Cryptography and Communications (2021) 13:995–1023
https://doi.org/10.1007/s12095-021-00541-8

Invariants for EA- and CCZ-equivalence of APN and AB functions

Nikolay S. Kaleyski[1]

Received: 16 February 2021 / Accepted: 6 July 2021 / Published online: 22 October 2021
© The Author(s) 2021

Abstract
An (n, m)-function is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. Such functions have numerous applications across mathematics and computer science, and in particular are used as building blocks of block ciphers in symmetric cryptography. The classes of APN and AB functions have been identified as cryptographically optimal with respect to the resistance against two of the most powerful known cryptanalytic attacks, namely differential and linear cryptanalysis. The classes of APN and AB functions are directly related to optimal objects in many other branches of mathematics, and have been a subject of intense study since at least the early 90's. Finding new constructions of these functions is hard; one of the most significant practical issues is that any tentatively new function must be proven inequivalent to all the known ones. Testing equivalence can be significantly simplified by computing invariants, i.e. properties that are preserved by the respective equivalence relation. In this paper, we survey the known invariants for CCZ- and EA-equivalence, with a particular focus on their utility in distinguishing between inequivalent instances of APN and AB functions. We evaluate each invariant with respect to how easy it is to implement in practice, how efficiently it can be calculated on a computer, and how well it can distinguish between distinct EA- and CCZ-equivalence classes.

Keywords CCZ-equivalence · EA-equivalence · Almost perfect nonlinear function · Almost bent function · Invariant

Mathematics Subject Classification 2010 06E30; 94A60

1 Introduction

A vectorial Boolean function, or (n, m)-function, is any mapping from the vector space $\mathbb{F}_2^n$ over the finite field $\mathbb{F}_2 = \{0, 1\}$ to the vector space $\mathbb{F}_2^m$, where n and m are arbitrary

This article belongs to the Topical Collection: *Boolean Functions and Their Applications V*
Guest Editors: Lilya Budaghyan, Claude Carlet, Tor Helleseth and Kaisa Nyberg

✉ Nikolay S. Kaleyski
Nikolay.kaleyski@uib.no

[1] Universitetet i Bergen, 5007 Bergen, Norway

natural numbers. In the particular case when $m = 1$, we refer to $(n, 1)$-functions simply as Boolean functions. In this sense, an (n, m)-function can be seen as an m-dimensional vector of $(n, 1)$-functions, hence the name. Vectorial Boolean functions are natural objects that have many applications within computer science and mathematics. Perhaps the simplest way to appreciate their utility is to observe that an (n, m)-function can be interpreted as an operation that accepts n bits as input, and returns m bits as output. Since virtually all data can be encoded as sequences of bits, this means that any transformation that operates on data of any kind can be naturally represented in terms of Boolean functions and vectorial Boolean functions. In particular, this is how all data is represented on an electronic computer; and one need look no further than the indicator function of a subset, or the incidence matrix of a graph to find examples of mathematical structures that can be encoded using binary functions. The spectrum of applications of vectorial Boolean functions includes areas as diverse as set theory, artificial intelligence, and algorithm design.

Vectorial Boolean functions also play a crucial role in cryptography, where they are used as building blocks of both stream and block ciphers. In both cases, the properties of the functions used directly influence the ultimate strength of the encryption; and thus, investigating the cryptographic properties of (n, m)-functions, and finding concrete instances of such functions that provide good resistance against various types of cryptanalytic attacks is an important topic of research. A prominent example is the Rijndael block cipher, selected as the Advanced Encryption Standard (AES) by the US National Institute of Standards and Technology (NIST), which is one of the most secure and arguably the most popular block cipher to date [29, 30]. One of the major factors contributing to the security of Rijndael is a strong vectorial Boolean function that lies at the core of the encryption. In general, if a particular form of cryptanalysis succeeds because of a cryptographically weak function, this is due to the function having some undesirable property that can be exploited by the attacker. Researchers have identified several properties and statistics that measure the resistance of a function to different kinds of attacks. This quantification of the cryptographic strength of vectorial Boolean functions has the great advantage that it allows the security of a given function to be measured in an objective and systematic way; and, what is of equal importance, provides researchers in the field with a concrete goal, namely to find functions attaining the optimum values of these properties.

Two of the most powerful attacks against modern block ciphers are the so-called differential cryptanalysis [5] and linear cryptanalysis [45]. The statistics that measure the resistance of a function F to these two attacks are called differential uniformity (denoted Δ_F) and nonlinearity (denoted $\mathcal{NL}(F)$), respectively. The differential uniformity of a function should be as low as possible, and the nonlinearity should be as high as possible in order to resist these two attacks. In the case of (n, m)-functions with $n = m$ (which is arguably the most practically significant and, therefore, well-studied case), the classes of (n, n)-functions that achieve optimum differential uniformity and nonlinearity are called almost perfect nonlinear (APN) and almost bent (AB) functions, respectively. These two classes were introduced in the early 90's [3, 46, 47], and have been the subject of intense study ever since. The fact that these functions are cryptographically optimal implies that they have very little structure or patterns that can be exploited by a malicious attacker; unfortunately, this also means that, in general, they have very few properties that can be used analytically or constructively (we note, however, that we are not claiming that e.g. APN and AB functions must lack any kind of structure whatsoever; and we remark that some cryptographically optimal functions do have strong structural properties; for instance, the earliest known APN functions are monomials, and so they have a very simple polynomial representation). Intuitively, this is one explanation for why their study is difficult, both theoretically and computationally. This is

witnessed by the fact that a number of long-standing questions and problems on APN and AB functions remain open to this day. The reader is referred to [25] for a comprehensive overview of the background and results in the area, including a list of open problems.

One promising vector of attack for resolving some of these problems is to find new instances of APN and AB functions. In this respect, it must be noted that any AB function is necessarily APN; the converse does not hold in general, although any quadratic APN function over $\mathbb{F}_{2^n}$ with odd n is AB [26]. This means that AB functions provide optimum resistance to both differential and linear cryptanalysis; and that searching for APN functions is a natural way to search for AB functions as well. The number of (n, n)-functions is astronomical even for small values of n: there are $(2^n)^{2^n}$ such functions over the finite field $\mathbb{F}_{2^n}$ for any natural number n, and so for values as small as $n = 6$, an exhaustive search is out of the question at the current level of computational technology. On the one hand, this makes it necessary to use more sophisticated combinations of mathematical characterizations and computational methods in order to find new APN functions; on the other hand, the large number of (n, n)-functions suggests that the total number of APN functions (despite them being a rather special class of functions) also grows rapidly with the dimension n, and their enumeration and classification quickly become infeasible.

To make the classification of some given class of objects manageable, a typical approach is to only classify them up to some suitable notion of equivalence that preserves the properties of interest to our study. In the case of APN and AB functions, these are the differential uniformity and the nonlinearity. At present, there are several known notions of equivalence on vectorial Boolean functions that preserve both of these properties. The most general known relation of this type is called Carlet-Charpin-Zinoviev equivalence (after the names of its inventors) [26], or CCZ-equivalence for short; results on APN and AB functions in the literature are typically given up to CCZ-equivalence. A less general, but also very frequently used equivalence relation is the extended affine equivalence, or EA-equivalence for short. Although it is known to be strictly less general than CCZ-equivalence (even if we allow taking inverses of permutations in addition to EA-equivalence) [20], it has been shown that CCZ- and EA-equivalence coincide in the case of quadratic APN functions (in the sense that two quadratic APN functions are CCZ-equivalent if and only if they are EA-equivalent) [53]; since the vast majority of known APN functions are quadratic, this makes the study of EA-equivalence and its properties almost as useful in practice as that of CCZ-equivalence.

Classifying functions up to CCZ-equivalence (or EA-equivalence) dispenses, at least partly, with the problem of their overwhelming number, but it raises another issue, namely: how to prove the equivalence or inequivalence of two given functions. This is a matter of great practical importance in the study of APN functions, since any "new" function obtained from a computational search or theoretical construction needs to be compared for equivalence against representatives from the equivalence classes of all known APN functions; this "new" function is then genuinely new only in the case that it is not equivalent to any of these representatives. Despite the definitions of both EA-equivalence and CCZ-equivalence being natural and simple, testing whether two given functions are equivalent is a very hard problem. Showing the inequivalence of two functions theoretically is only possible in some special cases, and is still very laborious; see e.g. [17], which contains a theoretical proof of inequivalence to the Gold, Kasami, and inverse power APN functions of an infinite family of APN binomials. Computationally testing equivalence is quite difficult too: to the best of our knowledge, there is currently no algorithm that can decide the CCZ-equivalence of any two functions directly from the definition; instead, one typically uses tests relying on the

isomorphism of linear codes [11, 37]. These tests have a number of shortcomings: they are difficult to implement (provided one does not have a working implementation of a test for linear code isomorphism); the computation time and memory consumption is significant; and, at least in the case of some implementations, these tests can give false negatives. More precisely, the procedure that we have available either outputs the form of the isomorphism showing that the linear codes corresponding to the tested functions are equivalent, or it outputs "false" in the case that no such isomorphism can be found; the latter, however, can happen either because the procedure has traversed the entire search space and found no such isomorphism (in which case we can correctly conclude that the functions are inequivalent); or it can be because the procedure has run out of memory and has been forced to abort early, in which case we can still get "false" as a result even though the functions may be equivalent. Despite this, the linear code test remains the only option for testing CCZ-equivalence in the general case. Other algorithms, operating from first principles, have been published for dealing with specialized cases of CCZ- and EA-equivalence, such as in the case of affine and linear equivalence [6], and the so-called restricted EA-equivalence [22, 48]; and, more recently, an algorithm relying on invariants for testing EA-equivalence for even dimensions [42], and an algorithm for testing the EA-equivalence of quadratic functions (for both even and odd dimensions) [23].

The classification of functions into equivalence classes can be significantly simplified and sped-up by means of invariants. An invariant is a property or statistic that is preserved under a given equivalence relation. For instance, if p is an integer-valued statistic computable for any (n, n)-function F, and is also a CCZ-invariant, then for any $F, G : \mathbb{F}_2^n \to \mathbb{F}_2^n$ that are CCZ-equivalent, we must have $p(F) = p(G)$. Invariants can facilitate the classification in several ways. Suppose that a tentative new instance of an APN function is obtained via some construction or computer search. First, the values of different invariants for this potentially new function can be computed and compared with those values for representatives from the known classes of APN functions. If the set of values for the new function does not match that of any known representative, then we can immediately conclude that the discovered function is indeed new, and no further tests need to be performed. If, however, the set of values does coincide with that of one or more of the known representatives, then only the representatives with that exact same set of values need to be tested for equivalence against the newly discovered function; this will typically be a significantly smaller set of functions. Most invariants are numerical values, which makes it easier to verify that the computation has produced a meaningful result, and precludes the possibility of false positives or false negatives. Finally, some invariants have a natural interpretation, and describe some property or structure of the function, which can be significant and useful in contexts other than deciding equivalence. Constantly updated tables of the known invariants for all known CCZ-inequivalent APN representatives can be found online at [1].

