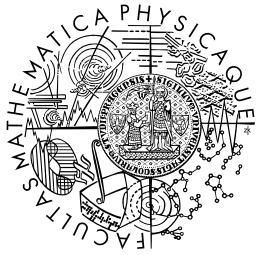

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Jiří Pavlů

Search for APN permutations among known APN functions

Department of Algebra

Supervisor of the master thesis:	Dr. rer. nat. Faruk Göloğlu
Study programme:	Mathematics
Study branch:	Mathematics for Information Technology

Prague 2018

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

I would like to thank my supervisor Dr. rer. nat. Faruk Göloğlu for consultations, tips for literature, and for asking the right questions.

Title: Search for APN permutations among known APN functions

Author: Jiří Pavlů

Department: Department of Algebra

Supervisor: Dr. rer. nat. Faruk Göloğlu, Department of Algebra

Abstract: In the thesis a new way of checking whether a function is CCZ–equivalent to a permutation is given. The results for known families of almost perfect nonlinear (APN) functions are presented functions defined over $\mathbb{F}_2^n$ for even $n \leq 12$. The ways how to reduce the number of polynomials from each family are studied. For functions of the form $x^3 + a^{-1}\mathrm{tr}_1^n(a^3x^9)$ it is shown, that they cannot be CCZ–equivalent to a permutation on fields $\mathbb{F}_2^{4n}$ for $n \in \mathbb{N}$.

Keywords: vectorial Boolean functions, APN permutations, CCZ–equivalenc,e computational proof

Contents

Introduction

The existence of APN permutations in even dimension is an interesting problem, for which only a partial answer is known. While for odd dimensions we know many APN permutations – for example Gold functions:

$$f(x) = x^{2^i+1},$$

for $\gcd(i, n) = 1$, but for even dimensions Gold functions are still APN, but they are not permutations.

The fact that no APN permutation in even dimension could be found led many people to conjecture, that actually no such function exist. This belief was then strengthened when it was found than in $\mathbb{F}_{2^4}$, there indeed is no APN permutation, as noted by Hou in [1]. However in [2] Dillon presents a new algorithm for checking whether an APN function is CCZ–equivalent to a permutation and used it on known APN functions in even dimensions up to dimension 10. This approach was successful in finding an APN permutation in dimension 6. The function that was found to be CCZ–equivalent to a permutation is *Kim function*:

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

where u is the primitive element whose primitive polynomial over $\mathbb{F}_2$ is $x^6 + x^4 + x^3 + x + 1$. The APN permutation found is then of the form $g = f_2 \circ f_1^{-1}$, for $f_1 = L_1(x) + L_2(\kappa(x))$, $f_1 = L_3(x) + L_4(\kappa(x))$, where L_1, L_2, L_3 and L_4 are linear functions on $\mathbb{F}_{2^6}$. Specifically:

$$\begin{aligned} f_1 =& w^{38}x^{48} + w^{33}x^{40} + w^{28}x^{34} + w^{25}x^{33} + w^{43}x^{32} \\ &+ w^5x^{24} + w^{42}x^{20} + x^{17} + w^2x^{16} + w^4x^{12} \\ &+ w^7x^{10} + w^{58}x^8 + w^{59}x^6 + w^5x^5 + w^{36}x^4 \\ &+ w^{47}x^3 + w^{30}x^2 + w^9x \end{aligned}$$

$$\begin{aligned} f_2 =& w^{26}x^{48} + w^{60}x^{40} + w^{46}x^{34} + w^6x^{33} + w^{61}x^{32} \\ &+ w^{51}x^{24} + w^{53}x^{20} + w^{61}x^{17} + w^{54}x^{16} + w^{55}x^{12} \\ &+ w^{33}x^{10} + w^{33}x^8 + w^{19}x^6 + w^{46}x^5 + w^{51}x^4 \\ &+ w^{16}x^3 + w^{37}x^2 + w^{27}x \end{aligned},$$

for $w = u^{-2}$. And it is up to equivalence the only known APN permutation in even dimensions.

A really interesting is the case $\mathbb{F}_{2^8}$. AES (Advanced Encryption Standard) uses an 8–bit substitution function, but as not a single APN permutation on field $\mathbb{F}_{2^8}$ is known, it uses the inverse function, which is resilient against known attack, e.g. differential and linear cryptanalysis, but it is not APN, and therefore not optimally resilient. So the existence of APN permutation on 8 bits would mean, that the substitution function of AES is not optimal.

The algorithm given by Dillon in [2] is very slow for higher dimensions. This thesis is concerned with improving it to be usable in higher dimensions. For this we present a new condition for a function to be CCZ–*inequivalent* to a permutation, along with some ideas to further speed up the checking process.

In first chapter we cover the basics of finite fields, vectorial Boolean functions and characters. The second chapter is then concerned with the previous work, done in this area. We show the algorithm used by Dillon, and reprove the condition that is used in [2] using the terminology we will use for our own condition. In the third chapter we then present our approach, along with the computational results we have got, and compare our condition with the condition used by Dillon in [2]. Additionally we present a proof of the fact, that the functions of the form

$$x^3 + a^{-1}\mathrm{tr}_1^n(a^3x^9)$$

cannot be CCZ–equivalent to a permutation on fields $\mathbb{F}_{2^{4n}}$ for $n \in \mathbb{N}$.

1. Preliminaries

This thesis deals with the theory of vectorial Boolean functions. Therefore it is important for the reader to have at least a basic knowledge of finite fields, Boolean functions, coding theory, and their importance in cryptology. This brief introduction should cover all the theory used in the thesis.

1.1 Finite fields

Definition 1 (Group)**.** *Let G be a set, and let us have an operation "$\cdot$" from $G \times G$ to G. We call $(G, \cdot)$ a* group *if the following holds:*

- *associativity:* $\forall a, b, c \in G : (a \cdot b) \cdot c = a \cdot (b \cdot c)$,
- *existence of neutral element:* $\exists e \in G \forall a \in G : a \cdot e = e \cdot a = a$,
- *existence of inverse element:* $\forall a \in G \exists b \in G : a \cdot b = b \cdot a = e$.

Moreover if the operation $\cdot$ is commutative ($\forall a, b \in G : a \cdot b = b \cdot a$), then $(G, \cdot)$ is called a commutative *group or an* abelian *group.*

Definition 2 (Field)**.** *Let $\mathbb{F}$ be a nonempty set with at least two elements, and let us have two operations: "$+$" : $\mathbb{F} \times \mathbb{F} \to \mathbb{F}$ (addition) and "$\cdot$" : $\mathbb{F} \times \mathbb{F} \to \mathbb{F}$ (multiplication). We call $(\mathbb{F}, +, \cdot)$ a* field *if the following holds:*

- $(\mathbb{F}, +)$ *is an abelian group,*
- $(\mathbb{F} \backslash \{0\}, \cdot)$ *is an abelian group (where 0 is the neutral element of $(\mathbb{F}, +)$),*
- *distributivity:* $\forall a, b, c \in \mathbb{F} : (a + b) \cdot c = a \cdot c + b \cdot c$.

If $\mathbb{F}$ is finite, then we call $(\mathbb{F}, +, \cdot)$ a finite *field. For the sake of brevity from now on we write just $\mathbb{F}$ instead of $(\mathbb{F}, +, \cdot)$ when appropriate.*

It can be shown, that a finite field with q elements exist if and only if q is a prime power. Moreover for a given prime power p^n the finite field is unique (up to isomorphism). Therefore by $\mathbb{F}_{p^n}$ we shall denote the unique finite field with p^n elements. The prime p is called the *characteristic* of the field.

Now why are we even interested in finite fields? Every vector space over field $\mathbb{F}_p$ can be interpreted as a finite field of characteristic p. Since computers operate on vectors over $\mathbb{F}_2$ (they are called registers) and 2 is a prime, we can interpret these operations as operations over a finite field. Now finite fields have nice properties that make our life easier – all functions defined over a finite field can be written as polynomials and generally fields have richer structure that can be used, compared to just vector spaces.

As finite fields have one–to–one correspondence to vector spaces over $\mathbb{F}_p$, we can see a finite field as a vector space and vice–versa. Therefore in the thesis we use notation like $\mathbb{F}_2^{n^*}$ meaning $\mathbb{F}_2^n$ without an all–zero vector as motivated by the theory of finite fields (the all-zero vector in the vector space corresponds to the zero element of the finite field).

Another property of finite fields is the so-called *freshman's dream.*

Remark 1 (freshman's dream). Let $\mathbb{F}_q$ be a finite field, with $q = p^n$ for some $n \in \mathbb{N}$. Then for every $a, b \in \mathbb{F}_q$ and for every $k \in \mathbb{N}$ it holds, that:

$$(a+b)^{p^k} = a^{p^k} + b^{p^k}.$$

The case when $k = 1$ is easily seen from the binomial theorem, and from the fact, that a finite field of characteristic p is a vector space over $\mathbb{F}_p$ where $0 = p$. For other $k \in \mathbb{N}$ the proof can then be done by induction.

1.2 Boolean functions

Boolean and vectorial Boolean functions are functions, that are defined on a finite field of characteristic 2 (or on a vector space over $\mathbb{F}_2$). Therefore they are of great interest in modern computer–oriented cryptology.

Definition 3 (Boolean function)**.** *Let f be a function. We call f a* Boolean function *if there is $n \in \mathbb{N}$ such that: $Dom(f) = \mathbb{F}_2^n$ and $Im(f) = \mathbb{F}_2$.*

Definition 4 (Vectorial Boolean function)**.** *Let f be a function. We say, that f is a* vectorial Boolean function *if there are $n, m \in \mathbb{N}$ such that: $Dom(f) = \mathbb{F}_2^n$ and $Im(f) = \mathbb{F}_2^m$.*

We can write every Boolean function in its *algebraic normal form*. That is in a form:

$$f(\mathbf{x}) = f(x_1, x_2, \ldots, x_n) = \sum_{u \in \mathbb{F}_2^n} a_u \mathbf{x}^u,$$

where $\mathbf{x}^u = \prod_{i=1}^n {x_i}^{u_i}$, and $a_u \in \mathbb{F}_2$. Moreover by a simple counting argument it can be shown, that there is a one–to–one correspondence between algebraic normal forms, and Boolean functions over the same vector space (or field). As the vectorial Boolean function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ can be seen as an m–dimensional vector of Boolean functions, the algebraic normal form of a vectorial Boolean function is:

$$f(\mathbf{x}) = f(x_1, x_2, \ldots, x_n) = \sum_{u \in \mathbb{F}_2^n} a_u \mathbf{x}^u,$$

where $a_u \in \mathbb{F}_2^m$.

This means that the all functions, we can be interested in, are in fact polynomials.

Nevertheless, as we noted before there is a connection between this notation is more vector space oriented (it is called *multivariate* for obvious reasons). But we can fix a basis $\beta_1, \ldots, \beta_n$ of the finite field $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$, and see, that each basis element β_i corresponds to a vector, whose i–th coordinate is 1,and the rest is zero. Note, that this correspondence is dependent on the given basis.

Therefore in this *univariate* notation all functions can be expressed as polynomials from $\mathbb{F}_{2^n}$ of degree less then or equal to $2^n - 1$ (since $\alpha^{2^n} = \alpha$ for every $\alpha \in \mathbb{F}_{2^n}$ by the order of its multiplicative group). So in univariate notation the algebraic normal form of a vectorial Boolean function becomes:

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

for $a_i \in \mathbb{F}_{2^n}$.

Since vectorial Boolean functions are still a relatively hard to deal with, we want to divide them into parts, and study each part separately. For this we will use the notions of *coordinante* and *component* functions.

Definition 5 (Coordinate and component functions)**.** *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a vectorial Boolean function. Then $f(x)$ can be written as $(f_1(x), f_2(x), \ldots, f_m(x))$, where $f_1, \ldots, f_m$ are Boolean functions from F_2^n. The functions $f_1, \ldots, f_m$ are called the* coordinate functions *of f. Any nontrivial linear combination of $f_1, \ldots, f_m$ is then called a* component function *(or component) of f.*

Note that, all component functions can be expressed as $< f(x)|u >$, where $< x|y >$ denotes a scalar product, and $u \in \mathbb{F}_2^n$ – different u give different linear combinations of coordinate functions. To be able to use these terms even while using univariate notation (and much more) we will need to use the *trace* function.

