Experimental construction of optimal cryptographic functions by expansion

Maren Hestad Aleksandersen

Department of Informatics
University of Bergen

2021

Acknowledgements

First of all, I would like to thank my supervisors, Lilya Budaghyan and Nikolay Kaleyski at the Selmer Center for guidance and support throughout my research.

A special thanks to Nikolay for helping me formulate my thesis, answering my questions and always being available through the process of writing my thesis. Thank you for always keeping an open door and making me feel welcome.

I would like to thank my fellow students at the Department of Informatics, especially Elise and Madelen for motivating me during my final year.

Last but not least, a big thanks to my family and friends, both at home and here in Bergen, for all the support and love throughout my studies.

Maren H. Aleksandersen

Bergen, 2021

Abstract

We document all APN functions over $\mathbb{F}_{2^n}$ for $n = 8, 9$ that can be obtained by adding a small number of quadratic terms with coefficients belonging to a certain subfield of $\mathbb{F}_{2^n}$ to one of the representatives from the known infinite APN families for $n = 8$ and $n = 9$. We discover one new APN function over $\mathbb{F}_{2^8}$, and find significantly shorter representatives of several already known APN instances over $\mathbb{F}_{2^8}$ and $\mathbb{F}_{2^9}$. We also introduce some theoretical simplifications that allow us to speed up the search (up to EA-equivalence) when the function to which terms are added is a monomial, and we document the running times of all of our searches. We conclude that this is a very promising approach that is worth exploiting further.

Contents

Acknowledgements i

Abstract iii

1 Introduction 1

2 Background 3
2.1 Vectorial Boolean functions . . . 3
2.2 Cryptographic properties . . . 8
2.2.1 Differential uniformity . . . 8
2.2.2 Nonlinearity . . . 10
2.3 Equivalence relations . . . 11
2.3.1 Linear and affine equivalence . . . 12
2.3.2 EA-equivalence . . . 12
2.3.3 CCZ-equivalence . . . 13
2.3.4 Testing equivalence relations . . . 14
2.3.5 Cyclotomic equivalence . . . 16
2.4 Invariants . . . 17
2.5 Survey of the known APN functions . . . 19
2.5.1 Infinite monomial APN families . . . 19
2.5.2 Infinite families of APN polynomials . . . 20
2.5.3 Sporadic APN instances . . . 20

3 Polynomial expansion 23
3.1 Basic notions . . . 23
3.2 Status quo . . . 25
3.3 Experimental setup . . . 25
3.4 Simplifying the search when the starting function is a monomial . . . 26

4 Computational results 29

5 Conclusion and future work 37

Chapter 1

Introduction

This thesis is dedicated to the class of almost perfect nonlinear (APN) functions, and some computational methods for searching for and classifying such functions. The study of APN functions is motivated, first and foremost, by their practical applications in the design of block ciphers in cryptography: APN functions provide the best possible security against the so-called differential attack, and so they allow us to design secure encryption algorithms. In general, strong encryption is crucial to the privacy of personal data, and this is the reason that methods of encryption have been a fundamental topic of study for a long time; for example, one of the most well-known block ciphers, the Data Encryption Standard (DES) was designed already in the late 70's (see e.g. [40]).

Finding APN functions is computationally difficult, and many search procedures and theoretical methods have been developed for doing so. One of the most basic but also most promising computational methods for searching for i.a. APN functions is called polynomial expansion; it is the main subject of this thesis.

Polynomial expansion has previously been used to find many CCZ-inequivalent instances of APN functions. Indeed, the first known examples of APN functions CCZ-inequivalent to monomials were discovered by a computational search of this type [26]. Later, the earliest known lists of CCZ-inequivalent APN representatives over $\mathbb{F}_{2^n}$ for $n \leq 8$ were also obtained in a similar fashion [6]. Recently, polynomial expansion was used as an auxiliary tool in the construction of an infinite family of APN quadrinomials [17]; in fact, this family generalized the APN binomial $x^3 + \beta x^{36}$ from [26] to an infinite construction (a problem that had been open since the publication of [26]).

In this thesis, we computationally investigate how far polynomial expansion can be pushed and what classes of APN functions can be obtained using it. We introduce some theoretical

simplifications that can be used to reduce the number of functions that have to be considered in a polynomial expansion search (up to EA-equivalence) when the initial function to which terms are added is a monomial. We run instances of polynomial expansion over $\mathbb{F}_{2^8}$ and $\mathbb{F}_{2^9}$ and carefully document the results in terms of running time and the number of functions that can be found in this way. We find one completely new APN function over $\mathbb{F}_{2^8}$. All the remaining functions that we find are CCZ-equivalent to known APN instances, but in many cases, the functions that we find have a significantly shorter representation than the currently known representatives from the same equivalence class. Furthermore, the previously known instances were found using complicated computational methods, e.g. [45], whereas our results rely on a simple computational principle. This shows that polynomial expansion is still a very promising technique for searching for APN functions, and it is worth investigating further.

In this thesis, we summarize the most important results of the search. The complete data from our experiments is available online at https://boolean.h.uib.no/mediawiki/index.php/APN_functions_obtained_via_polynomial_expansion_in_small_dimensions.

This thesis is organized as follows. In Chapter 2, we introduce the concept of APN functions, and present some related notions, definitions, and results that will be used throughout the rest of the text. In Chapter 3, we discuss the principle of polynomial expansion in some detail and describe the setup of our computational experiments. In Chapter 4, we give a detailed summary and discussion of our computational results. Finally, Chapter 5 provides a brief summary of our work and points out some directions for future study.

Chapter 2

Background

In this section, we discuss the background needed for the rest of the thesis. We introduce the notions of vectorial Boolean functions, their cryptographic properties, APN functions, and other important concepts, and we make a brief survey of important results in the area.

2.1 Vectorial Boolean functions

Let $\mathbb{F}_2$ denote the finite field with two elements. Let $\mathbb{F}_2^n$ denote the vector space of dimension n over $\mathbb{F}_2$ (for some natural number n), and let $\mathbb{F}_{2^n}$ denote the finite field of extension degree n over $\mathbb{F}_2$. For any two natural numbers n, m such that $m \mid n$, let Tr_m^n be the trace function from the finite field $\mathbb{F}_{2^n}$ to the finite field $\mathbb{F}_{2^m}$; we recall that the trace is defined as

$$\mathrm{Tr}_m^n(x) = \sum_{i=0}^{n/m-1} x^{2^{mi}}.$$

When $m = 1$, we will write Tr_n as shorthand for Tr_1^n.

An (n,m)**-function** (or **vectorial Boolean function**) is any function F from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$.

An (n,m)-function is a natural object which is why it occurs in many different areas in computer science and mathematics. The reason that these functions are so natural is that an (n,m)-function can be seen as a transformation of 0's and 1's. More precisely, an (n,m)-function takes a sequence of n bits as input, and produces a sequence of m bits as output. Since any data can be represented in binary, this means that any operation on any kind of data can be expressed as an (n,m)-function.

We will now look at the particular case when $m = 1$ which is one of the most important and well-studied subclasses of (n,m)-functions. We call an $(n,1)$-function an **n-dimensional Boolean function**. There are many ways to represent such a function, and the simplest way

is to represent it as a truth table (TT). A truth table lists the output value for every possible input. In this case, the output is either a 0 or a 1, and this can be interpreted as false or true, respectively. An example of a truth table of a (3,1)-function is given in Table 1.1. We thus have e.g. $f(0,1,0) = 1$, and $f(1,0,0) = 1$.

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

Table 1.1: Truth table of a (3,1)-function

As mentioned above, a Boolean function can only give one single bit as an output value. But for many practical applications (including the design of cryptographic primitives), having one bit as output is not enough. This is why we go back to the general case of (n,m)-functions for arbitrary values of m. One possible way to express functions having multiple bits as output is to combine the outputs of several Boolean functions into a vector; for this reason, (n,m)-functions with $m > 1$ are commonly called **vectorial Boolean functions**. Just like in the case of a Boolean function, we can use a TT to represent a vectorial Boolean function. We can see an example of such a function in Table 1.2, where we have 3 input and 2 output bits. To express this, we can define two Boolean functions, f_1 and f_2, that express the outputs of $F : \mathbb{F}_2^3 \rightarrow \mathbb{F}_2^2$ as a vector of two bits:

$$F(x_1,x_2,x_3) = (f_1(x_1,x_2,x_3), f_2(x_1,x_2,x_3))$$

As we can see in the TT (Table 1.2), the output is a vector of two bits. The first bit is the output of f_1, and f_2 gives the second output bit. Although this TT only consists of a vector of two bits, we can combine as many Boolean functions as we need into a multi-bit output vector.

Thus, any (n,m)-function F can be expressed as a vector

$$F(x_1,\ldots,x_n) = (f_1(x_1,\ldots,x_n),\ldots,f_m(x_1,\ldots,x_n)).$$

The functions $f_1, f_2, \ldots, f_m$ are called the **coordinates**, or **coordinate functions** of the function F. All non-zero linear combinations of the coordinate functions of a function F are called the **component functions** of F. Since any linear combination of the m coordinate functions can be identified with a vector from $\mathbb{F}_2^m$, the component functions of F are denoted by F_b for $b \in \mathbb{F}_2^m$ with $0 \neq b$. The component functions are needed in the definition of nonlinearity which we will discuss in a later section.

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

Table 1.2: Truth table of a (3,2)-function

Even though the TT is the simplest way to specify (n,m)-functions, it also has a lot of drawbacks that make other representations preferable. The TT of an (n,m)-function consists of 2^n entries, with each entry having m bits. The size of the TT increases exponentially with n, and it quickly becomes very large even for small values of n; we also need a lot of memory to store it. For example, for $n = 20$ we would need a table with $2^{20} = 1048576$ entries. Another shortcoming is that the TT reveals very little about the structure of the function. Because of this, other representations of vectorial Boolean functions can be more useful. One such representation is the algebraic normal form.

The **algebraic normal form** (ANF) of $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ is a polynomial in n variables, taking values in $\mathbb{F}_2^m$:

$$f(x_1,x_2,\ldots,x_n) = a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2 + \cdots + a_{2^n-1}x_1x_2\ldots x_n$$

with the coefficients $a_0, a_1, \ldots, a_{2^n-1}$ being in $\mathbb{F}_2^m$.