In this paper, we survey the known invariants for (n, n)-functions, with an emphasis on their utility for the classification of APN and AB functions. We evaluate each invariant with respect to several desirable properties. Ideally, we would like an invariant to be:

- simple: in other words, that it should not require any complicated algorithms or special software for its computation; ideally, an invariant would be easily implementable on any general purpose programming language without specialized tools or knowledge;
- efficient: that the invariant can be computed quickly, and without using too much memory for a reasonable range of dimensions n;
- useful: that it should take many different values for APN functions from distinct equivalence classes.

2 Preliminaries

Let n, m be natural numbers. An (n, m)-**function**, or **vectorial Boolean function** is a mapping between the vector spaces $\mathbb{F}_2^n$ and $\mathbb{F}_2^m$ over the finite field with two elements, $\mathbb{F}_2$. In the following, we concentrate on the case $n = m$, but we remark in passing that $(n, 1)$-functions are called simply **Boolean functions** (as opposed to vectorial Boolean functions). An (n, m)-function F can be seen as a vector $F = (f_1, f_2, \ldots, f_m)$ of Boolean $(n, 1)$-functions, each function $f_i(x)$ giving the value of the i-th coordinate of $F(x)$ for $x \in \mathbb{F}_2^n$. The functions f_i are called the **coordinate functions** of F. The **component functions** of F are all non-zero linear combinations of its coordinate functions. For $b \in \mathbb{F}_2^m$, the component function corresponding to the linear combination defined by b is denoted by F_b; that is, $F_b = \sum_{i=1}^m b_i f_i$, where $b = (b_1, b_2, \ldots, b_m)$ and f_i are the coordinate functions of F. As we shall see in Section 2.2, some cryptographic properties of an (n, m)-function F are expressed in terms of the Hamming distance between its components and certain other functions. We recall that the **Hamming distance** $d_H(F, G)$ between two (n, m)-functions F and G is the number of inputs $x \in \mathbb{F}_2^n$ on which the values of F and G are distinct, i.e. $d_H(F, G) = \#\{x \in \mathbb{F}_2^n : F(x) \neq G(x)\}$.

In the discussion of some of the invariants, we will use the notion of a multiset. Intuitively, a multiset is an unordered collection of elements in which the same element can occur multiple times. We will refer to the number of times that an element e occurs in a multiset M as the **multiplicity** of e in M, and will refer to the collection of the multiplicities of all elements in M as the **multiplicities of** M. Multisets will be denoted by square brackets, e.g. $[a, b, c]$, in contrast to ordinary sets, which are denoted by curly braces, e.g. $\{a, b, c\}$.

2.1 Representation of vectorial Boolean functions

A vectorial Boolean function F from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ can be represented in several different ways. Perhaps the simplest is the so-called truth-table representation, which is simply an exhaustive list of the values $F(x)$ for all possible inputs $x \in \mathbb{F}_2^n$. The truth table for a (3, 3)-function is given as an example in Table 1. The truth-table representation is rather efficient for implementing vectorial Boolean functions on a computer; in fact, the straightforward implementations of most invariants work best if the input (n, n)-function is given as a truth-table. Nonetheless, it has a number of serious shortcomings that make other representations preferable: the size of the table grows exponentially with n and m; it is difficult to observe any properties of the function from its truth table without performing non-trivial computations; and it is difficult to express infinite families and constructions via truth tables.

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

x	$F(x)$
000	000
001	101
010	110
011	111
100	100
101	011
110	001
111	010

Any (n, m)-function can be uniquely represented as a multivariate polynomial of the form

$$F(x_1, x_2, \ldots, x_n) = \sum_{I \subseteq \{1,2,\ldots,n\}} a_I \prod_{i \in I} x_j,$$

where $a_I \in \mathbb{F}_2^m$ for all $I \subseteq \{1, 2, \ldots, n\}$ and the variables $x_1, x_2, \ldots, x_n$ take values in $\mathbb{F}_2$. This representation is known as the **algebraic normal form (ANF)** of F. In some cases, the ANF allows for a significantly more compact representation of an (n, n)-function (when the dimensions n and m are large, so that the size of the truth table becomes prohibitive; while the number of terms with a non-zero coefficient in the ANF is small). The degree of the ANF (as a multivariate polynomial) is called the **algebraic degree** of F; due to the uniqueness of the ANF, this notion is well defined. The algebraic degree has cryptographic significance (it must be large in order to resist higher-order differential attacks) and, as we shall see later, it is invariant under EA-equivalence (but not under CCZ-equivalence).

A function of algebraic degree at most 1, resp. 2, resp. 3 is called **affine**, resp. **quadratic**, resp. **cubic**. Any affine (n, n)-function A satisfies

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

for any $x, y, z \in \mathbb{F}_2^n$, and thus this notion coincides with the usual definition of affinity. If an affine function L satisfies $L(0) = 0$ so that

$$L(x) + L(y) = L(x + y)$$

for any $x, y \in \mathbb{F}_2^n$, it is called **linear**. Affine and linear functions as defined here behave in the same way as they do over any vector space, and so all familiar notions and principles from linear algebra can be carried over to the case of vectorial Boolean functions.

For example, the function given by its truth-table in Table 1 has the ANF

$$\begin{aligned} F(x_1, x_2, x_3) \;=\; & (0, 1, 1)x_1x_2 + (0, 1, 0)x_1x_3 + (1, 0, 0)x_2x_3 + \\ & (1, 0, 0)x_1 + (1, 1, 0)x_2 + (1, 0, 1)x_3. \end{aligned} \tag{1}$$

We can immediately see that its algebraic degree is 2; in other words, this is a quadratic function. In this particular case, the size of the ANF is not significantly smaller than that of the truth-table; but as the dimension n increases, the disparity between the size of the two representations becomes more pronounced.

Let $\mathbb{F}_{2^n}$ denote the finite field with 2^n elements for some natural number n. Recall that $\mathbb{F}_{2^n}$ can be represented as an n-dimensional vector space over the prime field $\mathbb{F}_2$. Thus, $\mathbb{F}_2^n$ can be identified with $\mathbb{F}_{2^n}$, and (n, n)-functions can be seen as mapping from $\mathbb{F}_{2^n}$ to itself. This allows any (n, n)-function to be represented as a univariate polynomial of the form

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

where $a_i \in \mathbb{F}_{2^n}$ for $0 \leq 1 \leq 2^n - 1$. This is known as the **univariate representation** of F; just like the ANF, it always exists and is uniquely defined. The algebraic degree can be obtained directly from the univariate representation as the maximum binary weight of an exponent i with a non-zero coefficient a_i. The **binary weight** (also called 2-weight) of a natural number i, denoted by $w_2(i)$, is the number of distinct powers of 2 in its binary decomposition; that is, if we write i as $i = \sum_j a_j 2^j$ for $a_j \in \{0, 1\}$, then $w_2(i) = \sum_j a_j$. Equivalently, $w_2(i)$ is the number of non-zero bits in the binary representation of i (for

instance, $w_2(11) = 3$ since 11 can be written as $2^3 + 2^1 + 2^0$, or as 1011 in binary). The algebraic degree of F is then

$$\deg(F) = \max\{w_2(i) : 0 \leq i \leq 2^n - 1, a_i \neq 0\}.$$

We also remark that the component functions of F can be expressed using the absolute trace function $\mathrm{Tr} : \mathbb{F}_{2^n} \to \mathbb{F}_2$ as the Boolean functions $F_b : x \mapsto \mathrm{Tr}(bF(x))$ for all non-zero $b \in \mathbb{F}_{2^n}$. We recall that the absolute trace is defined by

$$\mathrm{Tr}(x) = x + x^2 + x^{2^2} + \cdots + x^{2^{n-1}}.$$

The univariate representation is by far the most widely used at the moment. A decisive reason for this is that many of the known APN functions (including infinite constructions and families) have a very simple form under this representation; a classic example is the Gold function x^3, which is known to be APN over $\mathbb{F}_{2^n}$ for any natural number n [38, 46]. Indeed, the $(3, 3)$-function represented by Table 1 and the ANF in (1) is precisely x^3 over $\mathbb{F}_{2^3}$. At the time of writing, most known infinite constructions of APN functions are expressed in the univariate representation; there are also a few constructions based on the so-called bivariate representation (that we do not treat here, but briefly discuss below). A list of the known infinite families of APN monomials is given in Table 2, and a list of the known infinite polynomial APN families is given in Table 3.

The above is by no means an exhaustive list of known representations of vectorial Boolean functions. Nonetheless, it should provide sufficient context for the sequel, and we shall not go into any further details on the topic of representation of (n, n)-functions. We will, however, mention the bivariate representation, in which a $(2n, 2n)$-function can be represented as pair of polynomials $(F_1(x, y), F_2(x, y))$ in two variables (see e.g. [25], p.47); the representation of a quadratic function by means of a so-called quadratic APN matrix (QAM) [56]; and the representation of a function by means of the values of its first-order derivatives [50].