Definition 6 (Trace)**.** *Let $n > m$, $m|n$. Then we call the function from $\mathbb{F}_{2^n}$ to $\mathbb{F}_{2^m}$ such that:*

$$\mathrm{tr}_m^n(\alpha) = \sum_{i=0}^{\frac{n}{m}-1} \alpha^{2^{mi}},$$

the trace *function from $\mathbb{F}_{2^n}$ to $\mathbb{F}_{2^m}$.*

Remark 2 (Linearity of trace). The trace from $\mathbb{F}_{2^n}$ to $\mathbb{F}_{2^m}$ is a $\mathbb{F}_{2^m}$–linear function:

$$\mathrm{tr}_m^n(ax + by) = \mathrm{tr}_m^n(ax) + \mathrm{tr}_m^n(by) = a\ \mathrm{tr}_m^n(x) + b\ \mathrm{tr}_m^n(y),$$

for every $a, b \in \mathbb{F}_{2^m}$ and every $x, y \in \mathbb{F}_{2^n}$.

Another easy observation is that:

$$\mathrm{tr}_m^n(\alpha) = \mathrm{tr}_m^n(\alpha^{2^{md}}),$$

for any $d \in \mathbb{N}$.

So we know, that for all β is $L_\beta(x) = \mathrm{tr}_1^n(\beta x)$ a linear mapping from $\mathbb{F}_{2^n}$ to $\mathbb{F}_2$. They are also pairwise different. Actually the trace map satisfies all conditions for being a scalar product. Now we recall, that every element in $\mathbb{F}_{2^n}$ can be represented in terms of a base of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$. Therefore we see, that there is one–to–one correspondence between mappings $< f(x)|u >$ and $\mathrm{tr}_1^n(\alpha f(x))$ and therefore we see, that all component functions of $f(x)$ are of the form $\mathrm{tr}_1^n(\alpha f(x))$ for some $\alpha \in \mathbb{F}_{2^n}$, and that in this way we indeed get all component functions of f.

An important characteristic of (vectorial) Boolean functions is their degree. Since we can see these function in two ways (over a field and over a vector space), we have two different notions of their degree.

Definition 7 (Univariate degree)**.** *Let $f = \sum_{i=0}^{2^n-1} a_i x^i$ be a vectorial Boolean function. We say that the* univariate degree of f *is d if $a_d \neq 0$, and for all $j > d$ it holds that $a_j = 0$.*

Definition 8 (Multivariate degree)**.** *Let $f = \sum_{i=0}^{2^n-1} a_i x^i$ be a vectorial Boolean function. We say that the* multivariate degree of f *is d if there exists i such that $wt(i) = d$ and $a_i \neq 0$, and for all j such that $wt(j) > d$ it holds that $a_j = 0$.*

The multivariate degree can be deduced using the *freshman's dream* property mentioned earlier – it can be seen that squaring, or iterated squaring (taking 2^k powers for some $k \in \mathbb{N}$) is a linear operation. Which motivates the following definitions:

Definition 9 (Linearized polynomial)**.** *We say that a polynomial $f \in \mathbb{F}_{2^n}[x]$ is* linearized *if it is of the form:*

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

such that all $a_i \in \mathbb{F}_{2^n}$.

Definition 10 (Quadratic function)**.** *We say that a (vectorial) Boolean function f from $\mathbb{F}_{2^n}$ is* quadratic *if it is of the form:*

$$f(x) = \sum_{i,j=0, i \neq j}^{n-1} a_{i,j} x^{2^i+2^j} + L(x),$$

where $L(x)$ is a linearized polynomial and all $a_{i,j} \in \mathbb{F}_{2^n}$.

Therefore from now on when we will talk about degree, or call a function linear, or quadratic, we will mean linear or quadratic with respect to its multivariate degree.

A useful property of linear mappings, which we will later use is the existence of an adjoint mapping.

Definition 11 (Adjoint polynomial)**.** *Let L be a linearized polynomial. We call a polynomial L^* such that:*

$$\mathrm{tr}_1^n(xf(y)) = \mathrm{tr}_1^n(yf^*(x)),$$

an adjoint *polynomial of L.*

More generally we could define an adjoint polynomial by $< L(x)|y > = < L^*(y)|x >$. We get our definition by choosing trace as the scalar product.

Lemma 1 (Existence and form of adjoint polynomial)**.** *Let L be a linearized polynomial over $\mathbb{F}_{2^n}$ of the form:*

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

Then there exists a linearized polynomial L^* over $\mathbb{F}_{2^n}$ of the form:

$$L^*(x) = \sum_{i=0}^{n-1} a_{n-i}^{2^i} x^{2^i},$$

such that for all $x, y \in F_{2^n}$ it holds that:

$$\mathrm{tr}_1^n(xL(y)) = \mathrm{tr}_1^n(yL^*(x).$$

Proof. We will check whether:

$$\mathrm{tr}_1^n(xL(y)) = \mathrm{tr}_1^n(yL^*(x)).$$

We have that:

$$\mathrm{tr}_1^n(xL(y)) = \mathrm{tr}_1^n(x\sum_{i=0}^{n-1} a_i y^{2^i}).$$

By using distributivity, and by reordering of the terms of the polynomial f we get:

$$\mathrm{tr}_1^n(x\sum_{i=0}^{n-1} a_i y^{2^i}) = \mathrm{tr}_1^n(\sum_{i=0}^{n-1} x a_{n-i} y^{2^{n-i}}).$$

Now by utilizing the freshman's dream property we rewrite as follows:

$$\mathrm{tr}_1^n(\sum_{i=0}^{n-1} x a_{n-i} y^{2^{n-i}}) = \mathrm{tr}_1^n(\sum_{i=0}^{n-1} (x^{2^i} a_{n-i}^{2^i} y)^{2^{n-i}}).$$

Finally we now use the linearity of trace to check the equality term by term. Therefore we are now asking whether:

$$\mathrm{tr}_1^n((x^{2^i} a_{n-i}^{2^i} y)^{2^{n-i}}) = \mathrm{tr}_1^n(x^{2^i} a_{n-i}^{2^i} y).$$

Which is, as we have observed before, true. □

We will also be using the following linear algebra lemmata about the dimensions of spaces defined by linear mapping. These are well know, and along with their proofs they can be found in almost any linear algebra textbook.

Lemma 2 (Dimension of the kernel of the adjoint)**.** *Let L be a linearized polynomial, and let $d = dim(Ker(L))$. Then the $d = dim(Ker(L^*))$.*

Lemma 3 (Sum of the dimension of the kernel and of the image)**.** *Let f be a linear mapping defined on $\mathbb{F}_{2^n}$. Then $dim(Im(f)) + dim(Ker(f)) = n$.*

Another lemma we will later use is about connection of the kernel of a linear mapping, and the image of its adjoint.

Lemma 4. *Let A be a linear mapping. Then $(Ker(A))^\perp = Im(A^*)$.*

Proof. By the definition of $Ker(A)$ we have:

$$\mathrm{tr}_1^n(x(A(u))) = 0,$$

for all $x \in \mathbb{F}$ and $u \in Ker(A)$. Now by the definition of adjoint mapping we have, that:

$$0 = \mathrm{tr}_1^n(xA(u)) = \mathrm{tr}_1^n(uA^*(x)).$$

This means, that:

$$Ker(A) = (Im(A^*))^\perp.$$

Which is equivalent to:

$$(Ker(A))^\perp = Im(A^*).$$

□

It can be also seen, by a simple counting argument, that there is a one–to–one correspondence between linearized polynomials from $\mathbb{F}_{2^n}$, and n–dimensional matrices with coefficients from $\mathbb{F}_2$. Indeed, there are $(2^n)^n$ linearized polynomials on $\mathbb{F}_{2^n}$, and they all correspond to different linear mappings. And we know, that every linear mapping on $\mathbb{F}_2^n$ can be expressed as an $n \times n$ matrix with coefficients from $\mathbb{F}_2$. Indeed, there is $(2^n)^n$ of these matrices, which gives us the correspondence.

An important tool to work with vectorial Boolean functions is the *Walsh transform* which has some nice properties.

Definition 12 (Walsh transform)**.** *Let f be a (vectorial) Boolean function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. We shall call the function that satisfies:*

$$\hat{f}(u,v) = \sum_{x \in \mathbb{F}} (-1)^{<v|f(x)>+<u|x>},$$

where $u \in \mathbb{F}_2^n, v \in \mathbb{F}_2^{m^*}$ the Walsh transform *of f. .*

Now the trace function is the scalar product of our choice, so from now on, we are going to use it. Therefore we can rewrite the Walsh transform in the following way:

$$\hat{f}(u,v) = \sum_{x \in \mathbb{F}} (-1)^{\mathrm{tr}_1^m(vf(x))+\mathrm{tr}_1^n(ux)},$$

where $u \in \mathbb{F}_{2^n}, v \in \mathbb{F}_{2^m}^*$.

Now, since the functions $\mathrm{tr}_1^n(ux)$ are linear functions, we can see, that the Walsh transform measures how similar are the components of $f(x)$ to linear functions. Indeed $(-1)^{\mathrm{tr}_1^m(vf(\alpha))+\mathrm{tr}_1^n(u\alpha)} = 1$ if and only if the component function $\mathrm{tr}_1^m(vf(x))$ and the linear function $\mathrm{tr}_1^n(ux)$ are equal when evaluated at α. Therefore the higher the value $\hat{f}(u,v)$ is, the more similar these two Boolean functions are. For Boolean functions f and g we can define the distance between f and g as a number of inputs such that $f(x) \neq g(x)$. It is then only natural to be interested in functions that are as far as possible from both linear and affine functions. Therefore it does make sense to define nonlinearity as $\max(|\hat{f}(u,v)|)$.

If we are considering the Walsh transform of Boolean functions defined over $\mathbb{F}_2^n$, then the following holds:

Lemma 5 (Fundamental equalities of Walsh transform)**.**

$$\sum_{u \in \mathbb{F}_2^n} \hat{f}(u) = 2^n (-1)^{f(0)},$$

$$\sum_{u \in \mathbb{F}_2^n} (\hat{f}(u))^2 = 2^{2n} (\text{Parseval's relation}).$$

Proof. For the first equality we have:

$$\sum_{u \in \mathbb{F}_2^n} \hat{f}(u) = \sum_{u \in \mathbb{F}_2^n} \sum_{x \in \mathbb{F}_2^n} (-1)^{\mathrm{tr}_1^n(f(x)+ux)} =$$

$$= \sum_{x \in \mathbb{F}_2^n} (-1)^{\mathrm{tr}_1^n(f(x))} \sum_{u \in \mathbb{F}_2^n} (-1)^{\mathrm{tr}_1^n(ux)} = (-1)^{f(0)} 2^n,$$

where the last equality comes from the fact that for $x \neq 0$ $\sum_{u \in \mathbb{F}_2^n} (-1)^{\mathrm{tr}_1^n(ux)} = 0$, because nonzero linear functions have exactly half of values equal to one (from linearity).

The second one is then proved as follows:

$$\sum_{u \in \mathbb{F}_2^n} (\hat{f}(u))^2 = \sum_{u \in \mathbb{F}_2^n} \sum_{x \in \mathbb{F}_2^n} (-1)^{\mathrm{tr}_1^n(f(x)+ux)} \sum_{y \in \mathbb{F}_2^n} (-1)^{\mathrm{tr}_1^n(f(x+y)+ux+uy)} =$$

$$= \sum_{u \in \mathbb{F}_2^n} \sum_{x \in \mathbb{F}_2^n} (-1)^{\mathrm{tr}_1^n(f(x))} \sum_{y \in \mathbb{F}_2^n} (-1)^{\mathrm{tr}_1^n(f(x+y)+uy)} =$$

$$= \sum_{x \in \mathbb{F}_2^n} (-1)^{\mathrm{tr}_1^n(f(x))} \sum_{y \in \mathbb{F}_2^n} (-1)^{\mathrm{tr}_1^n(f(x+y))} \sum_{u \in \mathbb{F}_2^n} (-1)^{\mathrm{tr}_1^n(uy)}.$$

Now the innermost sum is nonzero if and only if $y = 0$. Then we can rewrite as:

$$\sum_{u \in \mathbb{F}_2^n} (\hat{f}(u))^2 = \sum_{x \in \mathbb{F}_2^n} (-1)^{\mathrm{tr}_1^n(f(x))} (-1)^{\mathrm{tr}_1^n(f(x))} \sum_{u \in \mathbb{F}_2^n} (-1)^{\mathrm{tr}_1^n(uy)},$$

and after cancellations we get:

$$\sum_{x \in \mathbb{F}_2^n} (-1)^0 2^n = 2^n 2^n = 2^{2n},$$

which completes the proof.

□

Definition 13 (Extended Walsh spectrum)**.** *By the* extended Walsh spectrum *of f we shall mean the multiset of values of $\hat{f}(u, v)$. We will denote is as W_f.*

$$W_f = \{* \hat{f}(u, v) : u \in \mathbb{F}_{2^n}, v \in \mathbb{F}_{2^m}^* *\}.$$

As we noted, the values of the extended Walsh spectrum measure exactly how similar are the component functions of f to linear functions. We will be interested in functions that are as nonlinear as possible. That is, we want for them to be as far as possible from both linear *and* affine functions. So we will be interested in functions Walsh spectra values will be in absolute value as low as possible.

By utilizing Lemma 5 we know, that there is a limit on how nonlinear a function can be.

Lemma 6. *Let f be a Boolean function. Then $max(|W_f|) \geq 2^{n/2}$.*

Proof. This follows from Lemma 5. As we have:

$$\sum_{u \in \mathbb{F}_2^n} (\hat{f}(u))^2 = 2^{2n},$$

it follows, that $max((\hat{f}(u))^2) \geq 2^{2n}/2^n = 2^n$. Therefore

$$max(|W_f|) = \sqrt{max((\hat{f}(u))^2)} \geq 2^{n/2}.$$

□

For even n it is possible to achieve equality in Lemma 6. We will call the functions, that achieve this equality *bent*.

Definition 14 (Bent function)**.** *Let f be a Boolean function on $\mathbb{F}_2^n$, where n is even. We say that f is* bent *if the only values in its Walsh spectrum are $\pm 2^{n/2}$.*

One bent function is *the quadratic bent function* that can be given in multivariate notation as:

$$x_1x_2 + x_3x_4 + \cdots + x_{n-1}x_n.$$

Even vectorial quadratic functions are of great interest, since their low algebraic degree makes their analysis easier. Note that we will study functions from Table 1.2 and all of them are quadratic. Therefore we want to find functions that have similar properties to quadratic functions, which would make their study easier. This motivates the following definition.

Definition 15 (Plateaued function)**.** *Let f be a Boolean function on $\mathbb{F}_2^n$. We say that f is* plateaued *if there exist $k \in \mathbb{N}$ such there are no other values in the Walsh spectrum of f than 0 and $\pm 2^k$.*

Note that by this definition bent functions are plateaued as well.

To see that quadratic functions are plateaued, and that we have found a class of functions that generalize quadratics, we will use the following:

Definition 16 (Derivative)**.** *Let f be a (vectorial) Boolean function. We call a function* the derivative of f in the direction of *a a function for which it holds:*

$$D_af(x) = f(x) + f(x + a).$$

Lemma 7 (3, Proposition 28)**.** *Let f be a Boolean function on $\mathbb{F}$. Then f is plateaued if and only if there exists λ such that for every $x \in \mathbb{F}$:*

$$\sum_{a,b\in\mathbb{F}} (-1)^{D_aD_bf(x)} = \lambda^2$$

Therefore we can now say the following:

Lemma 8 (Quadratic functions are plateaued)**.** *Let f be a quadratic function on $\mathbb{F}$. Then f is plateaued.*

Proof. In the case of a quadratic function we will have, that its second derivative is a constant, and therefore by the Lemma 7 they must be plateaued. □

A proof of Lemma 7 along with a more detailed proof of Lemma 8 can be found in [3].

Now as we defined plateaued functions, because of their relation to quadratic functions, we can also be interested in vectorial functions that have all their components plateaued.

Definition 17 (Component–wise plateaued function)**.** *Let f be a vectorial Boolean function. We say that f is* component–wise plateaued *if all its components are plateaued.*

Now, using the Walsh spectra, we define sets corresponding to a function, that will be utilized in later chapters.

Definition 18 (Zeroes of Walsh transform)**.** *Let f be a vectorial boolean function. We will denote $Z_{\hat{f}}$ the set:*

$$Z_{\hat{f}} = \{(u, v) \in \mathbb{F} \times \mathbb{F} : \hat{f}(u, v) = 0\}.$$

Note that now we allow $v = 0$. That is because we will later search for subspaces in $Z_{\hat{f}} \bigcup \{(0, 0)\}$.

Definition 19 (Bent set of f)**.** *Let f be a vectorial Boolean function. We call the set*

$$\mathrm{B}_f = \{v \in \mathbb{F} : \hat{f}(0, v) \neq \pm 2^{n/2}\},$$

the bent set *of f. We will write* B_f*.*

Definition 20 (Nonbent set of f)**.** *Let f be a vectorial Boolean function. We call the set*

$$\mathrm{NB}_f = \{v \in \mathbb{F} : \hat{f}(0, v) = \pm 2^{n/2}\} = \mathbb{F} \setminus B_f,$$

the nonbent set *of f. We will write* NB_f*.*

Later we will use the following:

Lemma 9. *For $\chi(x) = (-1)^{\mathrm{tr}_1^n f(x)}$ the following holds:*

$$\sum_{x \in \mathbb{F}} (\chi(\alpha x)) = \begin{cases} 0 \text{ if } \alpha \neq 0 \\ 2^n \text{ if } \alpha = 0 \end{cases}$$

Proof. Follows from the fact, that all nonzero Boolean linear functions are balanced (they evaluate to 1 exactly half of the time). □

We will also use the following theorem that gives a necessary and sufficient condition for a vectorial Boolean function to be a permutation.

Theorem 10 (Necessary and sufficient condition for permutations on finite fields(4, Theorem 7.7))**.** *Let f be a function on $\mathbb{F}$. Then f is a permutation if and only if*

$$\forall \alpha \in \mathbb{F}^* : \sum_{x \in \mathbb{F}} \chi(\alpha f(x)) = 0,$$

where $\chi(\alpha f(x)) = (-1)^{\mathrm{tr}_1^n(\alpha f(x))}$.

1.3 APN functions

1.3.1 Motivation

The APN (Almost Perfect Nonlinear) functions are interesting from a cryptographic point of view. Two of the most general and powerful attacks against block ciphers are *linear cryptanalysis* and *differential cryptanalysis*. Both of these attacks somehow exploit the fact, that the functions used in ciphers are linear, or *almost* linear. It is the one of the reasons we are interested in nonlinear functions is simply for better ciphers. For this reason we want to quantify the nonlinearity of functions with respect to these attacks.

Table 1.1: Known infinite families of APN monomial functions on $\mathbb{F}_{2^n}$

Family	Monomial	Conditions	References
Gold	X^{2^i+1}	$\gcd(i,n)=1$	[5]
Niho	$X^{2^t+2^{t/2}-1}$, t even $X^{2^t+2^{(3t+1)/2}-1}$, t odd	$n=2t+1$	[6]
Kasami	$X^{2^{2i}-2^i+1}$	$\gcd(i,n)=1$	[7]
Welch	X^{2^t+3}	$n=2t+1$	[8]
Dobbertin	$X^{2^{4t}+2^{3t}+2^{2t}+2^t-1}$	$n=5t$	[9]
Inverse	$X^{2^{2t}-1}$	$n=2t+1$	[10]

Table 1.2: Known infinite families of APN multinomial functions on $\mathbb{F}_{2^{2n}}$

#	Polynomial	Conditions	References
1	$X^{2^s+1}+A^{2^t-1}X^{2^{it}+2^{rt+s}}$	$n=3t$, $\gcd(t,3)=\gcd(s,3t)=1$ $t\geq 3$, $i\equiv st \pmod 3$, $r=3-i$, $A\in\mathbb{F}$ is primitive	[11]
2	$X^{2^s+1}+A^{2^t-1}X^{2^{it}+2^{rt+s}}$	$n=4t$, $\gcd(t,2)=\gcd(s,2t)=1$ $t\geq 3$, $i\equiv st \pmod 4$, $r=4-i$, $A\in\mathbb{F}$ is primitive	[12]
3	$AX^{2^s+1}+A^{2^m}X^{2^{m+s}+2^m}+BX^{2^m+1}+\sum_{i=1}^{m-1}c_iX^{2^{m+i}+2^i}$	$n=2m$, m odd, $c_i\in\mathbb{F}_{2^m}$, $\gcd(s,m)=1$, s odd, $A,B\in\mathbb{F}$ is primitive	[13]
4	$AX^{2^{n-t}+2^{t+s}}+A^{2^t}X^{2^s+1}+bX^{2^{t+s}+2^s}$	$n=3t$, $\gcd(t,3)=\gcd(s,3t)=1$, $3\vert t+s$, $A\in\mathbb{F}$ is primitive, $b\in\mathbb{F}_{2^t}$	[13]
5	$A^{2^t}X^{2^{n-t}+2^{t+s}}+AX^{2^s+1}+bX^{2^{n-t}+1}$	$n=3t$, $\gcd(t,3)=\gcd(s,3t)=1$, $3\vert t+s$, $A\in\mathbb{F}$ is primitive, $b\in\mathbb{F}_{2^t}$	[14]
6	$A^{2^t}X^{2^{n-t}+2^{t+s}}+AX^{2^s+1}+bX^{2^{n-t}+1}+cA^{2^t+1}X^{2^{t+s}+2^s}$	$n=3t$, $\gcd(t,3)=\gcd(s,3t)=1$, $3\vert t+s$, $A\in\mathbb{F}$ is primitive, $b,c\in\mathbb{F}_{2^t}$, $bc\neq 1$	[14]
7	$X^{2^{2k}+2^k}+BX^{q+1}+CX^{q(2^{2k}+2^k)}$	$n=2m$, m odd, C is a $(q-1)$st power but not a $(q-1)(2^i+1)$st power, $CB^q\neq B$	[15]
8	$X(X^{2^k}+X^q+CX^{2^kq})+X^{2^k}(C^qX^q+AX^{2^kq})+X^{(2^k+1)q}$	$n=2m$, $\gcd(n,k)=1$, C satisfies $X^{2^i+1}+CX^{2^i}+C^{2^{n/2}}X+1$, $A\in\mathbb{F}\backslash\mathbb{F}_{2m}$	[15]
9	$X^3+a^{-1}\mathrm{tr}_1^n(a^3X^9)$		[16] [17]

The *differential cryptanalysis* [18] tries to find output difference b such that the equation

$$D_a f(x) = f(x) + f(x+a) = b$$

holds for as many input differences a as possible, and use it for attacking. Supposing f is linear there exist only one such b and for this b the equation holds for every a. Therefore we can use the number of solution of the equation above to measure the nonlinearity of f.

The *linear cryptanalysis* [19] tries to find linear relations between bits of plaintext, ciphertext and key, that hold with a high probability. An example would be:

$$P_1 \oplus P_3 \oplus C_1 \oplus C_5 \oplus K_2 \oplus K_3 = 1$$

where P_i are bits of plaintext, C_i bits of ciphertext, K_i bits of key and the $\oplus$ operation is the addition modulo 2. Clearly if all the functions in a cipher were linear, it would be easy to find relations, that hold with probability 1, and then use Gaussian elimination to recover the key. This usually isn't the case, but it is possible to recover the key from relations which hold with lower probability than 1 at the cost of requiring more plaintext-ciphertext pairs. We can deduce how resilient is a function against linear cryptanalysis by its Walsh transform.

These attacks are therefore the reasons that in ciphers we want to use functions that are highly nonlinear, functions for which all such relations hold with as small probability as possible.

Since modern ciphers are used almost exclusively on computers, to use the computational resources effectively the ciphers need to operate on bits. Therefore they can be defined over a finite–dimensional vector space over a binary field, which we can identify with a finite field. Here it holds that $f(x) + f(x+a) = f(x+a) + f(x+a+a) = f(x+a) + f(x)$ and so the number of solutions of $D_a f(x)$ will always be even. Which motivates us to define APN functions in the following way.

Definition 21 (APN function)**.** *We call a vectorial Boolean function $f : \mathbb{F}_{2^n} \to \mathbb{F}_{2^m}$* APN *if the equation*

$$D_a f(x) = f(x) + f(x+a) = b$$

has either 0 or 2 solutions for all $a \in \mathbb{F}_{2^n}^*, b \in \mathbb{F}_{2^m}$.

The APN functions are defined in such a way, that they are optimal with respect to differential cryptanalysis. Moreover it is known, that all APN functions from Table 1.2 have the same extended Walsh spectrum [20]. As such it is called the *classical Walsh spectrum*. It looks like this [20]:

$$\{*2^n(1), 2^{n/2+1}(\frac{1}{3}(2^n-1)(2^{n-3}+2^{(n-4)/2})),$$

$$2^{n/2}(\frac{2}{3}(2^n-1)(2^{n-3}+2^{(n-4)/2})), 0((2^n-1)(2^{n-2}+1)),$$

$$-2^{n/2}(\frac{2}{3}(2^n-1)(2^{n-3}-2^{(n-4)/2})), -2^{n/2+1}(\frac{1}{3}(2^n-1)(2^{n-3}-2^{(n-4)/2}))*\}.$$

Note that in [20] $v = 0$ is allowed in the extended Walsh spectrum. We see, that in the classical Walsh spectrum the number of bent coefficients is quite high.

And we see most of the numbers in the absolute value are quite small. Therefore quadratic APN functions from the Table 1.2 are resilient even with respect to linear cryptanalysis.

The values in W_f are not just a lucky coincidence. In [21] the following is proven:

Lemma 11 (21, Corrolary 3)**.** *Let f be a vectorial Boolean function, that is component–wise plateaued. Then if f is APN, $|\mathrm{B}_f| \geq 2(2^n - 1)/3$.*

It is also shown in [21], that no component–wise plateaued APN function can be a permutation.

But for usage in symmetric cryptography, it is beneficial for these nonlinear functions to be permutations. Moreover it would be nice for the dimension of the field where such a function is defined to divide the length of the computer register used (for optimal usage of computational resources). This implies that APN permutations over fields of even dimension are of great interest. Unfortunately, while many APN permutations over fields of odd dimension are known, in the even case there is only one known example (up to equivalence) – the Dillon permutation in dimension 6. The existence of APN permutations in higher even dimensions is still an open problem.

For better understanding, we will show the following classical example:

Example. We will show that x^3 is APN, that

$$\mathrm{NB}_{x^3} = \{x^3 : x \in \mathbb{F}\},$$

which means NB_{x^3} is set of cubes in $\mathbb{F}$, and we will also show that x^3 is a permutation if and only if n is odd.

To see, that x^3 is APN we will need to find the number of solutions to:

$$f(x) + f(x + a) = x^3 + x^3 + x^2a + xa^2 + a^3 = b.$$

Therefore let us consider two solutions and find what conditions they must fulfill:

$$f(x) + f(x + a) = x^3 + x^3 + x^2a + xa^2 + a^3 =$$
$$y^3 + y^3 + y^2a + ya^2 + a^3 = f(y) + f(y + a),$$

which after cancellations is:

$$x^2a + xa^2 = y^2a + ya^2.$$

Therefore we may rewrite in the following way:

$$x^2a + y^2a = ya^2 + xa^2.$$

Now we may do a substitution $x \mapsto va$ and $y \mapsto wa$ to get:

$$v^2a^3 + w^2a^3 = wa^3 + va^3.$$

As we require $a \neq 0$ we may divide, and get:

$$v^2 + w^2 = v + w.$$

Therefore we see, that the only possibilities for v and w – 0 and 1. Therefore when the equation has a solution, it must have exactly 2. Which proves the fact, that x^3 is APN.

To find NB_{x^3} we will try to find B_{x^3}. We will try to find solutions to:

$$\sum_{x\in\mathbb{F}} \chi(ax^3) = \pm 2^{n/2},$$

where $\chi(x) = (-1)^{\mathrm{tr}_1^n(x)}$ as before. As we are only concerned with the absolute value of $\pm 2^{n/2}$, we shall square both sides of the equation:

$$\sum_{x\in\mathbb{F}} \chi(ax^3) \sum_{y\in\mathbb{F}} \chi(ay^3) = 2^n.$$

Now instead of y we will write $x + z$, and we will multiply the sums:

$$\sum_{x,z\in\mathbb{F}} \chi(az^3 + ax^2z + axz^2) = 2^n.$$

Now we shall rewrite:

$$\sum_{z\in\mathbb{F}} \chi(az^3) \sum_{x\in\mathbb{F}} \chi(a(x^2z + xz^2)) = 2^n.$$

Now we can rewrite the inner sum as follows:

$$\sum_{x\in\mathbb{F}} \chi(a(x^2z + xz^2)) = \sum_{x\in\mathbb{F}} \chi(a(x^2z))\chi(axz^2)) =$$

$$\sum_{x\in\mathbb{F}} \chi(a(x^2z))\chi(a^2x^2z^4)) = \sum_{x\in\mathbb{F}} \chi(x^2(az + a^2z^4)).$$

Therefore we get:

$$\sum_{z\in\mathbb{F}} \chi(az^3) \sum_{x\in\mathbb{F}} \chi(a(x^2z + xz^2)) = \sum_{z\in\mathbb{F}} \chi(az^3) \sum_{x\in\mathbb{F}} \chi(x^2(az + a^2z^4)) = 2^n.$$

Now using Lemma 9 we know that to compute this sum we only need to know when $x^2(az + a^2z^4) = 0$. As $x = 0$ is a solution, we do not want any other. Another solution exists exactly when $az = a^2z^4$, and therefore $a = z^{-3}$. So we see, that $\mathrm{NB}_{x^3} = \{a^3 : a \in \mathbb{F}\}$.

Now to show when is x^3 a permutation. It is clear that x^d is a permutation if and only if $\gcd(d, 2^n - 1) = 1$. The proof of this can be found in [4, Theorem 7.8.]. Therefore x^3 is a permutation if and only if n is odd.

1.3.2 Equivalence of Boolean functions

When a new interesting function is found we want it to really be new, and not similar to another already known function in some sense. For example we consider functions x^4 and $x^4 + 1$ to be similar: $x^4 + 1$ is just a bitwise complement of x^4. And therefore if x^4 has some interesting properties, be it its Walsh spectrum, or the maximal number of solutions to $f(x) + f(x + a) = b$, we do not need to bother checking them for $x^4 + 1$ since we already know the answer we can derive it from the properties of x^4. Therefore we would not consider $x^4 + 1$ to be new. Apart

from the bitwise complement (or more generally an addition of a constant), we also consider functions to be similar if we can get one from the other by changing the basis. Which motivates the following notion of equivalence.

Since we want to study permutations, we will only be interested in functions $f : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$. That is functions whose domain and codomain is the same.

Definition 22 (EA–equivalence (Extended Affine)). *We say, that functions f and g are EA–equivalent, if $f = L_1(g(L_2)) + L_3$, where L_1, L_2 and L_3 are affine permutations. We will write $g \approx_{EA} f$.*

This notion of equivalence is useful, but it is still not as general as we would want it to be. We would want for a permutation to be equivalent to its inverse – clearly for symmetric ciphers the encryption and decryption are similarly difficult and are the same with respect to linear and differential cryptanalysis. Indeed, if we have an APN permutation, then its inverse is also APN. If

$$x + y = a$$

$$f(x) + f(y) = b$$

have only 0 or 2 solutions and f is a permutation then by substituing $x \leftarrow f^{-1}(s)$, $y \leftarrow f^{-1}(t)$ we get

$$f^{-1}(s) + f^{-1}(t) = a$$

$$s + t = b$$

also have only 0 or 2 solutions. Also we see, that the $W_f = W_{f^{-1}}$ (if allowing $v = 0$ as in [20]) is the same for a function and its inverse. Indeed if we take $x = f^{-1}(y)$ we get:

$$\hat{f}(u, v) \sum_{x \in \mathbb{F}} (-1)^{\mathrm{tr}_1^n(vf(x)+ux)} = \sum_{x \in \mathbb{F}} (-1)^{\mathrm{tr}_1^n(vy+uf^{-1}(y))} = \hat{f^{-1}}(v, u)$$

Nevertheless it is still powerful enough to allow us to deal only with such functions for which $f(0) = 0$, which we will now do. To find a more powerful notion of equivalence it is useful to understand the relationship between Boolean functions and linear codes.

1.3.3 APN functions and coding theory

Definition 23 (Error correcting code). *We say that an M-element subset C of $\mathbb{F}_q^n$, such that the minimum Hamming distance between its elements is d, is a q–ary (n, M, d)* error correcting code. *We say, that C has* length n, cardinality M *and* distance d. *The elements of C are called* codewords.

The *error correction* comes from the fact, that if we send a codeword c over a channel and the receivers receives a message c', then if at most $d/2$ errors happen, they are able to recover the sent codeword – it is the one that minimizes the Hamming distance between c and c'.

Definition 24 (Linear code). *A* linear code *is such an q–ary error correcting code, that its codewords are closed under addition and multiplication by a constant from $\mathbb{F}_q$. It is therefore a subspace of $\mathbb{F}_q^n$. We say that a* linear code C *is q–ary $[n, k, d]$ code, if it is over $\mathbb{F}_q^n$, its dimension as a subspace is k, and minimum distance between its codewords is d.*

Because a linear code is a subspace of $\mathbb{F}_q^n$ we can use matrices to represent it.

Definition 25 (Generating and parity check matrices)**.** *A* generating matrix *of a q–ary $[n, k, d]$ linear code C is such a $k \times n$ matrix G_C, that $Im(G_C^T) = C$. A* parity check *matrix of a linear code C is such a $n - k \times n$ matrix H_C, that $Ker(H_C) = C$.*

Definition 26 (Dual code)**.** *We say that a code is* the dual code *of C if its generator matrix is H_C and its parity check matrix is G_C. We will denote such a code $C^\perp$*

Remark 3. We have a bijection between the elements of the finite field $\mathbb{F}_{q^n}$, and the elements of the vector space $\mathbb{F}_q^n$. For the sake of brevity, when we consider linear codes (and their matrices) over the field $\mathbb{F}_q$, we will often describe them with the images of the elements of $\mathbb{F}_{q^n}$. Therefore if we write a generator matrix of a linear code over $\mathbb{F}_q$ in this form:

$$G_C = \begin{pmatrix} x_1 & \dots & x_{q^n} \end{pmatrix},$$

where x_i are pairwise different elements of $\mathbb{F}_q^n$, we mean that G_C is an $n \times q^n$ matrix whose columns are the elements of $\mathbb{F}_{q^n}$ taken as vectors from $\mathbb{F}_q^n$.

Definition 27 (Linear code corresponding to a function)**.** *We say, that a linear code C is corresponding to a function f if the code's generating matrix can be written in o form:*

$$G_C = \begin{pmatrix} x_1 & \dots & x_{q^n} \\ f(x_1) & \dots & f(x_{q^n}) \end{pmatrix},$$

where x_i are pairwise different elements of $\mathbb{F}_q^n$.

As we are interested in vectorial Boolean functions, from now on we will be only considering codes over $\mathbb{F}_2$

Definition 28 (Permutation equivalence)**.** *We say, that codes C and D are* permutation equivalent *if there exists a bijection between the sets of their respective codewords, that is a just permutation of the coordinates of the codewords.*

Definition 29 (Hamming code)**.** *We call binary $[2^n - 1, n, 3]$ code C a* Hamming code*, if its generator matrix can be written in form:*

$$G_C = \begin{pmatrix} x_1 & \dots & x_{2^n-1} \end{pmatrix},$$

where x_i are pairwise different elements of $\mathbb{F}_2^n$ apart from zero vector.

Definition 30 (Simplex code)**.** *We say a code is a* simplex code *if it is a dual code of a Hamming code.*

Definition 31 (Double simplex code)**.** *We say a code C is a* double simplex code *if it is a direct sum of two simplex codes. Therefore its parity check matrix can be written in form:*

$$H_C = \begin{pmatrix} x_1 & \dots & x_{2^n} \\ y_1 & \dots & y_{2^n} \end{pmatrix},$$

where x_i, y_i are pairwise different elements of $\mathbb{F}_2^n$.

One can now give another definition of an APN function:

Definition 32 (APN function (code-based definition))**.** *We say that a function f such that $f(0) = 0$ is APN if the code $C_f^{\perp}$ is double error correcting (its distance is at least 5).*

Definition 33 (Extended code corresponding to a function)**.** *We say that a code C is an* extended code corresponding to a function *f if its generating matrix is of the form:*

$$\tilde{G_C} = \begin{pmatrix} 1 & \dots & 1 \\ x_1 & \dots & x_{q^n} \\ f(x_1) & \dots & f(x_{q^n}) \end{pmatrix},$$

It can be shown, that these definitions of APN functions are equivalent. Indeed if a function is APN, then there exist no a, b, c, d such that

$$a + b = c + d$$

and

$$f(a) + f(b) = f(c) + f(d).$$

This can be seen both from the fact, that equation

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

cannot have four solutions for f APN, and the fact, that the sum of four columns of the parity check matrix of a double simplex code cannot vanish. Thus we see, that the extended codes have a direct connection to resilience against linear cryptanalysis.

Moreover it can be shown that the extended codes have also a connection to linear cryptanalysis. Their weight distribution has a connection with W_f as noted in [22].

If two functions f and g have the same corresponding code, than f is APN if and only if g is APN. Moreover it suffices if C_g is permutation equivalent to C_f Which motivates the following notion of equivalence. Note that the usage of extended codes will give us a larger equivalence class.

Definition 34 (CCZ–equivalence (Carlet, Charpin, Zinoviev))**.** *We say, that functions f and g are CCZ–equivalent, if $\tilde{C_f}$ and $\tilde{C_g}$ are permutation equivalent. We will write $g \approx_{CCZ} f$.*

Remark 4. We can see that this means, that for f and g are CCZ–equivalent if and only if there exists an invertible linear function L such that:

$$L \begin{pmatrix} x_i \\ f(x_i) \end{pmatrix} = \begin{pmatrix} x_j \\ g(x_j) \end{pmatrix},$$

for x_i, x_j iterating over the whole field.

It is easy to see, that for all permutations f it holds that $f \approx_{CCZ} f^{-1}$. To switch between $\tilde{C_f}$ and $\tilde{C}_{f^{-1}}$ we only need to rearange the columns of the generating matrices.

Furthermore it holds, that if an APN function is a permutation, then its corresponding code is a double simplex code. Actually for the code to be a double simplex code, it is only necessary for the APN function to only be CCZ–equivalent to a permutation.

But the extended codes are not the nicest objects to work with. Fortunately in [23] the following is proven:

Theorem 12 (23, Theorem 6.2)**.** *Let f and g be APN functions, that vanish at 0. Let $g \approx_{CCZ} f$, and let their CCZ–equivalence class contain a quadratic function. Then their corresponding codes are equivalent.*

The power of Theorem 12 is that as we are only considering quadratic APN functions that fix 0 we can work with the corresponding codes, and not the extended ones.

Motivated by the code definition of CCZ–equivalence we can also try to see the equivalence more as a characteristic of functions instead of codes. To do this we shall define a graph of a function. Moreover this definition encompasses even the functions that do not fix zero.

Definition 35 (Graph of a function)**.** *Let f be a (vectorial) Boolean function defined on $\mathbb{F}_{2^n}$. We call a set $\{(x, f(x)) : x \in \mathbb{F}_{2^n}\}$ a* graph of f.

It is straightforward to see a connection between the code of a function and the graph of a function. Therefore it is easy to reformulate the CCZ–equivalence with graphs.

Definition 36 (CCZ–equivalence (graph based definition))**.** *Let Gr_f be the graph of f, and Gr_g the graph of g. We say, that f and g are* CCZ–equivalent *if there exist an affine automorphism L from Gr_f to Gr_g ($L(Gr_f) = Gr_g$). That is*

$$\exists \begin{pmatrix} A & B \\ C & D \end{pmatrix}, \begin{pmatrix} u \\ v \end{pmatrix} : \begin{pmatrix} x' \\ g(x') \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} x \\ f(x) \end{pmatrix} + \begin{pmatrix} u \\ v \end{pmatrix},$$

where x and x' iterate over the whole field.

Remark 5. If we have g a permutation, then $A(x)+B(f(x))$ and $C(x)+D(f(x))$ must be permutations. For $A(x)+B(f(x))$ this is required from the Definition 36 (CCZ–equivalence), and for $C(x)+D(f(x))$ it follows from g being a permutation.

Now we see, that CCZ–equivalence is more general, than EA–equivalence. EA–equivalence just introduces additional constraints on function L.

2. Dillon's approach

In [2] a deterministic algorithm for deciding whether an APN function is CCZ–equivalent to a permutation is provided. With this algorithm, Dillon was able to check functions defined over $\mathbb{F}_{2^n}$ for $n \leq 10$ even.

As the only known APN multinomials are quadratic and vanish at 0, Dillon proves the following theorem in [23]:

Theorem 13 (Theorem 6.2 in [23]). *Let f and g be APN maps which vanish at 0. If f and g are CCZ–equivalent and their CCZ–equivalence class contains a quadratic function, then the codes C_f and C_g are equivalent.*

Therefore if a quadratic APN function f is not a permutation, but is CCZ–equivalent to one, its corresponding code C_f is still a double simplex code – its parity check matrix can be written in the form $\begin{pmatrix} f_1(x) \\ f_2(x) \end{pmatrix}$ for all $x \in \mathbb{F}^*$, where f_1 and f_2 are permutations such that $f_1(0) = f_2(0) = 0$. Now if we consider a function $g = f_2 \circ f_1^{-1}$, then it is certainly a permutation – it is a composition of permutations. And it is definitely APN – it is CCZ–equivalent to f. So if for any APN f we could find such f_1 and f_2 permutations we would actually have found an APN permutation. For this we need to find an invertible linear function $L = \begin{pmatrix} L_1 \\ L_2 \end{pmatrix}$ defined on $\mathbb{F}_{2^n} \times \mathbb{F}_{2^n}$, such that

$$L \begin{pmatrix} x \\ f(x) \end{pmatrix} = \begin{pmatrix} f_1(x) \\ f_2(x) \end{pmatrix}.$$

That is to find linear functions L_1, L_2 : such that

$$L_i \begin{pmatrix} x \\ f(x) \end{pmatrix} = f_i(x),$$

for $i = 1, 2$.

Dillon observed, that it is sufficient to use the Algorithm 1:

Algorithm 1 Algorithm for deciding whether an APN function is CCZ–equivalent to a permutation

INPUT: H_f, a parity check matrix of C_f
OUTPUT: TRUE if f is CCZ–equivalent to a permutation, FALSE otherwise
1: Generate a list of all codes with parity check matrix of the form LH_f, for L full–rank in reduced row echelon form.
2: **if** in the resulting list there are two codes with trivial intersection **then return** TRUE
3: **else return** FALSE
4: **end if**

As Dillon notes, it really is only necessary to consider full–rank matrices (by the definition of CCZ–equivalence), and we can consider only those in reduced row echelon forms, because for S invertible $S(L_iH_f) = (SL_i)H_f)$, and the codes

with generating matrices L_iH_f and SL_iH_f are the same. The matrix S can then be chosen to make SL_i in row reduced echelon form – indeed we could get S it by Gaussian elimination of L_i. Moreover when generating the matrices one can also subject them to additional constraint – f_1 and f_2 need to be permutations, therefore L_i cannot vanish on any sum of any two columns from H_f. Therefore Dillon used these constraints while generating the matrices L_i one column at a time.

It is possible to start with an identity matrix of appropriate dimension, and then add columns. The columns need to be added in such a way, that while adding j-th column (denoted $\boldsymbol{l}_j$), solutions to:

$$\begin{pmatrix} \boldsymbol{l}_1 & \boldsymbol{l}_2 & \dots & \boldsymbol{l}_j \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_j \end{pmatrix} = 0,$$

are avoided for all vectors s that are a sum of two vectors from H_f, or more precisely which are the first j coordinates of the respective sum.

Dillon notes, that for low dimensions it is actually possible to generate all the simplex codes and try to find two with trivial intersection – if the search is successful it gives an APN permutation. Unfortunately this procedure does not scale well with the dimension since the code parameters of simplex codes are $[2^n - 1, n, 2^{n-1}]$. Still it is powerful enough to check functions up to dimension 10. Moreover it is thanks to this algorithm that the only known APN permutation over a binary field of even dimension was found.

In Section 6 of [2] another randomized algorithm for checking whether an APN function is CCZ–equivalent to a permutation. To understand the idea behind the algorithm we will need the following lemma.

Lemma 14 (Sufficient and necessary condition for being CCZ–equivalent to a permutation)**.** *A function f is CCZ–equivalent to a permutation, if and only if there exist two spaces $U, V \subseteq Z_{\hat{f}} \bigcup \{(0,0)\}$, such that $U \bigcap V = \{(0,0)\}$ and $dim(U) = dim(V) = n$.*

Proof. Let us recall the graph definition of CCZ–equivalence (Definition 36) it says, that $f \approx_{CCZ} g$ requires an affine automorphism given by A, B, C, D, u, v such that:

$$\begin{pmatrix} x \\ f(x) \end{pmatrix} \mapsto \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} x \\ f(x) \end{pmatrix} + \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} x' \\ g(x') \end{pmatrix}.$$

where $x' = \pi(x)$ and π is a permutation on $\mathbb{F}$. And we know, that the matrix $\begin{pmatrix} A & B \\ C & D \end{pmatrix}$ must be full–rank to give us an automorphism. Let us have g a permutation and $g \approx_{CCZ} f$. Therefore for some π, a permutation of $\mathbb{F}$, we can write:

$$A(x) + B(f(x)) + u = \pi(x),$$

$$C(x) + D(f(x)) + v = g(\pi(x)).$$

Now by Remark 5 we know that $A(x) + B(f(x))$ and $C(x) + D(f(x))$ are permutations. Using the Theorem 10 we have that f is CCZ–equivalent to a permutation if and only if both of the following hold:

$$\sum_{x\in\mathbb{F}} \chi(\alpha A(x) + \alpha B(f(x))) = 0,$$

and

$$\sum_{x\in\mathbb{F}} \chi(\alpha C(x) + \alpha D(f(x))) = 0$$

for all $\alpha \in \mathbb{F}^*$.

Now, we can rewrite these conditions using the adjoint mappings of A, B, C, and D, and our condition for f to be CCZ–equivalent to a permutation transforms into the following equivalent one:

$$\sum_{x\in\mathbb{F}} \chi(xA^*(\alpha) + f(x)B^*(\alpha)) = \hat{f}(A^*(\alpha), B^*(\alpha)) = 0, \tag{2.1}$$

and

$$\sum_{x\in\mathbb{F}} \chi(xC^*(\alpha) + f(x)D^*(\alpha)) = \hat{f}(C^*(\alpha), D^*(\alpha)) = 0, \tag{2.2}$$

for all $\alpha \in \mathbb{F}^*$

Now let $U = \{(A^*(\alpha), B^*(\alpha)\}$, $V = \{(C^*(\alpha), D^*(\alpha)\}$. These are spaces as their are images of linear mappings, and clearly $U, V \subseteq Z_{\hat{f}} \bigcup \{(0,0)\}$, and $dim(U), dim(V) \leq n$.

To see, that U has dimension n it suffices to consider what it would mean if it were not the case. Clearly there would need to exist $\alpha \in \mathbb{F}$ such that $(A^*(\alpha), B^*(\alpha)) = (0,0)$ For this α would the Equation 2.1 become:

$$\sum_{x\in\mathbb{F}} \chi(0) = 2^n \neq 0,$$

which would contradict the assumption that $A(x) + B(f(x))$ is a permutation.

The proof of existence of another subspace V is analogous using C, D and Equation 2.2 instead of A, B and 2.1 respectively.

The fact, that $U \bigcap V = \{(0,0)\}$, follows from the fact, that the matrix $\begin{pmatrix} A & B \\ C & D \end{pmatrix}$ must be of a full rank. Indeed, let us suppose, that their intersection is not trivial. That is, that there exists $(\alpha, \beta) \in U \bigcap V$. Then:

$$(A^*(a), B^*(a)) = (\alpha, \beta) = (C^*(b), D^*(b)),$$

for some $(a, b) \neq (0, 0)$, which means:

$$A^*(a) + C^*(b) = 0 = B^*(a) + D^*(b).$$

And, as trace is a scalar product, that is equivalent with:

$$\mathrm{tr}_1^n(x(A^*(a) + C^*(b)) = 0,$$

and

$$\mathrm{tr}_1^n(y(B^*(a) + D^*(b)) = 0,$$

for all $x, y \in \mathbb{F}$. Rewriting these using adjoint mappings gives:

$$\mathrm{tr}_1^n(aA(x) + bC(x)) = 0,$$

and

$$\mathrm{tr}_1^n(a(B(y) + bD(y)) = 0,$$

for all $x, y \in \mathbb{F}$, which we can rewrite using matrix notation as:

$$\begin{pmatrix} a & b \end{pmatrix} \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = 0,$$

for some $(a, b) \neq (0, 0)$ and for all $x, y \in \mathbb{F}$. This means, that a nontrivial linear combination of rows of $\begin{pmatrix} A & B \\ C & D \end{pmatrix}$ adds up to zero, and therefore it cannot be full–rank, which completes the proof.

□

The algorithm simply searches for two n–dimensional subspaces in $Z_{\hat{f}} = \{(a, b) : \hat{f}(a, b) = 0\}$ with trivial intersection. It is this algorithm that we modify in the next chapter. We will rewrite this algorithm from [2] as Algorithm 2.

It can be seen, that Algorithm 2 has less memory requirement than Algorithm 1, as Algorithm 2 only keeps two subspaces of $\mathbb{F} \times \mathbb{F}$ in memory at a time, instead of keeping a long list of codes like the previous algorithm. Moreover it is faster – generating subspaces instead of generating matrices, and then comparing all resulting codes.

Algorithm 2 Algorithm for deciding whether an APN function is CCZ–equivalent to a permutation

INPUT: $B = \{(a, b) : \hat{f}(a, b) = 0\}$, s a parameter for restarting the search, a parameter for avoiding infinite loop

OUTPUT: PROVEN if it manages to prove, that f is CCZ–equivalent to a permutation, INCONCLUSIVE otherwise

1: $B_1 = B$, $q \leftarrow 1$
2: Choose $b_1 \in B_1$ randomly, $i \leftarrow 1$, $c \leftarrow 1$
3: **while** $i \neq n$ and $c \neq s$ **do**
4: Choose b_{i+1} randomly
5: **if** $b_{i+1} \notin B_1 \backslash < b_1, \ldots, b_i >$ **then**
6: $i \leftarrow i + 1$
7: **end if**
8: $c \leftarrow c + 1$
9: **end while**
10: **if** $q = a$ **then**
11: **return** INCONCLUSIVE
12: **end if**
13: **if** $i \neq n + 1$ **then**
14: $q \leftarrow q + 1$
15: **goto** 3
16: **end if**
17: $B_2 = B \backslash < b_1, \ldots, b_n >$
18: Choose $b'_1 \in B_2$ randomly, $i \leftarrow 1$, $c \leftarrow 1$
19: **while** $i \neq n$ and $c \neq s$ **do**
20: Choose b'_{i+1} randomly
21: **if** $b'_{i+1} \notin B_2 \backslash < b'_1, \ldots, b'_i >$ **then**
22: $i \leftarrow i + 1$
23: **end if**
24: $c \leftarrow c + 1$
25: **end while**
26: **if** $q = a$ **then**
27: **return** INCONCLUSIVE
28: **end if**
29: **if** $i \neq n + 1$ **then**
30: $q = q + 1$
31: **goto** 18
32: **end if**
33: **return** PROVEN

3. The results

In this chapter we show another deterministic algorithm (Algorithm 3) that is faster then Algorithms 1 and 2 in proving that a component–wise plateaued function is *not* CCZ–equivalent to a permutation. We present the results we obtained for functions over $\mathbb{F}_{2^n}$ for $n \in \{6, 8, 10, 12\}$.

Moreover we also present a theoretical proof of the fact, that functions from Family 9 in Table 1.2 are *not* CCZ–equivalent to a permutation for infinite number of dimensions, that is for doubly even dimensions, i.e. $4|n$.

For Families 1, 2, 4, 5 and 9 from the Table 1.2 we managed to computationally prove that no member of these families can be CCZ equivalent to a permutation in even extensions of $\mathbb{F}_2$ up to $\mathbb{F}_{2^{12}}$ – an improvement over previous results that were up to dimension 10.

3.1 Computational search

3.1.1 Modified condition

We will give a deterministic version of the randomized Algorithm 2 first presented in the Section 6 of [2]. That algorithm searches for two subspaces of dimension n with trivial intersection in $Z_{\hat{f}} \bigcup \{(0,0)\} \subseteq \mathbb{F} \times \mathbb{F}$. We use a deterministic version and show that it is sufficient to search for subspaces of dimension $n/2$ in $\mathrm{NB}_f \subseteq \mathbb{F}$. Note that by examining the classical Walsh spectrum and by using Lemma **?** , it can be deduced, that for functions we are interested in, it holds that $|\mathrm{NB}_f| < \sqrt{|Z_{\hat{f}}|}$. For this Algorithm 3 we will use the following result from [24]:

Theorem 15. *If a component–wise plateaued function f is CCZ–equivalent to a permutation, then there exist two subspaces $U, V \subseteq \mathrm{NB}_f$, such that $U^\perp \bigcap V^\perp = \{0\}$, and $dim(U) + dim(V) \geq n$.*

The fact, that f is component–wise plateaued, means that all non–bent components of f lack values $\pm 2^{n/2}$ in their Walsh spectra. Therefore if we consider the linear mappings A, B, C, D from the definition of CCZ–equivalence, then if $\hat{f}(A^*(\alpha), B^*(\alpha)) = 0$, then $\hat{f}(0, B^*(\alpha)) \neq \pm 2^{n/2}$. So definitely $Im(B^*) \subseteq \mathrm{NB}_f$ and analogously $Im(D^*) \subseteq \mathrm{NB}_f$.

Proof. Let f be a component–wise plateaued function that is CCZ–equivalent to a permutation g. Recall the definition of CCZ–equivalence (Definition 36). There exists a full–rank matrix $\begin{pmatrix} A & B \\ C & D \end{pmatrix}$, such that

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} x \\ f(x) \end{pmatrix} + \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \pi(x) \\ g(\pi(x)), \end{pmatrix}$$

for some permutation π. As we have noticed before $Im(B^*), Im(D^*) \subseteq \mathrm{NB}_f$. Let $U = Im(B^*)$, and $V = Im(D^*)$.

From Lemma 4 we have $U^\perp = Ker(B)$, and $V^\perp = Ker(D)$. And since the matrix $\begin{pmatrix} A & B \\ C & D \end{pmatrix}$ is full–rank, there cannot be $a \neq 0$ such that:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} 0 \\ a \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

which means, that

$$U^\perp \bigcap V^\perp = Im(B^*)^\perp \bigcap Im(D^*)^\perp = \{0\}.$$

It also means, that

$$dim(Im(B^*)^\perp) + dim(Im(D^*)^\perp) \leq n,$$

so

$$dim(Ker(B)) + dim(Ker(D)) \leq n,$$

which is

$$dim(U^\perp) + dim(V^\perp) \leq n,$$

which is equivalent to

$$dim(U) + dim(V) \geq n,$$

which completes the proof. □

From this lemma we immediately get the following corollary:

Corollary 1. *If a component–wise plateaued function is CCZ–equivalent to a permutation, then in* NB_f *there exists at least one subspace of dimension* $n/2$.

Proof. Since we have $dim(Im(B^*)) + dim(Im(D^*)) \geq n$, then either $dim(Im(B^*)) \geq n/2$ or $dim(Im(D^*)) \geq n/2$, which is exactly what we want. □

Thanks to this theorem and its corollary we only need to calculate the NB_f and try to find a subspace of dimension $n/2$, instead of searching for the subspaces of dimension n in $\mathbb{F} \times \mathbb{F}$. Based on our calculations which are summed up in Table 3.1, it is apparent, that for most functions we cannot find even a single $n/2$–dimensional subspace of NB_f. Indeed, by Lemma 11, the NB_f, is rather small, and so it is unlikely to contain high dimensional subspaces. On the other way it makes it rather easy to prove CCZ–inequivalence to a permutation.

Moreover we can show, that the maximal subspace dimension in NB_f is an EA invariant.

Lemma 16 (For plateaued functions NB_f is an EA invariant)**.** *Let* $g \approx_{EA} f$*, then the maximal subspace dimension in* NB_f *equals the maximal subspace dimension in* NB_g.

Proof. Let $g = L_1(f(L_2(x))) + L_3(x)$, and U be a maximal subspace in NB_g. Then for $\alpha \in U$ we have:

$$\sum_{x \in \mathbb{F}} (-1)^{\mathrm{tr}_1^n(\alpha g(x))} \neq \pm 2^{n/2}.$$

By rewriting g we get:

$$\sum_{x\in\mathbb{F}}(-1)^{\mathrm{tr}_1^n(\alpha(L_1(f(L_2(x)))+L_3(x)))} \neq \pm 2^{n/2}.$$

By substitution $y = L_2^{-1}(x)$ it is:

$$\sum_{x\in\mathbb{F}}(-1)^{\mathrm{tr}_1^n(\alpha(L_1(f(y))+L_3(L_2^{-1}(y))))} \neq \pm 2^{n/2},$$

and rewriting using adjoint mapping gives:

$$\sum_{x\in\mathbb{F}}(-1)^{\mathrm{tr}_1^n(f(y)(L_1^*(\alpha)+y(((L_3L_2)^{-1})^*(\alpha))))} \neq \pm 2^{n/2}.$$

Now since g is plateaued we have:

$$\sum_{x\in\mathbb{F}}(-1)^{\mathrm{tr}_1^n(f(y)(L_1^*(\alpha)} \neq \pm 2^{n/2}.$$

Therefore $U \subseteq \mathrm{NB}_g$ is mapped to $L_1^*(U) \subseteq \mathrm{NB}_f$, which completes the proof. □

Now we will use the following result by Yoshiara, to prove that for quadratic APN functions the EA invariant from Lemma 16 is even a CCZ invariant.

Theorem 17 (EA/CCZ–equivalence for quadratic APN functions [25, Theorem 1)**.** *] Let f and g be quadratic APN functions on $\mathbb{F}_{2^n}$, $n \geq 2 \in \mathbb{N}$. Then $g \approx_{EA} f$ if and only if $g \approx_{CCZ} f$.*

Lemma 18 (CCZ invariant for quadratic APN functions)**.** *Let f, g be quadratic APN functions such that $g \approx_{CCZ} f$. Then the maximal subspace dimension in* NB_f *equals the maximal subspace dimension in* NB_g*.*

Proof. This lemma is a direct consequence of Lemma 16 and Theorem 17. □

Now we can search in NB_f which is much smaller than $Z_{\hat{f}}$. As mentioned in the Subsection 3.1.3 of [26] we know, that for quadratic APN functions the size of NB_f is always less than or equal to $\frac{1}{3}(2^n - 1) + 1$.

Below we present our algorithm that we actually use for proving CCZ–inequivalence to a permutation. The full Sage code can be found in the attachments. The algorithm is basically a depth-first search, with some modifications to not visit the searched nodes multiple times. That is we use the fact, that if $S =< U \bigcup \{v\} >$, then we can get S as a linear span of $U \bigcup \{s\}$ for every $s \in S \setminus U$. Therefore when we generate S from U using $v \in S \setminus U$, we skip generating S from U using other elements of $S \setminus U$.

We considered even a breadth-first like approach for guaranteeing that every possible space will be considered just once, but this approach quickly proved itself unusable in higher dimension because of memory reasons.

The reason for including the F in the Algorithm 3 set is the one explained in more detail above – it is a set of *forbidden* elements that we do not need to consider, since they will not give us a new space.

Algorithm 3 Algorithm for finding the dimension of the biggest possible subspaces in a set

```
    INPUT: S set where we search for subspaces
    OUTPUT: n maximal possible dimension of a subspace of S
 1: function SEARCH(S # set where we search for subspaces,
                    C # current subspace of S,
                    F # set of already searched elements,
                    m # dimension of C)
 2:     if m > n then n ← m
 3:     end if
 4:     for all element in S do
 5:         if element in F then continue
 6:         end if
 7:         N ← C + element
 8:         if N ⊆ S and N ⋂ F = ∅ then SEARCH(S, C ⋃ N, F ⋃ N, m + 1)
 9:         end if
10:         F ← F ⋃ N
11:     end for
12: end function
13: n ← 0
14: SEARCH(S, {0}, {0}, 0)
15: return n
```

3.1.2 Reducing the number of polynomials

Looking at the list of APN functions 1.2 one realizes, that in some families in higher dimensions there are simply too many functions to check for their CCZ–equivalence to a permutation even with our algorithm. While there are computational proofs (for example in [27]), that for small n all APN functions in the Table 1.2 are CCZ–equivalent to each other *within* their respective families and therefore for these small fields, it would only be necessary to run our algorithm for one polynomial from each family only, for larger fields we have no such result, and checking CCZ–equivalence between all members of a family is a rather time–consuming task. The only approach we are aware of is just generating a whole CCZ–equivalence class and then checking whether there is a function within the CCZ–equivalence class that is also in the APN function family. So we would generate a large number a functions that we would not even use. SO for proving CCZ–(in)equivalence to a permutation for each polynomial within a class, we must use our algorithm on each polynomial within a class.

As mentioned earlier, for small n we can prove computationally that all polynomials within a family from the Table 1.2 are from the same CCZ–equivalence class. Therefore it is natural to expect, that for higher families there will also be relatively small number of CCZ–equivalence classes, if not only one. As we want to provide an actual computational proof of CCZ-(in)equivalence we cannot settle with the randomized approach, but it seems, that if there is only a small number of CCZ–equivalence classes, then it should be easy to prove theoretically, for at least some functions within a class, that they are indeed in the same CCZ–equivalence class.

To do this we choose an easy to analyze subset of linear mappings. We are definitely not pretending to select only one polynomial from each CCZ–equivalence class within a family. This would sped up our search dramatically, but as it is noted in [20] we do not know much about these intrafamily CCZ–equivalence classes as these results are hard to prove – hence the purely computational nature of the results in [27] in this area. But we are simply trying to select as few functions from one CCZ–equivalence class as possible while still having our code run fast. This is important, since if we had an amazing condition on CCZ–equivalence that would make our code run 1000000 times slower, then it would render our algorithm unusable. Luckily we are successful in finding ways to discard some functions even before the generation of the lists of functions to check for CCZ–equivalence to a permutation (as opposed to generating all functions from a family, and then deleting them from the list).

Below are notions of equivalence, that we used for the reduction of the number of polynomials to check for being CCZ–equivalent to a permutation. It is easy to see (the mapping used are linear) that they all are less general then CCZ–equivalence (actually even more than EA–equivalence).

Definition 37 (Frobenius equivalence)**.** *We say, that f and g are* Frobenius equivalent *if $f(x) = (g(x^{2^i}))^{2^j}$ for some $i, j \in \mathbb{N}$.*

Definition 38 (Left translation equivalence)**.** *We say, that f and g are* left translation equivalent *if $f(x) = ag(bx)$ for some $a, b \in \mathbb{F}$.*

Remark 6 (Subfield trick). If f is of the form: $f(x) = A^{2^t}x^i + Ax^j + bx^c$, where A is primitive and $b \neq 0$ is from a subfield of 2^t elements, then $b^{-1}f(x)$ is of the same form (since $\frac{A^{2^t}}{b} = \frac{A^{2^t}}{b^{2^t}} = (\frac{A}{b})^{2^t}$)

The usage of Frobenius equivalence is straightforward – if we can use it, we simply take an element a from a list of possible choices for a parameter p, and erase from the list all elements of the form $a^{2^i}, 1 < i < n$. Using the Family 1 from 1.2 we have a function $f(x)$ of the form, $x^{2^s+1} + A^{2^t-1}x^{2^{it}+2^{rt+s}}$. Now for every $1 \leq k \leq n$ we can take a function $g(x) = (f(x^{2^{-}k}))^{2^k}$. Thanks to the freshman's dream property we get:

$$g(x) = x^{(2^s+1)2^{k-k}} + A^{(2^t-1)2^k}x^{(2^{it}+2^{rt+s})2^{k-k}} = x^{2^s+1} + A^{2^k 2^t-1}x^{2^{it}+2^{rt+s}}.$$

And so we need only one element from the list $A^{2^0}, A^{2^1}, \ldots A^{2^{n-1}}$, which helps since if A is primitive, then all elements from the list $A^{2^0}, A^{2^1}, \ldots A^{2^{n-1}}$ are primitive as well.

The usage of left translation equivalence is a little bit more complex. We shall provide an example of how it is done. We will use the Family 1 from the table 1.2.

We have a function $f(x)$ of the form: $x^{2^s+1} + A^{2^t-1}x^{2^{it}+2^{rt+s}}$. We transform it into a function $g(x) = f(cx)$. Therefore we have:

$$g(x) = c^{2^s+1}x^{2^s+1} + A^{2^t-1}c^{2^{it}+2^{rt+s}}x^{2^{it}+2^{rt+s}}.$$

Since we know that the first coefficient corresponding to x^{2^s+1} should be 1, we multiply the whole function by its inverse. We get a function $h(x) = c^{(-1)(2^s+1)}g(x) = c^{(-1)(2^s+1)}f(cx)$. Therefore we have:

$$h(x) = x^{2^s+1} + A^{2^t-1}c^{2^{it}+2^{rt+s}-2^s-1}X^{2^{it}+2^{rt+s}}.$$

It follows, that for a given exponent $2^s + 1$ we can look if in the list of possible choices for the parameter A there are some choices A' A'' such that $A' = c^{2^{it}+2^{rt+s}-2^s-1}A''$ for some $c \in \mathbb{F}$. If that is a case we can remove one of them. Now this method cannot be used on a family as a whole, but the reduction must be done for each possible exponent from the family separately. In the list of reductions we will list just the found exponents. In this case it will be $2^{it}+2^{rt+s}-2^s-1$. Because of the inversion this reduction is most useful (most probable to be used) for binomials.

Now to show the usage of the subfield trick, we will use the Family 4. Therefore we have $f(x) = AX^{2^{n-t}+2^{t+s}} + A^{2^t}X^{2^s+1} + bX^{2^{t+s}+2^s}$. Let us suppose, that $b \neq 0$. We will consider a function $g(x) = b^{-1}f(x)$. Therefore we get:

$$g(x) = \frac{A}{b}X^{2^{n-t}+2^{t+s}} + \frac{A^{2^t}}{b}X^{2^s+1} + X^{2^{t+s}+2^s}.$$

As b is from the subfield $\mathbb{F}_{2^t}$ we know, that $b = b^{2^t}$. So we get:

$$g(x) = \frac{A}{b}X^{2^{n-t}+2^{t+s}} + \frac{A^{2^t}}{b2^t}X^{2^s+1} + X^{2^{t+s}+2^s}.$$

Which for the primitive element $A' = \frac{A}{b}$ equals:

$$g(x) = A'X^{2^{n-t}+2^{t+s}} + \frac{A'^{2^t}}{b2^t}X^{2^s+1} + X^{2^{t+s}+2^s}.$$

Therefore every function from the Family 4 with $b \neq 0$ is EA–equivalent to another function from the Family 4 with $b = 1$.

Here we present the list of all means we used to reduce the number of polynomials within each APN family to check for being CCZ–equivalent to a permutation:

Family 1: Frobenius equivalence with respect to A can be used, and we can also use left translation equivalence. We will find an exponent $2^{it}+2^{rt+s}-2^s-1$.

Family 2: Frobenius equivalence with respect to A can be used, and we can also use left translation equivalence. We will find an exponent $2^{it}+2^{rt+s}-2^s-1$ (same as with family 1).

Family 3: Frobenius equivalence with respect to A can be used, and we can also use subfield trick to force $c_{m-1} \in \mathbb{F}_2$.

Family 4: Frobenius equivalence with respect to A can be used, and we can also use the subfield trick to force $b \in \mathbb{F}_2$. Moreover left translation equivalence can be used to find exponent $2^{n-t} - 2^s$.

Family 5: Frobenius equivalence with respect to A can be used, and we can also use the subfield trick to force $b \in \mathbb{F}_2$. Moreover left translation equivalence can be used to find exponent $2^{n-t} - 2^s$ (same as with family 4).

Family 6: We can use the subfield trick to force $c \in \mathbb{F}_2$. Moreover when $c = 0$, we get exactly polynomials from family 5. Therefore we can set $c = 1$. Thus we can omit polynomials where $b = 1$.

Table 3.1: Calculated maximal dimensions of subspaces in NB_f

#	$n=6$	$n=8$	$n=10$	$n=12$
1	N/A	N/A	N/A	4
2	N/A	N/A	N/A	4
3	2	N/A	4	N/A
4	$3^{\dagger}$	N/A	N/A	4
5	$3^{\dagger}$	N/A	N/A	4
6	$3^{\dagger}$	N/A	N/A	3^{*}
7	2	N/A	4	N/A
8	2	2	4	3^{*}
9	3°	3	$5^{\circ\circ}$	4

"$*$" – we managed to check only some polynomials from the given family in the given dimension and therefore there is a small possibility that the number can be higher.
"$\dagger$" – in this family in this dimension there are functions which are equivalent to the APN permutation found in [2].
"$\circ$" – while there are subspaces of high enough dimension, in this particular dimension this function is actually just x^3 which is not CCZ–equivalent to a permutation.
"$\circ\circ$" – while there is a subspace of high enough dimension, there is only one – the subfield $\mathbb{F}_{2^{n/2}}$, and therefore by Theorem 15 it is not CCZ–equivalent to a permutation.
"N/A" – functions from the given family are not defined in this dimension

Family 7: We can use Frobenius equivalence with respect to C. Since the allowed Cs differ between exponents the reduction must be done for each exponent separately.

Family 8: No reductions were found.

Family 9: No reductions were found.

3.1.3 Results of computation

By applying the algorithm described above to functions from the Table 1.2 we have got results, that we summarized in the Table 3.1.

Now, using the corollary of Theorem 15 we can prove with the Table 3.1 that no polynomial from families 1,2,4,5 and 9 from the Table 1.2 are CCZ–equivalent to a permutation in even extensions up to dimension 12. For functions in other families we conjecture that they also are not CCZ–equivalent to a permutation. While we did not manage to try every single one, from each family we tried at least 20, and the maximal found dimension remained constant. In addition to this kind of results we have found, that in dimension 12 for Families 1,4 and 5, for exactly 50% of polynomials the maximal subspace dimension in NB_f was 4 and for the rest it was 3. Therefore since by Lemma 16 we know, that the maximal dimension of a subspace in NB_f is EA–invariant, and that for quadratic functions

Table 3.2: Running speed of Algorithm 3

#	$n = 6$	$n = 8$	$n = 10$	$n = 12$
time	2 s	10 s	3 min	2,5 h

are the notions of EA–equivalence and CCZ–equivalence the same, we conclude, that in these Families in these dimensions there are at least two CCZ–equivalence classes.

The reason we were not able to run the algorithm for some families in higher dimensions is the fact, that the number of those functions was to high. Nevertheless while checking the correctness of some parts of the Sage implementation using Magma, it became apparent that Sage is perhaps not the fastest environment for this kind of computation. Therefore one of the ways how to improve these results can be simply implementing the whole algorithm using Magma. Or rewriting it into C++, since there are also specialized libraries for finite field computations. The other way is proving better CCZ–equivalence results. For illustration we provide the approximate running time of our algorithm. For benchmarking we used a function from the Family 9 with $a = 1$, as functions from this family are defined over every $\mathbb{F}_{2^{2n}}$, and the found dimensions are quite high.

3.2 A theoretical result

We find, that for doubly even extensions maximal found dimension for polynomials in Family 9 is never greater than $n/2 - 1$. This is the first result of such kind for a non–monomial function.

Theorem 19. *An APN function of the form $x^3 + a^{-1}\mathrm{tr}_1^n(a^3x^9)$ is not CCZ–equivalent to a permutation in any field $\mathbb{F}_{2^{4m}}$.*

For the proof we will use the fact, that NB_{x^3} is the set of cubes in $\mathbb{F}$ (as shown in the first chapter), and the following result that appeared in [24]:

Theorem 20. *Let $C = \{\alpha^3 : \alpha \in \mathbb{F}_{2^n}\}$ and $U \subseteq C$ a subspace in C of maximal dimension. Then*

$$dim(U) \begin{cases} =n/2 & \text{if } n/2 \text{ is odd} \\ \leq n/2 - 2 & \text{if } n/2 \text{ is even} \end{cases}$$

Corollary 2. *Let $\mathbb{F}$ be a doubly even extension of $\mathbb{F}_2$. Let $C = \{\alpha^3 : \alpha \in \mathbb{F}\}$, and let $H_c = \{\alpha : \mathrm{tr}_1^n(c\alpha) = 0\}$. Let $U \subseteq C \bigcap H_c$. Then $dim(U) \leq n/2 - 2$.*

Proof. This can be seen from the fact, that for every subspace $U \subseteq C$, for every $x, y \in U \setminus H_c$ $x + y \in U \bigcap H_c$. Indeed:

$$\mathrm{tr}_1^n(c(x + y)) = \mathrm{tr}_1^n(cx) + \mathrm{tr}_1^n(cy) = 1 + 1 = 0.$$

And as $x, y \in U$, $x + y \in U$. Therefore in $U \bigcap H_c$ there can be only a half of elements of U. □

Let f denote $x^3 + a^{-1}\mathrm{tr}_1^n(a^3x^9)$. For f to be CCZ–equivalent to a permutation, by the Corrolary 1 of the Theorem 15, there would need to exist a subspace of dimension $n/2$ in the NB_f. We will show, that in $\mathbb{F}_2^n = \mathbb{F}_2^{4m}$ there exists none.

Proof. Let f denote $x^3 + a^{-1}\mathrm{tr}_1^n(a^3x^9)$. Let us suppose, that there is $U \subseteq NB_f$ such that $dim(U) = n/2$. Then we know, that $dim(U \bigcap H_{a^{-1}})$ has dimension at least $n/2 - 1$. So it is sufficient to show, that there is no big enough subspace in $\mathrm{NB}_f \bigcap H_{a^{-1}}$. We will show $\mathrm{NB}_f \bigcap H_{a^{-1}} = \mathrm{NB}_{x^3} \bigcap H_{a^{-1}}$. Then by using Corrolary 2 of Theorem 20 we will show ther is no such U.

Let us have $\alpha \in \mathrm{NB}_f \bigcap H_{a^{-1}}$. Then it holds that:

$$\sum_{x\in\mathbb{F}}(-1)^{\mathrm{tr}_1^n(\alpha(x^3+a^{-1}\mathrm{tr}_1^n(a^3x^9)))} \neq \pm 2^{n/2}$$

Now using the linearity of the trace we know that:

$$\sum_{x\in\mathbb{F}}(-1)^{\mathrm{tr}_1^n(\alpha(x^3+a^{-1}\mathrm{tr}_1^n(a^3x^9)))} = \sum_{x\in\mathbb{F}}(-1)^{\mathrm{tr}_1^n(\alpha(x^3))+\mathrm{tr}_1^n(\alpha a^{-1}\mathrm{tr}_1^n(a^3x^9))}.$$

Using the fact, that $\mathrm{tr}_1^n(a^3x^9)$ is an element of $\mathbb{F}_2$ only we need to consider two possibilities: $\mathrm{tr}_1^n(a^3x^9) = 0$ and $\mathrm{tr}_1^n(a^3x^9) = 1$. In the former case we get

$$\mathrm{tr}_1^n(\alpha a^{-1}\mathrm{tr}_1^n(a^3x^9)) = \mathrm{tr}_1^n(0\alpha a^{-1}) = \mathrm{tr}_1^n(0) = 0,$$

and in the latter case we get

$$\mathrm{tr}_1^n(\alpha a^{-1}\mathrm{tr}_1^n(a^3x^9)) = \mathrm{tr}_1^n(1\alpha a^{-1}) = \mathrm{tr}_1^n(\alpha a^{-1}) = 0$$

since $\alpha \in H_{a_{-1}}$. Therefore we get that:

$$\sum_{x\in\mathbb{F}}(-1)^{\mathrm{tr}_1^n(\alpha(x^3))+\mathrm{tr}_1^n(\alpha a^{-1}\mathrm{tr}_1^n(a^3x^9))} = \sum_{x\in\mathbb{F}}(-1)^{\mathrm{tr}_1^n(\alpha(x^3))},$$

which means that $\mathrm{NB}_f \bigcap H_{a^{-1}} = \mathrm{NB}_{x^3} \bigcap H_{a^{-1}}$ and therefore completes the proof. □

Therefore we managed to prove that the APN function $x^3 + a^{-1}\mathrm{tr}_1^n(a^3x^9)$ cannot be a permutation in infinite number of extension. To our knowledge the only similar results to this date are ones dealing only with monomials [24]. In [24] the inequivalence result is proved for Gold exponents. Indeed for Gold exponents $NB_f = C$ (x^3 is a function with a Gold exponent).

So, we have a proof for when the field is $\mathbb{F}_{2^{4m}}$. But what about the other (still even) case? The function seems to behave nicely with respect to dimension of its NB_f (except for dimension 6 where it is actually the function x^3). Therefore one would think it should be possible to proof the CCZ–inequivalence result for every $\mathbb{F}_{2^{2m}}$. We conjecture the same:

Conjecture 1. *The APN function $x^3 + a^{-1}\mathrm{tr}_1^n(a^3x^9)$ is not CCZ–equivalent to a permutation in any field $\mathbb{F}_{2^{2m}}$.*

Nevertheless our conjecture cannot be proven in similar way, since for $\mathbb{F}_{2^{4m+2}}$, there is at least one subspace in $C \bigcap H_1$ of dimension $n/2$ – the finite field $\mathbb{F}_{2^{2m+1}}$. $\mathbb{F}_{2^{2m+1}}$ as a subfield of $\mathbb{F}_{2^{4m+2}}$ is set of elements $\{a^{2^{2m+1}+1} : a \in \mathbb{F}_{2^{4m+2}}\}$, and for k odd we have that $3|2^k + 1$, so $\mathbb{F}_{2^{2m+1}} \subseteq C$. And as for $a \in \mathbb{F}_{2^{2m+1}}$ it holds that $a^{2^{2m+1}} = a$, then all terms in the $\mathrm{tr}_{2^{2m+1}}^{2^{4m+2}}$ cancel out, so $\mathbb{F}_{2^{2m+1}} \in H_1$. And indeed it is easy to see, that the dimension of $\mathbb{F}_{2^{2m+1}}$ is 2^{2m+1}.

But whether our conjecture will turn out to be right or wrong remains to be seen.

Conclusion

In the thesis we presented a new condition for checking whether an APN function is CCZ–inequivalent to a permutation. We used this condition to implement an algorithm to check known APN functions on $\mathbb{F}_{2^n}$ for n even for being CCZ–equivalent to a permutation.

While we did not find an APN permutation, we managed to computationally prove, that in some families in $\mathbb{F}_{2^{12}}$, there is no function that is CCZ–equivalent to a permutation. This is an improvement over the previous work done by Dillon in [2].

Also we found an EA–invariant and used it with [25, Theorem 1] to see, that in some families of APN functions in $\mathbb{F}_{2^{12}}$, there are at least 2 CCZ–equivalence classes.

Moreover we managed to prove, that the functions of the form:

$$x^3 + a^{-1}\mathrm{tr}_1^n(a^3x^9)$$

cannot be CCZ–equivalent to a permutation in any field $\mathbb{F}_{2^{4n}}$ for $n \in \mathbb{N}$. These kinds of results have been proved before only for monomials.

In the future work, it is then possible to speed up the algorithm by rewriting at least a part of it into C/C++. We believe, that by rewriting the program it should be possible to check all known APN functions in $\mathbb{F}_{2^{12}}$ and $\mathbb{F}_{2^{14}}$ in reasonable time.

Bibliography

[1] Xiang-dong Hou. Affinity of permutations of $\mathbb{F}_2^n$. *Discrete Applied Mathematics*, 154:313–325, 02 2006.

[2] K.A. Browning, J.F. Dillon, M.T. McQuistan, and A.J. Wolfe. An APN permutation in dimension six. *Postproceedings of the 9th International Conference on Finite Fields and Their Applications*, 518:33–42, 2010.

[3] Claude Carlet. Boolean functions for cryptography and error correcting codes. *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, pages 257–397, 2010.

[4] Harald Niederreiter Rudolf Lidl. *Finite Fields*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2 edition, 1997.

[5] R. Gold. Maximal recursive sequences with 3-valued recursive cross-correlation functions (corresp.). *IEEE Trans. Inf. Theor.*, 14(1):154–156, September 2006.

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

[7] T. Kasami. The weight enumerators for several classes of subcodes of the 2nd order binary reed-muller codes. *Information and Control*, 18(4):369 – 394, 1971.

[8] Hans Dobbertin. Almost perfect nonlinear power functions on GF(2^n): The Welch case. 45:1271 – 1275, 06 1999.

[9] Hans Dobbertin. Almost perfect nonlinear power functions on GF(2^n): A new case for n divisible by 5. In Dieter Jungnickel and Harald Niederreiter, editors, *Finite Fields and Applications*, pages 113–121, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[10] Kaisa Nyberg. Differentially uniform mappings for cryptography. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, pages 55–64, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[11] L. Budaghyan, C. Carlet, P. Felke, and G. Leander. An infinite class of quadratic APN functions which are not equivalent to power mappings. In *2006 IEEE International Symposium on Information Theory*, pages 2637–2641, July 2006.

[12] L. Budaghyan, C. Carlet, and G. Leander. Two classes of quadratic APN binomials inequivalent to power functions. *IEEE Trans. Inf. Theor.*, 54(9):4218–4229, September 2008.

[13] Carl Bracken, Eimear Byrne, Nadya Markin, and Gary McGuire. New families of quadratic almost perfect nonlinear trinomials and multinomials. *Finite Fields and Their Applications*, 14(3):703 – 714, 2008.

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

[15] Lilya Budaghyan and Claude Carlet. Classes of quadratic APN trinomials and hexanomials and related structures. 54:2354 – 2357, 06 2008.

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

[17] L. Budaghyan, C. Carlet, and G. Leander. On a construction of quadratic APN functions. In *2009 IEEE Information Theory Workshop*, pages 374–378, Oct 2009.

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

[19] Mitsuru Matsui and Atsuhiro Yamagishi. A new method for known plaintext attack of FEAL cipher. In Rainer A. Rueppel, editor, *Advances in Cryptology — EUROCRYPT' 92*, pages 81–91, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[20] Alexander Pott. Almost perfect and planar functions. *Des. Codes Cryptography*, 78(1):141–195, January 2016.

[21] T. P. Berger, A. Canteaut, P. Charpin, and Y. Laigle-Chapuy. On Almost Perfect Nonlinear functions over $\mathbb{F}_{2^n}$. *IEEE Trans. Inf. Theor.*, 52(9):4160–4170, September 2006.

[22] C. Xiang, K. Feng, and C. Tang. A construction of linear codes over $\mathbb{F}_{2^t}$ from Boolean functions. *IEEE Transactions on Information Theory*, 63(1):169–176, Jan 2017.

[23] K.A. Browning, J.F. Dillon, R.E. Kibler, and M.T. McQuistan. APN polynomials and related codes. 34, 01 2009.

[24] F. Gologlu and P. Langevin. APN families which are not equivalent to permutations. *preprint*, 2017.

[25] Satoshi Yoshiara. Equivalences of quadratic APN functions. *J. Algebraic Comb.*, 35(3):461–475, May 2012.

[26] Claude Carlet. Vectorial boolean functions for cryptography. *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, pages 398–469, 2010.

[27] Bo Sun. On equivalence of known families of APN functions in small dimensions. *PROCEEDING OF THE 20TH CONFERENCE OF FRUCT ASSOCIATION*, 04 2017.

List of Tables