To see how the indices of the coefficients (for example, a_3) correspond to the terms (for example x_1x_2), we need to introduce some auxiliary notation. We know that we have n variables $x_1, x_2, \ldots, x_n$ and so every term of $f(x_1, x_2, \ldots, x_n)$ can be identified with a binary vector in n bits. For instance, if $n = 5$, the vector (0,0,1,0,1) would correspond to the term x_1x_3. This vector can also be interpreted as the binary expansion of an integer; more precisely (0,0,1,0,1) is the binary representation of 5, and so a_5 is the coefficient in front of the term x_1x_3.

The function from Table 1.1 has the ANF $f(x_1, x_2, x_3) = x_1 + x_2 + x_3$.

The algebraic degree can be obtained easily from the ANF and it is important that the degree is high in order to resist higher-order differential attacks [22, 36]. The **algebraic degree** of a function F is the size of the largest term with a non-zero coefficient in its ANF, and is denoted by $deg(F)$. Thus, it is trivial to compute the algebraic degree if we know the ANF. This is in contrast to the TT representation where there is no obvious way to compute the algebraic degree. Besides the cryptographic significance related to the higher-order differential attacks, the algebraic degree is also important because it allows us to define affine, linear and quadratic functions, and is also an invariant under EA-equivalence.

Although the ANF can have up to 2^n terms with non-zero coefficients, on average a lot of the terms have zero coefficients, which leads to a more compact representation than the TT. Even though it is often needed to find functions of high algebraic degree, quadratic functions can be very useful in a number of contexts as well. It is frequently easier to search for functions of lower algebraic degree computationally, or to prove properties of such functions.

An (n, m)-function of algebraic degree 1 is called **affine**, and has the property

$$F(x) + F(y) + F(z) = F(x + y + z)$$

for any $x, y, z \in \mathbb{F}_2^n$. A function F is called **linear** if F is affine and satisfies F(0) = 0. We thus have

$$F(x) + F(y) = F(x + y)$$

for any $x, y \in \mathbb{F}_2^n$. We can see that an affine function is just a linear function plus a constant, and all linear functions are affine, but not vice-versa. Since cryptographically strong functions should not have any apparent structure or property that can be exploited, affine functions are

generally not suitable for cryptographic use by themselves.

When a function has algebraic degree 2 it is called a **quadratic function**. These functions play an important role in the study of APN functions. This is because most of the known instances of APN functions are quadratic, or CCZ-equivalent to quadratic.

Any (n,n)-function can be uniquely represented as a univariate polynomial over $\mathbb{F}_{2^n}$ of the form

$$F(x) = \sum_{i=0}^{2^n-1} a_i x^i,$$

with coefficients $a_i \in \mathbb{F}_{2^n}$. This is called the **univariate representation**, and is often used instead of the ANF. This is because some known APN functions have a very simple univariate representation and almost all of the infinite families of APN functions are given in this representation. For instance, the Gold function $F(x) = x^3$ is APN over $\mathbb{F}_{2^n}$ for any natural number n. Here, an advantage is that the univariate representation is simple, with only one term, while the size of its ANF and TT increases with the dimension n. Another advantage we get from this representation besides the fact that it is more compact is that we can easily compute the algebraic degree. In contrast to this there is no easy way to do this from the TT. More precisely, the algebraic degree of $F(x) = \sum a_i x^i$ is the largest binary weight of an exponent i with a non-zero coefficient a_i in the univariate representation of F.

Another representation that is used in the study of vectorial Boolean functions is the **bivariate representation**, which can be used when the dimension n is even. It has similar advantages to the univariate representation, but since we do not use this in our research we do not go into details.

Another way to represent vectorial Boolean functions is using the values of their Walsh transform. The **Walsh transform** of an (n,m)-function F is the function $W_F : \mathbb{F}_{2^n} \times \mathbb{F}_{2^m} \to \mathbb{Z}$ defined by

$$W_F(a,b) = \sum_{x \in \mathbb{F}_{2^n}} (-1)^{\mathrm{Tr}_m(bF(x)) + \mathrm{Tr}_n(ax)},$$

where Tr_n is the trace from $\mathbb{F}_{2^n}$ to $\mathbb{F}_2$.

The values of the Walsh transform for all possible choices of a and b are called the **Walsh coefficients** of F. The multiset of all of the Walsh coefficients of a function F is called the **Walsh spectrum** of F. The multiset of all of their absolute values is called the **extended Walsh spectrum** of F.

Many important properties of vectorial Boolean functions (including some of their cryptographic parameters) can be expressed using the Walsh transform. What is more, the Walsh transform is invertible; that is, if we know the values of the Walsh transform $W_F(a,b)$ for all a,b, then we can uniquely reconstruct the function F. In this way, a table of all the values of the Walsh transform is yet another possible representation of (n,m)-functions.

2.2 Cryptographic properties

In cryptography, it is important to have functions that are secure and can resist various cryptanalyic techniques. There are many different types of attacks but two of the most powerful ones against block ciphers are the differential cryptanalysis and linear cryptanalysis. Different types of attacks exploit different types of weaknesses in the functions, and the security of the entire cipher depends on the properties of the functions used in it. If the function has a bad cryptographic quality, the attacker can exploit patterns and regularities in the behavior of the function and might be able to crack the cipher. So identifying (n,m)-functions with high cryptographic security is crucial in the design and analysis of cryptographic primitives. For instance, if we measure the differential uniformity of a function, we know how well it resists differential cryptanalysis. A typical goal of research in this area is then to find instances of functions having good values of various such cryptographic properties.

2.2.1 Differential uniformity

One of the most important cryptographic properties of a function is the differential uniformity. As mentioned above, it measures the resistance of the function to differential cryptanalysis.

Differential cryptanalysis is one of the most powerful known attacks that can be employed against block ciphers [3]. The basic idea behind differential cryptanalysis is to consider pairs of inputs (x_1,x_2) to the function, and study how the difference in the outputs $b = F(x_2) - F(x_1)$ depends on the difference in the inputs $a = x_2 - x_1$. If for a fixed input difference a, a certain output difference b is much more likely than any other output difference, the function behaves predictably in some sense, and it may be vulnerable to differential attacks.

In order to prevent this, we would like the output differences b among all pairs of inputs (x_1,x_2) with a fixed input difference $a = x_2 - x_1$ to be as uniformly distributed as possible. In order to measure this, we introduce the notion of the derivative.

Let F be an (n,n)-function for some natural number n. The **derivative** D_aF of F in **direction** $a \in \mathbb{F}_{2^n}$ is the (n,n)-function

$$D_aF(x) = F(a+x) - F(x).$$

Clearly, the value of the derivative $D_aF(x)$ expresses the difference between the outputs of F at $a+x$ and x; that is, it expresses the output difference for two inputs with difference a.

Note that in the case of binary finite fields and vector spaces, addition and subtraction are the same operation. Therefore, the derivative is usually written as

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

instead of $D_aF(x) = F(a+x) - F(x)$ as we wrote above.

Having introduced the derivative, we can define the numbers $\delta_F(a,b)$ that describe how many input pairs with difference a give an output pair with difference b. Formally, we let

$$\delta_F(a,b) = \#\{x \in \mathbb{F}_{2^n} : D_aF(x) = b\}.$$

As discussed above, we want the values of $\delta_F(a,b)$ for any fixed value of $a \in \mathbb{F}_{2^n}$ to be as uniformly distributed as possible; in other words, we want $\delta_F(a,b)$ to be as small as possible for any choice of a and b. Note that $a = 0$ is an exception, since if the difference between x_1 and x_2 is 0, then $x_1 = x_2$, and so $F(x_1) = F(x_2)$; so if the input difference is 0, then the output difference is always 0 as well. This motivates the definition of the **differential uniformity** δ_F of F as the largest value of $\delta_F(a,b)$ over all meaningful choices of a and b; that is,

$$\delta_F = \max\{\delta_F(a,b) : a,b \in \mathbb{F}_{2^n}, a \neq 0\}.$$

A related notion is that of the differential spectrum of F, which is simply the multiset of all possible values of $\delta_F(a,b)$. In other words, the **differential spectrum** of an (n,n)-function F is the multiset $\{\delta_F(a,b) : a \in \mathbb{F}_{2^n}, b \in \mathbb{F}_{2^n}\}$.

Since we have 2^n inputs $x \in \mathbb{F}_{2^n}$ and 2^n possible output difference $b \in \mathbb{F}_{2^n}$, we would ideally like $\delta_F(a,b)$ to be equal to 1 for any choice of a and b (so that we would obtain $\delta_F = 1$). Unfortunately, this is not possible, since if $D_aF(x) = b$ for some $a,b,x \in \mathbb{F}_{2^n}$, then $D_aF(a+x) = b$ as well. This means that the smallest possible value of δ_F is 2 for any (n,n)-

function F.

Functions attaining this best possible value do exist, and are called almost perfect nonlinear. More formally, we say that an (n,n)-function F is **almost perfect nonlinear (APN)** if $\delta_F = 2$. Clearly, the differential spectrum of an APN function will only contain the values 0 and 2.

2.2.2 Nonlinearity

Another very important cryptographic property is the **nonlinearity**. This is a parameter which shows how well a function performs against linear cryptanalysis [38].

The basic idea behind linear cryptanalysis is as follows. Consider a cryptographic cipher having a vectorial Boolean function as part of its design. The attacker constructs a second cipher which is exactly the same as the original one, but he replaces the function targeted by the attack with an affine (or even linear) function. Linear and affine functions are never used for cryptographic purposes by themselves because they are predictable. So if we build a cipher around an affine function, it is likely to be cracked very easily. Thus, the attacker is able to crack the modified cipher where he has replaced the original function with an affine function. This is not terribly useful by itself because this is a different cipher; but if the original function happens to be close to the affine function that the attacker used, then he could be able to get information about the original cipher.

In order to measure how close two functions are to one another, we use the notion of **Hamming distance**. Recall that the Hamming distance is defined as

$$d_H(F,G) = \#\{x \in \mathbb{F}_2^n : F(x) \neq G(x)\}.$$

If the Hamming distance between two functions is low, we will consider the functions as being close to one another. Using the Hamming distance, one can define a cryptographic parameter called nonlinearity which measures how well a given function can resist linear cryptanalysis.