2.2 APN and AB functions

Let F be an (n, m)-function for some natural number n, and denote by $\delta_F(a, b)$ the number of solutions $x \in \mathbb{F}_{2^n}$ to the equation

$$F(x + a) + F(x) = b \tag{2}$$

for some $a, b \in \mathbb{F}_{2^n}$. Note that the equation expresses the difference between two outputs of F whose corresponding inputs are at distance a. If for some given a, some value of b is

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

Family	Exponent	Conditions	Algebraic degree	Source
Gold	$2^i + 1$	$\gcd(i, n) = 1$	2	[38, 46]
Kasami	$2^{2i} - 2^i + 1$	$\gcd(i, n) = 1$	$i + 1$	[40, 44]
Welch	$2^t + 3$	$n = 2t + 1$	3	[33]
Niho	$2^t + 2^{t/2} - 1$, t even $2^t + 2^{(3t+1)/2} - 1$, t odd	$n = 2t + 1$	$(t + 2)/2$ $t + 1$	[32]
Inverse	$2^{2t} - 1$	$n = 2t + 1$	$n - 1$	[3, 46]
Dobbertin	$2^{4i} + 2^{3i} + 2^{2i} + 2^i - 1$	$n = 5i$	$i + 3$	[34]

Table 3 Known infinite families of quadratic APN polynomials over $\mathbb{F}_{2^n}$

ID	Functions	Conditions	Source
F1-F2	$x^{2^s+1}+u^{2^k-1}x^{2^{ik}+2^{mk+s}}$	$n=pk, \gcd(k,3)=\gcd(s,3k)=1, p\in\{3,4\}, i=sk \bmod p, m=p-i, n\geq 12, u$ primitive in $\mathbb{F}_{2^n}^*$	[17]
F3	$sx^{q+1}+x^{2^i+1}+x^{q(2^i+1)}+cx^{2^iq+1}+c^qx^{2^i+q}$	$q=2^m, n=2m, gcd(i,m)=1, c\in\mathbb{F}_{2^n}, s\in\mathbb{F}_{2^n}\setminus\mathbb{F}_q, X^{2^i+1}+cX^{2^i}+c^qX+1$ has no solution x s.t. $x^{q+1}=1$	[15]
F4	$x^3+a^{-1}\mathrm{Tr}_n(a^3x^9)$	$a\neq 0$	[18]
F5	$x^3+a^{-1}\mathrm{Tr}_n^3(a^3x^9+a^6x^{18})$	$3\|n, a\neq 0$	[19]
F6	$x^3+a^{-1}\mathrm{Tr}_n^3(a^6x^{18}+a^{12}x^{36})$	$3\|n, a\neq 0$	[19]
F7-F9	$ux^{2^s+1}+u^{2^k}x^{2^{-k}+2^{k+s}}+vx^{2^{-k}+1}+wu^{2^k+1}x^{2^s+2^{k+s}}$	$n=3k, \gcd(k,3)=\gcd(s,3k)=1, v,w\in\mathbb{F}_{2^k}, vw\neq 1, 3\|(k+s), u$ primitive in $\mathbb{F}_{2^n}^*$	[9]
F10	$(x+x^{2^m})^{2^k+1}+u'(ux+u^{2^m}x^{2^m})^{(2^k+1)2^i}+u(x+x^{2^m})(ux+u^{2^m}x^{2^m})$	$n=2m, m\geqslant 2$ even, $\gcd(k,m)=1$ and $i\geqslant 2$ even, u primitive in $\mathbb{F}_{2^n}^*, u'\in\mathbb{F}_{2^m}$ not a cube	[57]
F11	$a^2x^{2^{2m+1}+1}+b^2x^{2^{m+1}+1}+ax^{2^{2m}+2}+bx^{2^m+2}+(c^2+c)x^3$	$n=3m, m$ odd, $L(x)=ax^{2^{2m}}+bx^{2^m}+cx$ satisfies the conditions of Lemma 8 of [14]	[14]
F12	$u(u^qx+x^qu)(x^q+x)+(u^qx+x^qu)^{2^{2i}+2^{3i}}+a(u^qx+x^qu)^{2^{2i}}(x^q+x)^{2^i}+b(x^q+x)^{2^i+1}$	$q=2^m, n=2m, \gcd(i,m)=1, x^{2^i+1}+ax+b$ has no roots in $\mathbb{F}_{2^m}$	[52]
F13	$x^3+a(x^{2^i+1})^{2^k}+bx^{3\cdot 2^m}+c(x^{2^{i+m}+2^m})^{2^k}$	$n=2m=10, (a,b,c)=(\beta,1,0,0), i=3, k=2, \beta$ primitive in $\mathbb{F}_{2^2}$ $n=2m, m$ odd, $3\nmid m$, $(a,b,c)=(\beta,\beta^2,1), \beta$ primitive in $\mathbb{F}_{2^2}, i\in\{m-2, m, 2m-1, (m-2)^{-1} \bmod n\}$	[21]

significantly more likely to occur than all others, an attacker can exploit this to derive a correlation between the input and output of the function. This is the basic idea of differential cryptanalysis, which is one of the most powerful known attacks against block ciphers [5]. In order to be resistant to such attacks, the number of solutions to (2) should be as uniform as possible over all possible choices of $b\in\mathbb{F}_{2^n}$ for any fixed non-zero $a\in\mathbb{F}_{2^n}$. The **differential uniformity** of F, denoted by Δ_F, is the largest value of $\delta_F(a,b)$ over all choices of $0\neq a\in\mathbb{F}_{2^n}$ and $b\in\mathbb{F}_{2^n}$. Symbolically:

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

The differential uniformity should be as low as possible in order to resist differential cryptanalysis, and since $x+a$ is a solution to (2) whenever x is, the differential uniformity of any (n,n)-function can be no lower than two. A function is called **almost perfect nonlinear (APN)** if it achieves this trivial lower bound with equality.

The **differential spectrum** $\mathcal{D}_F$ of F is the multiset of the values of $\delta_F(a,b)$ over all $a,b\in\mathbb{F}_{2^n}$ with $a\neq 0$; that is,

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

Clearly, a function is APN if and only if its differential spectrum consists of the two values 0 and 2. The differential spectrum is, in fact, invariant under CCZ-equivalence, but it is practically useless for the purpose of distinguishing inequivalent APN functions. It does, however, take a much more prominent role as an invariant of the ortho-derivatives of quadratic APN functions, as described in Section 4.6.

We remark that the function $D_aF(x) = F(x + a) + F(x)$ is called the (discrete first-order) **derivative** of F in direction $a \in \mathbb{F}_{2^n}$. An APN function can be equivalently defined as a function all of whose derivatives D_aF for $a \neq 0$ are 2-to-1 functions.

Another powerful attack against block ciphers is linear cryptanalysis [45], which attempts to approximate the behavior of a function by means of linear functions. Intuitively, a function F should be as far away from all linear functions as possible in order to be resistant to this attack. The nonlinearity of a Boolean function $f : \mathbb{F}_{2^n} \to \mathbb{F}_2$ is defined as the minimum Hamming distance between f and any affine $(n, 1)$-function. This notion can then be naturally generalized to the case of vectorial Boolean functions through the nonlinearity of their component functions. The **nonlinearity** of an (n, n)-function F, denoted by $\mathcal{NL}(F)$, is defined as the minimum Hamming distance between any component function of F, and any affine $(n, 1)$-function. The nonlinearity should be as high as possible in order to resist linear cryptanalysis, and it has been shown [27, 51] that it satisfies

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

for any (n, n)-function F. The functions that attain this upper bound with equality are called **almost bent (AB)** functions. Note that AB functions exist only for odd values of n. In the case of even n, functions with nonlinearity $2^{n-1} - 2^{n/2}$ are known, and they are conjectured to be optimal with respect to nonlinearity; nonetheless, the exact upper bound on the nonlinearity for $(2n, 2n)$-functions remains an open question.

It is also known that any AB function is necessarily APN. The converse does not hold in general, although it is known that any quadratic APN (n, n)-function is AB when n is odd [26]. Thus, AB functions provide optimal resistance against both linear and differential cryptanalysis. In particular, constructing new instances and families of APN functions is a natural approach to finding new constructions of AB functions.

Besides providing the best possible resistance to differential and linear cryptanalysis, respectively, APN and AB functions correspond to optimal objects in other areas of mathematics and computer science, including coding theory, combinatorics, projective geometry, and sequence design. Developments in the study of cryptographic vectorial Boolean functions therefore naturally lead to progress in other areas; and results and methods used in these related fields of study can be applied to the investigation of APN and AB functions. In essence, the natural definitions of APN and AB functions make them significant, universal objects that transcend the immediate practical needs of cryptography and have a much broader relevance.

In addition to the differential uniformity and nonlinearity, the algebraic degree of an (n, n)-function is also one of its important cryptographic properties; it should be high in order to provide good resistance against higher-order differential attacks. In addition, there are quite a few other statistics and properties for measuring the cryptographic strength of a function against various kinds of attacks. Since in this paper we mostly focus on invariants with respect to the classification of APN and AB functions, we do not go into further details here. We refer the reader to [25] for a more comprehensive treatment of the subject.

2.3 Equivalence relations

Equivalence relations on (n, n)-functions that preserve the differential uniformity and nonlinearity are used to reduce the number of functions that need to be studied and classified. Currently, CCZ-equivalence is the most general known equivalence relation preserving these properties, and so classification and computational results on APN and AB function are typically given up to CCZ-equivalence.

The notion of the CCZ-equivalence of two functions is expressed in terms of their graphs. Let F and G be (n, n)-functions for some natural number n. The graph of F is the set $\Gamma_F = \{(x, F(x)) : x \in \mathbb{F}_{2^n}\}$. Note that for any (n, n)-function F, its graph Γ_F is contained in the set $\mathbb{F}_{2^n}^2$ of pairs of elements from $\mathbb{F}_{2^n}$. The latter can be identified with $\mathbb{F}_{2^{2n}}$, and thus we can assume that Γ_F is contained in $\mathbb{F}_{2^{2n}}$. Then F and G are said to be **Carlet-Charpin-Zinoviev equivalent**, or **CCZ-equivalent** for short, if there exists an affine permutation A of $\mathbb{F}_{2^{2n}}$ mapping Γ_F to Γ_G, i.e.

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

In Section 3, we will concentrate on properties that are left invariant by CCZ-equivalence. For the time being, we remark that CCZ-equivalence preserves neither the algebraic degree, nor the bijectivity of the function. This is noteworthy, as both of these can (and have) been used constructively. In 2010, John Dillon constructed the only currently known instance of an APN (n, n)-permutation for even n by traversing the CCZ-equivalence class of a known (non-bijective) APN function over the same field [12]. In a similar vein, most of the known instances of APN functions listed in the literature are quadratic. As pointed out in Section 2.2, it is desirable for the algebraic degree to be high in order to resist higher-order differential attacks. Traversing the CCZ-class of a quadratic APN function may yield functions of higher algebraic degree that are CCZ-equivalent to it (and hence also APN).

A special case of CCZ-equivalence is the so-called extended affine equivalence, or EA-equivalence for short. Two (n, n)-functions F and G are said to be **EA-equivalent** if there exist affine (n, n)-functions A_1, A_2, A with A_1 and A_2 bijective such that

$$A_1 \circ F \circ A_2 + A = G. \tag{3}$$

CCZ-equivalence is strictly more general than EA-equivalence combined with taking inverses of permutations [20]. Nonetheless, the two equivalence relations coincide in the case when both F and G are quadratic; that is, if F and G are quadratic APN (n, n)-functions, then F and G are CCZ-equivalent if and only if they are EA-equivalent [53]. In practice, this makes EA-equivalence (and hence, properties that are invariant under EA-equivalence) almost as useful as CCZ-equivalence in the case of APN functions, since all known APN instances are equivalent to quadratic functions or monomials, with only a single known exception for $n = 6$ [36].

Further specializations of EA-equivalence can be obtained by imposing additional restrictions on A_1, A_2 and A. If $A = 0$ in (3), we say that F and G are **affine equivalent**. If $A = 0$ and $A_1(0) = A_2(0) = 0$ (so that A_1 and A_2 are linear instead of merely affine), we say that F and G are **linear equivalent**. Further restrictions have been investigated, such as the so-called restricted EA-equivalence [22, 48]. These relations are of limited interest in the APN case, and so we mostly concentrate on the notions of EA- and CCZ-equivalence.

