University of Bergen
Department of Informatics

Computational searches for quadratic APN functions with subfield coefficients

Author: Simon K. Berg
Supervisor: Nikolay S. Kaleyski

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

June, 2023

Abstract

Almost perfect nonlinear (APN) functions are important in fields such as algebra, combinatorics, cryptography, etc. Finding new APN functions is of special importance in cryptography. This is because when used in modern block ciphers, they are optimal against differential cryptanalysis. In this thesis, we discuss how the matrix approach for constructing quadratic APN functions developed by Yu et al. can be adapted to the case of functions over $\mathbb{F}_{2^n}$ with coefficients in a subfield $\mathbb{F}_{2^k}$. This adaptation allows us to search for functions of this form and using the notion of linear equivalence, we can significantly restrict the search space. Using this method, we classify all quadratic APN functions with coefficients in $\mathbb{F}_{2^2}$ over $\mathbb{F}_{2^8}$ up to CCZ-equivalence. To the best of our knowledge, no such search has been carried out before. The classification resulted in 27 CCZ-equivalence classes covering all quadratic APN functions with coefficients in $\mathbb{F}_{2^2}$ over $\mathbb{F}_{2^8}$ of which one seems to be new.

Acknowledgements

I would like express my sincere gratitude to the University of Bergen and the Selmer center for introducing me to the field of cryptography and Boolean functions. In particular, I want to thank my supervisor Nikolay Kaleyski for helping me with all aspects of my thesis, from formulating it, to providing invaluable input and always being available to help me. I would also like to thank my friends and family for their support throughout my studies here in Bergen.

`Simon K. Berg`
Thursday 1st June, 2023

Contents

Chapter 1

Introduction

Cryptography, the science of secure communication, is more important than ever in today's digital age. With the increasing reliance on digital technologies for communication, commerce, and storage of sensitive information, the need to protect the confidentiality, integrity, and authenticity of data has become paramount.

The study of Boolean functions is especially important for cryptographic purposes due to their usage in symmetric cryptography. Block ciphers such as the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES) both use vectorial Boolean functions as their S-boxes, which is a key part in providing confusion in their schemes. The choice of vectorial Boolean functions as S-boxes heavily determines the level of vulnerability of these block ciphers to many cryptanalytic attacks. Among these attacks are some of the most dangerous ones known to date, the linear- and differential attack.

Almost Perfect Nonlinear (APN) functions were introduced as a countermeasure to the differential attack [2, 15] and have been heavily researched by the cryptography community since their introduction. The reason APN functions are optimal against differential cryptanalysis is due to their low differential uniformity, but this property also makes it hard to find and study APN functions by mathematical means. Due to the large search space, it is not feasible to find all possible APN functions over a given field by exhaustive search. This is why targeted computational searches are an important tool for investigating and classifying APN functions.

A new method to efficiently search for quadratic APN functions was introduced by Yu et al. [18] in which they made use of a matrix structure, called the Quadratic APN Matrix

(QAM), to efficiently find new APN functions by modifying the QAM of a known APN function. By using their new method, they were able to increase the number of known APN functions in $\mathbb{F}_{2^7}$ from 19 [9, 10] to 470 functions. Additionally, they found more than 1000 new functions in $\mathbb{F}_{2^8}$, which was a significant increase from the 23 previously known ones. These results led to further research into QAMs. In 2020, Yu et al. adapted the QAM method for the special case when the APN functions had prime field coefficients. They did this by showing that the search space can be significantly reduced in this case. They used their adapted QAM method to perform a full classification of quadratic APN functions over $\mathbb{F}_{2^n}$ with coefficients in $\mathbb{F}_2$ for dimensions up to $n = 9$ [19]. Later, in 2022, a generalization of QAMs for any quadratic function over $\mathbb{F}_{p^n}$, called the derivative matrix, was introduced by Davidova and Kaleyski [8]. They also discussed a method of restricting the elements of the finite field to equivalence classes by considering functions up to linear equivalence, which reduced the search space. Both these techniques were then used to classify all planar functions with prime field coefficients over $\mathbb{F}_{3^n}$ up to $n = 7$.

In this thesis, we improve upon the above work by expanding upon the notion of using linear equivalence to restrict the search space in computational searches for functions with coefficients in $\mathbb{F}_{2^k}$ over $\mathbb{F}_{2^n}$ using the QAM method. Then we show how significant an improvement this brings by performing a computational classification of all quadratic APN functions in $\mathbb{F}_{2^8}$ with coefficients in the subfield $\mathbb{F}_{2^2}$, which to the best of our knowledge has not been done before. We performed our search using the *Magma* programming language [3] because of its ease of use, especially in representing complex mathematical structures, such as finite fields, polynomial rings and matrices, as well as its built-in features for operations on said structures. The classification resulted in finding 196863 APN functions that fell into 27 CCZ-equivalence classes. One of those classes seems to be new and can be represented by a 3-to-1 function, which we provide as part of our results. The classification works as a proof of concept for our improved search method. Hopefully, our results will pave the way for new searches over $\mathbb{F}_{p^n}$ with coefficients in $\mathbb{F}_{p^k}$ for higher values of p, n and k. To this end, we also show a method for approximating the complexity of such a search.

In Chapter 2, we introduce definitions and theorems necessary to understand our adaptation of the QAM method. The topics that will be covered are vectorial Boolean functions and their properties, such as differential uniformity and equivalence relations. We will also cover the theory of and surrounding the QAM.

In Chapter 3, we describe how to restrict the search space by considering linear equivalence between functions, as done in [8]. Then we show how our adapted method works and how significant of an improvement it brings by a simple example in $\mathbb{F}_{2^4}$. We then show how to construct a QAM for quadratic APN functions with coefficients in $\mathbb{F}_{2^2}$ over $\mathbb{F}_{2^8}$ and how to use our adapted method for said field. Lastly, we present the results of the classification of quadratic APN functions with coefficients in $\mathbb{F}_{2^2}$ over $\mathbb{F}_{2^8}$.

In Chapter 4, we draw our conclusion of the usefulness of our adapted method and discuss some future work that could benefit from implementing it.

Chapter 2

Background

In this chapter, we present background information needed for understanding cryptographic Boolean functions. We discuss topics such as vectorial Boolean functions, Almost Perfect Nonlinear (APN) functions, the quadratic APN matrix (QAM), equivalence relations between functions and their invariants.

2.1 Vectorial Boolean functions

Let $\mathbb{F}_2 = \{0, 1\}$ denote the binary finite field and $\mathbb{F}_2^n$ denote the n-dimensional vector space over $\mathbb{F}_2$ for some positive integer n. A **Boolean function** $F : \mathbb{F}_2^n \to \mathbb{F}_2$ is a function from the n-dimensional vector space $\mathbb{F}_2^n$ to $\mathbb{F}_2$, and is also called an **$(n, 1)$-function**.

In many scenarios, it is more natural to work with functions that give an entire vector of bits instead of a single bit as output, which leads to the introduction of vectorial Boolean functions. A **vectorial Boolean function (VBF)** $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ is a function that takes a Boolean vector of length n as input and output a Boolean vector of length m. A VBF is also called an **(n, m)-function** and can be expressed as

$$F(x_1, x_2, ..., x_n) = (f_1(x_1, x_2, ..., x_n), ..., f_m(x_1, x_2, ..., x_n)), \quad x_1, ..., x_n \in \mathbb{F}_2^n,$$

where $f_1, ...f_m : \mathbb{F}_2^n \to \mathbb{F}_2$ are called the **coordinate functions** of F. The nonzero linear combinations of the coordinate functions are called the **component functions** of F and are denoted by F_b, for $b \in \mathbb{F}_{2^m} \setminus \{0\}$.

Example 1 *Suppose we have a function $F = (f_1, f_2, f_3)$, then its component functions are $f_1,\ f_2,\ f_3,\ f_1 + f_2,\ f_1 + f_3,\ f_2 + f_3$ and $f_1 + f_2 + f_3$.*

The simplest way to represent a vectorial Boolean (n, m)-function is by its **truth table**, also called the **look-up table**, which lists every possible input to F and its corresponding output.

Example 2 *The truth table of a $(3, 2)$-function with coordinate functions $f_1(x_1, x_2, x_3) = x_1 + x_2$ and $f_2(x_1, x_2, x_3) = x_2 + x_3$ can be seen in Table 2.1.*

(x_1, x_2, x_3)	$F(f_1(x_1, x_2, x_3), f_2(x_1, x_2, x_3))$
$(0, 0, 0)$	$(0, 0)$
$(0, 0, 1)$	$(0, 1)$
$(0, 1, 0)$	$(1, 1)$
$(0, 1, 1)$	$(1, 0)$
$(1, 0, 0)$	$(1, 0)$
$(1, 0, 1)$	$(1, 1)$
$(1, 1, 0)$	$(0, 1)$
$(1, 1, 1)$	$(0, 0)$

Table 2.1: Truth table of a $(3, 2)$-function.

Although this is a simple way to represent a VBF, it becomes quite lengthy as n increases, since the size grows exponentially in n.

2.1.1 Algebraic normal form

The truth table is not always the most useful form of representing vectorial Boolean functions in cryptography, because the cryptographic properties of a vectorial Boolean function are not easily captured by such a representation. What is often used, is the following representation. The **Algebraic normal form**, or **ANF** of a vectorial Boolean (n, m)-function is the n-variable polynomial representation, of the form

$$F(x_1, x_2, ..., x_n) = \sum_{I \subseteq \{1,...,n\}} \alpha_I \left(\prod_{i \in I} x_i \right), \quad x_1, x_2, ..., x_n \in \mathbb{F}_2$$

with coefficients $\alpha_I \in \mathbb{F}_2^m$. We define the **algebraic degree $\boldsymbol{d^\circ(F)}$** of a function F as the degree of its ANF, which is equal to the highest number of variables in any term with a nonzero coefficient.

Example 3 *Let F be the function from Example 2. The ANF of F is as follows:*

$$F(x_1, x_2, x_3) = (0,1)x_3 + (1,1)x_2 + (1,0)x_1.$$

Since every term of F with a nonzero coefficient has degree equal to 1, we have $d^\circ(F) = 1$.