In order to define the nonlinearity of (n,n)-functions we first need to define it for Boolean functions. The nonlinearity of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is the minimum Hamming distance between f and any affine Boolean function on the same number of variables.

Having defined nonlinearity for Boolean functions we can now extend it to vectorial

Boolean functions. In the case of an (n,n)-function F, linear cryptanalysis can be successful even if F is far away from all affine (n,n)-functions, provided that one of its component functions is close to an affine $(n,1)$-function. For this reason, the nonlinearity NL(F) of a vectorial Boolean function F is defined as the minimum nonlinearity across all of its component functions; in other words, we define

$$NL(F) = min\{d_H(F_b,l) : b \in \mathbb{F}_{2^n}, b \neq 0, l \in A_n\},$$

where A_n is the set of all affine $(n,1)$-functions.
We want the nonlinearity to be as high as possible. Unlike in the case of differential uniformity where we can see that δ_F is at least 2 for any F, here it is not obvious how high the nonlinearity can be.

However, we do know (see e.g. [19]) that the nonlinearity of any (n,n)-function F satisfies

$$NL(F) \leq 2^{n-1} - 2^{(n-1)/2}. \tag{2.1}$$

The **almost bent (AB) functions** are the ones that achieve this upper bound with equality and provide the best resistance against linear cryptanalysis. From equation (2.1) we can see that AB functions only exist for odd values of n.

Recall that APN functions are optimal against differential cryptanalysis. It can also be shown that any AB function is APN, so AB functions provide the best resistance to both differential and linear attacks. Even though any AB function is APN [21], not every APN function is AB. We do know, however, that any quadratic APN function is AB when the dimension n is odd [20].

2.3 Equivalence relations

Classifying (n,m)-functions can be a difficult problem since the number of such functions grows exponentially as n increases and becomes too large to manage even for small values of n. Equivalence relations can be used to reduce the number of functions that have to be considered, provided that these equivalence relations preserve the cryptographic properties of the functions that we are interested in. Equivalence relations on (n,n)-functions can be defined in different ways. In our case, we are interested in equivalence relations that preserve,

for instance, the differential uniformity and the nonlinearity of the functions. Furthermore, it is typically better to use more general equivalence relations because this leads to a greater reduction in the number of equivalence classes that have to be studied.

Differential uniformity and nonlinearity are invariant under i.a. affine, EA-, and CCZ-equivalence. Some other important properties are not invariant under EA- and CCZ-equivalence, however. For instance, CCZ-equivalence does not preserve the algebraic degree; EA-equivalence, on the other hand, does. Equivalence relations can also be used to derive new functions from known ones.

In the following section, we introduce some of the most commonly used equivalence relations on vectorial Boolean functions, namely: linear and affine equivalence; EA-equivalence; CCZ-equivalence; and cyclotomic equivalence (in the case of power functions).

2.3.1 Linear and affine equivalence

One of the simplest notions of equivalence that can be defined on (n,m)-functions is the so-called linear equivalence. We say that two (n,n)-functions F and G are **linear-equivalent** if there exist linear permutations L_1 and L_2 of $\mathbb{F}_{2^n}$ such that

$$L_1 \circ F \circ L_2 = G. \tag{2.2}$$

While being very simple, linear equivalence is not commonly used in the study of cryptographic functions because we know more general equivalence relations that still preserve the properties of interest. One of these more general relations is the so-called affine equivalence, which is defined in the same way as linear equivalence, except that the permutations L_1 and L_2 in (2.2) are allowed to be affine (instead of just linear). More formally, we say that two (n,n)-functions F and G are **affine equivalent** if there exist affine permutations A_1, A_2 of $\mathbb{F}_{2^n}$ such that

$$A_1 \circ F \circ A_2 = G. \tag{2.3}$$

2.3.2 EA-equivalence

However, there are even more general equivalence relations that are used in the study of APN functions. For instance, affine equivalence can be generalized into extended affine equivalence

(or EA-equivalence, for short), which is one of the most commonly used equivalence relations in this area. The only difference between affine equivalence and EA-equivalence is that we allow an affine function (not necessarily a permutation) to be added to F. More precisely, we say that F and G are **EA-equivalent** if there exist affine permutations A_1, A_2 of $\mathbb{F}_{2^n}$, and an affine (n,n)-function A such that

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

It is not difficult to see that EA-equivalence preserves the differential uniformity, the nonlinearity, and the algebraic degree (among other properties) of (n,n)-functions (see e.g. [19]). For a long time, EA-equivalence was the standard equivalence relation used for classifying APN functions because it was the most general known equivalence relation that preserves differential uniformity; however, we now know that CCZ-equivalence is strictly more general than EA-equivalence (even if combined with taking inverses of permutations) while still preserving the differential uniformity and nonlinearity [16], and so APN functions are now typically classified up to CCZ-equivalence.

Despite this, we know that EA-equivalence coincides with CCZ-equivalence in some particular cases. Most importantly, we know that two quadratic APN functions are CCZ-equivalent if and only if they are EA-equivalent [42]. Since most of the APN functions that we currently know are quadratic (or CCZ-equivalent to quadratic), this makes EA-equivalence almost as important for the classification of APN functions as CCZ-equivalence. On the other hand, EA-equivalence is somewhat easier to work with than CCZ-equivalence, and for this reason it remains one of the most frequently used relations on vectorial Boolean functions.

2.3.3 CCZ-equivalence

The most general known equivalence relation on (n,n)-functions that preserves differential uniformity and nonlinearity is the **Carlet-Charpin-Zinoviev equivalence (CCZ-equivalence)**. This is why it is the most frequently used one in the study of APN functions. We say that two vectorial Boolean functions F and G are CCZ-equivalent if there exists an affine permutation that maps the graph $\{(x,F(x)) : x \in \mathbb{F}_{2^n}\}$ of F to the graph $\{(x,G(x)) : x \in \mathbb{F}_{2^n}\}$ of G.

It can be seen that EA-equivalence is a special case of CCZ-equivalence, but CCZ-

equivalence is strictly more general (even if combined with taking inverses of permutations) [16].

An important difference between CCZ-equivalence and EA-equivalence is that CCZ-equivalence does not preserve the algebraic degree or bijectivity. This allows us, for instance, to construct APN functions of higher algebraic degree by exploring the CCZ-equivalence class of quadratic APN functions.

One of the most remarkable constructive applications of CCZ-equivalence is a result due to Dillon in which he constructs an APN permutation in dimension 6 [7]. Much like in the case of algebraic degree, CCZ-equivalence does not preserve bijectivity, and so it is possible to take some known APN function (which is not a permutation itself), and to search for permutations in its CCZ-equivalence class. Since CCZ-equivalence does preserve APN-ness, all such functions are automatically APN themselves. This is essentially what Dillon and his collaborators did; and to date, this is the only known APN permutation on an even number of variables. Finding APN permutations in 8, 10, etc. variables (or showing that such do not exist) is arguably the most important open problem in the study of APN functions at the moment.

2.3.4 Testing equivalence relations

Since CCZ-equivalence preserves differential uniformity (and hence APN-ness), APN functions are typically classified up to CCZ-equivalence. This makes the study of APN functions simpler because we have fewer functions to consider. On the other hand, finding new APN functions becomes more difficult in a sense, since for every new APN function that we find, we have to verify that it is not CCZ-equivalent to any of the known ones. Thus, in order to find new APN functions and constructions, we need a way to test whether two given functions are CCZ-equivalent.

Unfortunately, the definition of CCZ-equivalence does not suggest any meaningful way to check whether some given functions F and G are CCZ-equivalent besides conducting an exhaustive search over all possible affine permutations. It turns out that testing the CCZ-equivalence between two functions is a very hard computational problem; and, to date, the only known way to do so relies on the isomorphism between linear codes.

Given an (n,n)-function F, we can associate with it a linear code C_F. In order to do so, we

first define a matrix M_F associated with F as

$$M_F = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & \alpha & \alpha^2 & \cdots & \alpha^{2^n-2} \\ F(0) & F(1) & F(\alpha) & F(\alpha^2) & \cdots & F(\alpha^{2^n-2}) \end{pmatrix},$$

where α is a primitive element of $\mathbb{F}_{2^n}$. We then use M_F as a parity-check matrix to define the code C_F. We then know that two (n,n)-functions F and G are CCZ-equivalent if and only if the codes C_F and C_G are isomorphic [6, 28].

The benefit of this is that the problem of the isomorphism of linear codes has been studied for a long time, and algorithms for solving it have already been developed. In particular, the Magma algebra system [4] that we use for our computations has a built-in implementation of such an algorithm. In this way, implementing a CCZ-equivalence test in Magma amounts to constructing the matrix M_F, defining the code C_F, and then running this built-in implementation.

Unfortunately, this approach has several problems. On the one hand, it requires a lot of memory, and performing the test is only possible for dimensions up to $n = 10$ (or $n = 11$ in some cases). Furthermore, if the implementation runs out of memory, it outputs "false", which is precisely the same output that it produces if the functions being tested are CCZ-inequivalent. This means that the test can give false negatives (especially in large dimensions), which is definitely something that we would like to avoid. Finally, the running times in larger dimensions can be rather long, although this is typically not such a big problem (at least, when compared to the false negatives).

In practice, we can often substitute a test for CCZ-equivalence with one for EA-equivalence. This is because two quadratic APN functions are CCZ-equivalent if and only if they are EA-equivalent [42], and because most of the known APN instances and constructions of APN functions are quadratic.

We can test EA-equivalence using linear codes [28]. The idea of the algorithm is the same as that for CCZ-equivalence, but it also has the same problem of the large memory consumption and the false negatives. We know two more algorithms for testing EA-equivalence without going through linear codes. In both cases, invariants are used to obtain restrictions and reduce the size of the search space when trying to find the equivalence between the functions. The first one is introduced in [32]. This approach is efficient for APN functions in even dimensions

n. Another algorithm testing EA-equivalence for quadratic functions is developed by Canteaut et al. [18]. This algorithm uses a similar approach, and is based on an invariant called the Jacobian matrix to restrict the search space. This approach only works for quadratic functions, but is effective for dimensions of any parity.

2.3.5 Cyclotomic equivalence

One very special equivalence notion that can be used for power functions is **cyclotomic equivalence**. Despite the fact that it is very specialized, it is extremely useful because it is simple to test.

If we have two (n,n)-power functions $F(x) = x^a$ and $G(x) = x^b$ where a,b,n are some natural numbers, then F and G are cyclotomic equivalent if there exists a natural number k such that

$$2^k \cdot a \equiv b \pmod{2^n - 1},$$

or

$$2^k \cdot a^{-1} \equiv b \pmod{2^n - 1},$$

where a^{-1} is the multiplicative inverse of a modulo $2^n - 1$ (if it exists).
To check whether two functions are CCZ-equivalent is difficult to do, both theoretically and computationally. But in the case of cyclotomic equivalence this is very easy. If two power functions are CCZ-equivalent they are necessarily cyclotomic equivalent as well [43]. So in the particular case of power functions, CCZ-equivalence reduces to cyclotomic equivalence. Thus, all that we have to do in that case is to solve a small number of modular equations; and if there is no solution, then the functions in question are inequivalent.

Although cyclotomic equivalence can only be applied to monomial functions and it seems that this makes it too restrictive to be of use, monomial functions are some of the most important APN functions in a number of ways. To begin with, monomial APN functions are the first examples of APN functions and of infinite constructions of APN functions. At the moment, the only known infinite APN families that are not quadratic consist of monomials.

2.4 Invariants

Invariants are properties or values that are preserved under equivalence. Suppose that P is a property that can be computed for any function F. If this P is an invariant for CCZ-equivalence, for instance, this means that for two functions F and G that are CCZ-equivalent the value of $P(F)$ and the value of $P(G)$ must necessarily be the same.

The differential uniformity is a well known CCZ invariant, which means that if F and G are CCZ-equivalent then the differential uniformity of F is equal to the differential uniformity of G. Therefore, if F is APN, then G is APN. This is the reason that CCZ-equivalence can be used for classifying APN functions.

Invariants offer a good way to show that two functions are inequivalent in some cases. Using invariants, we can prove that two functions are inequivalent, but we can never prove that they are equivalent. To prove that two functions are equivalent we have to use a special kind of test.

Another nice thing about invariants is that since they are concrete values (for instance, numbers), we can precompute their values for some given functions. Although this might take awhile, we can compute a table containing the values of some given invariant for all the known APN functions. Then, if we want to compare some given function for equivalence against the known ones, we can compute its value of the invariant, and then compare it to all the precomputed values in the table. If the value does not match anything in the table, then the function is definitely new; and if it does match some of the values in the table, then we only have to conduct further equivalence tests with those functions from among the known ones that have the same value of the invariant. This speeds up testing the equivalence for new functions.

As mentioned earlier, differential uniformity and nonlinearity are invariants under CCZ-equivalence. However, APN functions and AB functions have fixed values of both of these. APN functions have by definition differential uniformity equal to 2, and AB functions have a nonlinearity equal to $2^{n-1} - 2^{(n-1)/2}$. So in the study of APN functions and AB functions, these invariants cannot be used to distinguish between inequivalent functions.

Invariants based on combinatorial designs that are associated with the functions are introduced in [27]. They are called the Γ-rank, Δ-rank and the multiplier group. They can be easily computed in Magma.

According to the survey in [33] for all the known at the time 491 APN functions in dimension $n = 7$, the Γ-rank can take 14 distinct values, the Δ-rank can take 6 distinct values and the order of the multiplier group can take 5. Together, they can take 20 triplets of values. This means that these invariants can be rather useful for proving inequivalence of functions.

The extended Walsh spectrum is also an invariant, but it can only take e.g. 6 distinct values across all the more than 20 000 known APN functions in dimensions $n = 8$, so it is not a good invariant for the purpose of distinguishing inequivalent functions. On the positive side, it can be computed very quickly, and it can be useful in other ways.

Another invariant is the orthoderivative, which is a function π_F that can be associated with any quadratic APN function F. The EA-equivalence class of the orthoderivative is an invariant under EA-equivalence, and computing and comparing the Walsh spectrum and differential spectrum of the orthoderivative is almost as good as an EA-equivalence test in practice [18]. Another advantage of the orthoderivates is that they are fast to compute.

An **orthoderivative** of a quadratic (n,n)-function F is any non-zero (n,n)-function π_F such that for any $a \in \mathbb{F}_{2^n}$ we have

$$\pi_F(a) \cdot (F(x) + F(a+x) + F(a) + F(0)) = 0$$

for all $x \in \mathbb{F}_{2^n}$, with $\pi_F(0) = 0$ and $\pi_F(a) \neq 0$ for all $0 \neq a \in \mathbb{F}_{2^n}$. Here, "$\cdot$" denotes a scalar product on $\mathbb{F}_{2^n}$; we can take $a \cdot b = \mathrm{Tr}_n(ab)$ without loss of generality.

A quadratic function F has a uniquely defined orthoderivative if and only if F is APN. Furthermore, we know that if F and G are EA-equivalent quadratic APN (n,n)-functions, then their orthoderivatives π_F and π_G are EA-equivalent as well. This allows us to show that F and G are not EA-equivalent to one another by examining the values of the invariants of π_F and π_G (instead of those of F and G themselves). This makes invariants that are not very useful for classifying APN functions by themselves (such as the extended Walsh spectrum, or the differential uniformity) useful. For instance, we already remarked above that the Walsh spectrum can take 6 distinct values for all known APN functions for $n = 8$. On the other hand, the Walsh spectra and differential spectra (taken together) of the orthoderivatives of all EA-inequivalent quadratic APN functions for $n = 8$ are distinct. If we consider only e.g. the differential spectra, then collisions are very rare. In fact, the differential spectrum of the orthoderivative is the main invariant that we use for the classification of the functions that we find in our computational searches.

Family	Exponent	Conditions	Algebraic degree	Source
Gold	2^i+1	$\gcd(i,n)=1$	2	[29, 39]
Kasami	$2^{2i}-2^i+1$	$\gcd(i,n)=1$	$i+1$	[30, 35]
Welch	2^t+3	$n=2t+1$	3	[24]
Niho	$2^t+2^{t/2}-1$, t even $2^t+2^{(3t+1)/2}-1$, t odd	$n=2t+1$	$(t+2)/2$ $t+1$	[23]
Inverse	$2^{2t}-1$	$n=2t+1$	$n-1$	[2, 39]
Dobbertin	$2^{4i}+2^{3i}+2^{2i}+2^i-1$	$n=5i$	$i+3$	[25]

Table 2.1: Known infinite families of APN power functions over $\mathbb{F}_{2^n}$

2.5 Survey of the known APN functions

In this section, we give a brief survey of the known APN functions. The research that we present in the remainder of this thesis concentrates on documenting the range of the CCZ-inequivalent APN functions that can be obtained using a method called polynomial expansion, and so it is important to have a good overview of the APN instances that are already known.

The known APN functions can be split into two main categories: infinite families, and sporadic instances. The infinite families are theoretical constructions that allow us to find APN functions for infinitely many dimensions by substituting parameters in a general formula. The sporadic instances are APN functions that do not belong to any of the currently known infinite families. Such instances are typically found by computational searches.

2.5.1 Infinite monomial APN families

A power function, or monomial function, is a function whose univariate representation is a polynomial of the form $F(x)=x^d$ for some natural number d. An example of how nice such a function can be is the Gold function F(x) = x^3. It has, as we can see, a very simple representation which has only one term. This function is APN over all dimensions and it does not have any parameters. Functions like this are called exceptional, i.e. an exceptional function is one that is APN over infinitely many dimensions. In fact, the function x^3 is part of a more general construction known as the Gold family which is one of the families represented in Table 2.1. Power APN functions are also some of the earliest known APN functions.

A summary of the known infinite families of APN monomials is given in Table 2.1. A conjecture of Dobbertin states that any APN power function is CCZ-equivalent to one of the instances from Table 2.1 [25].

2.5.2 Infinite families of APN polynomials

While the infinite monomial APN families represent some of the earliest known and most remarkable APN functions, we can find a lot more functions that have a univariate form consisting of more than a single term. Many of these functions still have a pleasant univariate form (consisting of a very small number of terms), but some of them exhibit properties that none of the monomial functions do.

Currently, we know 14 infinite polynomial APN families, which are summarized in table 2.2.

2.5.3 Sporadic APN instances

Sporadic APN functions are the ones that have not yet been classified into an infinite family. We know a lot of sporadic instances up to dimension $n = 10$. For instance, for $n = 8$ we know more than 20 000 functions [1, 46].

These more than 20 000 instances show that the infinite families that we know are barely "scratching the surface". Sporadic instances are also worth investigating because they have properties that none of the APN functions from the known infinite families do. Just to consider the most interesting examples: the function discovered in [27] is the only known APN function that is CCZ-inequivalent to both quadratic functions and monomials; and Dillon's permutation for $n = 6$ is the only known APN permutation on an even number of variables [7].

One method that has been successfully used for finding new sporadic instances of APN functions is what we refer to as polynomial expansion. It has been historically used to find some of the earliest known instances of APN polynomial functions, as well as to construct some of the infinite families from Table 2.2 [C1-C2, C13]. Unfortunately, it has never been documented how far polynomial expansion can be pushed computationally. The main goal of this master thesis is to systematically and exhaustively catalogue the exact range of APN functions (up to CCZ-equivalence) that can be obtained in small dimensions using this method.