For the sake of completeness, we also mention a special kind of equivalence that can be defined for two power functions. If $F(x) = x^{e_1}$ and $G(x) = x^{e_2}$ are (n, n)-functions for some natural numbers e_1 and e_2, then we say that F and G are **cyclotomic equivalent** if there exists a natural number k such that $2^k e_1 = e_2 \pmod{2^n - 1}$ or $2^k e_1^{-1} =$

$e_2 \pmod{2^n - 1}$, where e_1^{-1} is the multiplicative inverse of e_1 modulo $2^n - 1$ (if it exists). We note that this is a special case of affine equivalence and taking inverses of permutations, since the function x^{e_2} with $2^k e_1 = e_2 \pmod{2^n - 1}$ can be obtained from x^{e_1} by composing the latter with the linear permutation $x \mapsto x^{2^k}$. It is known that if two power functions are CCZ-equivalent, they are necessarily cyclotomic equivalent [31, 54]. This greatly reduces the complexity of deciding equivalence in the case of power functions since, unlike EA- and CCZ-equivalence, testing cyclotomic equivalence is simple and amounts to solving modular equations.

2.4 Testing equivalence via linear codes

The simplicity of the definitions of the equivalence relations in Section 2.3 belies the complexity of testing them in practice. Indeed, doing so by exhaustive search (for instance, over all affine permutations of $\mathbb{F}_{2^n}^2$ in the case of CCZ-equivalence) is only feasible for very small dimensions. On the other hand, no efficient algorithm for deciding CCZ- or EA-equivalence from first principles in the general case is known; the algorithm for testing EA-equivalence from [42] is only effective for even dimensions; the one from [23] concerns only quadratic functions; and the methods described in [6, 22, 48] apply to certain special cases of EA-equivalence. We note that cyclotomic equivalence is a notable exception since deciding it involves verifying the solvability of a small number of modular equations; this can be done efficiently even for very high dimensions.

Testing CCZ-equivalence is typically done by means of linear codes as described in [11, 37]. Given any (n, n)-function F, we can associate with it a linear code $\mathcal{C}_F$ with the parity-check matrix

$$M_F = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & \alpha & \alpha^2 & \cdots & \alpha^{2^n-2} \\ F(0) & F(1) & F(\alpha) & F(\alpha^2) & \cdots & F(\alpha^{2^n-2}) \end{pmatrix}, \tag{4}$$

where α is a primitive element of $\mathbb{F}_{2^n}$. Then two (n, n)-functions F and G are CCZ-equivalent if and only if their associated codes $\mathcal{C}_F$ and $\mathcal{C}_G$ are isomorphic, i.e. if there is a permutation π of $\{1, 2, \ldots, 2^n\}$ such that $(x_1, x_2, \ldots, x_n)$ is a codeword of $\mathcal{C}_F$ if and only if $(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(2^n)})$ is a codeword of $\mathcal{C}_G$.

The problem of deciding the isomorphism of two linear codes is itself quite difficult. The advantage of the above reduction is that coding theory is a very well developed area, and algorithms for testing the isomorphism of linear codes are known and well studied. Mathematical libraries and programming languages (such as *Magma* [8]) typically have a working implementation of such algorithms, and implementing a CCZ-equivalence test then amounts to simply constructing the parity-check matrices in (4) and passing them to the relevant coding-theoretic framework.

Testing EA-equivalence can be done in the same way, except that a different associated code is used [37]. In this case, the parity-check matrix has a different form; once again, two functions are EA-equivalent if and only if their associated codes are isomorphic. In practice, the associated codes in the case of EA-equivalence have a larger length than those for CCZ-equivalence, and so testing EA-equivalence in this way is computationally more difficult. In practice, one typically uses the test for CCZ-equivalence in the case of quadratic APN functions since then two functions are CCZ-equivalent if and only if they are EA-equivalent. In the same paper, it is shown that affine equivalence can be tested in the same way as well; however, the length of the associated code is even larger than in the case of

EA-equivalence, and since vectorial Boolean functions are typically classified up to CCZ- or EA-equivalence, this last result is of limited practical utility.

Thus, deciding the CCZ-equivalence of two (n, n)-functions is currently only possible via the isomorphism of linear codes. Unfortunately, this has certain shortcomings. For one, the computation time and memory requirements increase exponentially with the dimension n. In practice, we are only able to test functions up to $n = 10$ on our server; for higher dimensions, the memory consumption becomes prohibitive. Another problem is that the current implementation of the isomorphism test in *Magma* can give false negatives if it runs out of memory during the computation. This behavior occurs only if the dimension is high; but it is possible for this to occur for dimensions as low as $n = 9$ in some cases.

Table 4 give some sample running times (in seconds) for verifying that two arbitrarily selected APN functions from among the known CCZ-inequivalent representatives over $\mathbb{F}_{2^n}$ are indeed CCZ-inequivalent for $6 \leq n \leq 10$. These sample times are measured by selecting some pairs of functions at random from among the known APN functions, and running the linear code isomorphism on them; the times given in the table are averaged over the tested pairs. The computation time in the case of CCZ-equivalent functions is typically less, since the algorithm terminates as soon as an equivalence is found; while in the case of inequivalent functions, all possibilities have to be exhausted before a negative answer can be given. The computation time also depends on the concrete pair of functions being tested; if one of the functions is a monomial, the computation is generally faster. For some pairs of functions, the running time can be significantly longer or shorter; the data given in Table 4 (as well as all other tables in this paper that contain computation times) are only meant to provide a general idea of how long the computation lasts in higher dimensions when compared to lower ones (of course, the actual computation depends on a lot of other factors, such as the computational equipment used, the concrete implementation, the number of processes running in parallel, etc.). We stress that the running times given in the following tables are only meant to give a general idea of the efficiency of different methods, and make no claims about the experiments being statistically accurate.

One remarkable observation that we can make from the table is that testing CCZ-equivalence for odd dimensions takes significantly longer than doing so for even dimensions; for instance, we can see that although 8 is greater than 7, the running time for comparing functions over $\mathbb{F}_{2^7}$ is typically several times longer than the time for doing the same over $\mathbb{F}_{2^8}$. A possible intuitive explanation for this surprising behavior could be that the known APN functions over fields of odd extension degree are much more "similar" to each other (and thus, harder to distinguish) than those over fields of even extension degree. Indeed, the vast majority of APN functions that we know over any finite field (regardless of the parity of its extension degree) are quadratic; and we know that any quadratic APN function over $\mathbb{F}_{2^n}$ for odd n is AB. Furthermore, AB functions behave more "predictably" than APN functions in some sense (for instance, they have a fixed

Table 4 Sample times (in seconds) for verifying CCZ-inequivalence over $\mathbb{F}_{2^n}$ via the linear code equivalence test

n	6	7	8	9	10
time (s)	0.020	7.170	0.180	8311.830	40.880

value of the nonlinearity, and do not have any bent components; the number of bent components is actually one of the invariants under EA-equivalence discussed in Section 4.3, and can take many distinct values across the known quadratic APN functions over $\mathbb{F}_{2^n}$ for even values of n). As we will observe in Sections 3 and 4, some invariants behave similarly, in the sense that they can taken many distinct values for APN functions over finite fields of even extension degree (and can thus be useful for distinguishing between EA- or CCZ-inequivalent ones); while the same invariants almost always take the same value for the known APN functions over $\mathbb{F}_{2^n}$ with n odd, and so are practically useless in this respect.

The above considerations further underline the practical importance of invariants. Computational searches can easily produce thousands of functions and, especially in the case of higher dimensions where testing equivalence can take a long time, partitioning the functions according to the values of an efficiently computable invariant can save a lot of computation time. In the case that two inequivalent functions have different values for a certain invariant (or a combination of invariants), this precludes any possibility of a false negative.

2.5 A note on the computational results

In the following Sections 3 and 4, we survey the most frequently used invariants for classifying APN and AB functions up to CCZ- and EA-equivalence, and report on some computational results for measuring how quickly these invariants can be computed, and how well they can distinguish between distinct CCZ- and EA-equivalence classes. In this section, we describe the computational equipment that we used and the sets of functions that we tested the invariants on.

Computations in C and Python were performed on an HP EliteDesk 800 G2 SFF computer, with a quad core 3.2 GHz processor with 15 GB of memory. Computations in *Magma* were performed on a server with 56 3.2 GHz cores and 500 GB of memory.

All the experiments that we conduct are for dimensions $n \geq 6$ since the classification of APN functions for $n < 6$ is already complete [10]. In the case of $n = 6$, we use the 14 CCZ-inequivalent representatives from the switching classes [36]; 13 of them are quadratic and represent all CCZ-classes of quadratic functions over $\mathbb{F}_{2^6}$ [35], while the remaining function is the only known example of an APN instance CCZ-inequivalent to monomial and quadratic functions. For $n = 7$, we use the list of 390 functions from [55, 56] along with the newly found quadratic APN function from [43]; as shown in [43], the quadratic representatives therein encompass all possible CCZ-classes of quadratic APN functions over $\mathbb{F}_{2^7}$. For $n = 8$, we use the 8181 functions from [55]; we note that recently more than 12 000 new inequivalent APN instances have been discovered in $\mathbb{F}_{2^8}$ [2], but we do not involve these in the computations since the goal is to give an empiric idea of how efficient the various invariants are in distinguishing distinct CCZ-classes (rather than to do a complete classification); a constantly updated list of invariants for the known functions is available at [1]. For $n = 9$ and $n = 10$, we take CCZ-inequivalent representatives from the known infinite APN families along with the new functions found in [2]; this is necessary, since some invariants tend to take a lot of different values among the known APN instances, but only one or two values for instances belonging to the currently known APN families. Since no classification of APN functions is available for dimensions $n \geq 11$, we restrict ourselves to only computing invariants for a few select functions in order to estimate the computation time, without making any claims about the distinguishing power of the invariants in those dimensions.

3 Invariants under CCZ-equivalence

3.1 Trivial invariants

The differential uniformity (and, more generally, the differential spectrum) and nonlinearity are invariant under CCZ-equivalence, but this does not help with the classification of APN and AB functions, since by definition they have a fixed value of the differential uniformity and nonlinearity, respectively.