Suppose we have an (n,m)-function F. If $d^\circ(F) \leq 1$, as in Example 3, then F is called **affine** and satisfies

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

for all $x, y, z \in \mathbb{F}_{2^n}$. In the special case when F is affine and $F(0, ..., 0) = (0, ..., 0)$, then F is also called **linear** and satisfies

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

for all $x, y \in \mathbb{F}_{2^n}$. If $d^\circ(F) \leq 2$, then F is called **quadratic** and if $d^\circ(F) \leq 3$, F is called **cubic**.

2.1.2 Univariate representation

In the special case when a function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ satisfies $m \mid n$, by identifying the vector space $\mathbb{F}_2^n$ with a finite field $\mathbb{F}_{2^n}$, F can be expressed as a univariate polynomial

$$F(x) = \sum_{i=0}^{2^n-1} c_i x^i, \quad c_i \in \mathbb{F}_{2^n}.$$

This is one of the representations of vectorial Boolean functions that will primarily be used in this thesis alongside the QAM, which we define later. To calculate the algebraic degree of a univariate polynomial, we simply look at the exponent with the highest Hamming weight. The **Hamming weight** of an integer i, denoted by $\boldsymbol{w_2(i)}$, is defined as the number of nonzero entries in the binary expansion of i. We can then calculate the algebraic degree of a univariate polynomial F as follows:

$$d^\circ(F) = \max_{0 \leq i < 2^n,\ c_i \neq 0} w_2(i).$$

If we have a univariate function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ of the form

$$F(x) = \sum_{i=0}^{n-1} c_i x^{2^i}, \quad c_i \in \mathbb{F}_{2^n},$$

then F is linear. We can see that the algebraic degree of F is equal to 1, because the Hamming weight of 2^i is equal to 1 for any positive i. A univariate function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ is quadratic if it can be expressed of the form

$$F(x) = \sum_{i=0}^{n-1} \sum_{j=0, j\neq i}^{n-1} c_{i,j} x^{2^i+2^j} + A(x)$$

for some affine function $A(x) : \mathbb{F}_2^n \to \mathbb{F}_2^m$.

2.1.3 The scale of Boolean functions

In this section, we showcase the reason why finding new Boolean functions and VBFs with certain cryptographic properties is a hard problem, especially as n increases.

Let the set of all Boolean functions over $\mathbb{F}_2^n$ be denoted by BF_n and let $|BF_n|$ denote its cardinality. The Boolean functions used in cryptography satisfying desirable conditions can not be determined or studied by an exhaustive computer investigation with current technology, because the number of n-variable Boolean functions $|BF_n| = 2^{2^n}$ increases exponentially with n, as can be seen in Table 2.2. This problem is even more pronounced when we look at the set of all vectorial Boolean (n, n)-functions BF_n^n, as the number of functions $|BF_n^n| = (2^n)^{2^n}$ increases even faster, with some examples given in Table 2.3. To give some perspective on how large the search space is, it is estimated that there exist approximately 10^{80} atoms in the known universe.

Computational searches are possible, but because the search space is so large, we need to restrict it as much as we can. In later sections, we will revisit how we restricted our search.

n	4	5	6	7	8
$\|BF_n\|$	65536	4294967296	1.84×10^{19}	3.40×10^{38}	1.16×10^{77}

Table 2.2: Growth of $|BF_n|$ as n increases

n	4	5	6	7
$\lvert BF_n^n \rvert$	1.84×10^{19}	1.46×10^{48}	3.94×10^{115}	5.28×10^{269}

Table 2.3: Growth of $|BF_n^n|$ as n increases

2.2 Cryptographic properties

Boolean functions and vectorial Boolean functions play an important part in representing the transformation of data in computer systems. Because of this, they have become a widely researched topic within the field of cryptography. One of the main objectives in studying such functions is to find functions $F : \mathbb{F}_{2^n} \to \mathbb{F}_{2^m}$ that can be used in an encryption scheme, such that the scheme cannot easily be decrypted by malicious actors. Vectorial Boolean functions are one of the fundamental aspects of modern day block ciphers, such as AES and DES, as they are used as S-boxes within the round function. Therefore, these S-boxes have to be constructed in a way that cryptanalytic attacks, such as the differential and linear attack, have as few properties to exploit as possible. The choice of S-boxes is crucial to the level of security of a block cipher, because they are the only nonlinear component in it.

2.2.1 Differential uniformity and APNness

The differential attack proposed by Biham and Shamir in [2] exploited the distribution of the difference between outputs $y_0, y_1 \in \mathbb{F}_{2^n}$ for some difference between inputs $x_0, x_1 \in \mathbb{F}_{2^n},\ x_0 \neq x_1$. The output differential b is the difference between the outputs y_0 and y_1, i.e. $b = y_0 \oplus y_1$. Similarly, the input differential a is the difference between the inputs x_0 and x_1, i.e. $a = x_0 \oplus x_1$. The larger the measured probability of getting b from a, which is the number of pairs with input differential a that produce output pairs with output differential b over all possible pairs, the more efficient the attack is. This leads to a criterion on the S-box functions, namely that the output differentials for any fixed input differential need to be as uniformly distributed as possible.

Let $D_aF(x)$ denote the **derivative** of F in direction a:

$$D_aF(x) = F(x) + F(x + a), \quad x, a \in \mathbb{F}_{2^n},\ a \neq 0$$

and let $\delta_F(a, b)$ denote the number of solutions to the equation $D_aF(x) = b$:

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

The **differential uniformity** of F, denoted by $\boldsymbol{\delta_F}$, is defined as the maximum value of $\delta_F(a, b)$ over all choices of $a, b \in \mathbb{F}_{2^n},\ a \neq 0$. If F has differential uniformity δ, F is called **differentially $\boldsymbol{\delta}$-uniform**.

The differential uniformity δ_F is necessarily even, because if x is a solution to $F(x) + F(x + a) = b$ then $x + a$ is also a solution. As shown by Nyberg in [15], the lower δ_F, the better the contribution of F to the resistance against the differential attack when used in an encryption scheme. When a function F is differentially 2-uniform, it is called **Almost Perfect Nonlinear (APN)**. APN functions provide the strongest resistance against differential attacks.

The **differential spectrum $\boldsymbol{\mathcal{D}_F}$** of a function F is defined as the multiset of the values of $\delta_F(a, b)$ over all $a, b \in \mathbb{F}_{2^n},\ a \neq 0$ and can symbolically be represented as

$$\mathcal{D}_F = [\delta_F(a, b) : \forall a, b \in \mathbb{F}_{2^n},\ a \neq 0],$$

where we use square brackets to denote a multiset. A function is APN if and only if its differential spectrum contains no value higher than 2. The differential spectrum will be revisited later when we discuss invariants.

Searching for new APN functions is a hard problem, which is why the main goal of this thesis is to investigate a computational method based on the work of Yu et al. in [18] for finding new APN functions.

2.2.2 Nonlinearity

The linear attack is another powerful attack against block ciphers and was proposed by Matsui in [14]. The idea of the linear attack is to approximate the VBF used in a cipher by an affine function, and use that approximation to attack the cipher. The reason for this is that affine functions behave predictably and are a well studied topic, making them easy to analyze. To ensure that ciphers are as safe as possible against the linear attack, only VBFs that have minimal correlation with any affine function should be used. We measure a function's resistance to linear attacks by its nonlinearity [6], which we will define in a bit. First, let us define the **Hamming distance** between two (n, n)-functions F and G as

$$d_H(F, G) = |\{x \in \mathbb{F}_{2^n} : F(x) \neq G(x)\}|.$$

Two functions are considered “close” to one another if the Hamming distance between those functions is low, in which case they are easy to approximate by one another. Now, let us define the **nonlinearity** of an $(n, 1)$-function f as

$$\mathcal{NL}(f) = min\{d_H(f, l) : l \in \mathcal{A}_n\},$$

where $\mathcal{A}_n$ denotes the set of all affine $(n, 1)$-functions. We define the nonlinearity of an (n, n)-function F as the minimum nonlinearity of all its component functions F_b:

$$\mathcal{NL}(F) = min\{\mathcal{NL}(F_b) : b \in \mathbb{F}_{2^n} \setminus \{0\}\}.$$

We know that the nonlinearity of any (n, n)-function is upper-bounded [5] by

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

and if F attains this upper bound, then F is said to be **Almost Bent (AB)**. AB functions are the most resilient functions against linear cryptanalysis, which is why it is optimal for VBFs used as S-boxes to be AB, as well as APN. All AB functions are APN as well, but not vice versa, except in the case when n is odd and F is quadratic [5].

2.3 Equivalence of vectorial Boolean functions

When a large set of objects needs to be studied, one way to approach it is to introduce an appropriate equivalence relation and then partition the set into classes under said relation. Then, every object belonging to the same class are considered equivalent and when studying them we only consider one representative from each class. This is the approach we take when studying APN functions. When we classify these functions up to some equivalence relation, it is desirable that the classes are as big as possible in order to reduce the number of functions we need to consider. At the moment, CCZ-equivalence is the most general known equivalence relation that preserves APNness. Therefore, APN functions are classified up to CCZ-equivalence and a function is only considered new if it is CCZ-inequivalent to the known ones. Hence, it is crucial that potential new APN functions are checked for CCZ-equivalence against the known CCZ-classes.

In this section, we will present the most commonly used equivalence relations between vectorial Boolean functions. These include **Linear equivalence**, **Affine equivalence**, **Extended Affine (EA) equivalence** and **Carlet-Charpin-Zinoviev (CCZ) equivalence**. All these equivalence relations preserve APNness and Nonlinearity. We will also present the hierarchy of these equivalence relations, as well as how to simplify testing CCZ-equivalence between two APN functions by using **invariants**.

2.3.1 Affine and Linear equivalence

Two functions F and G over $\mathbb{F}_{2^n}$ are said to be **Affine equivalent** if there exist some affine permutations $A_1, A_2 : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ such that

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

In the special case when A_1 and A_2 are linear permutations, F and G are called **Linear equivalent**.

2.3.2 EA-equivalence

Some functions F and G over $\mathbb{F}_{2^n}$ are said to be **EA-equivalent** if they satisfy

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

for some affine permutations $A_1, A_2 : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ and some affine function $A : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$.

2.3.3 CCZ-equivalence

CCZ-equivalence is currently the most general known form of equivalence that preserves APNness, which is why APN functions are usually classified with regards to it. The concept of CCZ-equivalence is to compare the graphs of some functions F and G. The **graph** of an (n, n)-function F is defined as

$$\Gamma_F = \{(x, F(x)) : x \in \mathbb{F}_{2^n}\},$$

such that Γ_F is a set of elements belonging to $\mathbb{F}_{2^{2n}}$. The functions F and G are said to be **CCZ-equivalent** [12] if there exists an affine $(2n, 2n)$-permutation $A(x)$ such that

$$\{A(x) : x \in \Gamma_F\} = \Gamma_G.$$

The hierarchy between the equivalence relations mentioned so far is as follows: linear equivalence is a special case of affine equivalence, which implies that if two functions are linear equivalent, then they are also affine equivalent, but not necessarily vice versa. Similarly, affine equivalence is a special case of EA-equivalence, and EA-equivalence is a special case of CCZ-equivalence. This hierarchy will become important, because in the special case of two quadratic APN functions (the type of APN functions we search for), they are EA-equivalent to one another if and only if they are CCZ-equivalent as well. We will use this property later when we classify the APN functions we find in our search up to CCZ-equivalence.

2.3.4 Checking functions for equivalence

When classifying newly found APN functions, it is crucial to check whether the functions do not lie in the CCZ-equivalence class of some known function. We do this with the use of linear codes, as shown in [11].

Given an (n, n)-function F, we can associate with it a linear code C_F given by the parity-check matrix

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

where α is a primitive element of $\mathbb{F}_{2^n}$. Note that M_F is a binary code, in which every element of $\mathbb{F}_{2^n}$ is expanded to its n-dimensional binary vector with respect to a basis of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$. Given a function G associated with a linear code C_G, F and G are CCZ-equivalent if and only if their respective linear codes C_F and C_G are isomorphic. Recall that isomorphism means that there exists a permutation π of $\{1, 2, ..., 2^n\}$ such that $(x_1, x_2, ..., x_n)$ is a codeword of C_F if and only if $(x_{\pi(1)}, x_{\pi(2)}, ..., x_{\pi(n)})$ is a codeword of C_G. The advantage of using these linear codes is that coding theory is a well-developed field and algorithms for testing code isomorphism already exist. When we test two functions for CCZ-equivalence in our research, we reduce the problem to a code isomorphism one and make use of the already existing test for code isomorphism in *Magma*. The downside with using this test is that it is rather slow, which is why ortho-derivatives are frequently used to speed up the process.

Ortho-derivatives

The notion of an ortho-derivative of a function was introduced in [4, 16] and is used to induce a number of invariants. These invariants can be used to distinguish between inequivalent quadratic APN function with respect to EA-equivalence. An **invariant** is a property of a function that is preserved by some equivalence relation. Ortho-derivatives are a powerful tool to use because the invariants they induce are fast to compute [12] and are highly accurate at distinguishing between inequivalent functions.

Let F be a quadratic function over $\mathbb{F}_{2^n}$ and let $x \cdot y$ denote the scalar product of x and y for some $x, y \in \mathbb{F}_{2^n}$. An **ortho-derivative** [4] of F is defined as any (n, n)-function π_F such that $\pi_F(0) = 0$ and

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

for all $x, a \in \mathbb{F}_{2^n}$. A quadratic function $F : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ is APN if and only if it has a unique ortho-derivative π_F such that $\pi_F(a) \neq 0$ for all nonzero $a \in \mathbb{F}_{2^n}$ [4].

Ortho-derivatives have the property that if two functions F and G over $\mathbb{F}_{2^n}$ are EA-equivalent via

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

for some affine permutations A_1, A_2 over $\mathbb{F}_{2^n}$ and some affine function A over $\mathbb{F}_{2^n}$, their ortho-derivatives π_F and π_G satisfy

$$\pi_F = A_1^{-1} \circ \pi_G \circ A_2,$$

meaning π_F and π_G are affine equivalent. Ortho-derivatives become useful when we consider their differential spectra as an invariant. Let the **ortho-derivative differential spectrum (ODDS)** of a function F be defined as the differential spectrum of π_F. We then note that the ODDS takes distinct values on almost all EA-inequivalent classes of quadratic APN functions [4]. This is one of the advantages of using ortho-derivatives as a tool to classify quadratic APN functions. The other reason to use ortho-derivatives, is that the ODDS is quickly computed compared to other invariants, such as the Γ- and

Δ-rank [12]. The drawback of using ortho-derivatives is that this only works for quadratic APN functions, whereas other invariants can be used as a tool to classify non-quadratic APN functions as well. This is not an issue for us, as we consider only quadratic APN functions in our search and classifying functions up to CCZ-equivalence (recall that EA-equivalence coincide with CCZ-equivalence in this case) is ideally done with the help of ODDS.

There are other invariants which can be used to distinguish between CCZ-inequivalent functions. Some of these invariants are, as previously mentioned, the **$\boldsymbol{\Gamma}$-rank** and the **$\boldsymbol{\Delta}$-rank**, as well as the **Multiplier Group $|\boldsymbol{\mathcal{M}}(\boldsymbol{G_F})|$** and the **extended Walsh spectrum $\boldsymbol{W_F}$**. As the invariants induced by ortho-derivatives are more useful in the case of this thesis, we do not define them formally here, but instead refer the reader to [12].

2.4 Quadratic APN matrices

The Quadratic APN matrix (QAM) was introduced by Yu et al. in [18] and is a matrix structure that can represent quadratic APN functions and be used to search for new APN functions. In [18], Yu et al. found 470 new classes of CCZ-inequivalent quadratic APN functions over $\mathbb{F}_{2^7}$ and over 1000 new classes over $\mathbb{F}_{2^8}$ by using QAMs. Their work was followed up by Yu and Perrin in [17], where they found 5412 new classes of quadratic APN functions over $\mathbb{F}_{2^8}$. Another paper by Yu et al. [19] was also published, where they introduced restrictions on QAMs corresponding to functions over $\mathbb{F}_{2^n}$ with prime field coefficients. They used this restriction to reduce the search space and to classify all quadratic APN functions over $\mathbb{F}_{2^n}$ with coefficients in $\mathbb{F}_2$ up to $\mathbb{F}_{2^9}$. Further research into QAMs was done by Davidova and Kaleyski in [8]. In their paper, the derivative matrix was introduced, which is a generalization of the notion of a QAM over $\mathbb{F}_{p^n}$ for any characteristic p. The derivative matrix could also represent any quadratic function, not just the ones that were APN. This generalization was used to classify all quadratic planar functions over $\mathbb{F}_{3^n}$ for $n \leq 7$. In this section, we present the state of knowledge behind derivative matrices and QAMs, which will lay the foundation for the work done in this thesis.

2.4.1 Definition and construction

Before we define what a derivative matrix is, it is necessary that we introduce some terminology. Let $M \in \mathbb{F}_{2^n}^{m \times r}$ denote a matrix with m rows and r columns that contains elements in $\mathbb{F}_{2^n}$ and let $M_{i,j}$ denote the entry in the i-th row and j-th column of M, for $0 \leq i \leq m-1$ and $0 \leq j \leq r-1$.

We will now present the concept of a derivative matrix and show how to construct it from a quadratic function F. Let $F : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ be a quadratic function of the form

$$F(x) = \sum_{1 \leq t < i \leq n} c_{i,t} x^{2^{i-1}+2^{t-1}}, \quad c_{i,t} \in \mathbb{F}_{2^n} \tag{2.1}$$

and let $B = \{b_1, b_2, ..., b_n\}$ be a basis of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$. We define the **derivative matrix** [8] of F with respect to B as the matrix $M_F \in \mathbb{F}_{2^n}^{n \times n}$ given by

$$M_F = \begin{bmatrix} \Delta_{b_1} F(b_1) & \Delta_{b_1} F(b_2) & \dots & \Delta_{b_1} F(b_n) \\ \Delta_{b_2} F(b_1) & \Delta_{b_2} F(b_2) & \dots & \Delta_{b_2} F(b_n) \\ \vdots & \vdots & \ddots & \vdots \\ \Delta_{b_n} F(b_1) & \Delta_{b_n} F(b_2) & \dots & \Delta_{b_n} F(b_n) \end{bmatrix},$$

where $\Delta_a F(x) = F(a+x) + F(x) + F(a)$ is a derivative.

We now know how to construct the derivative matrix M_F of a quadratic function F by evaluating the derivatives of F on the basis elements. When we perform computational searches using derivative matrices, it is necessary to also be able to reconstruct the univariate form of F given some derivative matrix M_F. To do this, we use the coefficient matrix C_F of F, which can be obtained by:

$$M_F = V_B^T C_F V_B.$$

Here, V_B is the Vandermonde matrix [13] of the basis B:

$$V_B = \begin{pmatrix} b_1 & b_2 & \dots & b_n \\ b_1^2 & b_2^2 & \dots & b_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ b_1^{2^{n-1}} & b_2^{2^{n-1}} & \dots & b_n^{2^{n-1}} \end{pmatrix},$$

C_F is the coefficient matrix of F:

$$C_F = \begin{pmatrix} 0 & c_{1,2} & \dots & c_{1,n} \\ c_{1,2} & 0 & \dots & c_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{1,n} & c_{2,n} & \dots & 0 \end{pmatrix}$$

and V_B^T is the transpose of V_B. Once we have C_F, we can reconstruct F by using Equation (2.1).

A very useful property of derivative matrices is that we can use them to characterize APN functions. We do this by introducing the notion of a quadratic APN matrix (QAM). To define what it means for a matrix to be QAM, we first need to define the rank of a vector. Let $V = (v_1, v_2, ..., v_m)$ be a vector in $\mathbb{F}_{2^n}^m$. Following [18], we define the **rank** of V to be the dimension of the subspace spanned by its elements $\{v_1, v_2, ..., v_m\}$ over $\mathbb{F}_2$. Now, let $M \in \mathbb{F}_{2^n}^{n\times n}$ be an $n \times n$ matrix defined on $\mathbb{F}_{2^n}$. Then M is called a **Quadratic APN Matrix (QAM)** [18] if:

1. M is symmetric and the elements in its main diagonal are all zeros.
2. Every nonzero linear combination of the n rows (or columns, since M is symmetric) of M has rank $n - 1$.

As shown in [8], a quadratic function F is APN if and only if its derivative matrix M_F is a QAM.

2.4.2 The structure of a derivative matrix over a normal basis

As shown in [8, 19], if a derivative matrix is constructed from a normal basis, it will have a certain structure which can be taken advantage of in a search for new APN functions.

Let $B_N = \{b, b^2, b^{2^2}, ..., b^{2^{n-1}}\}$ be a normal basis of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$ and let $F : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ be a quadratic function. We define the **normal derivative matrix** of F to be the derivative matrix M_F with respect to B_N given by:

$$M_F = \begin{bmatrix} \Delta_b F(b) & \Delta_b F(b^2) & \dots & \Delta_b F(b^{2^{n-1}}) \\ \Delta_{b^2} F(b) & \Delta_{b^2} F(b^2) & \dots & \Delta_{b^2} F(b^{2^{n-1}}) \\ \vdots & \vdots & \ddots & \vdots \\ \Delta_{b^{2^{n-1}}} F(b) & \Delta_{b^{2^{n-1}}} F(b^2) & \dots & \Delta_{b^{2^{n-1}}} F(b^{2^{n-1}}) \end{bmatrix}.$$

Now, if all coefficients c_i of F belong to a subfield $\mathbb{F}_{2^k}$, we know that

$$c_i \in \mathbb{F}_{2^k},\ c_i^{2^k} = c_i.$$

This property will be useful for restricting M_F, but first, let us recall that when we are in a finite field $\mathbb{F}_{2^n}$, we have by the freshmans' theorem:

$$(a+b)^2 = a^2 + b^2.$$

Then we recall that any function F can be written as

$$F(x) = \sum_{i=0}^{2^n-1} c_i x^i, \quad c_i \in \mathbb{F}_{2^n}.$$

Now, if the coefficients of F belong to a subfield $\mathbb{F}_{2^k}$, we can show that:

$$F(x)^{2^k} = \left(\sum_{i=0}^{2^n-1} c_i x^i\right)^{2^k} = \sum_{i=0}^{2^n-1} (c_i x^i)^{2^k} = \sum_{i=0}^{2^n-1} c_i^{2^k} (x^i)^{2^k} = \sum_{i=0}^{2^n-1} c_i (x^i)^{2^k} = F(x^{2^k}).$$

When we apply this to a derivative $\Delta_a F(x)$, we get:

$$\begin{aligned} \Delta_{a^{2^k}} F(x^{2^k}) &= F(x^{2^k} + a^{2^k}) + F(x^{2^k}) + F(a^{2^k}) \\ &= F(x+a)^{2^k} + F(x)^{2^k} + F(a)^{2^k} \\ &= (F(x+a) + F(x) + F(a))^{2^k} \\ &= (\Delta_a F(x))^{2^k}. \end{aligned} \tag{2.2}$$

Let M be the normal derivative matrix of a quadratic function $F : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ with coefficients in $\mathbb{F}_{2^k}$. By using Equation (2.2), we see that if the value at $M_{i,j}$ is known, then the value at $M_{i+k,j+k}$ is simply $(M_{i,j})^{2^k}$. If $i+k \geq n$ or $j+k \geq n$, we simply take $i+k$ *modulo* n or $j+k$ *modulo* n respectively. This is because for any element $b_i \in B_N$, we have $b_i^{2^n} = b_i$, so we get

$$(M_{n-k,n-k})^{2^k} = M_{0,0}.$$

Example 4 *Suppose we have some quadratic $F : \mathbb{F}_{2^4} \to \mathbb{F}_{2^4}$ with coefficients $c_i \in \mathbb{F}_2$, then due to the properties stated above, the normal derivative matrix M_F of F could be written as follows:*

$$M_F = \begin{bmatrix} 0 & A & B & A^{2^3} \\ A & 0 & A^2 & B^2 \\ B & A^2 & 0 & A^{2^2} \\ A^{2^3} & B^2 & A^{2^2} & 0 \end{bmatrix}, \tag{2.3}$$

for some variables $A \in \mathbb{F}_{2^4}, B \in \mathbb{F}_{2^2},\ A, B \neq 0$ (B belongs to $\mathbb{F}_{2^2}$ because we can see that

$B^{2^2} = B$). If the coefficients of F belonged to $\mathbb{F}_{2^2}$ instead, M_F could be written as:

$$M_F = \begin{bmatrix} 0 & A & B & C^{2^2} \\ A & 0 & C & D \\ B & C & 0 & A^{2^2} \\ C^{2^2} & D & A^{2^2} & 0 \end{bmatrix},$$

for $A, C \in \mathbb{F}_{2^4}, B, D \in \mathbb{F}_{2^2},\ A, B, C, D \neq 0$.

To see how much this reduces the search space, let us consider the general case where we do not have any additional information about the coefficients of F. In this case, M_F would have to be a regular derivative matrix and could be written as follows:

$$M_F = \begin{bmatrix} 0 & A & B & C \\ A & 0 & D & E \\ B & D & 0 & F \\ C & E & F & 0 \end{bmatrix},$$

for some variables $A, B, C, D, E, F \in \mathbb{F}_{2^4} \setminus \{0\}$. Not only do we have more variables to consider, but none of these variables are restricted to any subfield.

As we can see from Example 4, by choosing to construct the derivative matrix from a normal basis, we can significantly reduce the search time of computational searches for new APN functions with coefficients in a subfield. Another consequence of constructing the derivative matrix from a normal basis is that we are able to rule out some values of variables early. If A is the first variable set into the matrix M_F from Equation (2.3), then for certain values of A it is already known that M_F will not be QAM for any value of B. Since every linear combination of every row of M_F has to have rank $n-1$ for M_F to be QAM, one can already see after inserting $A = 1$, that row 1 and 2 of M_F will not attain this condition.

2.4.3 The submatrix method

When we search for new APN functions by using derivative matrices, we can make use of submatrices to reduce the amount of matrices that we have to check for being QAM. Let $M \in \mathbb{F}_{2^n}^{n\times n}$ be a derivative matrix and let $0 < m, r \leq n$. The matrix M is QAM if and only if every submatrix $S \in \mathbb{F}_{2^n}^{m\times r}$ of M is proper, for all m, r. We define S as **proper** if every nonzero linear combinations of the m rows has rank at least $r - 1$.

When searching for quadratic APN functions with coefficients in $\mathbb{F}_{2^k}$, due to the structure of the normal derivative matrices, submatrices can be found by taking values at specific row and column indices, as shown in [8].

Example 5 *Let $M_F \in \mathbb{F}_{2^8}^{8\times 8}$ be the normal derivative matrix of some function $F : \mathbb{F}_{2^8} \to \mathbb{F}_{2^8}$ with coefficients in $\mathbb{F}_{2^2}$ and let A, B be the first two variables we would consider in a computational search for new QAMs:*

$$M_F = \begin{bmatrix} 0 & A & B & - & - & - & - & - \\ A & 0 & - & - & - & - & - & - \\ B & - & 0 & - & - & - & - & - \\ - & - & - & 0 & - & - & - & - \\ - & - & - & - & 0 & - & - & - \\ - & - & - & - & - & 0 & - & - \\ - & - & - & - & - & - & 0 & - \\ - & - & - & - & - & - & - & 0 \end{bmatrix}.$$

Because of the subfield the coefficients belong to, we know that $F(x^{2^2}) = F(x)^{2^2}$. Hence, we can derive the following values:

$$M_F = \begin{bmatrix} 0 & A & B & - & - & - & B^{2^6} & - \\ A & 0 & - & - & - & - & - & - \\ B & - & 0 & A^{2^2} & B^{2^2} & - & - & - \\ - & - & A^{2^2} & 0 & - & - & - & - \\ - & - & B^{2^2} & - & 0 & A^{2^4} & B^{2^4} & - \\ - & - & - & - & A^{2^4} & 0 & - & - \\ B^{2^6} & - & - & - & B^{2^4} & - & 0 & A^{2^6} \\ - & - & - & - & - & - & A^{2^6} & 0 \end{bmatrix}.$$

By looking at the submatrices S_1 and S_2 corresponding to the values taken at row and column indices $(\{0,4\},\{2,6\})$ and $(\{0\},\{1,2,4\})$ respectively, the submatrices

$$S_1 = \begin{bmatrix} B & B^{2^6} \\ B^{2^2} & B^{2^4} \end{bmatrix}$$

and

$$S_2 = \begin{bmatrix} A & B & B^{2^6} \end{bmatrix}$$

can be constructed. Since every submatrix of M_F have to be proper for M_F to be QAM, we can already determine early in a search if certain values of variables will not produce QAMs.

In a computational search, as we guess more variables, we can construct even more submatrices. This will let us filter out values of variables that generate submatrices that are not proper early, which helps limit the search time.

2.4.4 EA-equivalence between derivative matrices

The submatrix method is important in a search for new QAMs, but it is not the only method that can be used to limit the search time. Linear functions, more specifically, linear permutations, can also be used. Let $M \in \mathbb{F}_{2^n}^{n \times n}$ be the QAM of some quadratic APN function $F : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ and let l be any linear permutation on $\mathbb{F}_{2^n}$. Following Theorem 3 in [18], any matrix M' such that

$$M'_{i,j} = l(M_{i,j}) \text{ for all } 0 \leq i, j \leq n-1$$

will be the QAM of some function F' that is EA-equivalent to F. The reason for this is that if $M' = l \circ M$, then $F' = l \circ F$. This is an important notion that we will use in Chapter 3.

Chapter 3

Contribution

In previous work, computational classifications of quadratic APN functions using the QAM method [18] were only done for functions with coefficients restricted to the prime field $\mathbb{F}_2$ [19] or $\mathbb{F}_3$ [8]. This thesis is dedicated to exploring the possibility of performing computational classifications with coefficients restricted to a larger subfield. In order to do this, we consider the method of restricting the values of one of the variables in the derivative matrix based on orbits used in [8]. We take this method further by applying it to more than one variable of the derivative matrix. We will explain how we use orbits in more detail later. We then propose a method of approximating the time complexity of this type of search. Using this method, we identify that it is feasible to perform a full classification of all quadratic APN functions with coefficients in $\mathbb{F}_{2^2}$ over $\mathbb{F}_{2^8}$, which has never been done before to the best of our knowledge. The classification also serves as a proof of concept that our method can be extended to even larger dimensions in the future, and that the method that we propose for estimating the time complexity can be used to assess the feasibility of such searches.

In this chapter, we explain how we restrict our search space by partitioning the finite field into orbits under the action of a set of linear permutations. We then estimate how much improvement this method brings. Finally, we give the classification of quadratic APN functions with coefficients in $\mathbb{F}_{2^2}$ over $\mathbb{F}_{2^8}$ and we observe that one of these functions is new.

3.1 Using orbits to restrict variables

The notion of using orbits to restrict the search space of a computational search is one of the core concepts explored in this thesis. In this section, we define an orbit and explain how orbits can be used to restrict a search space.

Let X be a set of elements and G be a group acting on X. Recall that a **group action** of G on X is a mapping $\alpha : G \times X \to X$ satisfying $\alpha(e, x) = x$ and $\alpha(g, \alpha(h, x)) = \alpha(gh, x)$ for any $x \in X$ and any $g, h \in G$, where e is the identity of G. The **orbit** [1] of an element $x \in X$ under the action of G is defined as the set

$$Orb(x, G) = \{g \circ x : g \in G\}.$$

Since G is a group, we know that the set of orbits of X under the action of G form a partition of X into equivalence classes.

Now, we observe that a finite field $\mathbb{F}_{2^n}$ is a set of elements and that a set of linear (n, n)-permutations L is a group acting on $\mathbb{F}_{2^n}$. We can define the orbit of an element $a_i \in \mathbb{F}_{2^n}$ under the action of L as

$$Orb(a_i, L) = \{l(a_i) : l \in L\}.$$

In a computational search for quadratic APN functions over $\mathbb{F}_{2^n}$ using the QAM method, we can use orbits to restrict the number of values we need to check for one entry in a derivative matrix. Let $M \in \mathbb{F}_{2^n}^{n \times n}$ be a derivative matrix and $A \in \mathbb{F}_{2^n} \setminus \{0\}$ be the variable at $M_{0,1}$. Now, we can restrict A to orbits under the action of L, such that we only need to consider one value from each orbit of A. The reason for this is that there exists a linear permutation $l \in L$ such that $l(u) = u'$ for any elements u, u' from the same orbit. By Theorem 3 in [18], for any derivative matrix M whose entry at $M_{0,1}$ is equal to u, there exists a derivative matrix M' whose entry at $M'_{0,1}$ is equal to u' such that M and M' represent EA-equivalent functions. Recall that a function is only considered new if it is CCZ-inequivalent to the known functions, and that EA-equivalent functions are also CCZ-equivalent. Therefore, it is enough to consider only one value from each orbit for the entry at position $M_{0,1}$, which greatly reduces the search space.

In our search, we only consider quadratic functions $F : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ with coefficients in a subfield $\mathbb{F}_{2^k}$, because the derivative matrix M_F of F has a certain structure which reduces the number of variables we need to consider. The problem we face when we restrict a variable in some M_F to orbits is that a derivative matrix $M'_{F'} = l(M_F)$ for some linear (n, n)-permutation l does not necessarily have the same structure as M_F. The reason for this is that the composition of a linear permutation and a function with coefficients in a subfield is not necessarily going to have coefficients in that same subfield. Similarly to what was done in [8], we solve this problem by only choosing linear permutations with coefficients in $\mathbb{F}_{2^k}$

$$L = \{c_0 x + c_1 x^2 + ... + c_{n-1} x^{2^{n-1}} \mid c_0, c_1, ..., c_{n-1} \in \mathbb{F}_{2^k}\}, \tag{3.1}$$

since a composition of two functions with coefficients in the same subfield also has coefficients in that subfield. Suppose we are performing a search over all derivative matrices by exhaustively going over all possible values of these matrices entry by entry. Let $M \in \mathbb{F}_{2^n}^{n \times n}$ be the derivative matrix, constructed from a normal basis, of a quadratic function $F : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ with coefficients in a subfield $\mathbb{F}_{2^k}$ and let $A \in \mathbb{F}_{2^n} \setminus \{0\}$ be the variable in $M_{0,1}$. To restrict A, we take the set of linear permutations in (3.1) and use it to partition all possible values of A into orbits:

$$\{Orb(a_i, L) : a_i \in \mathbb{F}_{2^n} \setminus \{0\}\}.$$

We then only need to evaluate one value from every orbit to obtain all possible CCZ-inequivalent quadratic APN functions with coefficients in $\mathbb{F}_{2^k}$ over $\mathbb{F}_{2^n}$.

Utilizing a set of linear permutations to restrict a single variable to orbits is what was done in [8]. Continuing forward, we will improve upon this method of using orbits to restrict the search space of a computational search. We do this by taking it further, not only restricting a single variable, but multiple variables, and we show just how big an improvement in efficiency this brings.

Let us start with the simple case of restricting a variable pair (A, B) to orbits. First, we start by using the set of linear permutations L in (3.1) to restrict the first variable A as shown previously. We then pick one representative from each orbit. We denote the set of representatives as $A_R = \{a_1, a_2, ..., a_t\}$, where t denotes the number of orbits. Now, for each representative a_i in A_R, we restrict the values of B by partitioning the field into orbits under the action of the stabilizer of a_i

$$L_{a_i} = \{l \in L \mid l(a_i) = a_i\}.$$

This results in the creation of t unique pairs of $(A = a_i, B)$, each corresponding to a set of linear permutations L_{a_i}, from which we restrict B. Since every linear permutation

$l \in L_{a_i}$ must fix a_i, when we restrict B to orbits, we do not change the value of A. When we restrict some variable V to orbits, we call those orbits for the orbits of V.

Example 6 *Suppose we are constructing a normal derivative matrix $M \in \mathbb{F}_{2^4}^{4\times 4}$ of some quadratic function $F : \mathbb{F}_{2^4} \to \mathbb{F}_{2^4}$ with coefficients c_i in $\mathbb{F}_{2^2}$, then M could be written as follows after considering the first variable $A \in \mathbb{F}_{2^4} \setminus \{0\}$:*

$$M = \begin{bmatrix} 0 & A & - & - \\ A & 0 & - & - \\ - & - & 0 & A^{2^2} \\ - & - & A^{2^2} & 0 \end{bmatrix}.$$

When restricting A to orbits we use all the linear permutations of the form

$$c_0x + c_1x^2 + c_2x^{2^2} + c_3x^{2^3},\ c_0, ..., c_3 \in \mathbb{F}_{2^2},$$

which gives us the orbits in Table 3.1.

Orbit number	Orbit values
1	$1, a^5, a^{10}$
2	$a, a^2, a^3, a^4, a^6, a^7, a^8, a^9, a^{11}, a^{12}, a^{13}, a^{14}$

Table 3.1: Orbits of the variable A. The orbit values are the elements of $\mathbb{F}_{2^4} \setminus \{0\}$.

We then choose the smallest value of each orbit as its representative, which gives us the following set of representatives:

$$A_R = \{1, a\}.$$

Instead of checking through 15 values, only 2 values need to actually be checked for the variable A, reducing the number of matrices we need to evaluate.

After considering the second variable B, M could look as follows:

$$M = \begin{bmatrix} 0 & A & - & B^{2^2} \\ A & 0 & B & - \\ - & B & 0 & A^{2^2} \\ B^{2^2} & - & A^{2^2} & 0 \end{bmatrix}.$$

Now, if A is fixed to the representative of its first orbit, $A = 1$, the orbits of B would then be defined by using the stabilizer of 1*:*

$$L_1 = \{l \in L \mid l(1) = 1\}.$$

From L_1, the orbits of $(A = 1, B)$ are the ones we see in Table 3.2.

Orbit number	Orbit values
1	1
2	a, a^2, a^4, a^8
3	$a^{11}, a^{12}, a^{13}, a^{14}, a^3, a^6, a^7, a^9$
4	a^5, a^{10}

Table 3.2: Orbits of the variable B for $A = 1$. The orbit values are the elements of $\mathbb{F}_{2^4} \setminus \{0\}$.

The orbits of $(A = 1, B)$ could be represented by $\{1, a, a^3, a^5\}$. As we can see, instead of checking 15 *values of B when $A = 1$, we only need to check* 4 *values. By performing the same process for $A = a$, we find that there are* 6 *orbits of $(A = a, B)$, with representatives $\{1, a, a^2, a^3, a^4, a^5\}$.*

After considering all variables in M, it could be written as follows:

$$M = \begin{bmatrix} 0 & A & C & B^{2^2} \\ A & 0 & B & D \\ C & B & 0 & A^{2^2} \\ B^{2^2} & D & A^{2^2} & 0 \end{bmatrix},$$

where $A, B \in \mathbb{F}_{2^4},\ C, D \in \mathbb{F}_{2^2},\ A, B, C, D \neq 0$ ($C^{2^2} = C$ and $D^{2^2} = D$ due to M being symmetric). After having restricted the variables A and B to orbits, we can see that instead of having to check $15^2 \times 3^2 = 2025$ matrices for being QAM, we only need to check $4 \times 3^2 + 6 \times 3^2 = 90$ matrices.

Extending this method to triples (A, B, C), we define the orbits of C given some fixed $a_i, b_i, \in \mathbb{F}_{2^n} \setminus \{0\}$, we use the stabilizer subset of L_{a_i} with respect to b_i:

$$L_{a_i,b_i} = \{l \in L_{a_i} \mid l(b_i) = b_i\}.$$

Since all $l \in L_{a_i}$ already satisfy the property $l(a_i) = a_i$, we only need to check the condition $l(b_i) = b_i$. Now, we repeat this process for an arbitrary amount of variables

$(A, B, C, D, ...)$ until the number of orbits does not offer any notable improvement. Since we are continually taking subsets of linear permutations as we restrict more variables to orbits, this usually happens when there are few or no linear permutations which satisfy the stabilizer conditions. Because of this, our method typically works better the larger the initial set of linear permutation L from Equation (3.1).

We can visualize the orbit representatives of some variables $A, B, C, ... \in \mathbb{F}_{2^n} \setminus \{0\}$ created by our method as a tree, which we will call an **orbit tree**. In this orbit tree, the branches connected to the root would be the orbit representatives A_R of the first variable A, then the branches from each $a_i \in A_R$ would be the orbit representatives B_R of the respective orbits (a_i, B) and so forth. The leaves of the tree would be the matrices we need to check for being QAM in a computational search. Figure 3.1 partly shows how such a tree representation would look for the orbits in Example 6.

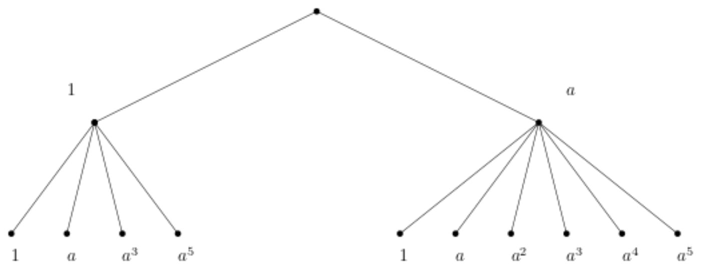

Figure 3.1: The first two levels of the orbit tree of the search from Example 6. The branches from the root are the orbit representatives of A, the next level of branches are the orbit representatives of B. The remaining branches are left out, as the figure would become too large.

Now, we propose a method to calculate an upper bound on the number of leaves in such a tree we must check for being QAM in a computational search for quadratic APN functions with coefficients in $\mathbb{F}_{2^k}$ over $\mathbb{F}_{2^n}$, where $k \mid n$. Let $M \in \mathbb{F}_{2^n}^{n\times n}$ be a normal derivative matrix of some quadratic function F over $\mathbb{F}_{2^n}$, and let $A, B, C, ... \in \mathbb{F}_{2^n} \setminus \{0\}$ denote the variables in M. We can calculate how many matrices we must check by recursively traversing the orbit tree and counting the number of leaves of the tree. To do this, we start by defining orbits of A from the set of linear permutations L from (3.1). We then traverse the orbit representatives A_R of A and define orbits of (a_i, B), $a_i \in A_R$. Similarly, we traverse every set of orbit representatives (a_i, B_R) and define orbits of (a_i, b_i, C), $b_i \in B_R$. We repeat this process recursively until it becomes inefficient to continue defining orbits, in which case we backtrack and count the number of leaves from that branch to be all possible values of the remaining variables. In Algorithm 1, we present the pseudocode of such a method. The pseudocode makes use of an auxiliary function called *Orbits*, which takes as inputs some value $u \in \mathbb{F}_{2^n} \setminus \{0\}$ and a set of linear permutations L and outputs a set of orbit representatives of the orbits defined by L_u, as well as the set L_u itself. It also makes use of some parameters, called max_{depth} and *orbitRepLimit*, which respectively determine how many levels of recursion we perform before backtracking and at what point it becomes inefficient to continue defining orbits.

Algorithm 1 Estimating how many matrices must be checked for being QAM after implementing orbits.

```
1: Input: Orbit representatives of the first variable: $As$, Linear permutations: $L$, remaining variables: $r$
2: Output: The number of matrices that must be checked
3: Parameters: $max_{depth}$, $orbitRepLimit$
4: Aux. functions: $Orbits(value, perms)$
5: Run Est(As, L, r, 1)
6: function Est($orbitReps, perms, r, level$)
7:     if $level \geq max_{depth}$ then
8:         return $|orbitReps| \times (2^n - 1)^r$
9:     $total = 0$
10:    for $value$ in $orbitReps$ do
11:        $nextOrbitReps, nextPerms \leftarrow Orbits(value, perms)$
12:        if $|nextOrbitReps| > orbitRepLimit$ then
13:            $total \leftarrow total + (2^n - 1)^r$
14:        else
15:            $total \leftarrow total + \mathrm{Est}(nextOrbitReps, nextPerms, r - 1, level + 1)$
16:    return $total$
```

3.2 Search for QAMs in $\mathbb{F}_{2^8}$ with coefficients in $\mathbb{F}_{2^2}$

Suppose we are constructing normal derivative matrices for quadratic functions $F : \mathbb{F}_{2^8} \to \mathbb{F}_{2^8}$ with coefficients in $\mathbb{F}_{2^2}$. The structure of such matrices follows a certain pattern, as shown in Section 2.4.2. Let $M \in \mathbb{F}_{2^8}^{8\times 8}$ be a normal derivative matrix representing some function $F : \mathbb{F}_{2^8} \to \mathbb{F}_{2^8}$ with coefficients $c_i \in \mathbb{F}_{2^2}$ and let A be the variable in M at position $M_{0,1}$. As shown in [19], we already know that the value at position $M_{2,3}$ is equal to A^{2^2} and that the value at position $M_{4,5}$ is equal to A^{2^4} and so forth. Because of this, all normal derivative matrices corresponding to quadratic functions $F : \mathbb{F}_{2^8} \to \mathbb{F}_{2^8}$ with coefficients $c_i \in \mathbb{F}_{2^2}$ can be written as follows:

$$M = \begin{bmatrix} 0 & A & B & C & D & E & B^{2^6} & F \\ A & 0 & F^{2^2} & G & E^{2^4} & H & C^{2^6} & G^{2^6} \\ B & F^{2^2} & 0 & A^{2^2} & B^{2^2} & C^{2^2} & D^{2^2} & E^{2^2} \\ C & G & A^{2^2} & 0 & F^{2^4} & G^{2^2} & E^{2^6} & H^{2^2} \\ D & E^{2^4} & B^{2^2} & F^{2^4} & 0 & A^{2^4} & B^{2^4} & C^{2^4} \\ E & H & C^{2^2} & G^{2^2} & A^{2^4} & 0 & F^{2^6} & G^{2^4} \\ B^{2^6} & C^{2^6} & D^{2^2} & E^{2^6} & B^{2^4} & F^{2^6} & 0 & A^{2^6} \\ F & G^{2^6} & E^{2^2} & H^{2^2} & C^{2^4} & G^{2^4} & A^{2^6} & 0 \end{bmatrix},$$

where the variables A, B, C, E, F, G can take every value of $\mathbb{F}_{2^8} \setminus \{0\}$. Note that because the QAM is always symmetric, we have $D = D^{2^4}$ and $H = H^{2^4}$ in this case, which implies $D, H \in \mathbb{F}_{2^4} \setminus \{0\}$.

If we were to go through all functions that could be represented by M by checking all possible value combinations of $A, B, ..., H$, we would have to check $255^6 \times 15^2 \approx 6.19 \times 10^{16}$ different matrices. It takes approximately 0.010 seconds on our server in the worst case to check an 8×8 matrix for being QAM, which means that checking all possible matrices without placing any restrictions on the search would take approximately 6.19×10^{15} seconds, or 196283612 years. This is why it is crucial to optimize the search algorithm in some way if we want to perform a classification with these parameters.

3.2.1 The orbits of our search parameters

Since our search was performed for quadratic functions with coefficients in $\mathbb{F}_{2^2}$ over $\mathbb{F}_{2^8}$, we present some data on the orbits of this special case here. Unfortunately, since the orbit tree expands so quickly for this dimension, it is not possible to show the tree structure visually like it was for Example 6. However, by taking the average number of orbits created from a set of orbit representatives, we can give some insight into how quickly the orbit tree branches nonetheless.

The orbits of the first variable A were constructed from the set of linear permutations of the form:

$$c_0x + c_1x^2 + c_2x^{2^2} + c_3x^{2^3} + c_4x^{2^4} + c_5x^{2^5} + c_6x^{2^6} + c_7x^{2^7}, \quad c_0, c_1, ..., c_7 \in \mathbb{F}_{2^2},$$

Orbit number	Number of elements in orbit	Representative element of orbit
1	3	1
2	192	a
3	48	a^7
4	12	a^{17}

Table 3.3: The orbits of the variable A over $\mathbb{F}_{2^8}$ with coefficients in $\mathbb{F}_{2^2}$

of which there were 24576. By using these permutations, we found 4 orbits for the first variable A, which we show in Table 3.3. From these representatives of A, we obtain the number of orbits of B and an average number of orbits of C, which we show in Table 3.4. We can see in Table 3.4 that the number of orbits seems to increase quite fast, but later

A	Number of orbits of B	Average number of orbits of C
1	8	22.1
a	30	56.7
a^7	22	43.5
a^{17}	14	41.5

Table 3.4: Representative elements of the orbits of A, the number of orbits of B and the average number of orbits of C, taken over all orbits of B

we will use Algorithm 1 to give an upper bound on how many matrices we need to check for our search.

3.3 Results

In this section, we give time estimates of what we will call a brute force search and a submatrix search for QAMs. We give these estimates as a comparison to how long it took to finish our classification, using our method. After that, we compare the difference between restricting only a single variable to orbits and our method of restricting multiple variables. We then show the results of our classification, which include the time it took to complete and the functions we found. All the following results are of searches done in the finite field $\mathbb{F}_{2^8}$ with coefficients in $\mathbb{F}_{2^2}$. We run our searches on a server with a single core speed of 3.2 GHz and 500 GB of memory. We ran 20 cores in parallel when performing our classification.

3.3.1 Time estimate of brute force search

We define a brute force search for QAMs in $\mathbb{F}_{2^8}$ with coefficients in $\mathbb{F}_{2^2}$ as a search where every possible value combination of the variables $A, B, C, E, F, G \in \mathbb{F}_{2^8} \setminus \{0\},\ D, H \in \mathbb{F}_{2^4} \setminus \{0\}$ is checked. In the previous section, we gave a worst case scenario of it taking 196283612 years to perform such a search. Here, we will give a more accurate time estimate by running a brute force search on our server for 24 hours and taking a timestamp at every iteration of the variable E. We then calculate the average time between these timestamps to get an estimate of how long it takes to check all QAMs for one value of E. Finally, we multiply this estimate by the number of possible choices of A, B, C, D, E to get our final time estimate of the search. The results were as follows:

Let T_I denote the average time it takes to check all QAMs for one value of E:

$$T_I = 146.31529610814582 \text{ seconds.}$$

We multiply T_I by the number of possible values of A, B, C, D, E:

$$T_I \times 255^4 \times 15.$$

This gives us an estimated search time of 294262 years, which is comparable to the estimate we got from the worst case scenario.

3.3.2 Time estimate of submatrix search

We define a submatrix search as we did a brute force search, but with the added condition of checking if every submatrix is proper. Estimating the search time after implementing the submatrix method was done similarly to how it was done with the brute force search. The difference is that we have to take into account that we do not check every possible value combination of the variables $\{B, C, D\}$, because some values of those variables will form submatrices that are not proper.

The results were as follows:

$$T_I = 16.594500768019856 \text{ seconds,}$$

where T_I is defined as for the brute force case. We pre-computed that approximately 37.5% of the values of $\{B, C, D\}$ are skipped in total due to forming submatrices that are not proper. Computing

$$T_I \times (255^2 \times 15 \times 0.625) \times 255^2$$

gives us an estimated search time of 20859 years. While this is significantly faster than the brute force estimate, it is still not optimal.

3.3.3 Time estimate of our method

For this method, it is not as straightforward to estimate the time of a search as with the two previous ones. The reason is that some branches of the orbit tree may take a lot longer to search through than others, which means that an average time interval taken at for example the variable E (as done in the previous two methods) will not necessarily give an accurate measurement. Which is why for this method, we use Algorithm 1, which we implemented in *Magma*, to estimate how many matrices we need to check after implementing our method. We then compare it with the number of matrices we need to check in the brute force search.

Let us start by showing how much improvement in search time we get by restricting one variable to orbits. This is simple to calculate, as we know from Table 3.3 that there are 4 orbits of the first variable A. We then calculate that there are

$$\frac{255^6 \times 15^2}{4 \times 255^5 \times 15^2} = 63.75$$

times fewer matrices that we need to check for being QAM than in the brute force method. Knowing this, we apply this factor to the brute force search time estimate to get a new search time estimate of

$$\frac{294262}{63.75} = 4615 \text{ years.}$$

Now, to get an upper bound on how many matrices we must check using our method, we run Algorithm 1. We run it with a max_{depth} of 5 and an $orbitRepLimit$ of $0.8 \times |\mathbb{F}_{2^8}|$ for the variables $\{A, B, C, E, F\}$ (D is skipped due to only belonging to $\mathbb{F}_{2^4}$). This gives us an upper bound of 3439404274500 matrices that we need to check. We compare this with the number of matrices we need to check when performing a brute force search, which gives us a factor of

$$\frac{255^6 \times 15^2}{3439404274500} = 17986. \tag{3.2}$$

After applying this factor to the time estimate of the brute force search, we get the time estimate of our method:

$$\frac{294262}{17986} = 16.36 \text{ years.}$$

There is the question of how much extra time is needed to compute all these orbits. In fact, this takes a negligible amount of time. It took 1355 seconds on our server to run our implementation of Algorithm 1 with the parameters we gave in the above paragraph, in which all orbits needed to get a search time of 16.36 years were computed.

While this approximation of 16.36 years is a good reduction in time from the 294262 years of a brute force search, it is still not optimal, as the submatrix method was not considered here. In the next part, we will discuss using both orbits and the submatrix method, as we discuss the results of our classification.

3.3.4 The time it took to perform the classification

We performed an exhaustive search over all possible matrices corresponding to CCZ-inequivalent quadratic APN functions with coefficients in $\mathbb{F}_{2^2}$ over $\mathbb{F}_{2^8}$, distributed over 20 cores which ran in parallel. The search ran for approximately 1.5 months. Some branches of the orbit tree took longer to search through than others, see Table 3.5 for details.

Orbit	Cores	Time
$A = 1$	4	40 hours
$A = a$	20	1 month
$A = a^7$	20	10 days
$A = a^{17}$	8	7 days

Table 3.5: Orbit, cores used and time spent

The parallelization of the search was managed such that the 20 cores used would each work on one part of a branch. In that way, after every core finished, said branch would be completely searched through. Before starting the search, we would also estimate the search time of the different orbit paths. This was done to decide how many cores needed to be allocated for a certain path. We estimated the time of a path similarly to how we did it for the submatrix method, the difference being we took into account the number of orbits in the path.

Variable	Without orbits	With orbits
B	97.7%	77.0%
C	97.1%	83.9%
D	65.8%	43.9%
E	62.0%	40.5%
F	10.9%	7.6%
G	10.0%	9.4%

Table 3.6: Values that were not filtered out by the submatrix method for each variable, given as a percentage.

One factor that contributed to the speed of the search, was that using our method seemed to synergize well with the submatrix method. In Table 3.6, we see how many values of a variable had to be checked after using the submatrix method, with and without orbits.

3.3.5 Functions found

In total, 196863 quadratic APN functions were found in our search, with 27 unique ortho-derivative differential spectra. A list of all functions sorted by CCZ-equivalence can be found on `https://github.com/Simon-Berg/thesis`. The ODDS found can be seen in Table 3.8 and representative polynomials of those ODDS can be seen in Table 3.7. All functions with equal ODDS were tested for CCZ-equivalence using the test from [11] and were verified to be CCZ-equivalent. After the classification, we established that one of the functions we found is new and is also 3-to-1. A representative polynomial of this new function and its ODDS is highlighted with bold text in Table 3.7 and 3.8 respectfully. Additionally, we provide some of the invariants of this new function in Table 3.9.

The ODDS are represented by a list of integers raised to some power. Recall that the differential spectrum of a function is a multiset of values defined by:

$$\mathcal{D}_F = [\delta_F(a,b) : a, b \in \mathbb{F}_{2^n},\ a \neq 0].$$

We represent such a multiset by the integers that occur in the multiset, raised to the power of how many times they occur. For example, the ODDS at index 1 in Table 3.8 is:

$$[0^{35700}, 2^{26520}, 4^{3060}]$$

which denotes the multiset in which the value 0 occurs 35700 times, 2 occurs 26520 times and 4 occurs 3060 times.

Index	Representative functions
1	$x^{192} + x^{96} + x^{72} + x^{33} + x^{3}$
2	$a^{170}x^{144} + a^{85}x^{72} + a^{85}x^{48} + a^{85}x^{24} + a^{85}x^{12} + x^{9} + x^{3}$
3	$x^{72} + x^{66} + x^{48} + x^{36} + a^{170}x^{33} + x^{18} + x^{9} + a^{170}x^{6} + a^{85}x^{3}$
4	$a^{170}x^{192} + a^{85}x^{144} + a^{85}x^{132} + a^{170}x^{66} + a^{85}x^{48} + x^{33} + x^{24} + a^{170}x^{18} + a^{85}x^{6} + x^{3}$
5	$a^{170}x^{144} + a^{85}x^{132} + x^{96} + a^{170}x^{72} + a^{170}x^{36} + x^{33} + a^{85}x^{24} + x^{12} + x^{3}$
6	$a^{170}x^{192} + a^{85}x^{132} + a^{170}x^{66} + a^{170}x^{24} + a^{170}x^{18} + a^{170}x^{12} + x^{6} + x^{3}$
7	$a^{85}x^{192} + a^{170}x^{144} + a^{170}x^{96} + a^{85}x^{72} + a^{170}x^{66} + x^{36} + x^{12} + x^{3}$
8	$a^{85}x^{144} + a^{170}x^{129} + a^{170}x^{72} + a^{170}x^{18} + x^{12} + a^{85}x^{9} + x^{3}$
9	$a^{170}x^{132} + a^{170}x^{96} + a^{85}x^{72} + a^{85}x^{66} + x^{48} + a^{170}x^{18} + x^{6} + x^{3}$
10	$\boldsymbol{a^{85}x^{96} + a^{85}x^{72} + a^{170}x^{24} + x^{18} + a^{85}x^{12} + a^{85}x^{9} + x^{6} + x^{3}}$
11	$x^{132} + a^{85}x^{96} + a^{170}x^{72} + a^{85}x^{48} + x^{33} + x^{24} + a^{170}x^{12} + a^{170}x^{6} + x^{3}$
12	$a^{170}x^{144} + a^{85}x^{132} + x^{72} + a^{170}x^{48} + a^{170}x^{24} + x^{18} + a^{170}x^{12} + x^{3}$
13	$a^{85}x^{144} + a^{85}x^{66} + a^{170}x^{65} + a^{85}x^{48} + x^{36} + a^{170}x^{33} + a^{170}x^{20} + a^{170}x^{18} + x^{9} + x^{3}$
14	$a^{170}x^{160} + a^{85}x^{144} + a^{85}x^{132} + a^{85}x^{96} + a^{85}x^{80} + a^{85}x^{68} + a^{85}x^{66} + a^{85}x^{48} + x^{18} + a^{170}x^{5} + x^{3}$
15	$a^{85}x^{160} + a^{85}x^{136} + a^{85}x^{96} + a^{170}x^{40} + a^{85}x^{36} + a^{85}x^{34} + x^{20} + a^{85}x^{17} + x^{12} + x^{9} + x^{3}$
16	$x^{160} + a^{170}x^{144} + a^{85}x^{129} + a^{170}x^{96} + a^{170}x^{68} + a^{170}x^{40} + x^{20} + a^{85}x^{18} + a^{170}x^{12} + x^{9} + x^{3}$
17	$a^{85}x^{192} + a^{85}x^{160} + a^{85}x^{132} + a^{170}x^{72} + x^{48} + x^{40} + x^{34} + x^{33} + x^{18} + a^{170}x^{10} + x^{3}$
18	$a^{85}x^{136} + a^{85}x^{132} + x^{96} + a^{170}x^{72} + a^{85}x^{68} + a^{85}x^{66} + x^{48} + x^{33} + a^{85}x^{17} + x^{3}$
19	$a^{170}x^{192} + a^{85}x^{132} + x^{129} + a^{85}x^{80} + a^{85}x^{68} + x^{48} + x^{24} + x^{20} + x^{10} + a^{85}x^{5} + x^{3}$
20	$a^{170}x^{144} + a^{170}x^{136} + x^{132} + a^{170}x^{80} + a^{85}x^{66} + a^{170}x^{65} + a^{170}x^{34} + a^{170}x^{33} + x^{24} + a^{170}x^{18} + x^{9} + a^{85}x^{6} + x^{3}$
21	$a^{170}x^{144} + x^{136} + x^{129} + a^{85}x^{68} + a^{85}x^{66} + x^{65} + a^{170}x^{48} + a^{170}x^{40} + a^{85}x^{36} + x^{33} + a^{170}x^{17} + a^{170}x^{12} + x^{3}$
22	$x^{192} + a^{85}x^{144} + a^{85}x^{68} + a^{170}x^{65} + a^{85}x^{48} + a^{170}x^{40} + a^{85}x^{24} + a^{170}x^{20} + a^{170}x^{18} + a^{85}x^{17} + a^{170}x^{10} + x^{3}$
23	$a^{85}x^{192} + a^{85}x^{144} + x^{36} + x^{33} + a^{170}x^{24} + a^{170}x^{18} + x^{12} + x^{6} + x^{3}$
24	$x^{192} + x^{160} + a^{85}x^{130} + a^{85}x^{96} + a^{170}x^{72} + x^{66} + a^{170}x^{48} + x^{40} + a^{85}x^{33} + a^{85}x^{18} + x^{5} + x^{3}$
25	$a^{85}x^{192} + a^{170}x^{160} + x^{144} + x^{130} + a^{170}x^{129} + x^{65} + a^{170}x^{40} + a^{85}x^{34} + a^{85}x^{24} + a^{170}x^{20} + a^{170}x^{18} + a^{170}x^{9} + x^{3}$
26	$a^{85}x^{129} + a^{85}x^{96} + x^{72} + a^{85}x^{66} + x^{12} + a^{170}x^{9} + x^{6} + x^{3}$
27	$a^{170}x^{160} + a^{170}x^{136} + a^{85}x^{132} + a^{170}x^{129} + a^{85}x^{68} + a^{170}x^{40} + x^{33} + a^{85}x^{24} + a^{85}x^{9} + a^{170}x^{6} + x^{3}$

Table 3.7: Representatives up to CCZ-equivalence of all quadratic APN functions with coefficients in $\mathbb{F}_{2^2}$ over $\mathbb{F}_{2^8}$. Bold text denotes the new function.

Index	ODDS
1	$0^{35700}, 2^{26520}, 4^{3060}$
2	$0^{36420}, 2^{25080}, 4^{3780}$
3	$0^{37872}, 2^{22788}, 4^{4068}, 6^{492}, 8^{60}$
4	$0^{37980}, 2^{22272}, 4^{4716}, 6^{312}$
5	$0^{38004}, 2^{22614}, 4^{4008}, 6^{630}, 10^{24}$
6	$0^{38040}, 2^{22461}, 4^{4218}, 6^{513}, 8^{36}, 10^{12}$
7	$0^{38160}, 2^{22104}, 4^{4536}, 6^{456}, 8^{24}$
8	$0^{38160}, 2^{22164}, 4^{4428}, 6^{492}, 8^{36}$
9	$0^{38184}, 2^{22179}, 4^{4338}, 6^{531}, 8^{48}$
10	$\mathbf{0^{38196}, 2^{22008}, 4^{4608}, 6^{456}, 8^{12}}$
11	$0^{38256}, 2^{22116}, 4^{4230}, 6^{648}, 8^{30}$
12	$0^{38592}, 2^{21426}, 4^{4590}, 6^{654}, 8^{18}$
13	$0^{38844}, 2^{20974}, 4^{4764}, 6^{654}, 8^{44}$
14	$0^{38880}, 2^{21165}, 4^{4230}, 6^{1005}$
15	$0^{39174}, 2^{20513}, 4^{4756}, 6^{749}, 8^{76}, 10^{8}, 12^{4}$
16	$0^{39290}, 2^{20399}, 4^{4686}, 6^{774}, 8^{112}, 10^{15}, 12^{4}$
17	$0^{39408}, 2^{20072}, 4^{4922}, 6^{798}, 8^{70}, 10^{10}$
18	$0^{39408}, 2^{20218}, 4^{4692}, 6^{838}, 8^{104}, 10^{12}, 12^{8}$
19	$0^{39444}, 2^{20042}, 4^{4912}, 6^{762}, 8^{112}, 10^{8}$
20	$0^{39446}, 2^{20067}, 4^{4896}, 6^{718}, 8^{138}, 10^{15}$
21	$0^{39504}, 2^{20127}, 4^{4674}, 6^{801}, 8^{138}, 10^{27}, 16^{6}, 18^{3}$
22	$0^{39544}, 2^{19996}, 4^{4752}, 6^{841}, 8^{130}, 10^{12}, 12^{2}, 14^{1}, 18^{2}$
23	$0^{39600}, 2^{19680}, 4^{5220}, 6^{600}, 8^{180}$
24	$0^{39692}, 2^{19752}, 4^{4756}, 6^{978}, 8^{72}, 10^{26}, 12^{4}$
25	$0^{39750}, 2^{19641}, 4^{4876}, 6^{845}, 8^{136}, 10^{29}, 14^{2}, 18^{1}$
26	$0^{39780}, 2^{21930}, 6^{3570}$
27	$0^{39840}, 2^{19707}, 4^{4644}, 6^{900}, 8^{120}, 10^{9}, 14^{60}$

Table 3.8: The ortho-derivative differential spectra of all quadratic APN functions with coefficients in $\mathbb{F}_{2^2}$ over $\mathbb{F}_{2^8}$. Bold text denotes the ODDS of the new function. We omit the square brackets around the ODDS for aesthetic reasons.

$f(x)$	$a^{170}x^{192} + a^{170}x^{144} + a^{85}x^{48} + x^{36} + a^{170}x^{24} + a^{170}x^{18} + x^{12} + x^{6}$
ODDS	$0^{38196}, 2^{22008}, 4^{4608}, 6^{456}, 8^{12}$
Γ-rank	14034
Δ-rank	438
$\lvert\mathcal{M}(G_F)\rvert$	3072
$\mathcal{NL}$	112
W_F	$-32^{2380}, -16^{20400}, 0^{16320}, 16^{23120}, 32^{3060}$

Table 3.9: Invariants of the new function, see Section 2.3.4 for details.

Orbits	Fs	ODDS
$A = 1$	3360	24
$A = a$	144379	27
$A = a^7$	40720	27
$A = a^{17}$	8404	27

Table 3.10: Orbits of A, number of APN functions found and number of ODDS found.

Now, let us look at the distribution of all the functions we found over the orbits of A and some of its implications. As we can see in Table 3.10, although many more functions were found for the largest orbit $A = a$ than for the orbits $A = a^7$ and $A = a^{17}$ (recall Table 3.3), we can see that functions having the same ODDS were found regardless of which representative was chosen to be the value of A. While this may only be the case for this particular search, it may also imply that we do not need to search through every orbit of A to find all CCZ-inequivalent functions for other searches as well.

Chapter 4

Conclusion

In this thesis, we have developed an improved method of using orbits to significantly speed up the running time of computational searches for quadratic APN functions with coefficients in a subfield, based on the QAM method by Yu et al. [18]. Previously, orbits had only been used to restrict the values of a single variable in a normal derivative matrix. Our method extended its use to restricting multiple variables, which greatly reduces the number of matrices that have to be checked. We used our improved method to classify all quadratic APN functions with coefficients in $\mathbb{F}_{2^2}$ over $\mathbb{F}_{2^8}$, which has never been done before to the best of our knowledge. The classification resulted in finding a new 3-to-1 APN function. Additionally, the classification serves as a proof of concept for our method, which can be applied to other searches, as described in the following section.

4.1 Future work

As for future work, we suggest exploring the possibility of using our method to perform a computational search for functions with coefficients in $\mathbb{F}_{2^4}$ over $\mathbb{F}_{2^8}$, $\mathbb{F}_{2^3}$ over $\mathbb{F}_{2^9}$ and $\mathbb{F}_{2^2}$ over $\mathbb{F}_{2^{10}}$. We also suspect that our method would work well for a computational search for planar functions (planar functions were not discussed in this thesis, but we refer the reader to [7] for details), which are defined over $\mathbb{F}_{p^n}$ for odd p. Additionally, our method might be used in a search for interesting functions, such as differentially 4-uniform functions.

Bibliography

[1] Michael Aschbacher. *Finite group theory*, volume 10. Cambridge University Press, 2000.

[2] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *J. Cryptology*, 4:3–72, 1991. doi: 10.1007/BF00630563.

[3] 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.

[4] Anne Canteaut, Alain Couvreur, and Léo Perrin. Recovering or testing extended-affine equivalence. *IEEE Transactions on Information Theory*, 68(9):6187–6206, 2022. doi: 10.1109/TIT.2022.3166692.

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

[6] Florent Chabaud and Serge Vaudenay. Links between differential and linear cryptanalysis. In *Advances in Cryptology—EUROCRYPT'94: Workshop on the Theory and Application of Cryptographic Techniques Perugia, Italy, May 9–12, 1994 Proceedings 13*, pages 356–365. Springer, 1995.

[7] Robert S Coulter and Rex W Matthews. Planar functions and planes of Lenz-Barlotti class II. *Designs, Codes and Cryptography*, 10(2):167–184, 1997.

[8] Diana Davidova and Nikolay Kaleyski. Classification of all DO planar polynomials with prime field coefficients over $GF(3^n)$ for n up to 7. *Cryptology ePrint Archive*, 2022.

[9] John F Dillon. APN polynomials: an update. In *International Conference on Finite fields and applications-Fq9*, 2009.

[10] Yves Edel and Alexander Pott. A new almost perfect nonlinear function which is not quadratic. *Cryptology ePrint Archive*, 2008.

[11] 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.

[12] Nikolay S Kaleyski. Invariants for EA-and CCZ-equivalence of APN and AB functions. *Cryptography and Communications*, 13:995–1023, 2021.

[13] Allen Klinger. The vandermonde matrix. *The American Mathematical Monthly*, 74 (5):571–574, 1967.

[14] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology—EUROCRYPT'93: Workshop on the Theory and Application of Cryptographic Techniques Lofthus, Norway, May 23–27, 1993 Proceedings 12*, pages 386–397. Springer, 1994.

[15] Kaisa Nyberg and Lars Ramkilde Knudsen. Provable security against a differential attack. *Journal of Cryptology*, 8:27–37, 1995.

[16] Léo Perrin. How to take a function apart with sboxu. In *BFA 2020-The 5th International Workshop on Boolean Functions and their Applications*, 2020.

[17] Yuyin Yu and Léo Perrin. Constructing more quadratic APN functions with the QAM method. *Cryptography and Communications*, 14(6):1359–1369, 2022.

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

[19] Yuyin Yu, Nikolay Kaleyski, Lilya Budaghyan, and Yongqiang Li. Classification of quadratic APN functions with coefficients in $\mathbb{F}_2$ for dimensions up to 9. *Finite Fields and Their Applications*, 68:101733, 2020.