ID	Functions	Conditions	Source
F1-F2	$x^{2^s+1}+u^{2^k-1}x^{2^{ik}+2^{mk+s}}$	$n=pk, \gcd(k,3)=\gcd(s,3k)=1, p\in\{3,4\}, i=sk \bmod p, m=p-i, n\geq 12, u$ primitive in $\mathbb{F}_{2^n}^*$	[13]
F3	$sx^{q+1}+x^{2^i+1}+x^{q(2^i+1)}+cx^{2^iq+1}+c^qx^{2^i+q}$	$q=2^m, n=2m$, $\gcd(i,m)=1$, $c\in\mathbb{F}_{2^n}, s\in\mathbb{F}_{2^n}\setminus\mathbb{F}_q, X^{2^i+1}+cX^{2^i}+c^qX+1$ has no solution x s.t. $x^{q+1}=1$	[10]
F4	$x^3+a^{-1}\mathrm{Tr}_n(a^3x^9)$	$a\neq 0$	[14]
F5	$x^3+a^{-1}\mathrm{Tr}_3^n(a^3x^9+a^6x^{18})$	$3\|n$, $a\neq 0$	[15]
F6	$x^3+a^{-1}\mathrm{Tr}_3^n(a^6x^{18}+a^{12}x^{36})$	$3\|n, a\neq 0$	[15]
F7-F9	$ux^{2^s+1}+u^{2^k}x^{2^{-k}+2^{k+s}}+vx^{2^{-k}+1}+wu^{2^k+1}x^{2^s+2^{k+s}}$	$n=3k, \gcd(k,3)=\gcd(s,3k)=1, v,w\in\mathbb{F}_{2^k}, vw\neq 1, 3\|(k+s), u$ primitive in $\mathbb{F}_{2^n}^*$	[5]
F10	$(x+x^{2^m})^{2^k+1}+u'(ux+u^{2^m}x^{2^m})^{(2^k+1)2^i}+u(x+x^{2^m})(ux+u^{2^m}x^{2^m})$	$n=2m, m\geq 2$ even, $\gcd(k,m)=1$ and $i\geq 2$ even, u primitive in $\mathbb{F}_{2^n}^*, u'\in\mathbb{F}_{2^m}$ not a cube	[47]
F11	$a^2x^{2^{2m+1}+1}+b^2x^{2^{m+1}+1}+ax^{2^{2m}+2}+bx^{2^m+2}+(c^2+c)x^3$	$n=3m, m$ odd, $L(x)=ax^{2^{2m}}+bx^{2^m}+cx$ satisfies the conditions of Lemma 8 of [9]	[9]
F12	$u(u^qx+x^qu)(x^q+x)+(u^qx+x^qu)^{2^{2i}+2^{3i}}+a(u^qx+x^qu)^{2^{2i}}(x^q+x)^{2^i}+b(x^q+x)^{2^i+1}$	$q=2^m, n=2m, \gcd(i,m)=1$, $x^{2^i+1}+ax+b$ has no roots in $\mathbb{F}_{2^m}$	[41]
F13	$x^3+a(x^{2^i+1})^{2^k}+bx^{3\cdot 2^m}+c(x^{2^{i+m}+2^m})^{2^k}$	$n=2m=10$, $(a,b,c)=(\beta,1,0,0)$, $i=3$, $k=2$, β primitive in $\mathbb{F}_{2^2}$ $n=2m$, m odd, $3\nmid m$, $(a,b,c)=(\beta,\beta^2,1)$, β primitive in $\mathbb{F}_{2^2}$, $i\in\{m-2,m,2m-1,(m-2)^{-1} \bmod n\}$	[17]

Table 2.2: Known infinite families of quadratic APN polynomials over $\mathbb{F}_{2^n}$

Chapter 3

Polynomial expansion

One of the main goals of this thesis is to document how far polynomial expansion can be pushed in the search of APN functions. In this chapter, we describe in detail the idea behind polynomial expansion, the current state of knowledge on this topic, and the exact experimental procedure that we follow in our work.

3.1 Basic notions

Due to the huge number of (n,n)-functions, conducting an exhaustive search for APN functions over all of them is impossible. In practice, we can find sporadic APN instances by performing exhaustive searches over very small subclasses of functions. There are many different ways to do this, and a natural idea is to take some given function F, and to examine all functions that have a representation (under ANF, or as TT, or in some other form) that is "close" to the representation of F in some sense.

For instance, we can take the TT of F, and then try to modify a few of its values in order to obtain new functions. We could then check which of these functions are APN, and in this way we can get new sporadic APN instances. Unfortunately, the approach of changing the TT of known APN functions does not appear to be promising: some negative results were obtained in [12] and [31], and in [11] it was shown that the number of modifications that need to be applied to the TT is so big that an exhaustive search becomes infeasible. Similarly, we do not know of any APN functions that have been obtained in this way using the ANF.

However, modifying the univariate representation of an APN function has been successfully used to construct many CCZ-inequivalent instances of APN functions, e.g. [6]. Furthermore, some APN instances found in this way have been generalized into infinite families [17], and so this appears to be a very promising method of finding new APN functions.

Unfortunately, there is currently no comprehensive documentation on how far polynomial expansion has been pushed, and what exactly exhaustive searches have been conducted. This is a significant problem since it means that researchers may keep repeating the same computational searches without finding new functions, which could significantly slow down progress in this area. In this thesis, we aim not only to conduct more and more extensive experiments using the polynomial expansion technique, but also to carefully document the exact state of our progress so that redundant experiments of this type can be avoided in the future.

The basic idea of polynomial expansion is that we take some initial (n,n)-function F, represented as a univariate polynomial. We then try to add a small number of terms to it and go over all possible combinations of coefficients and exponents. If we don't get any functions after adding one term, we then try to add two terms, three terms, four terms, and so forth, until the computation becomes too slow. If it gets to the point that it is too slow, we can restrict the coefficients to subfields of $\mathbb{F}_{2^n}$.

One approach that we can use to reduce the complexity of the search is to restrict the exponents to only quadratic ones. Although we will not be able to express any function of algebraic degree greater than 2 in this way, we know that most of the known APN functions (with a single exception [27]) are CCZ-equivalent to quadratic functions or monomials; furthermore, non-quadratic APN functions tend to be rare, especially if we restrict the univariate representation to a small number of terms. While this does hypothethically mean that we might miss a non-quadratic APN function, the advantange that we obtain by imposing this restriction is significant. The number of exponents in a full search without restrictions is $2^n - 1$ so in the case of $n = 8$ we have 255 exponents. But if we restrict this to only quadratic exponents we will only have $\binom{8}{2} = 28$ exponents in the search. By reducing the number of exponents, we significantly speed up the search.

After computing the functions, we need to classify them up to CCZ-equivalence. Since we have chosen to restrict the exponents to quadratics, we know that CCZ-equivalence is the same as EA-equivalence, and so we use the differential spectrum of the orthoderivative to distinguish between CCZ-inequivalent functions.

3.2 Status quo

Even though researchers have found many sporadic APN functions through polynomial expansion, there are few documented results on the efficiency of this method, and it seems that other computational methods have been preferred in the literature recently, e.g matrix representations [46] or the switching construction [27].
One disadvtange of these methods is that they give functions with a very complicated univariate representation, which makes it difficult to analyze the properties of these functions and to generalize them into infinite families. In fact, one of the goals of this thesis is to find the shortest possible representations of all classes of APN functions that can be obtained using polynomial expansion in small dimensions.

3.3 Experimental setup

As discussed above, our primary approach is to search for quadratic APN functions with a simple univariate representation over some finite fields $\mathbb{F}_{2^n}$. In order to do this, we conduct a number of experiments of the following type. We take some finite field $\mathbb{F}_{2^n}$; in our experiments, we consider $n = 8, 9$. We do not consider dimensions less than 8 since we know that the classification up to CCZ-equivalence of quadratic APN functions in those dimensions is complete [27, 34, 37]. We try to take the search as far as possible (with respect to the number of terms that we add), and we concentrate on dimensions $n = 8$ and $n = 9$ as being sufficiently small to allow for multiple searches of this form. For $n = 8, 9$, we take a starting function F. In our case, we let F be a representative from the CCZ-classes of the known infinite families of APN functions. Then we try to add terms to the function F. We restrict the exponents so that we only look at quadratic ones. In dimension 8, we are able to add up to $K = 6$ terms before the searches become too slow; and in dimension 9, we are able to go up to $K = 4$ in general; with the help of the EA-equivalence trick that we describe in Section 3.4, we are able to push this to $K = 5$ in the case when F is a monomial.

We also do not consider functions having all of their coefficients in $\mathbb{F}_2$ since these have been completely classified up to CCZ-equivalence for $n \leq 9$ [44].

Initially, we let the terms have coefficients from the entire field $\mathbb{F}_{2^n}$. Once the searches become too slow, we restrict the coefficients to a subfield $\mathbb{F}_{2^m}$, and continue adding more

terms. For $n = 8$, we can restrict to $m = 1, 2, 4, 8$; and in dimension $n = 9$, we can restrict to $m = 1, 3, 9$.

For every function obtained in this way, we check whether it is APN from the definition. Finally, for those functions that are APN, we compute the differential spectra and the Walsh spectra of their orthoderivatives, and we use them to split the functions into EA-equivalence classes (and since they are quadratic, CCZ-equivalence reduces to EA-equivalence for them). We do not perform any actual equivalence test on the functions (so, hypothetically, they might represent even more CCZ-equivalence classes) but since, according to computational observations, the Walsh spectrum and differential spectrum of the orthoderivatives allow us to distinguish between any two CCZ-inequivalent quadratic APN functions (with the single exception of some Gold functions in the case of odd dimensions), we believe that this should not be the case.

3.4 Simplifying the search when the starting function is a monomial

When expanding monomials we can use the following trick based on EA-equivalence to significantly reduce the number of functions that we have to consider. In particular, this allows us to push the expansion in dimension $n = 9$ to $K = 5$ terms for monomials, while in the general case of polynomials, we stop at $K = 4$ due to the time complexity.

Suppose that our starting function is $F(x) = x^d$ for some exponent d, and we are adding a term cx^e for some coefficient $c \in \mathbb{F}_{2^n}$ and some exponent e to obtain an expanded function $G(x) = x^d + cx^e$. If we compose G with the linear permutation $L_2(x) = xa$ for some $0 \neq a \in \mathbb{F}_{2^n}$, then we obtain the EA-equivalent function

$$G'(x) = G \circ L_2(x) = a^d x^d + ca^e x^e.$$

If we now compose G' on the left with the linear permutation $L_1(x) = x/a^d$, then we obtain the EA-equivalent function

$$G''(x) = L_1 \circ G \circ L_2(x) = x^d + ca^{e-d} x^e.$$

Comparing this with $G(x)$, we see that the only difference is the coefficient of x^e, which is

multiplied by a^{e-d}. Since we can do this for any non-zero $a \in \mathbb{F}_{2^n}$, this means that when selecting the coefficient c for the term cx^e in the search, we can restrict to only one coefficient from each set of the form

$$\{ca^{e-d} : 0 \neq a \in \mathbb{F}_{2^n}\}$$

without losing any functions up to EA-equivalence.

We can also use another similar trick that involves composing $G(x) = x^d + cx^e$ with the linear permutations $L_1(x) = x^2$ and $L_2(x) = x^{2^{n-1}}$. Recall that $x^{2^n} = x$ for any $x \in \mathbb{F}_{2^n}$. Composing G with L_1 and L_2, we obtain the EA-equivalent function

$$L_1 \circ G \circ L_2(x) = x^d + c^2 x^e.$$

Thus, we can raise the coefficient in front of the first term that we add to the second power and obtain an EA-equivalent function. Since we can do this as many times as we want, we can replace c with c^{2^k} for any k; and so it is enough to consider one coefficient c from each set of the form

$$\{c^{2^k} : k = 0, 1, \ldots, n-1\}$$

without losing any functions up to EA-equivalence.

Combining these two techniques, we can significantly reduce the number of choices for the coefficient of the first term that we add to our starting function, and so we can greatly reduce the complexity of the search.

Unfortunately, there does not appear to be any obvious way to extend these tricks to the general case when we are expanding polynomials, or to restrict the coefficients of terms that we add other than the first one.

Chapter 4

Computational results

In this chapter, we summarize the computational results that we obtain in our work. A full list of all the functions found by our searches is available online at https://boolean.h.uib.no/mediawiki/index.php/APN_functions_obtained_via_polynomial_expansion_in_small_dimensions. In Tables 4.1 in 4.2 we show the computation time for the polynomial expansion experiments that we perform, while in Tables 4.3 and 4.4, we list the representatives that we find from those classes of functions whose orthoderivative differential spectrum is different from that of any representative form the known APN families.

In Tables 4.1 and 4.2, we indicate the subfield to which we restrict the coefficients in the search using different colors. In the case of dimension $n = 8$, red means that we take coefficients from the entire field, white means that we take coefficients from the subfield $\mathbb{F}_{2^4}$, and green means that we take coefficients from the subfield $\mathbb{F}_{2^2}$. In the case of dimension $n = 9$, yellow means that we take coefficients from the entire field, and purple means that we take coefficients from the subfield $\mathbb{F}_{2^3}$. We recall that we do not consider functions all of whose coefficients are in the prime field $\mathbb{F}_2$ because all such APN functions up to dimension $n = 9$ have already been classified [44]. Finally, the colour gray means that we did not get any functions from the computations.

By examining the tables, we can see that pushing these searches further would require a significant amount of computation time: for instance, in dimension $n = 8$, running experiments in the subfield $\mathbb{F}_{2^2}$ took about 27 hours, and for dimension $n = 9$, experiments in the subfield $\mathbb{F}_{2^3}$ took 25 days to finish.

In dimension 9, we run experiments for expanding quadratic monomial APN functions (that is, the Gold APN functions) up to 5 terms, while in the case of polynomial APN functions we only go up to 4 terms. This is because the equivalence trick described in Section 3.4 in the

previous chapter only works for monomials; in the case of polynomials, we expect that the running time would be too long. The cells of Table 4.2 corresponding to searches that we did not run, are also colored in grey; to differentiate from searches that we did run but that did not yield any functions, we denote them by "N" in the table.

The running times are given in hours. If the time is very short (for instance, in the range of a few seconds to a few minutes), then we round it down to 0 hours.

Representatives from our searches that have differential spectra of the orthoderivative different from those of representatives from the known APN families are given in Tables 4.3 and 4.4. In both tables, the first column gives an index that can be used to refer to the function. The second column gives a univariate representative found using our search; we tried to give the simplest possible representative in all cases, meaning that we took the one with the fewest terms, and the most restricted coefficients (that is, with coefficients in the smallest possible subfield). The third column indicates which known APN function our representative is equivalent to (in terms of having the same differential spectrum of the orthoderivative). In the case of Table 4.3, "SW X" means that the representative has the same differential spectrum of the orthoderivative as a function obtained from Edel and Pott's switching construction in [27]; in particular, we refer to Table 9 from [27], and e.g. "SW 19" means that our representative corresponds to the one on the 19-th row of Table 9. Similarly, "B X" means that the representative has the same differential spectrum of the orthoderivative as a function obtained by Beierle and Leander [1], and "Y X" means that the representative has the same differential spectrum of the orthoderivative as a function obtained by Yu et al. [45]. In the case of Table 4.4, we have the similar "B X", which means that the representative has the same differential spectrum of the orthoderivative as a function obtained by Beierle and Leander [1], and "I X" means that the representative has the same differential spectrum of the orthoderivative as a function obtained by Budaghyan et al. using the so-called isotopic shift construction from [8]. The penultimate column gives the differential spectrum of the orthoderivative. Finally, the last column gives the number of terms in the univariate representation of the previously known representative with the same differential spectrum of the orthoderivative.

In dimension $n = 8$, we find 16 distinct orthoderivative differential spectra that are different from those of the known infinite APN families. Among these, the function with ID 8.7 can be seen to be CCZ-inequivalent to any known APN function over $\mathbb{F}_{2^8}$ based on the value of its orthoderivative differential spectrum. The remaining 15 functions have the same

differential spectrum of the orthoderivative as APN functions obtained from Edel and Pott's switching construction [27], the matrix construction of Yu et al. [46], and Beierle and Leander's self-equivalence method [1]. However, in a lot of cases, the functions that we find have a much shorter univariate representation that the currently known representatives from these constructions. For example, the function with ID 8.14 has the same differential spectrum of the orthoderivative as a function of Beierle and Leander, but the representative given in [1] consists of 36 terms and has the form

$$\begin{aligned}\alpha^{192}x^{192}+\alpha^{37}x^{160}+\alpha^{196}x^{144}+\alpha^{5}x^{136}+\alpha^{234}x^{132}+\alpha^{119}x^{130}+\alpha^{79}x^{129}+\\ \alpha^{155}x^{128}+\alpha^{74}x^{96}+\alpha^{171}x^{80}+\alpha^{172}x^{72}+\alpha^{116}x^{68}+\alpha^{98}x^{66}+\alpha^{127}x^{65}+\\ \alpha^{188}x^{64}+\alpha^{8}x^{48}+\alpha^{97}x^{40}+\alpha^{161}x^{36}+\alpha^{160}x^{34}+\alpha^{59}x^{33}+\alpha^{203}x^{32}+\\ \alpha^{129}x^{24}+\alpha^{236}x^{20}+\alpha^{180}x^{18}+\alpha^{181}x^{17}+\alpha^{118}x^{16}+\alpha^{150}x^{12}+\alpha^{118}x^{10}+\\ \alpha^{150}x^{9}+\alpha^{161}x^{8}+\alpha 192x^{6}+\alpha^{131}x^{5}+\alpha^{47}x^{4}+\alpha^{217}x^{3}+\alpha^{219}x^{2}+\alpha^{164}x,\end{aligned}$$

while the representative for 8.14 has the much simpler form

$$x^{144}+\alpha^{85}x^{96}+a^{170}x^{80}+\alpha^{85}x^{65}+\alpha^{85}x^{17}+x^{9}+x^{5}.$$

Note also that all coefficients in the representation that we find are in $\mathbb{F}_{2^2}$ (in contrast to the previously known representative).

It is interesting that a lot of the classes of APN functions obtained using complex computational methods (such as the ones of Edel and Pott, the generalized isotopic construction of Budaghyan et al., the matrix method of Yu et al., and the self-equivalence of Beierle and Leander) can be obtained in this much simpler way. Of course, we can see that not all of the functions from e.g. the switching classes are represented among our searches; however, our results suggests that polynomial expansion can still be useful for finding new classes of APN functions, and it would be interesting to see how much further it can be pushed, and whether more of the representatives given in e.g. [27] can be obtained using simpler methods like this.

For dimension $n = 9$, we find 19 distinct classes (according to the differential spectrum of the orthoderivative) distinct from those of the known infinite APN families. All of these functions have an orthoderivative differential spectrum matching one of the functions from Budaghyan et al.'s isotopic shift method [9] or Beierle and Leander's classes [1]; however, our representatives are simpler than the known ones in some cases. In the case of the functions

corresponding to [8], our representatives are not significantly shorter than the known ones. However, in the case of the Beierle-Leander functions (with ID 9.17, 9.18 and 9.19), our representatives are much shorter than the currently known ones; for example “B 31” has a polynomial form consisting of 44 terms given by

$$\begin{aligned}
&\alpha^{474}x^{384}+\alpha^{421}x^{320}+\alpha^{96}x^{288}+\alpha^{139}x^{272}+\alpha^{313}x^{264}+\alpha^{202}x^{260}+\alpha^{245}x^{258}+\\
&\alpha^{33}x^{257}+\alpha^{87}x^{256}+\alpha^{63}x^{192}+\alpha^{194}x^{160}+\alpha^{434}x^{144}+\alpha^{362}x^{136}+\alpha^{453}x^{132}+\\
&\alpha^{275}x^{130}+\alpha^{306}x^{129}+x^{128}+\alpha^{428}x^{96}+\alpha^{282}x^{80}+\alpha^{431}x^{72}+\alpha^{19}x^{68}+\alpha^{298}x^{66}+\\
&\alpha^{253}x^{65}+\alpha^{144}x^{64}+\alpha^{387}x^{48}+\alpha^{234}x^{40}+\alpha^{449}x^{36}+\alpha^{131}x^{34}+\alpha^{480}x^{33}+\\
&\alpha^{353}x^{32}+\alpha^{52}x^{24}+\alpha^{399}x^{20}+\alpha^{237}x^{17}+\alpha^{192}x^{16}+\alpha^{509}x^{12}+\alpha^{230}x^{10}+\\
&\alpha^{227}x^{9}+\alpha^{261}x^{8}+\alpha^{271}x^{6}+\alpha^{8}x^{5}+\alpha^{468}x^{4}+\alpha^{403}x^{3}+\alpha^{77}x^{2}+\alpha^{492}*x,
\end{aligned}$$

and our representative 9.17 has the form

$$x^{80}+\alpha^{73}x^{66}+x^{17}+\alpha^{73}x^{10}+x^{3}.$$

Once again, note that all coefficients are in $\mathbb{F}_{2^3}$.

Finally, we note that we find a completely new APN instance (which is CCZ-inequivalent to any previously known function), represented by

$$F_1(x)=\alpha^{170}x^{132}+\alpha^{85}x^{66}+\alpha^{85}x^{18}+x^{3}$$

over $\mathbb{F}_{2^8}$.

Dim 8	1	2	3	4	5	6
8.1	0	0	121	24.4	1.93	22.61
8.2	0	0	82.59	20.3	1428.87	26.63
8.4	0	2.3	0.74	55.78	1.11	9.99
8.5	0	2.19	0.58	33.66	0.57	3.99
8.6	0	2.24	0.6	36.67	0.58	4.18
8.7	0	2.8	0.64	37.69	0.5	3.68
8.8	0	2.14	0.55	32.96	0.57	4.09
8.9	0	2.08	0.35	14.73	0.17	0.88
8.10	0	6.22	1.3	74.67	1.22	8.58

Table 4.1: Computation times for the polynomial expansion experiments in dimension 8.

Dim 9	1	2	3	4	5
9.1	0	0.36	0.2	11.76	535.52
9.2	0	0.35	0.19	10.3	459.18
9.3	0	0.39	0.23	13	625.06
9.8	0	32.84	0.38	15.36	N
9.9	0	32.22	0.43	20.19	N
9.10	0	35.42	0.46	21.49	N
9.11	0	32.43	0.47	24.52	N

Table 4.2: Computation times for the polynomial expansion experiments in dimension 9.

ID	Representatives	Eq	Orthoderivative diff. spectrum	Terms
8.1	$\alpha^{170}x^{192}+\alpha^{85}x^{132}+x^6+x^3$	$SW19$	$0^{37872},2^{22788},4^{4068},6^{492},8^{60}$	4
8.2	$x^{66}+\alpha^{85}x^{33}+x^{18}+x^9+x^3$	$SW11$	$0^{38040},2^{22461},4^{4218},6^{513},8^{36},10^{12}$	16
8.3	$x^{66}+\alpha^{85}x^{33}+\alpha^{17}x^9+\alpha^{102}x^6+x^3$	$SW13$	$0^{38076},2^{22311},4^{4374},6^{495},8^{24}$	16
8.4	$\alpha^{85}x^{132}+\alpha^{85}x^{72}+x^9+x^6+x^3$	$SW12$	$0^{38160},2^{22104},4^{4536},6^{456},8^{24}$	16
8.5	$x^{66}+x^{12}+\alpha^{85}x^6+x^3$	$SW6$	$0^{38160},2^{22164},4^{4428},6^{492},8^{36}$	4
8.6	$x^{129}+\alpha^{85}x^{24}+x^{12}+x^9+x^3$	$SW8$	$0^{38184},2^{22179},4^{4338},6^{531},8^{48}$	16
8.7	$\alpha^{170}x^{132}+\alpha^{85}x^{66}+\alpha^{85}x^{18}+x^3$	new	$0^{38196},2^{22008},4^{4608},6^{456},8^{12}$	-
8.8	$\alpha^{85}x^{132}+\alpha^{85}x^{72}+x^{36}+x^{24}+x^3$	$SW9$	$0^{38256},2^{22116},4^{4230},6^{648},8^{30}$	16
8.9	$\alpha^{85}x^{192}+x^{72}+x^{33}+x^{24}+x^9+\alpha^{153}x^6$	$SW17$	$0^{38388},2^{21723},4^{4626},6^{507},8^{36}$	16
8.10	$\alpha^{221}x^{96}+\alpha^{221}x^{33}+x^{12}+x^9+x^6+\alpha^{187}*x^3$	SW 10	$0^{38439},2^{21618},4^{4671},6^{528},8^{24}$	16
8.11	$\alpha^{238}x^{144}+x^{132}+\alpha^{51}x^{96}+\alpha^{119}x^{48}+x^{33}+x^9$	$SW16$	$0^{38457},2^{21552},4^{4743},6^{510},8^{18}$	16
8.12	$\alpha^{204}x^{160}+\alpha^{51}x^{48}+\alpha^{102}x^{12}+\alpha^{204}x^{10}+x^9$	$SW22$	$0^{38844},2^{20974},4^{4764},6^{654},8^{44}$	5
8.13	$\alpha^{160}x^{132}+\alpha^{10}x^{72}+x^{48}+\alpha x^{34}+\alpha^3x^{33}+\alpha^{48}x^{18}+x^{17}+x^3$	B 31	$0^{39150},2^{20463},4^{4920},6^{675},8^{54},10^{12},12^6$	36
8.14	$x^{144}+\alpha^{85}x^{96}+\alpha^{170}x^{80}+\alpha^{85}x^{65}+\alpha^{85}x^{17}+x^9+x^5$	B 12668	$0^{39408},2^{20072},4^{4922},6^{798},8^{70},10^{10}$	36
8.15	$x^{66}+\alpha^{170}x^{40}+x^{18}+\alpha^{85}x^5+x^3$	Y 4346	$0^{39408},2^{20218},4^{4692},6^{838},8^{104},10^{12},12^8$	5
8.16	$x^{160}+x^{132}+x^{80}+x^{68}+x^6+x^3$	$SW20$	$0^{39692},2^{19752},4^{4756},6^{978},8^{72},10^{26},12^4$	6

Table 4.3: Representatives for $n=8$ with orthoderivative differential spectra distinct from those of the known APN families

ID	Representative	Eq	Orthoderivative diff. spectrum	Terms
9.1	$\alpha^{365}x^{257}+x^{96}+x^{68}+\alpha^{219}x^{33}+x^5$	I 4	0^{158529}, 2^{80829}, 4^{18144}, 6^{3283}, 8^{469}, 10^{294}, 12^{84}	5
9.2	$\alpha^{438}x^{129}+x^{66}+\alpha^{219}x^{10}+x^3$	I 8	0^{159418}, 2^{79275}, 4^{18690}, 6^{3213}, 8^{742}, 10^{252}, 12^{21}, 16^{21}	4
9.3	$x^{136}+x^{24}+x^{17}+\alpha^{73}x^{10}+x^3$	I 3	0^{159684}, 2^{78687}, 4^{19089}, 6^{3136}, 8^{777}, 10^{147}, 12^{84}, 14^{28}	5
9.4	$x^{68}+\alpha^{73}x^{40}+x^{33}+x^5$	I 10	0^{159684}, 2^{79590}, 4^{17871}, 6^{3283}, 8^{700}, 10^{273}, 12^{147}, 14^{84}	4
9.5	$\alpha^{73}x^{136}+\alpha^{146}x^{66}+\alpha^{219}x^{10}+x^3$	I 16	0^{159908}, 2^{79086}, 4^{18081}, 6^{3353}, 8^{721}, 10^{336}, 12^{105}, 14^{21}, 16^{21}	4
9.6	$x^{264}+\alpha^{73}x^{96}+\alpha^{219}x^{68}+x^5$	I 11	0^{160020}, 2^{79023}, 4^{17997}, 6^{3213}, 8^{868}, 10^{378}, 12^{133}	4
9.7	$\alpha^{219}x^{136}+x^{10}+x^3$	I 12	0^{160657}, 2^{77910}, 4^{18312}, 6^{3360}, 8^{952}, 10^{273}, 12^{147}, 14^{21}	4
9.8	$x^{192}+x^{66}+x^{17}+\alpha^{73}x^{10}+x^3$	I 14	0^{162183}, 2^{76482}, 4^{17388}, 6^{3871}, 8^{1162}, 10^{252}, 12^{126}, 14^{126}, 16^{21}, 22^{21}	5
9.9	$\alpha^{73}x^{192}+x^{136}+\alpha^{365}x^{129}+x^{17}+x^3$	I 5	0^{162708}, 2^{77175}, 4^{15498}, 6^{4270}, 8^{1260}, 10^{252}, 12^{168}, 14^{84}, 16^{126}, 18^{42}, 22^{42}, 26^7	5
9.10	$\alpha^{73}x^{129}+\alpha^{292}x^{66}+x^{10}+x^3$	I 9	0^{163009}, 2^{75537}, 4^{17283}, 6^{4116}, 8^{1071}, 10^{168}, 12^{231}, 14^{28}, 16^{84}, 18^{63}, 20^{42}	4
9.11	$x^{80}+\alpha^{146}x^{66}+\alpha^{73}x^{24}+x^{17}$	I 13	0^{163366}, 2^{75117}, 4^{17010}, 6^{4536}, 8^{966}, 10^{252}, 12^{63}, 14^{154}, 16^{63}, 18^{84}, 22^{21}	4
9.12	$x^{129}+\alpha^{73}x^{66}+x^{17}+x^{10}+\alpha^{365}x^3$	I 6	0^{163996}, 2^{74802}, 4^{16380}, 6^{4368}, 8^{1449}, 10^{231}, 12^{126}, 14^{84}, 16^{42}, 18^{84}, 20^{42}, 22^{21}, 32^7	5
9.13	$\alpha^{73}x^{136}+\alpha^{219}x^{66}+\alpha^{438}x^{10}+x^3$	I 15	0^{168994}, 2^{68712}, 4^{15141}, 6^{6279}, 8^{1659}, 10^{336}, 12^{21}, 14^{21}, 16^{105}, 18^{147}, 20^{189}, 24^{21}, 26^7	5
9.14	$\alpha^{438}x^{129}+x^{66}+\alpha^{219}x^{17}+x^3$	I 2	0^{169428}, 2^{68040}, 4^{15561}, 6^{6034}, 8^{1533}, 10^{420}, 12^{126}, 14^{21}, 16^{84}, 18^{189}, 20^{126}, 22^{63}, 26^7	5
9.15	$\alpha^{365}x^{80}+\alpha^{292}x^{24}+\alpha^{219}x^{17}+x^3$	I 17	0^{170079}, 2^{66297}, 4^{16737}, 6^{6160}, 8^{1407}, 10^{420}, 12^{21}, 14^{42}, 16^{63}, 18^{210}, 20^{133}, 22^{63}	4
9.16	$x^{257}+\alpha^{438}x^{68}+\alpha^{219}x^{12}+x^5$	I 7	0^{171430}, 2^{64617}, 4^{16842}, 6^{5733}, 8^{1932}, 10^{483}, 12^{105}, 14^{21}, 16^{147}, 18^{105}, 20^{154}, 22^{21}, 24^{42}	4
9.17	$x^{80}+\alpha^{73}x^{66}+x^{17}+\alpha^{73}x^{10}+x^3$	B 31	0^{160440}, 2^{78834}, 4^{17514}, 6^{3388}, 8^{777}, 10^{483}, 12^{126}, 14^{49}, 16^{21}	44
9.18	$\alpha^{365}x^{136}+x^{129}+\alpha^{73}x^{80}+x^{24}+x^{17}+x^3$	B 34	0^{164199}, 2^{76734}, 4^{13524}, 6^{4312}, 8^{2205}, 12^{147}, 16^{294}, 18^{147}, 20^{49}, 22^{21}	45
9.19	$\alpha^{73}x^{320}+x^{96}+\alpha^{219}x^{68}+x^{40}+x^{33}+x^5$	B 35	0^{172557}, 2^{68355}, 4^{12201}, 6^{3871}, 8^{1638}, 10^{735}, 12^{1470}, 14^{49}, 16^{147}, 18^{441}, 20^{147}, 42^{21}	45

Table 4.4: Representatives for $n = 9$ with orthoderivative differential spectra distinct from those of the known APN families

Chapter 5

Conclusion and future work

One of the main goals of this master thesis was to find out how far we could push polynomial expansion in the search of APN functions. To this end, we have carefully documented the exact configurations and running times of the experiments that we conducted for dimensions $n = 8$ and $n = 9$. From this, we can see that pushing polynomial expansion further is possible, but would require significant computational efforts.

On the other hand, we did obtain one new APN function for $n = 8$, and for both $n = 8$ and $n = 9$, we observed that many of the known APN functions that were previously found using complicated computational methods can also be obtained in this way. Furthermore, some of the representative that we find are significantly simpler than the previously known ones which were obtained using other methods. This suggests that polynomial expansion still has the potential to provide us with new APN functions, and with simpler representatives of known sporadic APN instances, and so it remains a viable method for investigating APN functions.

A natural continuation of our work would be to run more experiments and to see what other classes of APN functions can be obtained in this much simpler way. In particular, we have attempted to expand APN functions from the known infinite families of APN functions; a straightforward further step would be to attempt to expand some of the other known APN functions, or to start from other promising functions that are not themselves APN.

Another thing we can try to do is to find a way to speed up the search by using a technique similar to the EA-equivalence trick for polynomial functions (although it is not obvious what such a technique would look like).

We can also try to take a closer look at the representatives that we found (in particular, the new APN function for $n = 8$), and see if they can be generalized into infinite families. Although such a task is still far from easy, our shorter representations make this task more

realistic than the previously known representations.

Bibliography

[1] Christof Beierle and Gregor Leander. New instances of quadratic APN functions. *arXiv preprint arXiv:2009.07204*, 2020.

[2] Thomas Beth and Cunsheng Ding. On almost perfect nonlinear permutations. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 65–76. Springer, 1993.

[3] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of CRYPTOLOGY*, 4(1):3–72, 1991.

[4] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system I: The user language. *Journal of Symbolic Computation*, 24(3-4):235–265, 1997.

[5] Carl Bracken, Eimear Byrne, Nadya Markin, and Gary McGuire. A few more quadratic APN functions. *Cryptography and Communications*, 3(1):43–53, 2011.

[6] KA Browning, JF Dillon, MT McQuistan, and AJ Wolfe. APN polynomials and related codes. *Special volume of Journal of Combinatorics, Information and System Sciences*, 34:135–159, 2009.

[7] KA Browning, JF Dillon, MT McQuistan, and AJ Wolfe. An APN permutation in dimension six. *Finite Fields: theory and applications*, 518:33–42, 2010.

[8] Lilya Budaghyan, Marco Calderini, Claude Carlet, Robert Coulter, and Irene Villa. Generalized isotopic shift construction for APN functions. *Designs, Codes and Cryptography*, pages 1–14, 2020.

[9] Lilya Budaghyan, Marco Calderini, Claude Carlet, Robert S Coulter, and Irene Villa. Constructing APN functions through isotopic shifts. *IEEE Transactions on Information Theory*, 66(8):5299–5309, 2020.

[10] Lilya Budaghyan and Claude Carlet. Classes of quadratic APN trinomials and hexanomials and related structures. *IEEE Transactions on Information Theory*, 54(5):2354–2357, 2008.

[11] Lilya Budaghyan, Claude Carlet, Tor Helleseth, and Nikolay Kaleyski. On the distance between APN functions. *IEEE Transactions on Information Theory*, 66(9):5742–5753, 2020.

[12] Lilya Budaghyan, Claude Carlet, Tor Helleseth, Nian Li, and Bo Sun. On upper bounds for algebraic degrees of APN functions. *IEEE Transactions on Information Theory*, 64(6):4399–4411, 2018.

[13] Lilya Budaghyan, Claude Carlet, and Gregor Leander. Two classes of quadratic APN binomials inequivalent to power functions. *IEEE Transactions on Information Theory*, 54(9):4218–4229, 2008.

[14] Lilya Budaghyan, Claude Carlet, and Gregor Leander. Constructing new APN functions from known ones. *Finite Fields and Their Applications*, 15(2):150–159, 2009.

[15] Lilya Budaghyan, Claude Carlet, and Gregor Leander. On a construction of quadratic APN functions. In *2009 IEEE Information Theory Workshop*, pages 374–378, 2009.

[16] Lilya Budaghyan, Claude Carlet, and Alexander Pott. New classes of almost bent and almost perfect nonlinear polynomials. *IEEE Transactions on Information Theory*, 52(3):1141–1152, 2006.

[17] Lilya Budaghyan, Tor Helleseth, and Nikolay Kaleyski. A new family of APN quadrinomials. *IEEE Transactions on Information Theory*, 66(11):7081–7087, 2020.

[18] Anne Canteaut, Alain Couvreur, and Léo Perrin. Recovering or Testing Extended-Affine Equivalence. *arXiv preprint arXiv:2103.00078*, 2021.

[19] Claude Carlet. *Boolean functions for cryptography and coding theory*. Cambridge University Press, 2021.

[20] Claude Carlet, Pascale Charpin, and Victor A. Zinoviev. Codes, bent functions and permutations suitable for DES-like cryptosystems. *Designs, Codes and Cryptography*, 15(2):125–156, 1998.

[21] Florent Chabaud and Serge Vaudenay. Links between differential and linear cryptanalysis. In *Workshop on the Theory and Application of Cryptographic Techniques, EUROCRYPT 94*, volume 950, pages 356–365, 1994.

[22] Itai Dinur and Adi Shamir. Breaking Grain-128 with dynamic cube attacks. In *International Workshop on Fast Software Encryption*, pages 167–187. Springer, 2011.

[23] Hans Dobbertin. Almost perfect nonlinear power functions on $GF(2^n)$: the Niho case. *Information & Computation*, 151(1):57–72, 1999.

[24] Hans Dobbertin. Almost perfect nonlinear power functions on $GF(2^n)$: the Welch case. *IEEE Transactions on Information Theory*, 45(4):1271–1275, 1999.

[25] Hans Dobbertin. Almost perfect nonlinear power functions on $GF(2^n)$: A new case for n divisible by 5. *International Conference on Finite Fields and Applications*, pages 113–121, 2001.

[26] Yves Edel, Gohar Kyureghyan, and Alexander Pott. A new APN function which is not equivalent to a power mapping. *IEEE Transactions on Information Theory*, 52(2):744–747, 2006.

[27] Yves Edel and Alexander Pott. A new almost perfect nonlinear function which is not quadratic. *Advances in Mathematics of Communications*, 3(1):59–81, 2009.

[28] Yves Edel and Alexander Pott. On the equivalence of nonlinear functions. In *Enhancing cryptographic primitives with techniques from error correcting codes*, pages 87–103. IOS Press, 2009.

[29] Robert Gold. Maximal recursive sequences with 3-valued recursive cross-correlation functions (corresp.). *IEEE Transactions on Information Theory*, 14(1):154–156, 1968.

[30] Heeralal Janwa and Richard M Wilson. Hyperplane sections of Fermat varieties in P^3 in char. 2 and some applications to cyclic codes. In *International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pages 180–194. Springer, 1993.

[31] Nikolay Kaleyski. Changing APN functions at two points. *Cryptography and Communications*.

[32] Nikolay Kaleyski. Deciding EA-equivalence via invariants. *Cryptography and Communications*, pages 1–20, 2021.

[33] Nikolay S Kaleyski. Invariants for EA-and CCZ-equivalence of APN and AB functions. *IACR Cryptol. ePrint Arch.*, 2021:300, 2021.

[34] Konstantin Kalgin and Valeriya Idrisova. The classification of quadratic APN functions in 7 variables. *Cryptology ePrint Archive, Report 2020/1515*, 2020.

[35] Tadao Kasami. The weight enumerators for several classes of subcodes of the 2nd order binary Reed-Muller codes. *Information & Computation*, 18(4):369–394, 1971.

[36] Lars R Knudsen. Truncated and higher order differentials. In *International Workshop on Fast Software Encryption*, pages 196–211. Springer, 1994.

[37] Philippe Langevin, Elif Saygi, and Zulfukar Saygi. Classification of APN cubics in dimension 6 over GF (2). http://langevin.univ-tln.fr/project/apn-6/apn-6.html.

[38] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 386–397. Springer, 1993.

[39] Kaisa Nyberg. Differentially uniform mappings for cryptography. *Lecture Notes in Computer Science*, 765:55–64, 1994.

[40] Christof Paar and Jan Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media, 2009.

[41] Hiroaki Taniguchi. On some quadratic APN functions. *Designs, Codes and Cryptography*, pages 1–11, 2019.

[42] Satoshi Yoshiara. Equivalences of quadratic APN functions. *Journal of Algebraic Combinatorics*, 35(3):461–475, 2012.

[43] Satoshi Yoshiara. Equivalences of power APN functions with power or quadratic APN functions. *Journal of Algebraic Combinatorics*, 44(3):561–585, 2016.

[44] Yuyin Yu, Nikolay Kaleyski, Lilya Budaghyan, and Yongqiang Li. Classification of quadratic APN functions with coefficients in f2 for dimensions up to 9. *Finite Fields and Their Applications*, 68:101733, 2020.

[45] Yuyin Yu, Mingsheng Wang, and Yongqiang Li. Constructing differentially 4-uniform permutations from known ones. *Chinese Journal of Electronics*, 22(3):495–499, 2013.

[46] Yuyin Yu, Mingsheng Wang, and Yongqiang Li. A matrix approach for constructing quadratic APN functions. *Designs, codes and cryptography*, 73(2):587–600, 2014.

[47] Yue Zhou and Alexander Pott. A new family of semifields with 2 parameters. *Advances in Mathematics*, 234:43–60, 2013.