3.2 The extended Walsh spectrum

3.2.1 Definition

The Walsh transform $W_F : \mathbb{F}_{2^n}^2 \to \mathbb{Z}$ is an integer-valued function that can be associated with any (n, n)-function F. More precisely, the **Walsh transform** of F is defined as

$$W_F(a, b) = \sum_{x \in \mathbb{F}_{2^n}} (-1)^{b \cdot F(x) + a \cdot x}, \tag{5}$$

where "$\cdot$" denotes a scalar product, or dot product on $\mathbb{F}_{2^n}$ (or, equivalently, on $\mathbb{F}_2^n$). The properties of the Walsh transform do not depend on the concrete choice of the scalar product. There are at least two frequently used choices:

- if the multiplicands $a = (a_1, a_2, \ldots, a_n)$ and $b = (b_1, b_2, \ldots, b_n)$ are viewed as n-dimensional binary vectors in $\mathbb{F}_2^n$, the product can be defined as $a \cdot b = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$, with addition and multiplication being over $\mathbb{F}_2$;
- the product can also be defined as $a \cdot b = \mathrm{Tr}(ab)$, where $\mathrm{Tr} : \mathbb{F}_{2^n} \to \mathbb{F}_2$ is the absolute trace function from $\mathbb{F}_{2^n}$ to the prime field $\mathbb{F}_2$; in this case, the operands a and b are multiplied in the finite field, and then this product is mapped onto $\mathbb{F}_2$ via the trace.

The former realization tends to be more suitable for implementing the Walsh transform in a general-purpose programming language, while the latter is typically used in theoretical proofs and constructions.

The values $W_F(a, b)$ of the Walsh transform are referred to as **Walsh coefficients**. The **Walsh spectrum** of an (n, n)-function F is the multiset of the values of its Walsh coefficients for all possible $a, b \in \mathbb{F}_{2^n}$. The **extended Walsh spectrum** $\mathcal{W}_F$ of F is the multiset of the absolute values of all of the Walsh coefficients of F, that is

$$\mathcal{W}_F = [|W_F(a, b)| : a, b \in \mathbb{F}_{2^n}].$$

The extended Walsh spectrum is invariant under CCZ-equivalence.

As we will see in the evaluation below, the extended Walsh spectrum is not a very useful invariant in terms of its distinguishing power. Nonetheless, it is a good idea to compute it as a first step in analyzing a function, since the computation of several of the following invariants either require, or are facilitated by, knowledge of all values of the Walsh transform. Furthermore, it does become useful in distinguishing inequivalent quadratic APN functions when applied to their ortho-derivatives rather than to the functions themselves; ortho-derivatives are described in Section 4.6.

3.2.2 Evaluation

The Walsh transform can be implemented easily on any programming language; its computation from (5) requires nothing more complicated than basic arithmetic operations (addition, modulation, and XOR, and possibly finite field multiplication depending on how the scalar product is implemented). On algebra systems such as *Magma* [8] that include built-in functionality for computations over finite fields, both implementations can be easily realized. Furthermore, there are more sophisticated methods such as the so-called butterfly transform allowing the set of all Walsh coefficients of a given function to be computed even more efficiently (see Algorithm 9.3 on p. 276 in [41]).

Depending on the implementation, the Walsh transform can be computed very efficiently. A straightforward implementation in C taking the truth table of an (n, n)-function as input needs around 20 seconds to compute the entire Walsh spectrum for $n = 10$; further measurements are provided in Table 5 below. The memory consumption is negligible.

Unfortunately, the extended Walsh spectrum does not do a good job of distinguishing between inequivalent functions. Among all known APN instances F over $\mathbb{F}_{2^n}$ for odd values of n, there are only two possible values of W_F: all known APN functions except for the inverse power function x^{2^n-2} and the Dobbertin power function $x^{2^{4i}+2^{3i}+2^{2i}+2^i-1}$ for $n = 5i$ have the so-called Gold-like spectrum (that is, the same as the Gold function x^3); while the inverse power function and the Dobbertin power function share the same extended Walsh spectrum which is distinct from that of any other known APN function. We note that the Dobbertin power function is only defined for dimensions n that are divisible by 5 (so that the first two odd dimensions for which it is defined are $n = 5$ and $n = 15$), and so its characteristic Walsh spectrum is not reflected in Table 5, which only accounts for the values of the Walsh spectra in the range $6 \leq n \leq 10$.

For $n = 6$ and $n = 10$, we observe a similar partition. More precisely, all known APN functions over $\mathbb{F}_{2^6}$ have a Gold-like spectrum, except for one (function 2.5 from [36]); and all functions over $\mathbb{F}_{2^{10}}$ have a Gold-like spectrum, except for the Dobbertin power function x^{339}. For $n = 8$, the situation is a bit more varied. We know more than 20 000 CCZ-inequivalent APN functions over $\mathbb{F}_{2^8}$ [2], and these have six distinct values of the extended Walsh spectrum. Still, the vast majority of the known APN functions have a Gold-like spectrum; and it is remarkable that the same is true for all known polynomial (as opposed to monomial) APN functions (regardless of the dimension n) that have been classified into infinite families.

3.3 Invariants from associated designs

3.3.1 Definition

An **incidence structure** is a triple $(\mathcal{P}, \mathcal{B}, \mathcal{I})$, where $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$ is a set of points, $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$ is a set of blocks, and $\mathcal{I} \subseteq \mathcal{P} \times \mathcal{B}$ is an incidence relation.

Table 5 Time for computing the extended Walsh spectrum in C

n	6	7	8	9	10	11	12
time (s)	0.023	0.076	0.391	2.863	22.566	171.602	1410.009
total	14	491	21 115	46	21	–	–
values	2	2	6	2	2	–	–

Typically, we assume that the blocks in $\mathcal{B}$ are subsets of $\mathcal{P}$, and the incidence relation $\mathcal{I}$ is set membership. The notion of an incidence structure is quite natural and general; the field of combinatorial design theory studies incidence structures called block designs. We can associate to any incidence structure a so-called **incidence matrix**, which is a binary $m \times n$ matrix M representing the incidence relation $\mathcal{I}$. More precisely, for any i, j in the range $1 \leq i \leq m$ and $1 \leq j \leq n$, the element $M_{i,j}$ on the i-th row and j-th column of M is equal to 1 if $(p_i, b_j) \in \mathcal{I}$; and is equal to 0 otherwise. We refer the reader to [4] or [28] for more background on incidence structures and combinatorial designs.

Given any (n, n)-function F, two designs can be associated with it [36]. In both cases, the set of points is simply $\mathbb{F}_{2^n}^2$, that is, the set of all pairs of elements from the finite field. The first design is denoted by $dev(G_F)$, and its blocks are of the form

$$\{(x + a, F(x) + b) : x \in \mathbb{F}_{2^n}\}$$

for $a, b \in \mathbb{F}_{2^n}$. The second design is denoted by $dev(D_F)$, and its blocks are the sets

$$\{(x + y + a, F(x) + F(y) + b) : x, y \in \mathbb{F}_{2^n}\}$$

for $a, b \in \mathbb{F}_{2^n}$. The rank of the incidence matrix of $dev(G_F)$ is called the Γ**-rank** of F, and the rank of the incidence matrix of $dev(D_F)$ is called the Δ**-rank** of F. The Γ- and Δ-rank are shown to be invariant under CCZ-equivalence, and are two of the currently most widely used invariants in practice.

The orders of the automorphism groups of $dev(G_F)$ and $dev(D_F)$ are also CCZ-invariant, but their computation is only feasible for small dimensions, and so they are not quite as useful as the Γ- and Δ-rank. Nonetheless, these automorphism groups give rise to a CCZ-invariant that can be quite useful in practice: the order of the so-called multiplier group. The **multiplier group** is the subgroup of the automorphism group of $dev(G_F)$ consisting of automorphisms of a special form; its order, denoted by $\mathcal{M}(G_F)$, is invariant under CCZ-equivalence, and can be computed much more efficiently than the order of the full automorphism group. We note that the formal definition of the multiplier group is somewhat technical, and does not contribute anything to the discussion; we omit it, and refer the reader to [36] instead.

3.3.2 Evaluation

The definition of these invariants is conceptually simple, although it does require familiarity with the notion of a combinatorial design and its incidence matrix (and automorphism group, in the case of the $\mathcal{M}(G_F)$). In the case of the Γ- and Δ-rank, the incidence matrix corresponding to $dev(G_F)$ and $dev(D_F)$ can be constructed quite easily. The main issue is the computation of its rank, which requires a sophisticated implementation in order to be efficiently computable in practice. In both cases, we need to compute the rank of a binary $2^{2n} \times 2^{2n}$ matrix (whose construction takes significant time and which occupies a lot of memory, especially for higher dimensions), and a straightforward implementation of say Gaussian elimination is not nearly fast enough. One is thus forced to rely on software libraries or algebra systems (such as *Magma*) that implement efficient algorithms for computing the rank of a matrix; or, otherwise, to implement such highly non-trivial algorithms oneself.

Computing the Γ- and Δ-rank is fairly efficient for small dimensions, but the time complexity grows exponentially; for $n = 10$, computing a single Γ-rank can take more than a week. A summary of the average computation times (in seconds) is given in Table 6. The real bottleneck, however, is the memory consumption; the 500 GB of memory available

Table 6 Computation times (in seconds) for the Γ-rank, the Δ-rank, and the order of $\mathcal{M}(G_F)$

n	6	7	8	9	10
Γ-rank	2	15	138	4229	899024
Δ-rank	2	20	308	6976	–
$\mathcal{M}(G_F)$	0.010	0.120	0.100	11.330	8.710

on our server suffice only for computing Γ-ranks up to dimension 10, and Δ-ranks up to dimension 9.

Despite these shortcomings, the Γ-rank, Δ-rank, and the order $\mathcal{M}(G_F)$ of the multiplier group are rather useful invariants, as they can take on a lot of distinct values for the known functions. Table 7 gives a summary of the number of distinct values that the Γ-rank, Δ-rank, and $\mathcal{M}(G_F)$ can take individually, and the number of distinct combinations of values that they can take on together. Note that computing Δ-ranks for $n \geq 10$ and computing any of the invariants from this section for $n \geq 11$ is currently impossible due to insufficient memory on our server.

The values of the design invariants for the 35 new functions over $\mathbb{F}_{2^9}$ and the 5 new functions over $\mathbb{F}_{2^{10}}$ from [2] are given below in Tables 8 and 9. The functions are indexed in the order in which they appear in the dataset associated to [2]. A complete listing is available online at [1].

3.4 The distance invariant

3.4.1 Definition

A lower bound on the Hamming distance between an APN function F and the closest APN function to it in terms of Hamming distance is shown in [16]. The value of this lower bound is calculated from the minimum value contained in a multiset Π_F of natural numbers that can be associated with any (n, n)-function F. The multiset Π_F is shown to be invariant under CCZ-equivalence in the case of APN functions; that is, if F and G are CCZ-equivalent APN functions, then $\Pi_F = \Pi_G$. One can, however, easily find counterexamples to Π_F being invariant in the case when F and G are not APN.

The multiset Π_F is defined as follows. Let F be an (n, n)-function for some natural number n. For any $b, c \in \mathbb{F}_{2^n}$, we define a set

$$\pi_F(b, c) = \{a \in \mathbb{F}_{2^n} : (\exists x \in \mathbb{F}_{2^n}) F(x) + F(a + x) + F(a + c) = b\}.$$

Table 7 Number of distinct design invariants for some known APN functions

n	no. of functions	Γ-ranks	Δ-ranks	$\mathcal{M}(G_F)$	combinations
6	14	9	3	7	11
7	491	14	6	5	20
8	8192	24	11	10	49
9	46	31	15	8	41
10	16	15	–	7	15

Table 8 Design invariants for the 35 new APN functions over $\mathbb{F}_{2^9}$ [2]

ID	Γ-rank	Δ-rank	$\mathcal{M}(G_F)$	ID	Γ-rank	Δ-rank	$\mathcal{M}(G_F)$
1	48864	940	2560	19	48564	944	3584
2	48860	938	2560	20	48612	936	3584
3	48618	942	3584	21	48624	930	3584
4	48626	944	3584	22	48622	940	3584
5	48602	930	3584	23	48594	944	3584
6	48624	942	3584	24	48564	942	3584
7	48630	944	3584	25	48594	942	3584
8	48610	930	3584	26	48622	944	3584
9	48558	932	3584	27	48610	942	3584
10	48620	944	3584	28	48594	942	3584
11	48602	932	3584	29	48620	944	3584
12	48616	944	3584	30	48554	922	10752
13	48562	942	3584	31	48624	930	10752
14	48578	940	3584	32	48602	940	3584
15	48626	944	3584	33	48596	942	3584
16	48628	940	3584	34	48228	926	75264
17	48602	942	3584	35	48228	926	75264
18	48580	930	3584				

That is, $\pi_F(b, c)$ contains all directions $a \in \mathbb{F}_{2^n}$ for which the derivative $D_a F(x) = F(x + a) + F(x)$ maps to $F(a + c) + b$. The multiset Π_F then consists of the cardinalities of $\pi_F(b, c)$ for all possible choices of $b, c \in \mathbb{F}_{2^n}$; symbolically:

$$\Pi_F = [\#\pi_F(b, c) : b, c \in \mathbb{F}_{2^n}].$$

As indicated above, if F and G are APN and CCZ-equivalent, then we must necessarily have $\Pi_F = \Pi_G$. The lower bound on the Hamming distance mentioned previously can be computed as $\lceil (\min \Pi_F)/3 \rceil + 1$, where $\min \Pi_F$ is the smallest value in Π_F. It is also shown in [16] that the lower bound is not, in general, tight for $n \geq 5$. It is remarkable that while this lower bound is CCZ-invariant, the exact value of the Hamming distance to the closest APN function is not.

If F is quadratic, it is shown that

$$[\#\pi_F(b, c) : b \in \mathbb{F}_{2^n}] = [\#\pi_F(b, c') : b \in \mathbb{F}_{2^n}]$$

Table 9 Design invariants for the 5 new APN functions over $\mathbb{F}_{2^{10}}$ [2]

ID	Γ-rank	$\mathcal{M}(G_F)$
1	163400	31744
2	163398	31744
3	163308	158720
4	164026	31744
5	164026	31744

for any $c, c' \in \mathbb{F}_{2^n}$. It is thus sufficient to compute only the reduced multiset

$$\Pi_F^0 = [\#\pi_F(b, 0) : b \in \mathbb{F}_{2^n}],$$

which then completely determines the full multiset Π_F. Recall that the quadratic case is by far the most practically significant, as almost all known APN instances are quadratic, or at least CCZ-equivalent to quadratic ones. This reduction is then quite valuable, as it allows us to reduce the time for computing this invariant by a factor of 2^n.

3.4.2 Evaluation

Computing Π_F (or Π_F^0) is simple as it only requires summing finite field elements (which can be implemented via the binary XOR operation on most programming languages) and counting how many times a certain value occurs.

The computation time is quite fast, especially in the case of quadratic functions when only the reduced multiset Π_F^0 has to be computed. Table 10 shows some example computation times (in seconds) of a straightforward C implementation for functions over $\mathbb{F}_{2^n}$ for $5 \leq n \leq 11$. The last two columns give the total number of functions on which the invariant was tested and the number of distinct values that Π_F takes over those functions, respectively.

As indicated by Table 10, this invariant is not very effective for distinguishing inequivalent functions for odd dimensions. Indeed, for all odd n in the range $5 \leq n \leq 11$, we observe only two values of Π_F; one is attained by the inverse power function, while all the remaining known APN functions take the second value. This is particularly remarkable for $n = 7$, where we know a large number of CCZ-inequivalent APN instances. Furthermore, as shown in [10, 43], the considered representatives for $n = 5$ and $n = 7$ cover all CCZ-classes of quadratic APN functions over $\mathbb{F}_{2^5}$ and $\mathbb{F}_{2^7}$, respectively; we can thus conclude that all quadratic APN functions over $\mathbb{F}_{2^5}$ and $\mathbb{F}_{2^7}$ have a Gold-like value of Π_F.

In the case of even dimensions, on the other hand, Π_F proves to be quite useful for distinguishing between CCZ-equivalence classes. For $n = 8$, we get 6669 distinct values for the tested 8181 functions from [55]. What is particularly noteworthy, is that instances over even dimensions from the known infinite polynomial (as opposed to monomial) APN families always take the same value of Π_F as the Gold function x^3. All the other values of Π_F that we observe correspond to instances discovered by computational searches that have not been classified into infinite families yet. As in the case of the extended Walsh spectrum, the Dobbertin power function for $n = 5$ has the same value of Π_F as the inverse power function. Thus, the inverse and Dobbertin power functions are currently the only known

Table 10 Computation times and number of distinct values of Π_F for some known APN instances

n	Π_F^0	Π_F	all	values
5	0.002	0.064	3	2
6	0.003	0.192	14	5
7	0.004	0.512	491	2
8	0.004	1.024	8181	6669
9	0.005	2.56	11	2
10	0.031	31.744	21	4
11	0.066	135.168	13	2

infinite families of APN functions to have a value of Π_F distinct from that of the Gold functions.

4 Invariants under EA-equivalence

Despite EA-equivalence being strictly less general than CCZ-equivalence, we know that two quadratic APN functions are CCZ-equivalent if and only if they are EA-equivalent. Since almost all known APN functions are CCZ-equivalent to monomials or to quadratic functions, this makes tests and invariants for EA-equivalence almost as useful in practice as ones for the more general CCZ-equivalence.

4.1 Trivial invariants

Since EA-equivalence is a particular case of CCZ-equivalence, any two functions that are EA-equivalent are also CCZ-equivalent. Consequently, all invariants described in Section 3 remain invariants under EA-equivalence as well.

4.2 Algebraic degree and minimum degree

The algebraic degree of any (n, n)-function is invariant under EA-equivalence since the composition of an arbitrary function F with an affine function does not change the algebraic degree of F. Since the majority of known APN functions are quadratic, and since most computational searches tend to produce quadratic APN functions, the algebraic degree is of somewhat limited use in practice. In particular, we note that APN functions are typically classified up to CCZ-equivalence, and a major reason for the study of the less general EA-equivalence and its associated invariants is due to the fact that the two notions of equivalence coincide for quadratic APN functions; that is, two quadratic APN functions are CCZ-equivalent if and only if they are EA-equivalent. If the functions involved are not of algebraic degree 2, then EA-equivalence no longer coincides with CCZ-equivalence in this sense. However, the algebraic degree can be a useful invariant if we truly need to classify functions up to EA-equivalence (instead of using EA-equivalence as an indirect method for testing CCZ-equivalence as in the case of quadratic APN functions). In particular, the CCZ-class of a quadratic APN function can contain functions of various algebraic degrees, and the latter can be used to distinguish EA-inequivalent functions. The algebraic degree can play a prominent role in theoretical arguments and, in particular, proofs of inequivalence; as a noteworthy example, it was used to show that CCZ-equivalence is strictly more general than EA-equivalence combined with taking inverses [20].

A related invariant is the minimum degree, introduced in [20]. The **minimum degree** of an (n, n)-function F, denoted $\min d^\circ(F)$, is the minimum algebraic degree of a component function of F:

$$\min d^\circ(F) = \min\{\deg(F_b) : 0 \neq b \in \mathbb{F}_{2^n}\}.$$

In general, the minimum degree and algebraic degree of an (n, n)-function can be different. Furthermore, as long as the minimum degree is greater than 1, it is invariant under EA-equivalence. As in the case of the algebraic degree, the minimum degree is not useful for distinguishing between quadratic functions, but can be effective in other contexts. In [20], for instance, the first infinite families of APN and AB functions inequivalent to power functions are constructed, and the minimum degree is used in the proof of this inequivalence.

4.3 Number of subspaces in the set of non-bent components

4.3.1 Definition

Recall that a Boolean $(n, 1)$-function f is called bent if its non-linearity equals $2^{n-1} - 2^{n/2-1}$. Equivalently, f is bent if and only if its Walsh transform satisfies $W_f(a) = \pm 2^{n/2}$ for all $a \in \mathbb{F}_{2^n}$. Given an (n, n)-function F, we can define the set $\mathcal{S}_F$ of those elements $b \in \mathbb{F}_{2^n}$ for which the component function $F_b = \mathrm{Tr}(bF(x))$ is not bent; symbolically, we can write

$$\mathcal{S}_F = \{b \in \mathbb{F}_{2^n} : (\exists a \in \mathbb{F}_{2^n}) W_F(a, b) \neq \pm 2^{n/2}\},$$

or, equivalently,

$$\mathcal{S}_F = \{b \in \mathbb{F}_{2^n} : (\exists a \in \mathbb{F}_{2^n}) W_F(a, b) = 0\}$$

in the case of quadratic F.

If we now denote by $n_F(i)$ the number of i-dimensional linear subspaces contained in $\mathcal{S}_F$, the value $n_F(i)$ is an EA-invariant for any natural number i; that is, if F and G are EA-equivalent, then $n_F(i) = n_G(i)$ for all i [13, 39]. This is easy to see from the fact that if $G = A_1 \circ F \circ A_2 + A$, with all involved functions defined as in (3), then $b \in \mathcal{S}_F$ if and only if $A_1'(b) \in \mathcal{S}_G$, where $A_1'(x)$ is the adjoint operator of $A_1(x) + A_1(0)$. Clearly, if $n_F(i) = 0$ for some i, then $n_F(j) = 0$ for all $j \geq i$ as well. This allows us to compute $\mathcal{S}_F$ and the values of $n_F(i)$ for $i = 1, 2, 3, \ldots$ until an i' is found for which $n_F(i') = 0$, and to use the vector $N_F = (n_F(1), n_F(2), \ldots, n_F(i' - 1))$ for distinguishing between EA-classes.

Note that this invariant is only useful in the case of even dimensions. In the case of odd n, any quadratic APN (n, n)-function is AB, so that we have $\{W_F(a, b) : a \in \mathbb{F}_{2^n}\} = \{0, \pm 2^{(n+1)/2}\}$ for any $0 \neq b \in \mathbb{F}_{2^n}$. Consequently, $\mathcal{S}_F = \mathbb{F}_{2^n}$, and N_F has no distinguishing power.

4.3.2 Evaluation

Implementing N_F as an invariant is simple and easily doable in any general-purpose programming language. As observed in Section 3.2, the Walsh transform can be readily implemented via standard arithmetic and logic operations, and the entire Walsh spectrum of a given function can be computed quite quickly. In this case, the entire Walsh spectrum does not need to be computed (for every $b \in \mathbb{F}_{2^n}$, we can stop computing $W_F(a, b)$ as soon as we find an $a \in \mathbb{F}_{2^n}$ which witnesses that F_b is not bent), so the computation of $\mathcal{S}_F$ is even faster. In particular, if the Walsh spectrum or extended Walsh spectrum has already been computed for the given function (say, in the process of computing some other invariant), $\mathcal{S}_F$ can be determined almost immediately.

Computing the number $n_F(i)$ of subspaces of a given dimension i is the most computationally heavy part of the calculation. The simplest possible approach is to perform an exhaustive search, which is usually sufficiently fast, and does not require anything more complicated than using finite field (or, equivalently, vector space) addition via XOR, and verifying that a set of elements is closed under addition. Furthermore, an algorithm for recovering the subspaces contained in a given set of elements has been developed recently [7], and can be used to further speed up the computations.

The memory consumption is also very modest, although the computation time does increase exponentially with the dimension. Table 11 shows the time (in seconds) for computing N_F for x^3 over $\mathbb{F}_{2^n}$ for all even dimensions n in the range $6 \leq n \leq 12$ on a simple implementation in C.

Table 11 Computation times and number of distinct values for N_F for some known APN instances

n	6	8	10	12
time (s)	0.004	0.168	11.508	1151.773
number of functions	14	8181	21	–
values	6	641	11	–

4.4 The thickness spectrum

4.4.1 Definition

The notion of the thickness of a vector space is introduced in [24]. Given an (n, n)-function F, we begin by finding the set $\mathcal{Z}_F$ of its Walsh zeros, that is, the pairs $(a, b) \in \mathbb{F}_{2^n}^2$ on which the Walsh transform of F evaluates to zero:

$$\mathcal{Z}_F = \{(a, b) : a, b \in \mathbb{F}_{2^n}, W_F(a, b) = 0\}.$$

The **thickness** of any subspace $V \subseteq \mathcal{Z}_F$ is defined as the dimension of its projection onto $\{(0, x) : x \in \mathbb{F}_{2^n}\}$. Equivalently, if V is an n-dimensional space and L is a linear permutation of $\mathbb{F}_{2^{2n}}$ mapping $\{(x, 0) : x \in \mathbb{F}_{2^n}\}$ to V, we can write L in matrix form as

$$L = \begin{bmatrix} a & b \\ c & d \end{bmatrix}.$$

The thickness of V is then the rank of c.

Let i be any natural number, and let us denote by $t_F(i)$ the number of n-dimensional subspaces of $\mathcal{Z}_F$ that have thickness i. The **thickness spectrum** of F is then the vector $T_F = (t_F(1), t_F(2), \ldots, t_F(i))$, where i is the smallest natural number for which $t_F(i) = 0$. As with the number $n_F(i)$ of subspaces in the set of non-bent components, if $t_F(i) = 0$ for some i, then $t_F(j) = 0$ for all $j \geq i$ as well, so the previous definition is justified. The thickness spectrum can then be shown to be invariant under EA-equivalence.

4.4.2 Evaluation

The computation of the thickness spectrum involves computing $\mathcal{Z}_F$ (which essentially involves computing the Walsh spectrum of F), then going through all n-dimensional subspaces of $\mathcal{Z}_F$ and computing the thickness of each subspace. Finding these subspaces can be done using the same approaches used in the computation of N_F discussed in Section 4.3. The rest of the implementation is conceptually simple: after finding the n-dimensional subspaces, it is trivial to project them onto $\{(0, x) : x \in \mathbb{F}_{2^n}\}$ by restricting the coordinates on the left-hand side to zero; and computing the dimension of the projections amounts to counting the number of elements that they contain.

On the other hand, the time for computing T_F is longer than that for computing N_F, since we have to work over a $2n$-dimensional (instead of n-dimensional) vector space. It is intuitively clear that if $k < l < n$ are natural numbers, then finding all k-dimensional subspaces in a set of elements from $\mathbb{F}_2^{2n}$ is easier than finding all l-dimensional subspaces. In the case of N_F, one might restrict to the computation of $n_F(i)$ for only small values of i, and still hope to obtain a useful invariant. In the case of T_F, all n-dimensional subspaces have to be found before any further computations can take place. Table 12 gives a summary of the time (in seconds) needed for computing the thickness spectrum of x^3 over $\mathbb{F}_{2^n}$ with

Table 12 Computation time and number of values of the thickness spectrum for some known APN instances

n	6	7	8	9	10
time (s)	0.86	1.12	0.92	188.174	8.91
total	14	–	8181	46	21
values	8	–	185	40	7

$6 \leq n \leq 10$ using the SboxU library [49]. As we can see from Table 11, the running times for computing N_F and T_F are comparable; however, N_F tends to have more distinguishing power than the thickness spectrum.

4.5 A family of invariants based on zero sums

4.5.1 Definition

An algorithm for computationally testing two given (n, n)-functions F and G for EA-equivalence is developed in [42]. The algorithm is based on computing two multisets, one for F and one for G, which contain some information about the mapping A_1 from (3). More precisely, each element of $\mathbb{F}_{2^n}$ occurs in each of the two multisets with a certain multiplicity, and it is shown that x and $A_1(x)$ must have the same multiplicity for any $x \in \mathbb{F}_{2^n}$ if (3) holds. In particular, the multiplicities of the two associated multisets must be the same, i.e. they are invariant under EA-equivalence.

To make this more precise, we let k be a natural number, and define

$$\Sigma_F^k = [F(x_1) + F(x_2) + \cdots + F(x_k) : x_1, x_2, \ldots, x_k \in \mathbb{F}_{2^n}, x_1 + x_2 + \cdots + x_k = 0];$$

in other words, Σ_F^k is the multiset of the sums of F on all k-tuples of elements adding up to 0. If F and G are EA-equivalent, and k is even, then the multiplicities of Σ_F^k and Σ_G^k are the same. Note that it does not matter whether these k-tuples are taken to be ordered, unordered, or whether we allow repetitions among their elements; all of these variations lead to what is essentially the same invariant.

The smallest possible value for which the invariants make sense is $k = 4$ (for $k = 2$, the multiset Σ_F^2 would consist only of zeros). The multiplicities of Σ_F^k can always be computed via the Walsh transform; if $m_F^k(s)$ denotes the multiplicity of $s \in \mathbb{F}_{2^n}$ in Σ_F^k, then

$$m_F^k(s) = \frac{1}{2^{2n}} \sum_{a,b \in \mathbb{F}_{2^n}} (-1)^{b \cdot s} W_F^k(a, b). \tag{6}$$

This method for computing the multiplicities has the advantage that its time complexity does not depend on the choice of k, which only affects the power to which the Walsh coefficients in (6) have to be raised.

The proof of the above statement can be found in the full version of the conference paper [42], which is under review at *Cryptography and Communications* at the moment;

for the sake of completeness, we describe the basic idea below. The statement follows by straightforward manipulations using the properties of the Walsh transform. We have

$$\sum_{a,b}(-1)^{b\cdot s}W_F^k(a,b) = \sum_{a,b}\sum_{x_1,x_2,\ldots,x_k}(-1)^{b\cdot(F(x_1)+F(x_2)+\cdots+F(x_k)+s)+a\cdot(x_1+x_2+\cdots+a_k)}$$

$$= 2^{2n}\#\{(x_1,x_2,\ldots,x_k)\in\mathbb{F}_{2^n}^k : \sum_{i=1}^k x_i = 0, \sum_{i=1}^k F(x_i) = s\},$$

which then easily implies (6).

It must be noted that in the case of APN functions, the Σ_F^k invariant is essentially the same as the Π_F invariant described in Section 3.4; that is, if F and G are both APN, then $\Pi_F = \Pi_G$ if and only if Σ_F^k and Σ_G^k have the same multiplicities. An advantage of Σ_F^k is that it remains invariant for functions of any differential uniformity, while Π_F is invariant only in the case of APN functions; Σ_F^k also provides information about what the EA-equivalence between the two tested functions might look like, while Π_F does not. On the other hand, Π_F is invariant under CCZ-equivalence (and it is easy to find counterexamples showing that Σ_F^k is not), and provides a lower bound on the distance between F and the closest APN function.

4.5.2 Evaluation

One of the advantages of the algorithm in [42] is that it does not require any complicated mathematical or algorithmic machinery, and can be implemented from first principles (as opposed to the linear code test for CCZ-equivalence). Regardless of whether the computation is done from the definition, or via the Walsh transform, it only requires finite field (or vector space) addition, which can be represented as XOR, and counting multiplicities of elements from the finite field. If the Walsh spectrum of F is available, computing the multiplicities of Σ_F^k via (6) is quite straightforward. For the sake of completeness, we mention also the recently published algorithm for testing the EA-equivalence of quadratic functions [23].

As noted above, Π_F and Σ_F^4 have the same distinguishing power in the case of APN functions. In particular, these invariants are only useful in the case of even dimensions, in which case they can take a lot of distinct values. Once again, representatives from the known infinite families of APN functions (with the notable exception of the inverse power function over odd dimensions) have the same value of Σ_F^4 as the Gold function x^3.

4.6 Ortho-derivatives

4.6.1 Definition

The concept of an ortho-derivative is introduced in [23, 49], and appears to be quite useful for partitioning quadratic APN functions into EA-equivalence classes. Given a quadratic (n,n)-function F, an **ortho-derivative** of F is any (n,n)-function π_F such that

$$\pi_F(a)\cdot(F(x)+F(a+x)+F(a)+F(0)) = 0 \tag{7}$$

for all $x\in\mathbb{F}_{2^n}$. In this sense, π_F is orthogonal to all values of the expression $F(x+a)+F(x)+F(a)+F(0) = D_aF(x)+D_aF(0)$. Note that ortho-derivatives can be defined for any (n,n)-function F; however, F is APN if and only if $\pi_F(a)$ is uniquely defined for all non-zero $a\in\mathbb{F}_{2^n}$. Thus, in the APN case, a unique (n,n)-function π_F can be associated

with any quadratic APN (n, n)-function F. Furthermore, it is known that if F and G are two EA-equivalent (n, n)-functions via $A_1 \circ F \circ A_2 + A = G$, then

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

thus, the ortho-derivatives of two EA-equivalent APN functions are EA-equivalent themselves. The advantage is that the ortho-derivatives have a much more "varied" structure than the APN functions that they are associated with; as pointed out in [23], it seems that the algebraic degree of π_F is $n-2$ whenever F is an APN (n, n)-function. This is intuitively one of the reasons that the ortho-derivatives can take on many different values for the various invariants under EA-equivalence and CCZ-equivalence, and the latter become immensely more useful for partitioning functions into EA-classes.

4.6.2 Evaluation

Computing the truth table of the ortho-derivative itself is a very easy task; the simplest way involves guessing the value of $\pi_F(a)$ for each $0 \neq a \in \mathbb{F}_{2^n}$. For every possible candidate for the value of $\pi_F(a)$, it suffices to verify that (7) is satisfied. Computing (7), on other hand, requires nothing more than vector space addition and the implementation of the scalar product. The truth table of π_F can thus be reconstructed very quickly, even for relatively large dimensions. Table 13 gives some sample running times (in seconds) for computing the ortho-derivative of x^3 using SboxU. The row labeled "functions" gives the number of functions that were tested (all of them being CCZ-inequivalent to each other); while "values" gives the number of distinct values obtained across all of these functions. The set of functions used for the tests consists of all known quadratic APN functions over $\mathbb{F}_{2^n}$ for the appropriate value of n. We note that there are only two known cases in which CCZ-inequivalent functions can have an orthoderivative with the same differential spectrum and the same Walsh spectrum: these are the Gold functions x^3 and x^9 over $\mathbb{F}_{2^7}$, and the Gold functions x^3 and x^{33} over $\mathbb{F}_{2^9}$. The CCZ-classes of all other known quadratic APN functions (including the other Gold functions, e.g. x^5, x^9 and x^{17} over $\mathbb{F}_{2^9}$) can be distinguished with the help of orthoderivatives in this way.

Once the ortho-derivative is computed, it remains to apply some of the previously examined invariants, such as the Walsh spectrum and differential spectrum, to distinguish between the EA-classes of the ortho-derivatives. The number of values in Table 13 refers to the number of distinct combinations of the Walsh spectrum and differential spectrum observed among the ortho-derivatives. Note that the ortho-derivatives themselves are not APN, and so the differential spectrum becomes a useful invariant. In fact, as indicated by the data in Table 13, this is sufficient to distinguish between all currently known classes of quadratic APN functions (with the exception of some Gold functions). Thus, a partitioning by means of the ortho-derivatives appears to have the same strength in practice as an

Table 13 Computation time for finding the truth table of π_F, and number of values of the Walsh and differential spectra of π_F for some known APN instances

n	6	7	8	9	10	11	12
time (s)	0	0.00021	0.0006	0.002	0.34	0.22	0.84
functions	14	488	21 103	42	19	–	–
values	14	487	21 103	41	19	–	–

EA-equivalence test. Nonetheless, invariants can only be used to disprove the equivalence between two functions; and so, if two (n, n)-functions F and G have the same invariants for π_F and π_G, this does not constitute a proof of their EA-equivalence. It would thus be very interesting to find more examples of EA-inequivalent functions whose ortho-derivatives have the same spectrum of invariant values.

Despite this, as indicated by Table 13, the combination of the extended Walsh spectrum and the differential spectrum of the ortho-derivative is sufficient to distinguish between any pair of EA-inequivalent quadratic APN functions (with the exception of the Gold functions).

5 Conclusion

We have surveyed some known invariants on (n, n)-functions under CCZ- and EA-equivalence. The list of invariants is not meant to be complete, but aims to include all invariants that are commonly used in the classification of APN and AB functions in practice. Each of the invariants is evaluated in terms of how easy it is to implement in a general-purpose programming language, how efficient it is to compute, and how well it can distinguish between distinct equivalence classes of (n, n)-functions.

The considered CCZ-invariants include:

- the differential uniformity and non-linearity (trivial);
- the extended Walsh spectrum $\mathcal{W}_F$;
- the invariants from the associated combinatorial designs $dev(G_F)$ and $dev(D_F)$, viz. the Γ-rank, Δ-rank, and order of the multiplier group, as well as the orders of the automorphism groups of $dev(G_F)$ and $dev(D_F)$;
- the multiset Π_F used in the computation of the lower bound on the Hamming distance between two APN functions (invariant only for APN functions).

The considered EA-invariants include:

- all CCZ-invariants;
- the algebraic degree $\deg(F)$ and the minimum degree $\min d^\circ(F)$;
- the number $n_F(i)$ of i-dimensional subspaces in the set $\mathcal{S}_F$ of non-bent components of F;
- the number $t_F(i)$ of n-dimensional subspaces of thickness i in the set $\mathcal{Z}_F$ of Walsh zeros of F;
- the multiplicities of the elements in the multiset Σ_F^k for all even k, i.e. the number of times that each element of $\mathbb{F}_{2^n}$ can be expressed as a sum $F(x_1) + F(x_2) + \cdots + F(x_k)$ with $x_1 + x_2 + \cdots + x_k = 0$;
- the ortho-derivatives π_F (or, to be more precise, the EA-equivalence class of the ortho-derivative).

For each invariant, we have used a straightforward implementation in a suitable programming language (typically C in the case of the invariants that can be defined and computed from first principles, and *Magma* when more sophisticated mathematical structures or algorithms are involved) and have given example running times over the range of dimensions n in which the invariant can be reasonably used in practice. We have also counted how many different values each invariant can take over some of the known APN instances in each dimension, and have remarked on the further properties and significance of some of the invariants.

Acknowledgements The research presented in this paper was supported by the Trond Mohn Foundation.

I would like to express my gratitude to the anonymous reviewers for their thorough proofreading and helpful comments. I would also like to thank Lilya Budaghyan and Christof Beierle for various useful remarks and discussions.

Funding Open access funding provided by University of Bergen (incl Haukeland University Hospital).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Encyclopedia of Boolean functions. http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.ht://boolean.h.uib.no/mediawiki/index.php/Main_Page
2. Beierle, C., Leander, G.: New instances of quadratic APN functions. arXiv:2009.07204 (2020)
3. Thomas, B., Cunsheng, D.: On almost perfect nonlinear permutations. In: Workshop on the Theory and Application of of Cryptographic Techniques, pp. 65–76. Springer (1993)
4. Beth, T., Jungnickel, D., Lenz, H.: Design Theory, vol. 1. Cambridge, Cambridge University Press (1999)
5. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. J. Cryptol. **4**(1), 3–72 (1991)
6. Biryukov, A., De Canniere, C., Braeken, A., Preneel, B.: A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In: International Conference on the Theory and Applications of Cryptographic Techniques, pp. 33–50. Springer (2003)
7. Bonnetain, X., Perrin, L., Tian, S.: Anomalies and vector space search: Tools for s-box analysis. In: International Conference on the Theory and Application of Cryptology and Information Security. Springer (2019)
8. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system I: The user language. J. Symb. Comput. **24**(3-4), 235–265 (1997)
9. Bracken, C., Byrne, E., Markin, N., McGuire, G.: A few more quadratic APN functions. Cryptogr. Commun. **3**(1), 43–53 (2011)
10. Brinkmann, M., Leander, G.: On the classification of APN functions up to dimension five. Des. Codes Crypt. **49**, 273–288 (2008)
11. Browning, K.: APN polynomials and related codes. Special Vol. J. Comb. Inf. Syst. Sci. **34**, 135–159 (2009)
12. Browning, K.A., Dillon, J.F., McQuistan, M.T., Wolfe, A.J.: An APN permutation in dimension six. Finite Fields: Theory Appl. **518**, 33–42 (2010)
13. Budaghyan, L., Calderini, M., Carlet, C., Coulter, R., Villa, I.: Generalized isotopic shift construction for APN functions Des. Codes Crypt. **89**, 1–14 (2020)
14. Budaghyan, L., Calderini, M., Carlet, C., Coulter, R.S., Villa, I.: Constructing APN functions through isotopic shifts. IEEE Trans. Inf. Theory **66**(8), 5299–5309 (2020)
15. Budaghyan, L., Carlet, C.: Classes of quadratic APN trinomials and hexanomials and related structures. IEEE Trans. Inf. Theory **54**(5), 2354–2357 (2008)
16. Budaghyan, L., Carlet, C., Helleseth, T., Kaleyski, N.: On the distance between APN functions. IEEE Trans. Inf. Theory **66**(9), 5742–5753 (2020)
17. Budaghyan, L., Carlet, C., Leander, G.: Two classes of quadratic APN binomials inequivalent to power functions. IEEE Trans. Inf. Theory **54**(9), 4218–4229 (2008)
18. Budaghyan, L., Carlet, C., Leander, G.: Constructing new APN functions from known ones. Finite Fields Their Appl **15**(2), 150–159 (2009)
19. Budaghyan, L., Carlet, C., Leander, G.: On a construction of quadratic APN functions. In: 2009 IEEE Information Theory Workshop. pp 374–378 (2009)

20. Budaghyan, L., Carlet, C., Pott, A.: New classes of almost bent and almost perfect nonlinear polynomials. IEEE Trans. Inf. Theory **52**(3), 1141–1152 (2006)
21. Budaghyan, L., Helleseth, T., Kaleyski, N.: A new family of APN quadrinomials. IEEE Trans. Inf. Theory **66**(11), 7081–7087 (2020)
22. Budaghyan, L., Kazymyrov, O.: Verification of restricted ea-equivalence for vectorial boolean functions. In: International Workshop on the Arithmetic of Finite Fields, pp. 108–118. Springer (2012)
23. Canteaut, A., Couvreur, A., Perrin, L.: Recovering or testing extended-affine equivalence. arXiv:2103.00078 (2021)
24. Canteaut, A., Perrin, L.: On CCZ-equivalence, extended-affine equivalence, and function twisting. Finite Fields Their Appl. **56**, 209–246 (2019)
25. Carlet, C.: Boolean functions for cryptography and coding theory (2021)
26. Carlet, C., Charpin, P., Zinoviev, V.A.: Codes, bent functions and permutations suitable for DES-like cryptosystems. Des. Codes Crypt. **15**(2), 125–156 (1998)
27. Chabaud, F., Vaudenay, S.: Links between differential and linear cryptanalysis. In: Workshop on the Theory and Application of Cryptographic Techniques, EUROCRYPT 94, vol. 950, pp. 356–365 (1994)
28. Colbourn, C.J., Dinitz, J.H.: Handbook of combinatorial designs. CRC press (2006)
29. Daemen, J., Rijmen, V.: AES proposal: Rijndael (1999)
30. Daemen, J., Rijmen, V.: The design of Rijndael, vol. 2. Springer, Berlin (2002)
31. Dempwolff, U.: Ccz equivalence of power functions. Des. