2^n$, then $D_e x^I = x^I \oplus \prod_{i\in I}(x_i \oplus e_i)$. Where the product $\prod_{i\in I}(x_i \oplus e_i)$ is of the form $\prod_{i\in I}(x_i \oplus e_i) = x^I \oplus g(x)$, for some $g \in \mathcal{BF}_n$ with $d^\circ(g) < r$, hence

$$d^\circ(D_e x^I) = d^\circ(x^I \oplus \prod_{i\in I}(x_i \oplus e_i)) = d^\circ(g(x)) \leq r - 1.$$

□

4.2 Fourier and Walsh Transform

A number of properties of Boolean functions can be shown using the discrete Fourier transform which we will refer to as just the Fourier transform. We define the Fourier transform for pseudo-Boolean functions. A pseudo-Boolean function φ is a real valued function on $\mathbb{F}_2^n$, i.e. $\varphi : \mathbb{F}_2^n \to \mathbb{R}$.

Definition 4.24. Let φ be a pseudo-Boolean function and $u \in \mathbb{F}_2^n$. We define the Fourier transform at u as

$$\widehat{\varphi}(u) = \sum_{x\in\mathbb{F}_2^n} \varphi(x)(-1)^{u\cdot x}.$$

Where $u \cdot x$ is the inner product.

Observe that the Fourier transform itself is again a pseudo-Boolean function. From this definition we deduce two easy properties: $\widehat{\varphi + \psi} = \widehat{\varphi} + \widehat{\psi}$ and $\widehat{r \cdot \varphi} = r \cdot \widehat{\varphi}$ for $r \in \mathbb{R}$. An important pseudo-Boolean function is the sign function of a Boolean function f, $f_\chi(x) = (-1)^{f(x)}$. This function has instead of the outcome $0, 1$ the outcome $1, -1$ respectively. The Fourier transform of the sign function is called the Walsh transform[7].

Definition 4.25. Let f be a Boolean function on $\mathbb{F}_2^n$ and f_χ the sign function of f. The Fourier transform of f_χ is called the Walsh transform.

The Walsh transform at $u \in \mathbb{F}_2^n$ equals;

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

[7]In some literature this is called the Walsh-Hadamard transform.

The set of all values of the Fourier transform, $\widehat{\varphi}(a)$ for $a \in \mathbb{F}_2^n$, is called the Fourier spectrum of φ. Analogue we have the Walsh spectrum.

To compute the Fourier of Walsh spectrum of a function we can use the elegant Fast Fourier Transform FFT algorithm. The main idea of the algorithm is the following relation.

Let φ be a pseudo-Boolean function on $\mathbb{F}_2^n$, we write $a' = (a_1, \ldots, a_{n-1})$, $x' = (x_1, \ldots, x_{n-1})$ and deduce

$$\widehat{\varphi}(a_1, \ldots, a_n) = \sum_{(x_1,\ldots,x_n)\in\mathbb{F}_2^n} \varphi(x)(-1)^{(a_1,\ldots,a_n)\cdot(x_1,\ldots,x_n)}$$

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

When we make a table of the values of $\varphi(x)$ with the bits of $x \in \mathbb{F}_2^n$ in lexicographic order and the bit of higher order on the right. (Note that if φ is a Boolean function then this is just a truth table.) We can divide the table in an upper and a lower half. Together with the relation above we get the following. Define the two $(n-1)$ variable pseudo-Boolean functions

$$\begin{aligned} \varphi_0(x_1, \ldots, x_{n-1}) &= \varphi(x_1, \ldots, x_{n-1}, 0) \\ \varphi_1(x_1, \ldots, x_{n-1}) &= \varphi(x_1, \ldots, x_{n-1}, 1). \end{aligned}$$

To compute the values of $\widehat{\varphi}(a)$ for the upper half of the table we need to compute the Fourier transform at $(a_1, \ldots, a_{n-1})$ of $\varphi_0 + \varphi_1$. For the lower half we need to compute the Fourier transform at $(a_1, \ldots, a_{n-1})$ of $\varphi_0 - \varphi_1$. Observe that we can easily compute the values of $\varphi_0 + \varphi_1$ and $\varphi_0 - \varphi_1$ since we already have a table of values of $\varphi(x)$.

Algorithm The FFT algorithm is as follows:
Input: $\varphi(x_1, \ldots, x_n)$ a pseudo-Boolean function on $\mathbb{F}_2^n$.
Output: $\widehat{\varphi}(a)$ for every $a \in \mathbb{F}_2^n$.

1. Make a table of values φ with the bits in lexicographic order and the bit of higher order is on the right.

2. Replace every value of $\varphi(x_1, \ldots, x_n)$ in the table as follows:
   If $x_n = 0$, replace $\varphi(x_1, \ldots, x_n)$ by $\varphi_0(x_1, \ldots, x_{n-1}) + \varphi_1(x_1, \ldots, x_{n-1})$.
   If $x_n = 1$, replace $\varphi(x_1, \ldots, x_n)$ by $\varphi_0(x_1, \ldots, x_{n-1}) - \varphi_1(x_1, \ldots, x_{n-1})$.

3. Repeat step 2 on the new functions $\varphi_0 + \varphi_1$ and $\varphi_0 - \varphi_1$.

x_1	x_2	x_3	$f(x)$	$f_\chi(x)$	I	II	$\widehat{f_\chi}(x)$
0	0	0	1	-1	-2	-4	-2
1	0	0	0	1	2	2	-6
0	1	0	1	-1	-2	0	2
1	1	0	1	-1	0	2	-2
0	0	1	1	-1	0	0	-2
1	0	1	0	1	0	-2	2
0	1	1	1	-1	0	0	2
1	1	1	0	1	-2	2	-2

Table 2: Table of values of a FFT algorithm applied on the function $f(x) = 1 \oplus x_1 \oplus x_1x_2 \oplus x_1x_2x_3$.

The algorithm ends when there are no variables left. The complexity of the FFT algorithm on a Boolean function is $n2^n$ XORs. Which is much better then the complexity of a brute force computation 2^{2n} XORs. The command "replace" in the algorithm can, of course, be satisfied by adding a extra column to the table consisting of the new obtained values. This gives a more constructive view of the FFT algorithm. We will show this in an example.

Example 4.26. We compute the Walsh transform $\widehat{f_\chi}$ of the Boolean function $f(x) = 1 \oplus x_1 \oplus x_1x_2 \oplus x_1x_2x_3$ from table 1. The results of the FFT algorithm can be seen in table 2. On top of each column is stated what is computed in that particular column. In column I the table is divided into an upper and a lower half. For the upper half the values of $f_\chi(x_1, x_2, 0) + f_\chi(x_1, x_2, 1)$ are computed and for the lower half the values of $f_\chi(x_1, x_2, 0) - f_\chi(x_1, x_2, 1)$ are computed. Note that the two values of each addition or subtraction can be found in the previous column (That is the strength and the elegance of this algorithm!). So the first value is computed by $f_\chi(0, 0, 0) + f_\chi(0, 0, 1) = -1 + (-1) = -2$, the second value by $f_\chi(1, 0, 0) + f_\chi(1, 0, 1) = 1 + 1 = 2$ and so on. The last value of the column is computed by $f_\chi(1, 1, 0) - f_\chi(1, 1, 1) = -1 - 1 = -2$. This method is repeated in the next column but then the upper and lower half are seen as separate tables. We apply step 2 of the algorithm on the two functions;

$$\begin{aligned} g_{\chi 0}(x_1, x_2) &= f_\chi(x_1, x_2, 0) + f_\chi(x_1, x_2, 1) \\ g_{\chi 1}(x_1, x_2) &= f_\chi(x_1, x_2, 0) - f_\chi(x_1, x_2, 1). \end{aligned}$$

We explain the computation of the upper half, that is the upper half of column II. The first two entries are computed by $g_{\chi 0}(x_1, 0) + g_{\chi 0}(x_1, 1)$. Again note that the values of $g_{\chi 0}(x_1, 0)$ and $g_{\chi 0}(x_1, 1)$ can be found in column I. We compute $-2 + (-2) = -4$ if $x_1 = 0$ and $2 + 0 = 2$ if $x_1 = 1$. In the same way we compute the second two entry's by $g_{\chi 0}(x_1, 0) - g_{\chi 0}(x_1, 1)$, that is $-2 - (-2) = 0$ and $2 - 0 = 2$. For the last column the values of $\widehat{f_\chi}(x)$ are computed by applying the algorithm on the following functions;

$$\begin{aligned} h_{\chi 00}(x_1) &= g_{\chi 0}(x_1, 0) + g_{\chi 0}(x_1, 1) \\ h_{\chi 01}(x_1) &= g_{\chi 0}(x_1, 0) - g_{\chi 0}(x_1, 1) \\ h_{\chi 10}(x_1) &= g_{\chi 1}(x_1, 0) + g_{\chi 1}(x_1, 1) \\ h_{\chi 11}(x_1) &= g_{\chi 1}(x_1, 0) - g_{\chi 1}(x_1, 1) \end{aligned}$$

For instance the value of $\widehat{f_\chi}(0, 0, 0)$ is now equal to $h_{\chi 00}(0) + h_{\chi 00}(1)$ and the values of $h_{\chi 00}(0)$ and $h_{\chi 00}(1)$ are the first two values of the previous column. We obtain $\widehat{f_\chi}(0, 0, 0) = -4 + 2 = -2$.

4.2.1 Properties of Fourier transform

In order to use the Fourier transform for describing more properties of Boolean functions we first need to derive some general results. We start with a lemma and two propositions for pseudo-Boolean functions. The lemma is a simple observation from lemma 4.8.

Lemma 4.27. *Let $E \subseteq \mathbb{F}_2^n$ be a linear subspace and let l be a nonzero linear Boolean function on E. Then we have*

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

Proof. Lemma 4.8 can be extended to the vector space $E \subseteq \mathbb{F}_2^n$. A nonzero linear Boolean function $l(x)$ on E is balanced, that is $|\{x \in E : l(x) = 0\}| = |\{x \in E : l(x) = 1\}|$. It follows that $\sum_{x \in E} (-1)^{l(x)} = 0$. □

Observe that this also holds for l a non constant affine function, since for $l \oplus 1$ we have $\sum_{x \in E} (-1)^{l(x) \oplus 1} = -\sum_{x \in E} (-1)^{l(x)} = 0$.

The first proposition is more technical. Recall that, when $n > 1$, we use $+$ for the addition of two vectors a, b in the vector space $\mathbb{F}_2^n$.

Proposition 4.28. *Let φ be a pseudo-Boolean function on $\mathbb{F}_2^n$ and $a, b, u \in \mathbb{F}_2^n$. For the pseudo-Boolean function $\psi(x) = (-1)^{a\cdot x}\varphi(x+b)$ we have that*

$$\widehat{\psi}(u) = (-1)^{b\cdot(a+u)}\widehat{\varphi}(a+u).$$

Proof. We deduce,

$$\begin{aligned}
\widehat{\psi}(u) &= \sum_{x\in\mathbb{F}_2^n} \psi(x)(-1)^{u\cdot x} = \sum_{x\in\mathbb{F}_2^n} \varphi(x+b)(-1)^{a\cdot x\oplus u\cdot x} \\
&= \sum_{x\in\mathbb{F}_2^n} \varphi(x)(-1)^{(a+u)\cdot(x+b)} = \sum_{x\in\mathbb{F}_2^n} \varphi(x)(-1)^{(a+u)\cdot x\oplus(a+u)\cdot b} \\
&= (-1)^{b\cdot(a+u)} \sum_{x\in\mathbb{F}_2^n} \varphi(x)(-1)^{(a+u)\cdot x} = (-1)^{b\cdot(a+u)}\widehat{\varphi}(a+u).
\end{aligned}$$

□

For the second proposition we first define some simple notions. Let $E \subseteq \mathbb{F}_2^n$ be some subset, then 1_E is the indicator function of the set E, i.e.

$$1_E(x) := \begin{cases} 1 & \text{if } x \in E \\ 0 & \text{if } x \notin E \end{cases}$$

Further we write $E^\perp$ for the set of orthogonal elements in $\mathbb{F}_2^n$, i.e. $E^\perp = \{x \in \mathbb{F}_2^n : \forall y \in E,\ x\cdot y = 0\}$. Note that, similar to the dual codes in definition 3.12, we have that $\{\mathbf{0}\} \subseteq E \cap E^\perp$ and this relation can be strict.

Proposition 4.29. *Let $E \subseteq \mathbb{F}_2^n$ be a linear subspace and let 1_E be the indicator function of E, then*

$$\widehat{1_E} = |E|1_{E^\perp}.$$

Proof. For $u \in \mathbb{F}_2^n$ we have that,

$$\widehat{1_E}(u) = \sum_{x\in\mathbb{F}_2^n} 1_E(x)(-1)^{u\cdot x} = \sum_{x\in E}(-1)^{u\cdot x}$$

Since $u\cdot x$ is a linear function we can use lemma 4.27 and deduce

$$\sum_{x\in E}(-1)^{u\cdot x} = \begin{cases} 0 & \text{if } u\cdot x \text{ is a nonzero linear function, i.e.} \quad u \notin E^\perp. \\ |E| & \text{if } u\cdot x \text{ is the zero function, i.e.} \qquad\quad u \in E^\perp. \end{cases}$$

This proves that $\widehat{1_E}(u) = |E|1_{E^\perp}(u)$ as desired. □

In particular, if $E = \mathbb{F}_2^n$ we have that $\widehat{1_{\mathbb{F}_2^n}} = |\mathbb{F}_2^n| 1_{\{\mathbf{0}\}} = 2^n \delta_0$, where δ_0 is the Dirac symbol which is defined by

$$\delta_0(u) := \begin{cases} 1 & \text{if } u \text{ is the null vector,} \\ 0 & \text{otherwise.} \end{cases}$$

With these two propositions we will show more properties of pseudo-Boolean functions using the Fourier transform. We start by deducing the *Poisson summation formula* in its general form.

Theorem 4.30. *Let φ be a pseudo-Boolean function on $\mathbb{F}_2^n$ and $E \subseteq \mathbb{F}_2^n$ a linear subspace. We have for all $a, b \in \mathbb{F}_2^n$ that*

$$\sum_{u \in a+E} (-1)^{b \cdot u} \widehat{\varphi}(u) = |E|(-1)^{a \cdot b} \sum_{x \in b+E^\perp} (-1)^{a \cdot x} \varphi(x).$$

We denote $a+E$ for the subspace $\{x \in \mathbb{F}_2^n : \exists e \in E \text{ such that } a+e = x\}$. These subspaces of $\mathbb{F}_2^n$ are affine subspaces and are also called *flats*.

Example 4.31. Let $E \subseteq \mathbb{F}_2^3$ be the vector space spanned by $(x_1, x_2, 0)$ and $a = (0, 1, 1)$, then $a+E$ is the set of all vectors in $\mathbb{F}_2^3$ of the form $(x_1, 1 \oplus x_2, 1)$, or just $(x_1, x_2, 1)$.

Note that for all $a \in \mathbb{F}_2^n$ we have $\dim E = \dim(a + E)$. We give a proof of the theorem.

Proof. We first prove case I were $a = b = \mathbf{0}$, that is

$$\sum_{u \in E} \widehat{\varphi}(u) = |E| \sum_{x \in E^\perp} \varphi(x).$$

We deduce

$$\sum_{u \in E} \widehat{\varphi}(u) = \sum_{u \in E} \sum_{x \in \mathbb{F}_2^n} \varphi(x)(-1)^{u \cdot x} = \sum_{x \in \mathbb{F}_2^n} \varphi(x) \sum_{u \in E} (-1)^{u \cdot x} = \sum_{x \in \mathbb{F}_2^n} \varphi(x) \widehat{1_E}.$$

By proposition 4.29, it follows that

$$\sum_{x \in \mathbb{F}_2^n} \varphi(x) \widehat{1_E} = \sum_{x \in \mathbb{F}_2^n} \varphi(x) |E| 1_{E^\perp} = |E| \sum_{x \in E^\perp} \varphi(x).$$

To prove the general Poisson summation formula we apply proposition 4.28 to the case I. We take the pseudo-Boolean function $\psi(x) = (-1)^{a \cdot x} \varphi(x+$

b), by proposition 4.28 we have that $\widehat{\psi}(u) = (-1)^{b\cdot(a+u)}\widehat{\varphi}(a+u)$. Substituting this into case I, we get

$$\begin{aligned}
\sum_{u\in E}\widehat{\psi}(u) &= |E|\sum_{x\in E^\perp}\psi(x)\\
\sum_{u\in E}(-1)^{b\cdot(a+u)}\widehat{\varphi}(a+u) &= |E|\sum_{x\in E^\perp}(-1)^{a\cdot x}\varphi(x+b)\\
\sum_{u\in a+E}(-1)^{b\cdot u}\widehat{\varphi}(u) &= |E|\sum_{x\in b+E^\perp}(-1)^{a\cdot(x+b)}\varphi(x)\\
\sum_{u\in a+E}(-1)^{b\cdot u}\widehat{\varphi}(u) &= |E|(-1)^{a\cdot b}\sum_{x\in b+E^\perp}(-1)^{a\cdot x}\varphi(x).
\end{aligned}$$

This proves the Poisson summation formula. □

We state a simple consequence of the Poisson summation formula.

Corollary 4.32. *For all pseudo-Boolean functions on $\mathbb{F}_2^n$ we have*

$$\widehat{\widehat{\varphi}} = 2^n\varphi.$$

Proof. Let φ be a pseudo-Boolean function. We use the Poisson summation formula for $a = \mathbf{0}$ and $E = \mathbb{F}_2^n$ and obtain

$$\sum_{u\in\mathbb{F}_2^n}(-1)^{b\cdot u}\widehat{\varphi}(u) = 2^n\sum_{x\in b+\{\mathbf{0}\}}\varphi(x).$$

Which is equivalent to

$$\widehat{\widehat{\varphi}}(b) = 2^n\varphi(b).$$

□

We see that the Fourier transform is its own inverse up to division of a constant. Together with proposition 4.29 we deduce another corollary.

Corollary 4.33. *Let φ be a pseudo-Boolean function on $\mathbb{F}_2^n$. We have that φ is constant if and only if the Fourier transform $\widehat{\varphi}(a)$ is zero for all $a \in \mathbb{F}_2^n\backslash\{\boldsymbol{0}\}$.*

Proof. To prove corollary 4.33 we first assume that φ is constant. We have that φ is of the form $\varphi = r \times 1_{\mathbb{F}_2^n}$ for some value $r \in \mathbb{R}$. From proposition 4.29 it follows that

$$\widehat{\varphi} = \widehat{r \times 1_{\mathbb{F}_2^n}} = r \times \widehat{1_{\mathbb{F}_2^n}} = r|\mathbb{F}_2^n|1_{\{\mathbf{0}\}}.$$

This proves that $\widehat{\varphi}(a) = 0$ for all $a \in \mathbb{F}_2^n \backslash \{\mathbf{0}\}$.
For the other direction we assume that $\widehat{\varphi}(a) = 0$ for all $a \in \mathbb{F}_2^n \backslash \{\mathbf{0}\}$. We get that

$$\widehat{\widehat{\varphi}}(a) = \sum_{y \in \mathbb{F}_2^n} \widehat{\varphi}(y)(-1)^{a \cdot y} = \widehat{\varphi}(\mathbf{0}) = \sum_{x \in \mathbb{F}_2^n} \varphi(x).$$

From corollary 4.32 we know that $\widehat{\widehat{\varphi}}(a) = 2^n \varphi(a)$. Combining this we have that $\sum_{x \in \mathbb{F}_2^n} \varphi(x) = 2^n \varphi(a)$ for all $a \in \mathbb{F}_2^n$. Hence φ must be constant. □

The roles of φ and $\widehat{\varphi}$ in corollary 4.33 can be interchanged. Since $\widehat{\varphi}$ is again a pseudo-Boolean function, we can substitute φ for $\widehat{\varphi}$. Together with corollary 4.32 it follows that $\widehat{\widehat{\varphi}}(a) = 2^n \varphi(a) = 0$, so $\varphi(a) = 0$ for all $a \in \mathbb{F}_2^n \backslash \{\mathbf{0}\}$.

Another important result is Parseval's relation. To derive this relation we need the convolutional product.

Definition 4.34. The convolutional product $\otimes$ between two pseudo-Boolean functions φ, ψ on $\mathbb{F}_2^n$ is defined as

$$(\varphi \otimes \psi)(x) = \sum_{y \in \mathbb{F}_2^n} \varphi(y)\psi(x + y).$$

There exists a nice relation between the convolutional product and the normal multiplication, denoted as $\times$, of the Fourier transform of two pseudo-Boolean functions.

Lemma 4.35. *Let φ and ψ be two pseudo-Boolean functions on $\mathbb{F}_2^n$, then*

i. $\widehat{\varphi \otimes \psi} = \widehat{\varphi} \times \widehat{\psi}$,

ii. $\widehat{\varphi} \otimes \widehat{\psi} = 2^n \widehat{\varphi \times \psi}$.

Proof. To prove i we first observe that $u \cdot x = u \cdot y \oplus u \cdot (x+y)$ in the vector space $\mathbb{F}_2^n$. Then it is just a matter of writing out the definitions;

$$\begin{aligned}
\widehat{\varphi \otimes \psi}(u) &= \sum_{x \in \mathbb{F}_2^n} (\varphi \otimes \psi)(x)(-1)^{u \cdot x} \\
&= \sum_{x \in \mathbb{F}_2^n} \sum_{y \in \mathbb{F}_2^n} \varphi(y)\psi(x+y)(-1)^{u \cdot y \oplus u \cdot (x+y)} \\
&= \sum_{y \in \mathbb{F}_2^n} \varphi(y)(-1)^{u \cdot y} \left(\sum_{x \in \mathbb{F}_2^n} \psi(x+y)(-1)^{u \cdot (x+y)} \right) \\
&= \left(\sum_{y \in \mathbb{F}_2^n} \varphi(y)(-1)^{u \cdot y} \right) \left(\sum_{z \in \mathbb{F}_2^n} \psi(z)(-1)^{u \cdot z} \right) = \widehat{\varphi}(u)\widehat{\psi}(u).
\end{aligned}$$

To prove ii we apply i on the two functions, $\widehat{\varphi}$ and $\widehat{\psi}$ and use two times corollary 4.32. We get,

$$\widehat{\widehat{\varphi} \otimes \widehat{\psi}} = \widehat{\widehat{\varphi}} \times \widehat{\widehat{\psi}} = 2^{2n}(\varphi \times \psi) = 2^n \widehat{\widehat{\varphi \times \psi}}.$$

From which it follows that $\widehat{\varphi} \otimes \widehat{\psi} = 2^n \widehat{\varphi \times \psi}$. □

With these results it is an easy task to deduce *Parseval's relation.*

Theorem 4.36. *Let φ be a pseudo-Boolean function, then*

$$\sum_{u \in \mathbb{F}_2^n} \widehat{\varphi}^2(u) = 2^n \sum_{x \in \mathbb{F}_2^n} \varphi^2(x).$$

Proof. We take $\varphi = \psi$ and evaluating the relation of lemma 4.35ii in zero, we get

$$(\widehat{\varphi} \otimes \widehat{\varphi})(\mathbf{0}) = 2^n \widehat{\varphi \times \varphi}(\mathbf{0}) = 2^n \sum_{x \in \mathbb{F}_2^n} \varphi^2(x).$$

By definition of the convolutional product we also have that

$$(\widehat{\varphi} \otimes \widehat{\varphi})(\mathbf{0}) = \sum_{u \in \mathbb{F}_2^n} \widehat{\varphi}^2(u).$$

This proves Parseval's relation. □

In the next paragraph we will apply the results above on Boolean functions and the Walsh transform. Here we give already two consequences of Parseval's relation. First we apply Parseval's relation on the sign function, this gives a simple but strong result.

Corollary 4.37. *Let f be a Boolean function on $\mathbb{F}_2^n$ and f_χ its sign function, then*

$$\sum_{u\in\mathbb{F}_2^n} \widehat{f_\chi}^{\,2}(u) = 2^{2n}.$$

Proof. We apply theorem 4.36 on f_χ. The corollary follows immediate since the values of f_χ are ± 1. □

Second, we apply Parseval's relation on a Boolean function. It follows that the weight of f equals the mean of $\widehat{f^2}(u)$ of all $u \in \mathbb{F}_2^n$.

Corollary 4.38. *Let f be a Boolean function on $\mathbb{F}_2^n$, then*

$$\frac{1}{2^n}\sum_{u\in\mathbb{F}_2^n} \widehat{f^2}(u) = w_H(f).$$

Proof. For every $f \in \mathcal{BF}_n$ we have that $f^2 = f$, we deduce

$$\frac{1}{2^n}\sum_{u\in\mathbb{F}_2^n} \widehat{f^2}(u) = \sum_{x\in\mathbb{F}_2^n} f^2(x) = \sum_{x\in\mathbb{F}_2^n} f(x) = w_H(f).$$

□

4.2.2 Properties of the Walsh transform and nonlinearity

The results of the previous paragraph are useful when we apply them to Boolean functions or the sign function, i.e. Walsh transform. This way we can derive important properties of Boolean functions.

The Walsh transform of a Boolean function evaluated in $\mathbf{0}$ is given its own notation.

Definition 4.39. Let f be a Boolean function on $\mathbb{F}_2^n$, the map $f \mapsto \widehat{f_\chi}(0)$ is denoted with $\mathcal{F}$, i.e.

$$\mathcal{F}(f) := \widehat{f_\chi}(\mathbf{0}) = \sum_{x\in\mathbb{F}_2^n} (-1)^{f(x)}.$$

Recall definition 4.22 of the derivative of a Boolean function $D_a f(x) = f(x) + f(x+a)$, $a \in \mathbb{F}_2^n$. We will show the relation between the Walsh transform and the derivative using the *Wiener Khintchine Theorem* which we state as a lemma.

Lemma 4.40. *Let f be a Boolean function on $\mathbb{F}_2^n$ and f_χ its sign function, then*

$$\widehat{f_\chi \otimes f_\chi} = \widehat{f_\chi}^2.$$

Proof. Let $f \in \mathcal{BF}_n$ and apply lemma 4.35i to the sign function f_χ. □

We deduce the following relation.

Corollary 4.41. *Let f be a Boolean function on $\mathbb{F}_2^n$, f_χ its sign function and $u \in \mathbb{F}_2^n$, then*

$$\widehat{f_\chi}^2(u) = \sum_{b \in \mathbb{F}_2^n} \mathcal{F}(D_b f)(-1)^{u \cdot b}.$$

Proof. Let $b \in \mathbb{F}_2^n$, $f \in \mathcal{BF}_n$ and f_χ the sign function. By definition of the convolutional product we have that

$$(f_\chi \otimes f_\chi)(b) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)} (-1)^{f(x+b)} = \sum_{x \in \mathbb{F}_2^n} (-1)^{D_b f(x)} = \mathcal{F}(D_b f).$$

We take the Fourier transform of $f_\chi \otimes f_\chi$ evaluated in u and get

$$\widehat{f_\chi \otimes f_\chi}(u) = \sum_{b \in \mathbb{F}_2^n} \mathcal{F}(D_b f)(-1)^{u \cdot b}.$$

Applying the Wiener-Khintchine Theorem proves the desired. □

We will deduce a relation between the Walsh transform and the weight. Recall that for the weight of a function f we have that $w_H(f) = \sum_{x \in \mathbb{F}_2^n} f(x)$. First we make the following observation.

Lemma 4.42. *Let f a Boolean function on $\mathbb{F}_2^n$. Then $\widehat{f}(\boldsymbol{0})$ equals the Hamming weight of f.*

Proof. We have

$$\widehat{f}(\mathbf{0}) = \sum_{x \in \mathbb{F}_2^n} f(x)(-1)^{\mathbf{0} \cdot x} = w_H(f).$$

□

In particular, let g be another Boolean function on $\mathbb{F}_2^n$. The Hamming distance between f and g equals the Fourier transform of the sum $f \oplus g$ evaluated in $\mathbf{0}$, i.e.

$$d_H(f,g) = w_H(f \oplus g) = \widehat{f \oplus g}(\mathbf{0}).$$

We use lemma 4.42 to describe the weight of a Boolean function by its Walsh transform evaluated in zero.

Lemma 4.43. *Let f be a Boolean function on $\mathbb{F}_2^n$, then*

$$w_H(f) = 2^{n-1} - \frac{\widehat{f_\chi}(\boldsymbol{0})}{2}.$$

Proof. Observe that we can write the sign function of f as $f_\chi = (-1)^f = 1 - 2f$. We get

$$\widehat{f_\chi} = \widehat{1 - 2f} = \widehat{1} - 2\widehat{f} = 2^n\delta_0 - 2\widehat{f}.$$

The last equality is obtained from proposition 4.29 since the constant 1 equals the indicator function of $\mathbb{F}_2^n$, i.e. $1 = 1_{\mathbb{F}_2^n}$, and therefore $\widehat{1} = 2^n\delta_0$, where δ_0 is the Dirac symbol. Rewriting this gives

$$\widehat{f} = 2^{n-1}\delta_0 - \frac{\widehat{f_\chi}}{2}.$$

Together with lemma 4.42 we obtain the desired;

$$w_H(f) = f(\mathbf{0}) = 2^{n-1} - \frac{\widehat{f_\chi}(\mathbf{0})}{2}.$$

□

Nonlinearity We use lemma 4.43 to look at the distance of a Boolean function f to all affine functions, this way we can see how 'nonlinear' the function f is. The nonlinearity of a Boolean function measures the minimal distance to all affine functions.

Definition 4.44. Let f be a Boolean function on $\mathbb{F}_2^n$, the nonlinearity of f is defined as

$$\mathcal{NL}(f) := \min_{a \in \mathbb{F}_2^n,\ a_0 \in \mathbb{F}_2} |\{x \in \mathbb{F}_2^n : f(x) \neq a_0 \oplus a \cdot x\}|.$$

Recall that for $a \in \mathbb{F}_2^n$ and $a_0 \in \mathbb{F}_2$, $l(x) = a \cdot x \oplus a_0$ defines a affine function on $\mathbb{F}_2^n$. We use lemma 4.43 to prove that the nonlinearity is related to the maximum value of the Walsh transform.

Proposition 4.45. *Let f be a Boolean function on $\mathbb{F}_2^n$ and let $\widehat{f}_\chi$ its Walsh transform. Then we have,*

$$\mathcal{NL}(f) = 2^{n-1} - \frac{1}{2} \max_{a \in \mathbb{F}_2^n} |\widehat{f}_\chi(a)|.$$

Proof. Let $l(x)$ be an affine function on $\mathbb{F}_2^n$. We can write $l(x) = a \cdot x \oplus a_0$ for $a \in \mathbb{F}_2^n$ and $a_0 \in \mathbb{F}_2$. With lemma 4.43 we get

$$d_H(f, l) = w_H(f \oplus l) = 2^{n-1} - \frac{1}{2}\widehat{(f \oplus l)}_\chi(\mathbf{0}),$$

where

$$\widehat{(f \oplus l)}_\chi(\mathbf{0}) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus a \cdot x \oplus a_0} = (-1)^{a_0} \sum_{x \in \mathbb{F}_2^n} f_\chi(x)(-1)^{a \cdot x} = (-1)^{a_0} \widehat{f}_\chi(a).$$

This proves the proposition since we have that

$$\mathcal{NL}(f) = \min_{a \in \mathbb{F}_2^n,\ a_0 \in \mathbb{F}_2} d_h(f, a \cdot x \oplus a_0) = 2^{n-1} - \frac{1}{2} \max_{a \in \mathbb{F}_2^n} |\widehat{f}_\chi(a)|.$$

□

So the nonlinearity of a function is high when the amplitude of the Walsh transform $\widehat{f}_\chi(a)$ is low. From this proposition and corollary 4.37 of Parseval's relation we deduce an upper bound on the nonlinearity of a function.

Corollary 4.46. *Let f be a Boolean function on $\mathbb{F}_2^n$, then*

$$\mathcal{NL}(f) \leq 2^{n-1} - 2^{\frac{n}{2}-1}.$$

Proof. From corollary 4.37 we have that $\sum_{a \in \mathbb{F}_2^n} \widehat{f}_\chi^{\,2}(a) = 2^{2n}$. We deduce $\max_{a \in \mathbb{F}_2^n} \widehat{f}_\chi^{\,2}(a) \geq 2^n$, from which its follows that

$$\max_{a \in \mathbb{F}_2^n} |\widehat{f}_\chi(a)| \geq 2^{\frac{n}{2}}.$$

Together with proposition 4.45 this proves that $\mathcal{NL}(f) \leq 2^{n-1} - 2^{\frac{n}{2}-1}$. □

This upper bound is called the covering radius bound[8]. A Boolean function which reaches the covering radius bound is called *bent*. Note that bent functions on $\mathbb{F}_2^n$ can only exist for n is even (and they do exist for every even n). The Walsh transform at point $a \in \mathbb{F}_2^n$ of a bent function equals $2^{\frac{n}{2}}$ for every a, hence the Walsh spectrum has the same average value for every a.

Another extreme is the nonlinearity of an affine function. For such an affine function l we have that $\mathcal{NL}(l) = 0$, so there exists a value $a \in \mathbb{F}_2^n$ such that $|\widehat{l_\chi}(a)| = 2^n$. From corollary 4.37 of Parseval's relation it follows that for all other vectors $b \in \mathbb{F}_2^n$, $b \neq a$ we have $\widehat{f_\chi}(b) = 0$. In other words, the Walsh spectrum has one high peak and is low for all other vectors. More generally, functions with low nonlinearity have low values for most points but have high peaks at some points. While functions with high nonlinearity have only values near the average $2^{\frac{n}{2}}$ and no real peaks.

4.3 Linear and Affine Isomorphisms

Recall the definition of the ANF of a Boolean function on $\mathbb{F}_2^n$,

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

A permutation on $N = \{1, \ldots, n\}$ gives another representation and by uniqueness of the ANF, a different Boolean function. But how different are these two functions?

Let L be an affine isomorphism of dimension n, that is $L(x) = Mx + a$, where M is a non-singular $n \times n$ matrix and a is a vector in $\mathbb{F}_2^n$. It follows that L^{-1} is also an affine isomorphism with $L^{-1}(x) = M^{-1}x + b$ and $b = M^{-1}a$.

Definition 4.47. Two Boolean functions f, g on $\mathbb{F}_2^n$ are called affine equivalent if there exist an affine isomorphism $L : \mathbb{F}_2^n \to \mathbb{F}_2^n$, $L(x) = Mx + a$ as above, such that $f \circ L = g$.
If the vector $a = \mathbf{0}$ then f and g are called linear equivalent.

Since L^{-1} is again an affine isomorphism it is easy to show that this defines an equivalent relation. All non constant affine functions are affine equivalent and form a equivalence class.

[8]The name, covering radius bound, comes from coding theory. This bound equals the covering radius bound for Reed-Muller codes of order 1. In this code every codeword corresponds to a Boolean function of order 1, see van Lint [20].

We are interested in which properties of Boolean functions are invariant under affine isomorphisms and therefore hold for a whole equivalence class. We take a closer look at the following notions; the algebraic degree, the Hamming weight, the nonlinearity and the Fourier transform.

Lemma 4.48. *The algebraic degree is an affine invariant. Let f be a Boolean function on $\mathbb{F}_2^n$ and L an affine isomorphism on $\mathbb{F}_2^n$, then*

$$d^\circ(f \circ L) = d^\circ(f).$$

Proof. Let L be an affine isomorphism, $f \in \mathcal{BF}_n$ and x^I be a monomial of the ANF of f. For each monomial x^I we have that $d^\circ(x^I \circ L) \leq d^\circ(x^I)$ and therefore we have $d^\circ(f \circ L) \leq d^\circ(f)$. Since the inverse L^{-1} is also an affine isomorphism we can apply the above to the function $f \circ L^{-1}$. We get $d^\circ(f \circ L^{-1} \circ L) = d^\circ(f) \leq d^\circ(f \circ L^{-1})$ and it follows that $d^\circ(f \circ L) = d^\circ(f)$. This shows that the algebraic degree is an affine invariant. □

Lemma 4.49. *The Hamming weight is an affine invariant. Let f be a Boolean function on $\mathbb{F}_2^n$ and L an affine isomorphism on $\mathbb{F}_2^n$, then*

$$w_H(f \circ L) = w_H(f).$$

Proof. Let $f \in \mathcal{BF}_n$ and L an affine isomorphism, then L^{-1} is also an affine isomorphism and $w_H(f \circ L) = \sum_{x \in \mathbb{F}_2^n} f(L(x)) = \sum_{L^{-1}(x) \in \mathbb{F}_2^n} f(x) = w_H(f)$. □

With lemma 4.49 we can prove that the nonlinearity is an affine invariant. In particular the property that a Boolean function is bent is invariant under an affine isomorphism.

Lemma 4.50. *The nonlinearity is an affine invariant. Let f be a Boolean function on $\mathbb{F}_2^n$ and L an affine isomorphism on $\mathbb{F}_2^n$, then*

$$\mathcal{NL}(f \circ L) = \mathcal{NL}(f).$$

Proof. Let $f, l \in \mathcal{BF}_n$, l a nonconstant affine function and L an affine isomorphism. Since the weight is an affine invariant we have that

$$w_H((f \oplus l) \circ L) = w_H(f \oplus l).$$

For the LHS we have

$$w_H((f \oplus l) \circ L) = d_H((f \circ L) \oplus (l \circ L), \mathbf{0}) = d_H(f \circ L, l \circ L)$$

and for the RHS we have

$$w_H(f \oplus l) = d_H(f, l).$$

So $d_H(f \circ L, l \circ L) = d_H(f, l)$ and since all nonconstant affine functions are affine equivalent we have that $\mathcal{NL}(f \circ L) = \mathcal{NL}(f)$. □

In the last lemma of this subsection we look at the effect on the Fourier transform of a composition of a pseudo-Boolean function and a linear isomorphism. Let M be a nonsingular $n \times n$ matrix, $L(x) = Mx$ a linear isomorphism and let $L^{-1}(x) = M^{-1}x$ be the inverse of L. We write M' for the transpose of M^{-1} and $L'(x) = M'x$ for the adjoint operator of L^{-1}, i.e. $u \cdot L^{-1}(x) = L'(u) \cdot x$.

Lemma 4.51. *Let φ be a pseudo-Boolean function on $\mathbb{F}_2^n$ and L be a linear isomorphism, then*

$$\widehat{\varphi \circ L} = \widehat{\varphi} \circ L'.$$

Proof.

$$\begin{aligned} \widehat{\varphi \circ L}(u) &= \sum_{x \in \mathbb{F}_2^n} \varphi(L(x))(-1)^{u \cdot x} = \sum_{x \in \mathbb{F}_2^n} \varphi(x)(-1)^{u \cdot L^{-1}(x)} \\ &= \sum_{x \in \mathbb{F}_2^n} \varphi(x)(-1)^{L'(u) \cdot x} = \widehat{\varphi} \circ L'(u). \end{aligned}$$

□

4.4 Cryptographic Properties

In section 2.2 we explained the two principles of C.E. Shannon; confusion and diffusion. In a substitution permutation network SPN an S-box and a linear permutation need to satisfy these principles. The S-box consists of a vector of nonlinear Boolean functions. As we have seen, there are more properties for Boolean functions then nonlinearity. We will analyze some of these properties on their consequences for the cipher and the principles confusion and diffusion.

Nonlinearity The first criterion of an S-box is that it is nonlinear, since otherwise the SPN would be easily solvable by linear algebra. Nonetheless, as in a linear attack, a linear approximation of the S-box can be made. It

gets harder to make a good linear approximation when the nonlinearity of the S-box is higher. Hence a good (high) nonlinearity ensures more confusion. In this view it is desirable to use an S-box that is bent. Unfortunately this can not be derived together with other properties such as the algebraic degree and balancedness which will be explained below. We need to be satisfied with S-boxes which are highly nonlinear but not bent.

Algebraic degree An S-box with high algebraic degree makes the relation between the ciphertext and the encryption key more complex, hence causes more confusion. We state Rothaus Bound without a proof. The proof makes use of the dual of a bent function and this is beyond the scope of this thesis. For a proof see Carlet [5].

Proposition 4.52. *Let f be a Boolean function on $\mathbb{F}_2^n$ with $n \geq 4$. If f is bent, then the algebraic degree is less or equal to $n/2$.*

With this result, bent function are clearly not optimal and trade-offs must be made when one chooses an S-box.

Balancedness Recall that a function $f \in \mathcal{BF}_n$ is balanced if $w_H(f) = 2^{n-1}$. If a Boolean function is not balanced one output value will have a higher probability to occur. Since this disadvantage can be used in an attack, only balanced (coordinate) functions are used in an S-box. In particular, the S-box of an SPN needs to be a permutation, hence needs to consist of balanced coordinate functions. More of this will be explained in the next section about vectorial Boolean functions.

In lemma 4.43 we have seen that $w_H(f) = 2^{n-1} - \frac{1}{2}\widehat{f}_\chi(\mathbf{0})$. And from corollary 4.46 we know that if f is a bent function, then the Walsh transform $\widehat{f}_\chi(a)$ equals $\pm 2^{\frac{n}{2}}$ for all $a \in \mathbb{F}_2^n$. Hence the weight of a bent function equals $2^{n-1} \pm 2^{\frac{n}{2}-1}$ and a bent function is not balanced. This is another reason why bent functions are not optimal S-boxes.

Strict avalanche criteria and propagation criteria The idea of a differential attack is to make a small difference in the input of the cipher and look at the behavior of the output. Ensuring that many derivative functions are balanced makes it hard to assign good statistics on the output differences. This causes a better diffusion.

Definition 4.53. A Boolean function f on $\mathbb{F}_2^n$ satisfies the propagation criteron with respect to $E \subset \mathbb{F}_2^n$ if for each $x \in E$ the derivative $D_x f$ is balanced.

We say that a Boolean function f satisfies PC (k) if E consists of all nonzero vectors of weight less or equal to k, i.e.

$$E = \{x \in \mathbb{F}_2^n : 1 \leq w_H(x) \leq k\}.$$

This is equivalent with $\mathcal{F}(D_e f)$=0, for all $0 < w_H(e) \leq k$, since $\mathcal{F}(D_e f) = \sum_{x \in \mathbb{F}_2^n} (-1)^{D_e f(x)}$. Propagation criteria PC is a general form of *strict avalanche criteria* SAC. A function satisfies SAC if the derivative is balanced for all vectors of weight one, which is equivalent to PC (1). There is a relation between the maximal nonlinearity and the maximal propagation criterion.

Proposition 4.54. *Let f be a Boolean function on $\mathbb{F}_2^n$, f is bent if and only if f satisfies PC(n).*

Proof. Recall the corollary 4.41 of the Wiener-Khintchine theorem, we have that $\widehat{f_\chi}^2(a) = \sum_{x \in \mathbb{F}_2^n} \mathcal{F}(D_x f)(-1)^{a \cdot x}$. We deduce,

$$\widehat{f_\chi}^2(a) = \sum_{x \in \mathbb{F}_2^n} \mathcal{F}(D_x f)(-1)^{a \cdot x} = 2^n + \sum_{x \in \mathbb{F}_2^n \backslash \{\mathbf{0}\}} \mathcal{F}(D_x f)(-1)^{a \cdot x}.$$

From this it follows that $\widehat{f_\chi}^2(a) = 2^n$ if and only if $\mathcal{F}(D_x f) = 0$ for all $x \in \mathbb{F}_2^n \backslash \{\mathbf{0}\}$. That is, $\widehat{f_\chi}(a) \in \{\pm 2^{\frac{n}{2}}\}$ if and only if $D_x f$ is balanced for all $x \in \mathbb{F}_2^n \backslash \{\mathbf{0}\}$. So f is bent if and only if f satisfies PC (n). □

We have seen that we cannot use bent functions as an S-box; it follows that we can not use functions which satisfy PC (n) either.

Other properties These are the most important properties of Boolean function which are used as an S-box. There are more properties of Boolean functions which are interesting for cryptographic applications, such as PC (k) of order l, resiliency, linear structures and algebraic immunity. More about these properties can be found in Carlet [5].

5 Vectorial Boolean functions

In this section we will introduce vectorial Boolean functions. These functions are used in cipher systems, in particular for S-boxes. We extend properties of the previous chapter, introduce new properties such as almost bent AB and almost perfect nonlinear APN, look at two different kind of notions of equivalence; EA-equivalence and CCZ-equivalence and we show a connection between Vectorial Boolean functions and coding theory. The theory developed here will be used in the next section to deduce the Dillon-Wolfe function.

5.1 Extensions of Definitions and Properties

Most notions and properties that we have seen in section 4.1 can be extended to vectorial Boolean functions $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$, where n, m are positive integers. Such a function F is called an (n, m)-function and can be viewed as a vector of Boolean functions $f_i : \mathbb{F}_2^n \to \mathbb{F}_2$,

$$F(x) \quad = \quad \begin{pmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{pmatrix}.$$

These Boolean functions f_i, $1 \leq i \leq m$, are called the coordinate functions of F and for $b \in \mathbb{F}_2^m \backslash \{\mathbf{0}\}$, the inner product $b \cdot F$ is called a component function. This component function $b \cdot F$ is used in most extensions of previous definitions. Note that a component function is a Boolean function on $\mathbb{F}_2^n$, since it is a linear combination of coordinate functions.

5.1.1 ANF

When we write the coordinate functions in the *algebraic normal form* ANF, we obtain the ANF of a vectorial Boolean function.

Definition 5.1. Let F be a (n, m)-function and $N = \{1, \ldots, n\}$. We can represent F uniquely in the algebraic normal form ANF by,

$$F(x_1, \ldots, x_n) = \sum_{I \in \mathcal{P}(N)\}} a_I \left(\prod_{i \in I} x_i \right), \quad a_I \in \mathbb{F}_2^m.$$

Or we just write

$$F(x) = \sum_{I \in \mathcal{P}(N)} a_I x^I, \qquad a_I \in \mathbb{F}_2^m.$$

Recall our convention to write $+$ for the addition of vectors in $\mathbb{F}_2^m$. But for a single coordinate, the addition is done modulo 2 and is written as $\oplus$.

Example 5.2. Let F be a $(3,3)$ function and for $I \in \mathcal{P}(\{1,2,3\})$ the vectors a_I are:

$$a_\emptyset = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix},\ a_{\{1\}} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix},\ a_{\{2\}} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix},\ a_{\{3\}} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix},$$

$$a_{\{1,2\}} = a_{\{1,3\}} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix},\ a_{\{2,3\}} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix},\ a_{\{1,2,3\}} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

We get the vectorial Boolean function,

$$\begin{aligned} F(x_1,x_2,x_3) &= \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} x_1 + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} x_2 + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} x_3 \\ &\quad + \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} (x_1x_2 \oplus x_1x_3) + \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} x_2x_3 + \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} x_1x_2x_3 \\ &= \begin{pmatrix} 1 \oplus x_3 \oplus x_1x_2x_3 \\ x_1 \oplus x_1x_2 \oplus x_1x_3 \oplus x_1x_2x_3 \\ 1 \oplus x_1 \oplus x_2 \oplus x_1x_2x_3 \end{pmatrix}. \end{aligned}$$

For the ANF of a vectorial Boolean function we have similar relations as relation 3 and lemma 4.15 for Boolean functions, see section 4.1.2.

Lemma 5.3. *Let F be an (n,m)-function and $\sum_{I \in \mathcal{P}(N)} a_I x^I$ its ANF, then*

i.

$$F(u) = \sum_{I \subseteq supp(u)} a_I,$$

ii.

$$a_I = \sum_{x \in \mathbb{F}_2^n, supp(x) \subseteq I} F(x).$$

Proof. Note that we can compute each coordinate of F and a_I separately. Let $F = (f_1, \ldots, f_m)$ the coordinate functions and $a_I = (a_{I1}, \ldots, a_{Im})$. Then for $1 \leq i \leq m$ we have

i.
$$f_i(u) = \bigoplus_{I \subseteq \mathrm{supp}(u)} a_{Ii},$$

ii.
$$a_{Ii} = \bigoplus_{x \in \mathbb{F}_2^n, \mathrm{supp}(x) \subseteq I} f_i(x).$$

Applying relation (3) on i and lemma 4.15 on ii proves the desired. □

5.1.2 The algebraic degree and balancedness

The algebraic degree of a vectorial Boolean function is defined as the maximum degree of all coordinate functions.

Definition 5.4. Let F be an (n, m)-function and $F = (f_1, \ldots, f_m)$ the coordinate functions. The algebraic degree of F is defined as the maximum algebraic degree of all f_i, that is

$$d^\circ(F) := \max_{1 \leq i \leq n} d^\circ(f_i).$$

For any two coordinate functions f_i, f_j, we have that

$$d^\circ(f_i \oplus f_j) \leq \max(d^\circ(f_i), d^\circ(f_j)).$$

From this it follows that the algebraic degree of F is equal to the maximum degree of all component functions, i.e.

$$d^\circ(F) = \max_{b \in \mathbb{F}_2^m} d^\circ(b \cdot F).$$

In a similar way as for Boolean functions, a vectorial Boolean function F is called affine if $d^\circ(F) \leq 1$. If F is affine and $F(\mathbf{0}) = \mathbf{0}$ (where the input $\mathbf{0} \in \mathbb{F}_2^n$ and the output $\mathbf{0} \in \mathbb{F}_2^m$), then F is called linear. For (n, m)-functions we have also a minimum degree, that is the minimum degree of all component functions,

$$d^\circ_{min}(F) := \min_{b \in \mathbb{F}_2^m \setminus \{\mathbf{0}\}} d^\circ(b \cdot F).$$

Example 5.5. We compute the algebraic degree for the function F from example 5.2

$$F(x) = \begin{pmatrix} 1 \oplus x_3 \oplus x_1x_2x_3 \\ x_1 \oplus x_1x_2 \oplus x_1x_3 \oplus x_1x_2x_3 \\ 1 \oplus x_1 \oplus x_2 \oplus x_1x_2x_3 \end{pmatrix}.$$

Clearly $d^\circ(F) = 3$ since every coordinate function is of algebraic degree 3. We make a table of all component functions $b \cdot F$, but leave out the coordinate functions.

$$\begin{aligned} (1,1,0) \cdot F(x) &= 1 \oplus x_1 \oplus x_3 \oplus x_1x_2 \oplus x_1x_3 \\ (1,0,1) \cdot F(x) &= x_1 \oplus x_2 \oplus x_3 \\ (0,1,1) \cdot F(x) &= 1 \oplus x_2 \oplus x_1x_2 \oplus x_1x_3 \\ (1,1,1) \cdot F(x) &= x_2 \oplus x_3 \oplus x_1x_2 \oplus x_1x_3 \oplus x_1x_2x_3 \end{aligned}$$

The minimum degree of the component functions is 1 and therefore $d^\circ_{min}(F) = 1$.

We define the notion of a balanced vectorial Boolean function.

Definition 5.6. A vectorial Boolean function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ with $m \leq n$, is called balanced if each output value in $\mathbb{F}_2^m$ is reached equally many times, that is 2^{n-m} times.

It follows that (n, n)-functions which are balanced, are permutations on $\mathbb{F}_2^n$. The question arises if, for a (n, m)-function F to be balanced, the component functions need to be balanced as well. To answer this, we define the pre-image of F for $b \in \mathbb{F}_2^m$ as $F^{-1}(b) := \{x \in \mathbb{F}_2^n : F(x) = b\}$.

Proposition 5.7. *Let F be an (n, m)-function and $b \in \mathbb{F}_2^m$ nonzero. Then F is balanced if and only if all component functions $b \cdot F$ are balanced.*

Proof. We start with two observations:

I. By definition, F is balanced if and only if $|F^{-1}(y)| = |F^{-1}(z)|$ for all $y, z \in \mathbb{F}_2^m$.

II. We define the pseudo-Boolean function $\psi(b) := \sum_{x \in \mathbb{F}_2^n} (-1)^{b \cdot F(x)}$. For all nonzero $b \in \mathbb{F}_2^m$ we have, $\psi(b) = 0$ if and only if $b \cdot F(x)$ is balanced.

Corollary 4.33, applied to the Fourier transform $\widehat{\varphi}$ of a pseudo-Boolean function φ, states that: $\widehat{\varphi}$ is constant if and only if $\varphi(a) = 0$ for all $a \in \mathbb{F}_2^n \backslash \{\mathbf{0}\}$. Hence, by II, we need to prove that $\widehat{\psi}$ is constant if and only if F is balanced.

To prove this, we first observe that for an (n, m)-function F, $x \in \mathbb{F}_2^n$ and $y \in \mathbb{F}_2^m$ the map $b \mapsto b \cdot (F(x) + y)$ is a linear Boolean function on $\mathbb{F}_2^m$. Hence this map is either balanced or zero and we obtain

$$\begin{aligned} \sum_{b \in \mathbb{F}_2^m} (-1)^{b \cdot (F(x)+y)} &= \begin{cases} 2^m & \text{if } x \in F^{-1}(y), \\ 0 & \text{else.} \end{cases} \\ &= 2^m 1_{F^{-1}(y)}(x). \end{aligned}$$

Where $1_{F^{-1}(y)}$ is the indicator function of the set $F^{-1}(y)$. We take the sum over all $x \in \mathbb{F}_2^n$ and get

$$\sum_{x \in \mathbb{F}_2^n} \sum_{b \in \mathbb{F}_2^m} (-1)^{b \cdot (F(x)+y)} = 2^m |F^{-1}(y)|.$$

Now we are ready to compute the Fourier transform of ψ at $y \in \mathbb{F}_2^m$;

$$\begin{aligned} \widehat{\psi}(y) &= \sum_{b \in \mathbb{F}_2^m} \psi(b)(-1)^{b \cdot y} = \sum_{b \in \mathbb{F}_2^m} \sum_{x \in \mathbb{F}_2^n} (-1)^{b \cdot F(x)} (-1)^{b \cdot y} \\ &= \sum_{x \in \mathbb{F}_2^n} \sum_{b \in \mathbb{F}_2^m} (-1)^{b \cdot (F(x)+y)} = 2^m |F^{-1}(y)|. \end{aligned}$$

This proves that $\widehat{\psi}$ is constant if and only if $|F^{-1}(y)| = |F^{-1}(z)|$ for all $y, z \in \mathbb{F}_2^m$. Which means, by observation I, that F is balanced. Hence F is balanced if and only if $b \cdot F$ is balanced for all $b \in \mathbb{F}_2^m \backslash \{\mathbf{0}\}$. □

5.1.3 The Walsh transform and nonlinearity

The Walsh transform of an (n, m)-function at the value $a \in \mathbb{F}_2^n$ is defined for each component function $b \cdot F$ with $b \in \mathbb{F}_2^m \backslash \{\mathbf{0}\}$.

Definition 5.8. Let F be an (n, m)-function, $a \in \mathbb{F}_2^n$ and nonzero $b \in \mathbb{F}_2^m$. The Walsh transform at (a, b) is defined as

$$\lambda_F(a, b) = \sum_{x \in \mathbb{F}_2^n} (-1)^{b \cdot F(x) + a \cdot x}.$$

The values of the Walsh transforms determine an important property of a vectorial Boolean function F. The set of all values at $a \in \mathbb{F}_2^n$ and $b \in \mathbb{F}_2^m$, $b \neq \mathbf{0}$, is called the Walsh spectrum of F:

$$\Lambda_F := \{\lambda_F(a,b) : (a,b) \in \mathbb{F}_2^n \times \mathbb{F}_2^m \backslash \{\mathbf{0}\}\}.$$

We have seen in the previous chapter that the Walsh transform of a Boolean function is related to the nonlinearity. We define the nonlinearity of a vectorial Boolean function.

Definition 5.9. Let F be an (n,m)-function and $b \in \mathbb{F}_2^m$ nonzero. The nonlinearity of F is defined by the minimum nonlinearity of all component functions $b \cdot F$, that is

$$\mathcal{NL}(F) := \min_{b \in \mathbb{F}_2^m \backslash \{\mathbf{0}\}} \mathcal{NL}(b \cdot F).$$

In words; the minimum Hamming distance between the component functions and all affine functions on $\mathbb{F}_2^n$. Since the component function is just a Boolean function on $\mathbb{F}_2^n$ we can use our results from section 4.2.2. We extend proposition 4.45 and corollary 4.46.

Proposition 5.10. *Let F be an (n,m)-function and $\lambda_F(a,b)$ the Walsh transform of F at $(a,b) \in \mathbb{F}_2^n \times \mathbb{F}_2^m \backslash \{\boldsymbol{0}\}$, then*

i.
$$\mathcal{NL}(F) = 2^{n-1} - \frac{1}{2} \max_{a \in \mathbb{F}_2^n,\ b \in \mathbb{F}_2^m \backslash \{\boldsymbol{0}\}} |\lambda_F(a,b)|.$$

ii.
$$\mathcal{NL}(F) \leq 2^{n-1} - 2^{\frac{n}{2}-1}.$$

Proof. For i: By proposition 4.45 we have for $b \in \mathbb{F}_2^m \backslash \{\mathbf{0}\}$

$$\mathcal{NL}(b \cdot F) = 2^{n-1} - \frac{1}{2} \max_{a \in \mathbb{F}_2^n} |\widehat{(b \cdot F)}_\chi(a)|.$$

This proves i since $\widehat{(b \cdot F)}_\chi(a) = \lambda_F(a,b)$.

For ii: The covering radius bound from corollary 4.46 holds for every component function $b \cdot F$ and therefore holds for any vectorial Boolean function as well. □

A vectorial Boolean function F is called *bent* if it attains the covering radius bound. Hence F is bent if and only if all component functions of F are bent. Proposition 4.54 states that a Boolean function is bent if and only if all derivatives are balanced. We can extend this result to vectorial Boolean functions. The derivative of a vectorial Boolean function is defined similar as for Boolean functions.

Definition 5.11. Let F be an (n, m)-function and $a \in \mathbb{F}_2^n \backslash \{\mathbf{0}\}$. The derivative of F in the direction of a is:

$$D_a F(x) := F(x) + F(x + a).$$

Note that the derivative is an (n, m)-function as well.

Proposition 5.12. *An (n, m)-function F is bent if and only if the derivative $D_a F$ is balanced for all nonzero $a \in \mathbb{F}_2^n$.*

Proof. An (n, m)-function F is bent if and only if the component functions $b \cdot F$ are bent for all $b \in \mathbb{F}_2^m \backslash \{\mathbf{0}\}$. By proposition 4.54, $b \cdot F$ is bent if and only if it satisfies PC(n), i.e. $D_a(b \cdot F)$ is balanced $\forall a \in \mathbb{F}_2^n$, $a \neq \mathbf{0}$. We rewrite this derivative and deduce, $D_a(b \cdot F)(x) = (b \cdot F)(x) + (b \cdot F)(x + a) = b \cdot (F(x) + F(x + a))$, hence

$$D_a(b \cdot F)(x) = b \cdot D_a F(x).$$

By proposition 5.7, $b \cdot D_a F(x)$ is balanced for all nonzero $a \in \mathbb{F}_2^n$ if and only if $D_a F(x)$ is balanced for all nonzero $a \in \mathbb{F}_2^n$. This proves the desired. □

A vectorial Boolean (n, m)-function F is called *perfect nonlinear* if all derivatives $D_a F$, $a \in \mathbb{F}_2^n \backslash \{\mathbf{0}\}$, are balanced. For nonzero $a \in \mathbb{F}_2^n$ and $b \in \mathbb{F}_2^m$, we denote $\delta_F(a, b)$ for the cardinality of the pre-image $(D_a F)^{-1}(b)$, that is

$$\delta_F(a, b) := |\{x \in \mathbb{F}_2^n : F(x) + F(x + a) = b\}|.$$

An (n, m)-function with $n = m$, is perfect nonlinear[9] if for all $(a, b) \in \mathbb{F}_2^n \backslash \{\mathbf{0}\} \times \mathbb{F}_2^m$ we have that $\delta_F(a, b) = 1$. This is impossible since solutions of $F(x) + F(x + a) = b$ come in pairs, if x is a solution, then $x + a$ is another solution. Hence there exist no perfect nonlinear or bent (n, m)-functions for

[9]Perfect nonlinear function are also called *planar* functions.

$n = m$. It can be shown that bent, and hence perfect nonlinear, functions only exist for $m \leq \frac{n}{2}$, see Carlet [6].

For an (n, m)-function F, the values of δ_F determine an important property. The set of all values of δ_F is called the differential spectrum:

$$\Delta_F := \{\delta_F(a, b) : (a, b) \in \mathbb{F}_2^n \backslash \{\mathbf{0}\} \times \mathbb{F}_2^m\}.$$

The Walsh spectrum Λ_F and the differential spectrum Δ_F of an S-box F determine the resistance to cryptographic attack. High values of $\lambda_F(a, b)$ make a linear attack easier, hence bent function are optimal against linear attacks. Unfortunately bent functions do not exist for (m, m)-functions. For the differential spectrum, small values of $\delta_F(a, b)$ posses the best optimum against a differential attack, but we have seen that there exist no perfect nonlinear (m, m)-functions either. In the next section we will say more about (m, m)-functions and their optimum properties.

5.2 AB and APN

There are several criteria on which a vectorial Boolean function can be chosen to be used as an S-box. One criteria is that the S-box is a permutation. This ensures that no information is lost (although there are ciphers known with S-boxes which are not permutations). From now on we assume that $n = m$. We have seen that (m, m)-functions can not be bent and perfect nonlinear. Below, we will give other optimal bounds for (m, m)-functions.

We state a better upper bound for the nonlinearity of an (m, m)-function without a proof[10].

Lemma 5.13. *Let F be a (m, m)-function, then*

$$\mathcal{NL}(F) \leq 2^{m-1} - 2^{\frac{m-1}{2}}.$$

Clearly, this bound can only be attained for m odd.

Definition 5.14. An (m, m)-function F with m odd is called almost bent AB if

$$\mathcal{NL}(F) = 2^{m-1} - 2^{\frac{m-1}{2}}.$$

[10]This is a particular case of the Sidelnikov-Chabaud-Vaudenay bound which was introduced in [7]

The name 'almost bent' is somewhat misleading, note that this bound is an optimum for an (m, m)-function F. From proposition 5.10 i we deduce a consequence for the Walsh transform of a AB function.

Lemma 5.15. *An (m, m)-function F is AB if and only if $\lambda_F(a, b) \in \{0, \pm 2^{\frac{m+1}{2}}\}$ for all $a, b \in \mathbb{F}_2^m$, $b \neq \boldsymbol{0}$.*

Proof. Clearly, if $\lambda_F(a, b) \in \{0, \pm 2^{\frac{m+1}{2}}\}$ for all $a, b \in \mathbb{F}_2^m$, $b \neq \mathbf{0}$, then $\max_{a,b \in \mathbb{F}_2^m, b \neq \mathbf{0}} |\lambda_F(a, b)| = 2^{m+1}$ and hence F is AB by proposition 5.10 i.

For the other direction observe that

$$\lambda := \max_{a,b \in \mathbb{F}_2^m, b \neq \mathbf{0}} \lambda_F(a, b)^2 \geq \frac{\sum_{a,b \in \mathbb{F}_2^m, b \neq \mathbf{0}} \lambda_F(a, b)^4}{\sum_{a,b \in \mathbb{F}_2^m, b \neq \mathbf{0}} \lambda_F(a, b)^2}.$$

Equality is attained only if for all $a, b \in \mathbb{F}_2^m, b \neq \mathbf{0}$,

$$\lambda_F(a, b)^2 \in \{0, \lambda\}.$$

If F is AB, then by proposition 5.10 i we have $\max_{a,b \in \mathbb{F}_2^m, b \neq \mathbf{0}} |\lambda_F(a, b)| = 2^{\frac{m+1}{2}}$ and hence $\lambda = 2^{m+1}$. This proves the desired. □

We state an optimal bound for the values of the differential spectrum of an (m, m)-function.

Definition 5.16. An (m, m)-function F is called almost perfect nonlinear APN if for all $a, b \in \mathbb{F}_2^m$, $a \neq \mathbf{0}$,

$$F(x) + F(x + a) = b$$

admits at most two solutions, i.e. $\delta_F(a, b) \in \{0, 2\}$.

Again, the name 'almost perfect nonlinear' is a little misleading. We have seen that solutions of $\delta_F(a, b)$ come in pairs, hence APN is an optimum for (m, m)-functions. We can rewrite this definitions as follows.

Lemma 5.17. *Let F be an (m, m)-function, then F is APN if and only if the system of equations*

$$\begin{cases} x + y & = & a \\ F(x) + F(y) & = & b \end{cases}$$

admits at most two solutions for all $a, b \in \mathbb{F}_2^m$,$a \neq \boldsymbol{0}$.

Proof. This follows immediate since we can substitude $y = x + a$. □

Remark 5.18. It was long a big open question if there exists a vectorial Boolean function (S-box) of even dimension which is a permutation and APN. The Dillon-Wolfe function is an $(6, 6)$-function and is the only known function which achieves both properties. This function has the best possible resistance against differential attacks. We will say more on the construction of a Dillon-Wolfe function in section 6.

5.3 EA- and CCZ-Equivalence

We have already seen an equivalence relation on Boolean functions. In this section we will define two equivalent relations on vectorial Boolean (m, m)-functions and will see which notions are invariant under these equivalences.

5.3.1 Equivalences

We extend the equivalence relation on Boolean functions to vectorial Boolean functions. Since a vectorial Boolean (m, m)-function F maps from $\mathbb{F}_2^m$ to $\mathbb{F}_2^m$, we can compose F on the left and right by an affine permutation.

Definition 5.19. Let F, G be two (m, m)-functions and $A, A_1, A_2 : \mathbb{F}_2^m \to \mathbb{F}_2^m$ such that, A is an affine function and A_1, A_2 are two affine permutations. The two functions F and G are called extended affine EA equivalent if

$$G = A_1 \circ F \circ A_2 + A.$$

If $A = 0$, we call F and G affine equivalent.

We will show that the above defines an equivalence relation. Let

$$I_m, A, A_1, A_2, B, B_1, B_2 : \mathbb{F}_2^m \to \mathbb{F}_2^m,$$

be functions with I_m the identity map, A, B affine functions, A_1, A_2, B_1, B_2 affine permutations. Let F, G, H be (m, m) functions, we obtain an equivalent relation with linear algebra:

- Reflexive: $F = I_m \circ F \circ I_m + \mathbf{0}$.
- Symmetric: If $G = A_1 \circ F \circ A_2 + A$, then

  $$F = A_1^{-1} \circ G \circ A_2^{-1} + (A_1^{-1} \circ A \circ A_2^{-1}),$$

  with $A_1^{-1} \circ A \circ A_2^{-1}$ an affine function.

- Transitive: If $G = A_1 \circ F \circ A_2 + A$ and $H = B_1 \circ G \circ B_2 + B$, then

$$H = (B_1 \circ A_1) \circ F \circ (B_2 \circ A_2) + (B_1 \circ A \circ B_2 + B)$$

  with $(B_1 \circ A_1)$ and $(B_2 \circ A_2)$ affine permutations and $(B_1 \circ A \circ B_2 + B)$ an affine function.

There exists a more general notion of equivalence, called CCZ-equivalence, which was first stated by Carlet-Charpin-Zinoviev in [4]. We define the *graph* of an (m, m)-function as:

$$G_F := \{(x, F(x)) : x \in \mathbb{F}_2^m\}.$$

Definition 5.20. Let F, F' be two (m, m)-functions and $\mathcal{L}$ a linear permutation on $\mathbb{F}_2^{2m}$. The functions F and F' are called CCZ-equivalent if

$$\mathcal{L}(G_F) = G_{F'}.$$

We give a step by step description of the linear permutation $\mathcal{L}$ and the relation between F and F'.

- Two vectorial Boolean functions $F, F' : \mathbb{F}_2^m \to \mathbb{F}_2^m$ and a linear permutation $\mathcal{L} : \mathbb{F}_2^m \times \mathbb{F}_2^m \to \mathbb{F}_2^m \times \mathbb{F}_2^m$ of the form $\mathcal{L} = (L_1, L_2)$, where $L_1, L_2 : \mathbb{F}_2^{2m} \to \mathbb{F}_2^m$ are two linear functions.

- Apply this $\mathcal{L} = (L_1, L_2)$ on the graph G_F we get

$$\mathcal{L}(G_F) = \mathcal{L}(x, F(x)) = \mathcal{L}(L_1(x, F(x)), L_2(x, F(x))).$$

  We define two functions F_1 and F_2 as

$$F_1(x) := L_1(x, F(x)) \quad \text{and} \quad F_2(x) := L_2(x, F(x)).$$

  So we can write $\mathcal{L}(x, F(x)) = (F_1(x), F_2(x))$.

- Now, if $F_1(x)$ is a permutation, then F_1^{-1} exists and we have

$$\mathcal{L}(x, F(x)) = (x, F_2 \circ F_1^{-1}(x)).$$

  So, if the function F' is of the form $F' = F_2 \circ F_1^{-1}$, then $\mathcal{L}(G_F) = G_{F'}$ and F and F' are CCZ-equivalent.

This CCZ-equivalence defines an equivalence relation. Let F, F', F'' be (m,m)-functions and $\mathcal{L}, \mathcal{L}'$ two affine permutations as described in the definition above.

- Reflexive: Take $\mathcal{L} = (L_1, L_2)$ such that $L_1(a,b) = a$ and $L_2(a,b) = b$, then $\mathcal{L}(x, F(x)) = (x, F(x))$ and $\mathcal{L}(G_F) = G_F$.
- Symmetric: Suppose $\mathcal{L}(G_F) = G_{F'}$, since $\mathcal{L}$ is an affine permutation the inverse $\mathcal{L}^{-1}$ is also an affine permutation and $\mathcal{L}^{-1}(G_{F'}) = \mathcal{L}^{-1} \circ \mathcal{L}(G_F) = G_F$.
- Transitive: Suppose that $\mathcal{L}(G_F) = G_{F'}$ and $\mathcal{L}'(G_{F'}) = G_{F''}$. Since $\mathcal{L}' \circ \mathcal{L}$ is again an affine permutation we have $\mathcal{L}' \circ \mathcal{L}(G_F) = \mathcal{L}'(G_{F'}) = G_{F''}$.

If F and F' are CCZ-equivalent and the function $L_1(a,b)$ depends only on one variable, then $L_2(a,b)$ depends on one variable as well. Hence L_1 and L_2 are linear permutations and therefore F_1 and F_2 are linear permutations. This makes F and F' EA-equivalent. The converse is not true in general.

Example 5.21. Let $L_1(a,b) = L_1(a)$ and $L_2(a,b) = b$, then $F_1(x) = L_1(x)$ and $F_2(x) = L_2(F(x))$. We get that $F' = F_2 \circ F_1^{-1} = L_2 \circ F \circ L_1^{-1}$, hence F and F' are EA-equivalent.

If F and F' are CCZ-equivalent and the function $\mathcal{L}(a,b) = (L_1, L_2)(a,b) = (b,a)$, then $F' = F^{-1}$. Hence F and the inverse F^{-1} are CCZ-equivalent. We will see below that F and F^{-1} are not EA-equivalent in general, hence CCZ-equivalence is strictly more general then EA-equivalence.

It can be hard to prove that two functions are not CCZ-equivalent. One way is to look at the invariant properties of the two functions. We will say more about invariant properties in the next section.

5.3.2 Invariants

Some properties of vectorial Boolean functions are invariant under EA- or CCZ-equivalence. We have seen that EA-equivalence is more strict that CCZ-equivalence, hence if a property is EA-invariant it is also CCZ-invariant. We take a closer look at: the algebraic degree, the Walsh spectrum, nonlinearity and the differential spectrum.

Proposition 5.22. *The algebraic degree is EA-invariant for (m,m)-functions F with $d^\circ(F) \geq 2$.*

Proof. Let $F = (f_1, \ldots, f_m)$ be an (m, m)-functions and $A, A_1, A_2 : \mathbb{F}_2^m \to \mathbb{F}_2^m$ functions such that A an affine function and A_1, A_2 two affine permutations. Suppose $d^\circ(F) \geq 2$, we proof that $d^\circ(F) = d^\circ(A_1 \circ F) = d^\circ(F \circ A_2)$. This proves the desired since adding an affine function A does not change the algebraic degree, i.e. $d^\circ(F + A) = d^\circ(F)$.

First, observe that $A_1 \circ F$ is a vectorial Boolean function for which each coordinate function is equal to a component function of F. Therefore we have that $d^\circ(A_1 \circ F) = \max_{b \in \mathbb{F}_2^m \setminus \{\mathbf{0}\}} d^\circ(b \cdot F) = d^\circ(F)$.

Second, observe that for $1 \leq i \leq m$, the ith coordinate function of $(F \circ A_2)$ equals $f_i \circ A_2$. Therefore we have that $d^\circ(F \circ A_2) = \max_{1 \leq i \leq m} d^\circ(f_i \circ A_2)$. By lemma 4.48 the algebraic degree of a Boolean function is an affine invariant. Hence $d^\circ(F \circ A_2) = \max_{1 \leq i \leq m} d^\circ(f_i) = d^\circ(F)$. □

If $d^\circ(F) \leq 1$ then F is affine. So if $A = F$, then the two functions F and $F + A = 0$ are EA-equivalent but do not have that same algebraic degree in general. Hence the algebraic degree is only an EA-invariant for functions of algebraic degree greater or equal then two.

The proof of the minimal degree being an EA-invariant if $d^\circ_{min}(F) \geq 2$ is very similar.

We have seen that an (m, m)-function F and the inverse F' (if it exists) are CCZ-equivalent. Since in general $d^\circ(F) \neq d^\circ(F')$, the algebraic degree is not CCZ-invariant.

Proposition 5.23. *The Walsh spectrum is CCZ-invariant.*

Proof. Let F and F' be two (m, m)-functions which are CCZ-equivalent, with $\mathcal{L}(G_F) = G_{F'}$. We write $\mathcal{L}^\star$ for the adjoint operator of $\mathcal{L}$, i.e. $(a, b) \cdot \mathcal{L}(x, y) = \mathcal{L}^\star(a, b) \cdot (x, y)$. For $a, b \in \mathbb{F}_2^m$, $b \neq \mathbf{0}$, we have that

$$\begin{aligned} \lambda_{F'}(a, b) &= \sum_{x \in \mathbb{F}_2^m} (-1)^{b \cdot F'(x) + a \cdot x} = \sum_{x \in \mathbb{F}_2^m} (-1)^{(a,b) \cdot (x, F'(x))} \\ &= \sum_{x \in \mathbb{F}_2^m} (-1)^{(a,b) \cdot \mathcal{L}(x, F(x))} = \sum_{x \in \mathbb{F}_2^m} (-1)^{\mathcal{L}^*(a,b) \cdot (x, F(x))} = \lambda_F(\mathcal{L}^\star(a, b)). \end{aligned}$$

Since $\mathcal{L}$ is a permutation, $\mathcal{L}^\star$ is also a permutation and the two Walsh spectra Λ_F and $\Lambda_{F'}$ are equal up to a permutation of the elements. Hence the Walsh spectrum is CCZ-invariant. □

By lemma 5.10, the nonlinearity is directly related to the Walsh transform. It follows that the nonlinearity is CCZ-invariant as well. We state a particular consequence.

Corollary 5.24. *The property almost bent AB is CCZ-invariant.*

For the differential spectrum we have a similar result.

Proposition 5.25. *The differential spectrum is CCZ-invariant.*

Proof. Let F and F' be two (m,m)-functions which are CCZ-invariant, with $\mathcal{L}(G_F) = G_{F'}$. Let $a, b \in \mathbb{F}_2^m$, $a \neq \mathbf{0}$, by lemma 5.17 we have

$$\begin{aligned}\delta_{F'}(a,b) &= |\{F'(x) + F'(x+a) = b\}| = |\{(x, F'(x)) + (y, F'(y)) = (a,b)\}| \\ &= |\{\mathcal{L}(x, F(x)) + \mathcal{L}(y, F(y)) = (a,b)\}| \\ &= |\{(x, F(x)) + (y, F(y)) = \mathcal{L}^{-1}(a,b)\}| = \delta_F(\mathcal{L}^{-1}(a,b)).\end{aligned}$$

Since $\mathcal{L}$ is a permutation, $\mathcal{L}^{-1}$ is a permutation as well and the differential spectra of Δ_F and $\Delta_{F'}$ is equal up to a permutation. Hence the differential spectrum is CCZ-invariant. □

We state a particular consequence.

Corollary 5.26. *The property APN is CCZ-invariant.*

We have seen that the Walsh spectrum and the differential spectrum are CCZ-invariant. Since the values of the Walsh spectrum and the differential spectrum determine the success of a linear or differential attack respectively, we have that CCZ-equivalent S-boxes have the same resistance against such attacks.

6 The Dillon-Wolfe Function

We have seen some criteria according a vectorial Boolean function to be selected as an S-box. Two of these criteria are: the (m, m)-function is a permutation and low values of the differential spectrum, ideally APN. There are many examples of APN permutations known for odd dimensions m, see for instance lemma 6.1 below, but for even dimension this was a big open question if such a function would exist. It was even conjectured by Hou in [12] that no APN permutation exits in even dimension.

For the first nontrivial case $m = 4$ it was proved by Hou [12] that this conjecture was true. For higher dimensions $m \geq 6$, no proof was known. If the conjecture is false and an APN permutation would exist, where do we need to look? The space of bijective (m, m)-functions is huge, i.e. $2^6! \simeq 2^{296}$, hence the first step is to narrow this space down. In which subspace could such a function occur and do we need to investigate? When the subspace is small enough we can do an exhaustive search.

Building on previous results of K.A. Browning and M.T. McQuistan [14], J.F. Dillon and A.J. Wolfe published the article [13] in 2009, in which they proved that the conjecture was false. Dillon and Wolfe found a vectorial Boolean $(6, 6)$-function which is an APN permutation. We will explain how Dillon and Wolfe found this so called Dillon-Wolfe function.

Dillon and Wolfe used a connection between Boolean functions and coding theory. This will be explained in the first section of this chapter. In the second section we give the general idea in which direction we need to search to find an APN permutation. And we give an algorithm to deduce an APN permutation from a known APN (m, m)-function. In the third section we give the results stated by Dillon and Wolfe and in the fourth section we give the cryptographic properties of a Dillon-Wolfe function and compare these to other S-boxes. When Dillon and Wolfe discovered the existence of an APN permutation, they tried to find such a function with a randomized search algorithm. This we will be explained in section five.

6.1 Boolean Functions and Coding Theory

There exist some connections between Boolean functions and (binary linear) codes. To show this we can use the identification of the vector space $\mathbb{F}_2^m$ with the finite field $\mathbb{F}_{2^m}$ as described in section 3.2.1 about Hamming codes. An element in $\mathbb{F}_{2^m}$ can be identified with a vector in $\mathbb{F}_2^m$ and using a primitive

element ω of $\mathbb{F}_{2^m}$ which generates the multiplicative group $\mathbb{F}_{2^m}^*$ of $\mathbb{F}_{2^m}$, i.e. $\langle\omega\rangle = \mathbb{F}_{2^m}^*$. This identification can be used to represent a vector Boolean (n, m)-function using the finite fields $\mathbb{F}_{2^n}$ and $\mathbb{F}_{2^m}$. We do not go into detail how to derive the ANF from this representation and vice versa, for more of this representation see Carlet [5]. We first use this representation to give an example of an APN function. Secondly, we show a relation between codes and APN functions and thirdly, we prove that CCZ-equivalent functions generate equivalent codes. Recall our convention for addition of vectors in $\mathbb{F}_2^m$, we write $+$ for addition in $\mathbb{F}_{2^m}$.

We give an example of a (m, m)-function which is APN.

Lemma 6.1. *The function $F : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$, $F(x) = x^3$, is APN for every $m \geq 1$ and a permutation for every odd m.*

Proof. We first prove that F is APN. For $a, b \in \mathbb{F}_{2^m}$, $a \neq 0$, we have

$$F(x) + F(x + a) = x^3 + (x + a)^3 = x^3 + x^3 + ax^2 + a^2x + a^3 = b.$$

Note that in the finite field $\mathbb{F}_{2^m}$ the factor 2 vanishes. The equation

$$ax^2 + a^2x + a^3 = b$$

admits at most two solutions since it is quadratic. Hence $F(x)$ is APN for all $m \geq 1$.

To prove that F is a permutation for odd m, we need to prove that F does not generate a subgroup of the multiplicative group $\mathbb{F}_{2^m}^*$ since $F(0) = 0$. That is, we want to prove that $3 \not| (2^m - 1)$. We have that

$$2^m - 1 \equiv (-1)^m - 1 \mod 3 \equiv \begin{cases} 0 \mod 3 & \text{if } m \text{ is even,} \\ 1 \mod 3 & \text{if } m \text{ is odd.} \end{cases}$$

It follows that $3 \not| (2^m - 1)$ if and only if m is odd, hence F is a permutation if m is odd. □

We recall the construction of definition 3.23. A vectorial Boolean (m, m)-function F generates the $2m \times 2^m - 1$ matrix H_F by

$$H_F = \begin{bmatrix} x \\ F(x) \end{bmatrix}.$$

The columns $(x, F(x))$ in H_F are nonzero and pairwise distinct. To do computations on H_F it is useful to give an order on the columns $(x, F(x))$. Using

the identification of $\mathbb{F}_{2^m}$ with $\mathbb{F}_2^m$ we can give an order on the columns of H_F by

$$H_F = \begin{bmatrix} x \\ F(x) \end{bmatrix} = \begin{pmatrix} 1 & \omega & \omega^2 & \dots & \omega^{2^m-2} \\ 1 & F(\omega) & F(\omega^2) & \dots & F(\omega^{2^m-2}) \end{pmatrix}.$$

The parity check matrix H_F defines the code C_F and is a generator matrix for the dual code $C_F^\perp$. Let the parameters n be the length of a code and k be the dimension of the code. Recall that if C_F is an $[n, k]$ code, then the dual code $C_F^\perp$ is an $[n, n-k]$ code. Therefore both codes are of length $n = 2^m - 1$. Since H_F consists of $2m$ rows the dimension of the code at least $k \geq 2^m - 1 - 2m$. If F is a permutation and $F(\mathbf{0}) = \mathbf{0}$, then by definition 3.23 $C_F^\perp$ is a double simplex code.

There is an equivalent definition of a APN function using the code C_F which was first stated in the paper about CCZ-equivalence of Carlet et al. [4]. We will not give the full proof since it uses some deep results from coding theory, but we give the main idea such that the connection becomes clear.

Proposition 6.2. *Let F be a vectorial Boolean function, $F : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$, with $F(0) = 0$. Let $\omega \in \mathbb{F}_{2^m}$ be a primitive element and let C_F be the code of length $2^m - 1$, defined by the $2m \times (2^m - 1)$ parity check matrix*

$$H_F = \begin{pmatrix} 1 & \omega & \omega^2 & \dots & \omega^{2^m-2} \\ 1 & F(\omega) & F(\omega^2) & \dots & F(\omega^{2^m-2}) \end{pmatrix}.$$

Then F is APN if and only if C_F has minimum distance 5.

Proof. The proof consists of two steps.

First step: We show that the code C_F has minimal distance $3 \leq d \leq 5$. By lemma 3.14 we have that d equals the minimal number of dependent columns of H_F. Since any two columns of H_F are distinct we have $d \geq 3$. The code C_F is an $[n, k, d]$ code with $n = 2^m - 1$ and $k \geq 2^m - 1 - 2m$. It is known that there exist no such codes with $d \geq 6$, see [4]. Hence the minimal distance of C_F is $3 \leq d \leq 5$.

Second step: We show that F is APN if and only if $d = 5$. We have seen in the section on coding theory that $c = (c_1, \dots, c_n) \in C_F$ if and only if $c(H_F)^T = 0$, so we have

$$c \in C_F \quad \Leftrightarrow \quad \sum_{i=1}^{n} c_i \omega^i = 0 \quad \text{and} \quad \sum_{i=1}^{n} c_i F(\omega^i) = 0.$$

Suppose $d = 3, 4$, then by lemma 3.14, there are three or four pairwise distinct columns $(\omega^i, F(\omega^i))$ of H_F which are linear dependent. That is, there exist four distinct elements $x, y, x', y' \in \mathbb{F}_{2^m}$ such that

$$x + y + x' + y' = 0 \quad \text{and} \quad F(x) + F(y) + F(x') + F(y') = 0.$$

For $d = 3$ one of these elements is zero and $F(0) = 0$. For $d = 4$ none element equals zero. Hence there exist $a, b \in \mathbb{F}_{2^m}$, $a \neq 0$, such that $a = x+y = x'+y'$ and $b = F(x) + F(y) = F(x') + F(y')$ and the set of equations

$$\begin{cases} x + y & = & a \\ F(x) + F(y) & = & b \end{cases}$$

admits four solutions. It follows from lemma 5.17 that F is not APN.

Conversely, if F is APN, then $d \neq 3, 4$. Therefore $d = 5$ if and only if F is APN. □

We can prove that two CCZ-equivalent (m, m)-functions generate equivalent codes. Recall definition 3.11, two $[n, k]$ codes C_1 and C_2, with generator matrices G_1 and G_2 respectively, are equivalent if

$$MG_1 = G_2P,$$

where M is an $k \times k$ invertible matrix and P is an $n \times n$ permutation matrix.

Theorem 6.3. *Let F, F' be two (m, m)-functions, $F(\boldsymbol{0}) = F'(\boldsymbol{0}) = \boldsymbol{0}$ and $H_F, H_{F'}$ the corresponding $2m \times (2^m - 1)$ generator matrices of the two codes $C_F^\perp, C_{F'}^\perp$ respectively. If F and F' are CCZ-equivalent functions, then $C_F^\perp$ and $C_{F'}^\perp$ are equivalent codes.*

Proof. Assume that F, F' are two (m, m)-functions which are CCZ-equivalent. Then there exist a linear permutation $\mathcal{L}$ on $\mathbb{F}_2^{2m}$ such that for the two corresponding graphs we have

$$\mathcal{L}(G_F) = G_{F'}.$$

Since $\mathcal{L}$ is a linear permutation, we can write $\mathcal{L}$ as a $2m \times 2m$ invertible matrix. Since $F(\mathbf{0}) = \mathbf{0}$ we have that each element of the graph $G_F \backslash \{(\mathbf{0}, F(\mathbf{0}))\}$ is a column of H_F, this holds for F' as well. Hence

$$\mathcal{L}H_F = H_{F'}P,$$

where P is some $(2^m - 2) \times (2^m - 1)$ permutation matrix which induces a permutation on the columns of $\mathcal{L}H_F$. Therefore the two codes generated by $H_F, H_{F'}$ are equivalent. □

We take a closer look at this construction. Recall that the linear permutation $\mathcal{L}$ is of the from $\mathcal{L} = (L_1, L_2)$ with $L_1, L_2 : \mathbb{F}_2^{2m} \to \mathbb{F}_2^m$ two linear functions. We can define $\mathcal{L}, L_1, L_2$ in terms of matrices and get

$$\mathcal{L} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \quad L_1 = (A\ B) \quad L_2 = (C\ D).$$

Where each A, B, C, D is an $m \times m$ matrix. With this notation we get

$$F_1(x) = L_1(x, F(x)) = Ax + BF(x),$$

$$F_2(x) = L_2(x, F(x)) = Cx + DF(x).$$

Apply the matrix $\mathcal{L}$ on H_F gives use

$$\mathcal{L}H_F = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{bmatrix} x \\ F(x) \end{bmatrix} = \begin{bmatrix} Ax + BF(x) \\ Cx + DF(x) \end{bmatrix} = \begin{bmatrix} F_1(x) \\ F_2(x) \end{bmatrix}.$$

If the functions F and F' are CCZ-equivalent, then $F_1(x)$ is a permutation and $F' = F_2 \circ F_1^{-1}$. Hence, for $F_1(x) = y$, we have

$$\begin{bmatrix} F_1(x) \\ F_2(x) \end{bmatrix} = \begin{bmatrix} y \\ F_2 \circ F_1^{-1}(y) \end{bmatrix} = \begin{bmatrix} x \\ F'(x) \end{bmatrix} P$$

where P is an $(2^m - 1) \times (2^m - 1)$ permutation matrix which induces a permutation on the columns.

The converse is true if the function F' is a permutation. By definition 3.23, these are precisely the functions for which the corresponding code $C_{F'}^{\perp}$ is a double simplex code.

Corollary 6.4. *Let F, F' and the corresponding codes $C_F^{\perp}$ and $C_{F'}^{\perp}$ as in theorem 6.3. If $C_F^{\perp}$ and $C_{F'}^{\perp}$ are equivalent codes and $C_{F'}^{\perp}$ is a double simplex code, i.e. if F' is a permutation with $F'(\boldsymbol{0}) = \boldsymbol{0}$, then the functions F and F' are CCZ-equivalent.*

Proof. Let H_F and $H_{F'}$ be the generator matrices of the corresponding codes. Since the two codes are equivalent there exist a invertible $2m \times 2m$ matrix M and a $(2^m - 1) \times (2^m - 1)$ permutation matrix P such that $MH_F = H_{F'}P$, i.e.

$$M \begin{bmatrix} x \\ F(x) \end{bmatrix} = \begin{bmatrix} x \\ F'(x) \end{bmatrix} P.$$

Since P induces a permutation on the columns of $H_{F'}$ and since the two functions x and $F'(x)$ are permutations which leaves zero fixed, we have that

$$\begin{bmatrix} x \\ F'(x) \end{bmatrix} P = \begin{bmatrix} F_1'(x) \\ F_2'(x) \end{bmatrix}.$$

Where F_1' and F_2' are again two permutations with $F_1'(\mathbf{0}) = F_2'(\mathbf{0}) = \mathbf{0}$. In particular we have that

$$M \begin{bmatrix} x \\ F(x) \end{bmatrix} = \begin{bmatrix} F_1'(x) \\ F_2'(x) \end{bmatrix},$$

where M is a linear permutation on $\mathbb{F}_2^{2m}$ and F_1' a permutation. It follows that $F' = F_2' \circ F_1'^{-1}$ and $M(G_F) = G_{F'}$. Hence the two functions F and F' are CCZ-equivalent. □

6.2 An APN Permutation

The general idea of Dillon and Wolfe is to start with a $(6, 6)$-function F which is APN but *not* a permutation. Suppose we find a function F' which is CCZ-equivalent to F. Then there exist two (m, m)-functions $F_1(x, F(x)), F_2(x, F(x))$ such that $F' = F_2 \circ F_1^{-1}$ and F_1 is a permutation. If we can find a CCZ-equivalent function F' such that the above holds *and* F_2 is a permutation, then F' is an permutation as well. By proposition 5.25 APN is CCZ-invariant, hence we have found an APN permutation F'. We give a more detailed description of this idea below.

6.2.1 A double simplex code

To find CCZ-equivalent functions we will use results from coding theory but first we make the following observation.

Lemma 6.5. *Let F be an (m, m)-function and a permutation. Then there exists an (m, m)-permutation F', with $F'(\boldsymbol{0}) = \boldsymbol{0}$, which is CCZ-equivalent to F.*

Proof. Let $a \in \mathbb{F}_2^m$ such that $F(a) = \mathbf{0}$ and define the linear permutation $A_2(x) = x + a$ on $\mathbb{F}_2^m$. Then, by definition 5.19, the two functions F and $F' = F \circ A_2$ are EA-equivalent, hence also CCZ-equivalent and $F'(\mathbf{0}) = \mathbf{0}$. □

Without loss of generality we will assume that the permutations F on $\mathbb{F}_2^m$ will vanish at zero, $F(\mathbf{0}) = \mathbf{0}$. This makes it possible to use some results of the previous section.

We recall some observations from coding theory about double simplex codes. By definition 3.22 a double simplex code is the direct sum of two different simplex codes. From corollary 3.20 it follows that a double simplex code $C^\perp$ is defined by the $2m \times (2^m - 1)$ generator matrix

$$H = \begin{bmatrix} F_1(x) \\ F_2(x) \end{bmatrix},$$

where F_1 and F_2 are two distinct permutations on $\mathbb{F}_2^m$ with $F_1(\mathbf{0}) = F_2(\mathbf{0}) = \mathbf{0}$. We denote such a double simplex code $C^\perp$ by

$$C^\perp = \langle F_1(x) \rangle \oplus \langle F_2(x) \rangle.$$

In particular if F is an (m, m)-function and an APN permutation with $F(\mathbf{0}) = \mathbf{0}$. By definition 3.23, F defines a $2m \times (2^m - 1)$ generator matrix H_F and a double simplex code $C_F^\perp$ with

$$H_F = \begin{bmatrix} x \\ F(x) \end{bmatrix} \quad \text{and} \quad C_F^\perp = \langle x \rangle \oplus \langle F(x) \rangle.$$

Recall from relation (2) that if we identify an (m, m)-function F with a function on $\mathbb{F}_{2^m}$, then for a primitive element ω of $\mathbb{F}_{2^m}$, the codewords of $\langle F(x) \rangle$ are of the form $[a \cdot F(\omega^i) : i \in \{0, 1, \ldots, 2^m - 2\}]$.

We use these double simplex codes to show how we can make a CCZ-equivalent function which is an APN permutation. We take the following steps:

Step 1 Let F be an APN (m, m)-function with $F(\mathbf{0}) = \mathbf{0}$, but not a permutation. Now, suppose that the code $C_F^\perp$ is nonetheless equivalent to a double simplex code. Then we can write the code $C_F^\perp$ as

$$\begin{aligned} C_F^\perp &= \langle x \rangle \oplus \langle F(x) \rangle \\ &= \langle F_1(x) \rangle \oplus \langle F_2(x) \rangle. \end{aligned}$$

Where F_1 and F_2 are two (m, m)-functions which are permutations with $F_1(\mathbf{0}) = F_2(\mathbf{0}) = \mathbf{0}$. We can define the (m, m)-function $F' = F_2 \circ F_1^{-1}$, which

is also a permutation with $F'(\mathbf{0}) = F_2 \circ F_1^{-1}(\mathbf{0}) = \mathbf{0}$. It follows that the two codes $C_F^\perp$ and $C_{F'}^\perp$ are equivalent, since

$$H_F = \begin{bmatrix} x \\ F(x) \end{bmatrix} = \begin{bmatrix} F_1(x) \\ F_2(x) \end{bmatrix} = \begin{bmatrix} x \\ F'(x) \end{bmatrix} P = H_{F'} P,$$

where P is a $(2^m - 1) \times (2^m - 1)$ permutation matrix which induces a permutation on the columns. Since $F(\mathbf{0}) = F'(\mathbf{0}) = \mathbf{0}$, it follows from corollary 6.4 that the functions F and F' are CCZ-equivalent. Hence the function F' is an APN permutation.

Step 2 How can we find such an equivalent double simplex code? If F and F' are CCZ-equivalent, then there is an linear permutation $\mathcal{L}$ on $\mathbb{F}_2^{2m}$ such that $\mathcal{L}(G_F) = G_{F'}$. We will analyze this $\mathcal{L}$ to see if we can produce such map ourself. As in the proof of theorem 6.3 we can view the map $\mathcal{L} = (L_1, L_2)$ as a matrix which we can multiply with the generator matrix H_F, we get

$$\mathcal{L}H_F = \mathcal{L} \begin{bmatrix} x \\ F(x) \end{bmatrix} = \begin{bmatrix} L_1 H_F \\ L_2 H_F \end{bmatrix} = \begin{bmatrix} F_1(x) \\ F_2(x) \end{bmatrix}.$$

If $\mathcal{L}H_F = H_{F'}$ generates a double simplex code, then each matrix $L_1 H_F$ and $L_2 H_F$ generates a different simplex subcodes. We have that $\mathcal{L}$ is of rank $2m$, since it is invertible. Therefore the matrices L_1, L_2 are of rank m.

Final step The two simplex subcodes generated by $L_1 H_F, L_2 H_F$ need to be different, hence the row spaces only have the zero codeword in common. We call such two codes *disjoint* since every (linear) code contains the zero codeword. Therefore, to find a CCZ-equivalence, we can generate a list of all simplex subcodes LH_F with L a $m \times 2m$ matrix of rank m and look for any two codes from this list which are disjoint. If we succeed, the code $C_F^\perp$ is equivalent to a double simplex code from which we can deduce a CCZ-equivalent function F' of F which is an APN permutation.

6.2.2 Simplex subcodes LH_F

We will deduce an algorithm which generates all simplex subcodes LH_F. Since the matrix L is of rank m and since the code LH_F needs to generate a simplex code, we can make two restrictions for our search.

First With linear algebra we have for an $m \times m$ invertible matrix S, that

$$(SL)H_F = S(LH_F).$$

Therefore we can let L be of reduced row echelon form. That is, we can take L of the form

$$L = (I_m \ B'),$$

where I_m is the $m \times m$ identity matrix and B' is a $m \times m$ matrix.

Second The $m \times (2^m - 1)$ matrix LH_F need to generate a simplex code, hence each column must be nonzero and pairwise distinct. That is, each column is nonzero and each sum of two columns is nonzero. To ensure this, we make the following restriction for L. We define the set

$$\Sigma := \{ \begin{bmatrix} x+y \\ F(x)+F(y) \end{bmatrix} : x, y \in \mathbb{F}_2^m, \ x \neq y \}$$

and consider matrices L which do not vanish on the set Σ, i.e. $Lv \neq \mathbf{0}$ for all $v \in \Sigma$.

Using these two restrictions we can generate L such that LH_F will be a simplex code. To do this we divide Σ into subsets Σ_k, $0 \leq k \leq 2m$, which consists of vectors of which the last $2m - k$ bits are zero, i.e.

$$\Sigma_k := \{c \in \Sigma : c = (c_1, \ldots, c_k, 0, \ldots, 0), \ c_k \neq 0\}.$$

The vectors in Σ are of length $2m$, but we can view a vector $c = (c_1, \ldots, c_k, 0, \ldots, 0) \in \Sigma_k$ as a vector of length k. The set Σ_0 is empty, since $(\mathbf{0}, \mathbf{0})$ is not contained in Σ.

We can generate L one column at the time working from left to right, i.e. $L = [\ldots, l_i, \ldots]$, $1 \leq i \leq 2m$. We start with m columns from the $m \times m$ identity matrix $I_m = [l_1, \ldots, l_m]$. Observe that for all vectors $c = (c_1, \ldots, c_k, 0, \ldots, 0) \in \Sigma_k$, $1 \leq k \leq m$, we have that $c_k \neq 0$ and therefore

$$[l_1, \ldots, l_k] \begin{bmatrix} c_1 \\ \vdots \\ c_k \end{bmatrix} \neq \mathbf{0}.$$

Therefore we are only interested in the sets $\Sigma_{m+1}, \ldots, \Sigma_{2m}$.

For each next column $l_j \in \mathbb{F}_2^m$, $m+1 \leq j \leq 2m$, we check for all vectors $(c_1, \ldots, c_j, 0, \ldots, 0) \in \Sigma_j$ if

$$[l_1, \ldots, l_{j-1}, l_j] \begin{bmatrix} c_1 \\ \vdots \\ c_j \end{bmatrix} \neq \mathbf{0}. \tag{4}$$

If such vector l_j exists, we repeat this step for l_{j+1}.

Relation (4) ensures that LH_F will be a generator matrix of a simplex code. If the equation equals zero, then there are two possibilities:

I. The vector $c = (c_1, \ldots, c_j, 0, \ldots, 0) \in \Sigma_j$ is a column of H_F. No matter how we complete L, LH_F will contain the zero column.

II. The vector c is of the form

$$c = \begin{bmatrix} x + y \\ F(x) + F(y) \end{bmatrix}.$$

Hence, no matter how we complete L, LH_F will contain two equal columns

$$[l_1, \ldots, l_{j-1}, l_j] \begin{bmatrix} x \\ F(x) \end{bmatrix} = [l_1, \ldots, l_{j-1}, l_j] \begin{bmatrix} y \\ F(y) \end{bmatrix}.$$

Hence, if we find an L, the matrix LH_F is of the form $LH_F = [F_1(x)]$ with; by I, $F_1(\mathbf{0}) = \mathbf{0}$ and by II, F_1 a permutation on $\mathbb{F}_2^m$. We summarize this construction in the following algorithm which can generate a list of all simplex codes.

Algorithm

Input: An (m, m)-function F and the corresponding matrix H_F.
Output: All $m \times 2m$ matrices $L = [l_1, \ldots, l_{2m}]$ such that LH_F generates a simplex code.

i. Compute the set Σ and order it into the subsets $\Sigma_1, \ldots, \Sigma_{2m}$.

ii. Set the first m columns of L equal to the $m \times m$ identity matrix, i.e. $[l_1, \ldots, l_m] = I_m$.

iii. Given $[l_1, \ldots, l_j]$, $m \leq j \leq 2m$, check for each $l_{j+1} \in \mathbb{F}_2^m$ if, for all $c = (c_1, \ldots, c_{j+1}, 0, \ldots, 0) \in \Sigma_{j+1}$,

$$[l_1, \ldots, l_{j+1}] \begin{bmatrix} c_1 \\ \vdots \\ c_{j+1} \end{bmatrix} \neq \mathbf{0}. \tag{5}$$

For each l_{j+1} such that (5) holds for all $c \in \Sigma_{j+1}$, start a branch by repeating this step for $[l_1, \ldots, l_{j+1}]$. If no vector $l_{j+1} \in \mathbb{F}_2^m$ exists, then quit the branch with $[l_1, \ldots, l_j]$.

iv. For each vector l_{2m} that is found in iii, output $L = [l_1, \ldots, l_{2m}]$.

The algorithm outputs one or more matrices L and each LH_F generates a simplex subcode of $C_F^{\perp}$. Note that the algorithm always outputs the matrix $L = [l_1, \ldots, l_m, \mathbf{0}, \ldots, \mathbf{0}]$ since $[l_1, \ldots, l_m, \mathbf{0}, \ldots, \mathbf{0}]H_F = [x]$ generates a simplex subcode. With this algorithm we obtain a list of all simplex subcodes of C_F.

We compare each pair of simplex subcodes to see if there exists two which are disjoint. If there are no two disjoint simplex subcodes, then it is proven that F is not CCZ-equivalent to a permutation with $F(\mathbf{0}) = \mathbf{0}$. Therefore, by lemma 6.5, if F is APN, then F is not CCZ-equivalent to an APN permutation.

Dillon and Wolfe did not give any indication on the computation time of this algorithm. We will show a general upper bound. The algorithm will produce a tree of results since each vector l_{j+1}, found at iii, induces a new branch. The number of times that relation (5) need to be checked depends on the number of branches at each iteration of step iii. Hence it is hard to determine a tight bound on the computation time. We computed an upper bound which is far from being strict.

Proposition 6.6. *Relation (5) needs to be checked at most* 2^{m^2+2m-1} *times.*

Proof. We prove the proposition in four observations.

I. We compute an upper bound for the cardinality of $\Sigma = \{(x, F(x)) + (y, F(y)) : x, y \in \mathbb{F}_2^m,\ x \neq y\}$. Recall that F is an (m, m)-function. For $x, y \in \mathbb{F}_2^m$, we have that $|\{(x, F(x)) + (y, F(y))\}| \leq 2^m \cdot 2^m$. We leave out all 2^m vectors with $x = y$, and divide by 2 since $(x, F(x)) + (y, F(y)) = (y, F(y)) + (x, F(x))$. Hence we have

$$|\Sigma| \leq \frac{2^{2m} - 2^m}{2} = 2^{2m-1} - 2^{m-1}.$$

II. We also compute an upper bound of the cardinality of the subsets Σ_k, $1 \leq k \leq 2m$. For a vector $c \in \Sigma_k$, the k-th bit is nonzero and all i-th bits $i > k$ are zero, i.e. $c = (c_1, \ldots, c_{k-1}, 1, 0, \ldots, 0)$. Hence the cardinality of Σ_k is at most 2^{k-1}. Note that $|\Sigma_{2m}| \leq 2^{2m-1} - 2^{m-1}$ since $\Sigma_{2m} \subseteq \Sigma$.

III. Let B_j be the number of branches after each j-th iteration of step iii of the algorithm. Then $B_j \leq 2^{mj}$, since at each iteration there can arise $|\mathbb{F}_2^m| = 2^m$ branches.

IV. If F is given, we can compute Σ and $\Sigma_{m+1}, \ldots, \Sigma_{2m}$. From III we deduce that $2^{jm}|\Sigma_{m+j}|$ is an upper bound on the number of times that relation (5) is checked at the j-th iteration. Hence the sum of the upper bound of each iteration $1 \leq j \leq m$ is an upper bound for the whole algorithm, that is

$$\sum_{j=1}^{m} 2^{jm}|\Sigma_{m+j}| = 2^m|\Sigma_{m+1}| + \cdots + 2^{m^2}|\Sigma_{2m}|. \tag{6}$$

Using the results from I and II we get a general upper bound R,

$$\sum_{j=1}^{m} 2^{jm}|\Sigma_{m+j}| \leq 2^m 2^m + \cdots + 2^{(m-1)m}2^{2m-2} + 2^{m^2}(2^{2m-1} - 2^{m-1}) =: R.$$

This bound R consists of m terms. We compare the last term to the first $m-1$ terms to prove the proposition. For the last term in this sum we have

$$2^{m^2}(2^{2m-1} - 2^{m-1}) = 2^{m^2+2m-1} - 2^{m^2+m-1}.$$

And for the first $m-1$ terms of the sum we have

$$\begin{aligned} & 2^m 2^m + 2^{2m}2^{m+1} + \cdots + 2^{(m-1)m}2^{2m-2} \\ = \; & 2^{m+m} + 2^{2m+m+1} + \cdots + 2^{jm+m+j-1} + \cdots + 2^{m^2+m-2} \\ = \; & \sum_{j=1}^{m-1} 2^{jm+m+j-1} \end{aligned}$$

Each term $1 \leq j \leq m-2$ is at least 2^{m+1} times smaller then the previous term 2^{m^2+m-2}, therefore we have

$$\sum_{j=1}^{m-1} 2^{jm+m+j-1} \leq (m-2)\left(\frac{2^{m^2+m-2}}{2^{m+1}}\right) + 2^{m^2+m-2} \leq 2^{m^2+m-1}.$$

From this we get an upper bound on R,

$$R = 2^{m^2+2m-1} - 2^{m^2+m-1} + \sum_{j=1}^{m-1} 2^{jm+j+m-1} \leq 2^{m^2+2m-1}.$$

This proves the upper bound of the number of times relation (5) needs to be checked. □

Note that this bound is far from being tight. Branches will pour out quickly hence the algorithm is done much faster. For a given function F relation (6) gives a bound which is tighter, especially when the subset Σ_{2m} does not reach its maximal cardinality.

Nonetheless proposition 6.6 shows that for dimension 6, relation (5) need to be checked 2^{47} times, which can be done by a strong computer. This is a considerably lower amount compared to checking all possible permutations $2^6! \simeq 2^{269}$.

6.3 A Dillon-Wolfe Function

Dillon and Wolfe have applied the algorithm on every function appearing on the Banff, Edel-Pott or MAGMA lists of all known CCZ-inequivalent APN functions of dimension $6, 8$ and 10. Unfortunately I could not find the articles of the Banff list and the MAGMA list. Carlet devotes a section in [6] on known APN functions including the so called "non power mappings" from Edel, Kyureghyan and Pott. Most results on the list of Carlet are recently discovered and it could very well be that since the publication of this list, new APN functions have been found. Brinkmann and Leander classified in [16] all APN functions up to dimension five. For dimension six and higher, it is not proven that all CCZ-classes of APN functions have been found.

The search of Dillon and Wolfe was successful in dimension 6, for the APN function

$$\kappa(x) = x^3 + x^{10} + ux^{24},$$

where u is a primitive element of $\mathbb{F}_{2^6}$ with minimal polynomial $f^u_{min} = x^6 + x^4 + x^3 + x + 1$. The APN function κ is called the Kim map, since it was discovered by Kim Browning[11].

The algorithm generated a list of 222 matrices LH_κ of which each matrix generates a different simplex subcodes of $C^\perp_\kappa$. There where two sets of 16 simplex subcodes for which any pair from different sets are disjoint. Hence they found $16 \times 16 = 256$ pairs of simplex subcodes for which each pair L_1H_κ, L_2H_κ generates a double simplex code by

$$\begin{bmatrix} L_1H_\kappa \\ L_2H_\kappa \end{bmatrix} = \begin{bmatrix} F_1(x) \\ F_2(x) \end{bmatrix}.$$

[11] This is stated by Dillon in [8] without a reference. In [13], Dillon calls the map the #5 map of the Banff list.

For each of the 256 pairs F_1, F_2, the functions $F' := F_2 \circ F_1^{-1}$ and $F'^{-1} = F_1 \circ F_2^{-1}$ defines an APN permutation. Hence Dillon and Wolfe have found 512 different APN permutations which are, by construction, all CCZ-equivalent to the Kim map κ.

We investigate the cryptographic properties of a Dillon-Wolfe function in the next section.

6.4 Cryptographic properties of a Dillon-Wolfe function

In [13], Dillon and Wolfe present the first APN permutation F_{WD} that they found in an 8×8 matrix. Such representation is called a *look up table*. The function F_{DW} can be seen as a permutation on $\{0, \ldots, 63\}$. The six bits binary representation of each integer is seen as a vector $(i, j) \in \mathbb{F}_2^3 \times \mathbb{F}_2^3$ and i, j can again be seen as an integer $0 \leq i, j \leq 7$. The value of $F_{DW}(i, j)$ can be found in the i-th row of the j-th column. The APN permutation F_{DW} is defined by the matrix

$$\begin{pmatrix} 0 & 54 & 48 & 13 & 15 & 18 & 53 & 35 \\ 25 & 63 & 45 & 52 & 3 & 20 & 41 & 33 \\ 59 & 36 & 2 & 34 & 10 & 8 & 57 & 37 \\ 60 & 19 & 42 & 14 & 50 & 26 & 58 & 24 \\ 39 & 27 & 21 & 17 & 16 & 29 & 1 & 62 \\ 47 & 40 & 51 & 56 & 7 & 43 & 44 & 38 \\ 31 & 11 & 4 & 28 & 61 & 46 & 5 & 49 \\ 9 & 6 & 23 & 32 & 30 & 12 & 55 & 22 \end{pmatrix}.$$

We used Mathematica to compute the ANF of this function and some cryptographic properties. With this, we can compare a Dillon-Wolfe function to other S-boxes to see if it is a cryptographically strong S-box.

6.4.1 ANF

The ANF of the $(6, 6)$-Dillon-Wolfe-function F_{DW} above, can be computed with Mathematica. We write $F_{WD} = (f_1, \ldots, f_6)$ where each f_i is a coordi-

nate function, $1 \leq i \leq 6$. The result is stated below.

$$
\begin{aligned}
f_1(x) &= x_1 \oplus x_2 \oplus x_5 \oplus x_6 \\
&\quad \oplus x_1x_3 \oplus x_1x_5 \oplus x_3x_5 \oplus x_3x_6 \\
&\quad \oplus x_1x_2x_4 \oplus x_1x_2x_5 \oplus x_1x_3x_5 \oplus x_1x_4x_5 \oplus x_2x_4x_5 \oplus x_3x_4x_5 \\
&\quad \oplus x_1x_2x_6 \oplus x_2x_3x_6 \oplus x_1x_4x_6 \oplus x_2x_4x_6 \oplus x_2x_5x_6 \oplus x_3x_5x_6 \\
&\quad \oplus x_1x_2x_3x_4 \oplus x_1x_2x_3x_5 \\
f_2(x) &= x_1 \oplus x_2 \oplus x_4 \oplus x_5 \\
&\quad \oplus x_1x_4 \oplus x_3x_4 \oplus x_3x_5 \oplus x_4x_5 \oplus x_3x_6 \oplus x_4x_6 \\
&\quad \oplus x_1x_2x_4 \oplus x_1x_3x_4 \oplus x_2x_3x_4 \oplus x_1x_2x_5 \oplus x_1x_3x_5 \\
&\quad \oplus x_1x_2x_6 \oplus x_1x_3x_6 \\
f_3(x) &= x_3 \oplus x_4 \oplus x_5 \\
&\quad \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3 \oplus x_1x_5 \oplus x_2x_5 \oplus x_3x_5 \oplus x_4x_5 \oplus x_1x_6 \oplus x_3x_6 \\
&\quad \oplus x_1x_3x_6 \oplus x_2x_3x_6 \oplus x_1x_4x_6 \oplus x_2x_4x_6 \oplus x_3x_4x_6 \oplus x_3x_5x_6 \\
&\quad \oplus x_1x_2x_3x_6 \\
f_4(x) &= x_1 \oplus x_3 \oplus x_6 \\
&\quad \oplus x_2x_4 \oplus x_3x_4 \oplus x_3x_5 \oplus x_4x_5 \\
&\quad \oplus x_1x_2x_4 \oplus x_2x_3x_4 \oplus x_1x_2x_5 \oplus x_1x_3x_5 \\
f_5(x) &= x_1 \oplus x_3 \oplus x_5 \oplus x_6 \\
&\quad x_1x_2 \oplus x_1x_3 \oplus x_2x_3 \oplus x_3x_5 \oplus x_4x_5 \oplus x_1x_6 \oplus x_2x_6 \oplus x_5x_6 \\
&\quad \oplus x_1x_3x_4 \oplus x_1x_3x_5 \oplus x_2x_4x_5 \oplus x_3x_4x_5 \oplus x_1x_2x_6 \oplus x_1x_3x_6 \\
&\quad \oplus x_1x_4x_6 \oplus x_2x_4x_6 \oplus x_3x_4x_6 \oplus x_3x_5x_6 \\
&\quad \oplus x_1x_2x_3x_6 \\
f_6(x) &= x_3 \oplus x_4 \oplus x_5 \oplus x_6 \\
&\quad \oplus x_1x_2 \oplus x_1x_3 \oplus x_3x_4 \oplus x_1x_5 \oplus x_2x_5 \oplus x_4x_6 \oplus x_5x_6 \\
&\quad \oplus x_2x_4x_5 \oplus x_1x_2x_6 \oplus x_2x_3x_6 \oplus x_1x_4x_6 \oplus x_1x_5x_6 \oplus x_3x_5x_6 \\
&\quad \oplus x_1x_2x_3x_5
\end{aligned}
$$

By definition 5.4, the algebraic degree of F_{DW} is equal the maximum algebraic degree of the coordinate functions.

Proposition 6.7. *The algebraic degree of every Dillon-Wolfe function is* 4.

With the ANF it is possible to compute the Walsh spectrum of F_{DW}. This can be done by applying the Fast Fourier Algorithm on F_{DW} in Mathematica.

This algorithm is described in section 4.2. In particular we can compute the maximum value of the Walsh transform of F_{DW}.

Proposition 6.8. *The maximum value of the Walsh transform of every Dillon-Wolfe function is* 16.

By proposition 5.23, the Walsh spectrum is invariant under CCZ-equivalence. Therefore all Dillon-Wolfe functions have the same maximum value of the Walsh transform.

Further more we have, by proposition 5.10, that the nonlinearity is directly related to the Walsh transform by

$$\mathcal{NL}(F) = 2^{m-1} - \frac{1}{2} \max_{a \in \mathbb{F}_2^m,\ b \in \mathbb{F}_2^m \setminus \{\mathbf{0}\}} |\lambda_{F_{DW}}(a, b)|.$$

Hence we have the following consequence.

Corollary 6.9. *The nonlinearity of every Dillon-Wolfe function equals* 24.

By lemma 5.13 the upper bound on the nonlinearity of $(6, 6)$-functions equals $\lfloor 32 - 2^{2\frac{1}{2}} \rfloor = 26$. Hence a Dillon-Wolfe function is considered to be highly nonlinear.

We summarize the properties of a Dillon-Wolfe function in dimension six.

- Permutation, in particular the function is balanced.
- The algebraic degree is 4.
- Almost perfect nonlinear, i.e. $\max\{\delta(a, b) \in \Delta_F\} = 2$.
- Nonlinearity is 24.

In the next section we will compare these cryptographic properties to other S-boxes.

6.4.2 Compared to other S-boxes

In [15], Kavut classifies $(6, 6)$-functions which are permutations, so called rotation-symmetric, and he gives their cryptographic properties. This is a class of functions which are considered as good S-boxes. For instance all power mappings $x^d : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$ are contained in this class, in particular the

so called *inverse functions* $x^{-1} = x^{2^m-2}$, for more about such mappings see Carlet [6].

The inverse functions have good cryptographic properties and are mostly chosen as an S-box, for instance in the Advanced Encryption Standard AES.

We list the cryptographic properties of the inverse function for even and odd dimension in table 3. We see that in odd dimension the inverse function can be AB and APN, only the algebraic degree is significantly lower as in even dimension. Nevertheless there is often chosen for an S-box in even dimension, this is mainly for implementation reasons, [15].

	m is even	m is odd
Permutation	yes	yes
Algebraic degree	$m-1$	$\frac{m+1}{2}$
$\max\{\delta(a,b) \in \Delta\}$	4	2
Nonlinearity	$2^{m-1} - 2^{\frac{m}{2}}$	$2^{m-1} - 2^{\frac{m-1}{2}}$

Table 3: Cryptographic properties of the inverse function.

In table 4 we compare the cryptographic properties of an inverse function in dimension six to a Dillon-Wolfe function. We see that a Dillon-Wolfe function has almost the same cryptographic properties as the inverse function, hence we can conclude that a Dillon-Wolfe function is a good candidate for an S-box.

	inverse function	Dillon-Wolfe
Permutation	yes	yes
Algebraic degree	5	4
$\max\{\delta(a,b) \in \Delta\}$	4	2
Nonlinearity	26	24

Table 4: The inverse function in dimension six compared to the Dillon-Wolfe function.

6.5 Additional Remarks

6.5.1 A randomized search algorithm

After Dillon and Wolfe discovered an APN permutation in dimension six, they tried to find one by a randomized search. To do this they used the properties of an extended (double) simplex code. We will give the general idea of the algorithm, but we will not go into detail. For more about this randomized search algorithm see the article of Dillon and Wolfe [13].

It can be proved that all nonzero codewords of a simplex code $C^{\perp}$ are of weight 2^{m-1} and therefore that all nonzero codewords of a extended simplex code $\overline{C^{\perp}}$ are of length 2^m and have weight 2^{m-1}, see Hall [11]. We call such codewords *balanced*.

Let F be an APN (m,m)-function, $F(\mathbf{0}) = \mathbf{0}$ and let $\overline{\overline{H_F}}$ be the generator matrix of the extended code $\overline{C_F^{\perp}}$, i.e.

$$\overline{C_F^{\perp}} = \{(a,b)\overline{\overline{H_F}} : (a,b) \in \mathbb{F}_2^m \times \mathbb{F}_2^m\}.$$

We define the set of all balanced codewords in $\overline{C_F^{\perp}}$ as

$$B := \{(a,b) \in \mathbb{F}_2^m \times \mathbb{F}_2^m : (a,b)\overline{\overline{H_F}} \text{ is balanced}\}.$$

Dillon and Wolfe claim that, since the map $(a,b) \mapsto (a,b)\overline{\overline{H_F}}$ is an isomorphism between $\mathbb{F}_2^m \times \mathbb{F}_2^m$ and $\overline{C_F^{\perp}}$ and therefore also an isomorphism between $\mathbb{F}_2^{2m}$ and $C_F^{\perp}$, we need to find two m-dimensional subspaces $S_1, S_2 \subset \mathbb{F}_2^m \times \mathbb{F}_2^m$ which have all nonzero elements in the set B and intersect only in $(\mathbf{0},\mathbf{0})$. The matrices $S_1\overline{\overline{H_F}}$ and $S_2\overline{\overline{H_F}}$ will then define two 'disjunct' extended simplex subcodes, hence S_1H_F and S_2H_F define two 'disjunct' simplex subcodes. We have seen that the functions $[F_1(x)] = S_1H_F$ and $[F_2(x)] = S_2H_F$ are permutations which leaves zero fixed, that F and $F' = F_2 \circ F_1^{-1}$ are CCZ-equivalent and that therefore F' will be an APN permutation.

Randomized search algorithm We can use the following randomized search algorithm to find S_1 and S_2. We write $\langle a_1, \ldots, a_i \rangle$ for the vector space spanned by the vectors $a_1, \ldots, a_i \in \mathbb{F}_2^{2m}$.

Input: An APN (m,m)-function F, with $F(\mathbf{0}) = \mathbf{0}$. An upper bound on the number of restarts at II and an upper bound on the total number of starts at I.
Output: Two 'disjoint' simplex subcodes of $C_F^{\perp}$, if they exist.

I. Compute the set $B = \{(a,b) \in \mathbb{F}_2^m \times \mathbb{F}_2^m : (a,b)\overline{\overline{H_F}} \text{ is balanced}\}$ and set $B_1 = B$.

II. Choose $b_1 \in B_1$ randomly.

III. Given $b_1, \ldots, b_i$, $1 \leq i \leq m$, choose $b_{i+1} \in B_1 \backslash \langle b_1, \ldots, b_i \rangle$ randomly such that $b_{i+1} + s \in B_1$ for all $s \in \langle b_1, \ldots, b_i \rangle$.

IV. If b_m is found, set $S_1 = \langle b_1, \ldots, b_m \rangle$ and proceed to step V.
If no b_i can be found, $1 \leq i \leq m$, make a restart at step II.

V. Set $B_2 = B \backslash S_1$.

VI. Repeat step II and III for the set B_2 instead of B_1.

VII. If b_m is found at step IV, then set $S_2 = \langle b_1, \ldots, b_m \rangle$.
If no b_i can be found, $1 \leq i \leq m$, make a restart at step VI.
If the upper bound on the number of restarts at II is reached, then start again at step I.
If the upper bound on the number of total starts at I is reached, then quit the algorithm.

Dillon and Wolfe applied this algorithm on the Kim map $\kappa(x)$ in dimension $m = 6$. They found that the set B of all balanced codewords in $\overline{C_\kappa^\perp}$ has cardinality $|B| = 1071$, but the solutions poured out quickly.

6.5.2 Decomposition

Dillon and Wolfe [13] discovered a surprisingly simple decomposition of the code $C_F^\perp$, where the function F is linear equivalent to the Kim map κ. We give an overview of this decomposition.

Recall that for a $(6,6)$-function F and the code $C_F^\perp$ with generator matrix H_F, every codeword $c \in C_F^\perp$ is of the form $c = (a,b)H_F$ for some $(a,b) \in \mathbb{F}_2^6 \times \mathbb{F}_2^6$. We identify the vector space $\mathbb{F}_2^6$ with the field $\mathbb{F}_{2^6}$ as we have seen in section 3.2.1. Using the trace function $\mathrm{Tr}(x)$ instead of the usual inner product "$\cdot$", we have for every $a, b \in \mathbb{F}_2^{2m}$ a codeword $c \in C_F^\perp = \langle x \rangle \oplus \langle F(x) \rangle$ of the form

$$c = [\mathrm{Tr}(ax) \oplus \mathrm{Tr}(bF(x))] \quad \text{where } x \text{ ranges over } \mathbb{F}_{2^6}.$$

Let u be a primitive element of $\mathbb{F}_{2^6}$ with minimal polynomial $f^u_{min} = x^6 + x^4 + x^3 + x + 1$, the Kim map is defined as

$$\kappa(x) = x^3 + x^{10} + ux^{24}.$$

The function Dillon and Wolfe used for the decomposition is $F(x) = u \cdot \kappa(x)$. Note that F and κ are linear equivalent and since $\kappa(x)$ is APN, $F(x)$ is APN as well.

Since u is a primitive element of $\mathbb{F}_{2^m}$ and $\{1, u, u^2, u^3, u^4, u^5\}$ forms a basis of $\mathbb{F}_{2^6}$, we can decompose this field as

$$\mathbb{F}_{2^6} = \mathbb{F}_{2^3} \oplus u\mathbb{F}_{2^3}.$$

So we can write $C_F^{\perp} = \mathcal{A} \oplus \mathcal{B}$, where

$$\mathcal{A} = \{[\mathrm{Tr}(ax)] : a \in \mathbb{F}_{2^6}\} \quad \text{and} \quad \mathcal{B} = \{\mathrm{Tr}(bF(x)) : b \in \mathbb{F}_{2^6}\}.$$

And using the decomposition $\mathbb{F}_{2^6} = \mathbb{F}_{2^3} \oplus u\mathbb{F}_{2^3}$, we can write

$$\mathcal{A} = \mathcal{A}_1 \oplus \mathcal{A}_2, \quad \mathcal{B} = \mathcal{B}_1 \oplus \mathcal{B}_2,$$

with

$$\begin{array}{rclcrcl} \mathcal{A}_1 & = & \{[\mathrm{Tr}(ax)] : a \in \mathbb{F}_{2^3}\}, & & \mathcal{A}_2 & = & \{[\mathrm{Tr}(ax)] : a \in u\mathbb{F}_{2^3}\}, \\ \mathcal{B}_1 & = & \{[\mathrm{Tr}(bF(x))] : b \in \mathbb{F}_{2^3}\}, & & \mathcal{B}_2 & = & \{[\mathrm{Tr}(bF(x))] : b \in u\mathbb{F}_{2^3}\}. \end{array}$$

Changing the pairs of the decomposition we get

$$\mathcal{A}_1 \oplus \mathcal{B}_1 \quad \text{and} \quad \mathcal{A}_2 \oplus \mathcal{B}_2.$$

We have obtained two codes in which all nonzero codewords are balanced, hence they are two simplex codes. The two functions $\langle F_1(x) \rangle = \mathcal{A}_1 \oplus \mathcal{B}_1$ and $\langle F_2(x) \rangle = \mathcal{A}_2 \oplus \mathcal{B}_2$ are permutations and F is CCZ-equivalent to $F_2 \circ F_1^{-1}$. Therefore the function $F_2 \circ F_1^{-1}$ is an APN permutation.

Dillon and Wolfe conclude that this surprisingly nice decomposition makes it likely that such a decomposition can be generalized to higher dimensions. Hence this needs to be investigated in the future.

6.6 Conclusion

Using the relation between coding theory and vectorial Boolean functions we deduced the construction of Dillon and Wolfe to show the existence of an APN permutation in dimension six. This so called Dillon-Wolfe function is the first known example of an APN permutation in even dimension.

We have seen the algorithm of Dillon and Wolfe in section 6.2.2. For a given APN (m, m)-function F, this algorithm finds all APN permutations which are CCZ-equivalent to F. Hence it can be applied to every known CCZ-equivalent class of APN functions. In particular we have seen that, in dimension six, this algorithm has a reasonable computation time.

Dillon and Wolfe applied the algorithm to all known APN functions, up to CCZ-equivalence, in dimension $6, 8$ and 10. Only the Kim map κ in dimension six produced an APN permutation. Dillon and Wolfe expect that the surprisingly nice structured decomposition, described in section 6.5.2, can be extended to higher dimensions. Once they discovered the existence of an APN permutation they developed a randomized search algorithm to find such functions.

We have further seen what a substitution permutation network SPN is and which cryptographic properties are needed for an S-box to make the SPN a strong cipher. The cryptographic properties of a Dillon-Wolfe function appeared to be very good, that is comparably good as the inverse function which is frequently used as an S-box. Therefore, when designing an SPN and an S-box need to be chosen, a Dillon-Wolfe function need to be considered.

References

[1] E. Biham and A. Shamir. Differential cryptanalysis of des-like cryptosystems. *Journal of Cryptology*, Vol 4(no.1):3–72, 1991.

[2] J.A. Buchmann. *An Introduction to Cryptography*. Undergraduate Texts in Mathematics. Springer-Verlag New York, second edition, 2003.

[3] L. Budaghyan. *The Equivalence of Almost Bent and Almost Perfect Nonlinear Functions and their Generalizations*. PhD thesis, University Magdeburg, 2005.

[4] V. Zinoviev C. Carlet, P. Charpin. Codes, bent functions and permutations suitable for des-like cryptosystems. *Designs, Codes and Cryptography*, 15(2):125–156, 1998.

[5] C. Carlet. Boolean functions for cryptography and error correcting codes. In Y. Crama and P.L. Hammer, editors, *Boolean Methods and Models in Mathematics, Computer Science, Engineering*, volume 134 of *Encyclopedia of Mathematics and it's Applications*, pages 257–395. Cambridge University Press, 2010.

[6] C. Carlet. Vectorial boolean functions for cryptography. In Y. Crama and P.L. Hammer, editors, *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, volume 134 of *Encyclopedia of Mathematics and it's Applications*, pages 398–469. Cambridge University Press, 2010.

[7] F. Chabaud and S. Vaudenay. Links between differential and linear cryptanalysis. *Proceedings of EUROCRYPT'94, Lecture Notes in Computer Science*, (950):356–365, 1995.

[8] J.F. Dillon. Apn polynomials: An update. Slides from a talk at the International Conference on Finite Fields and their Applications, july 2009. at the University College in Dublin.

[9] Federal Information Processing Standard Publication. *Announcing the Advanced Encrypting Standard (AES)*, 2001. Publication is available at http://csrc.nist.gov/publications/.

[10] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Number 17 in Algorithms and Combinatorics. Springer-Verlag, 1999.

[11] J.I. Hall. Notes on coding theory. Manuscript of a book, available at www.nth.msu.edu/ jhall.

[12] Xiang-Dong Hou. Affinity of permutations on $\mathbb{F}_2^n$. *Discrete Applied Mathematics*, 154:313–325, 2006.

[13] M.T. McQuistan A.J. Wolfe K.A. Browning, J.F. Dillon. An apn permutation in dimension six. *American Mathematical Society*, 518:33–42, 2010.

[14] R.E. Kibler M.T. McQuistan K.A. Browning, J.F. Dillon. Apn polynomials and related codes. *Journal of Combinatorics, Information and System Science, Special Issue in honor of Prof. D.K Ray-Chaudhuri on the occasion of his 75th birthday.* to appear.

[15] S. Kavut. Results on rotation-symmetric s-boxes. *Information Sciences*, 201:93–113, 2012.

[16] G. Leander M. Brinkmann. On the classification of apn functions up to dimension five. *Designs, Codes and Cryptography*, (49):273–288, 2008.

[17] M. Matsui. Linear cryptanalysis method for des cipher. *Advances in Cryptology - EUROCRYPT'93, Lecture Notes in Computer Science 765*, pages 386–397, 1994.

[18] C.E. Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28:656–715, 1949.

[19] A. Sinkov. *Elementary Cryptanalysis: a mathematical approach.* Mathematical Association of America, second printing edition, 1978.

[20] J.H. van Lint. *Introduction to Coding Theory.* Number 86 in Graduate Texts in Mathematics. Springer-Verlag Berlin Heidelberg New York, third edition, 1999.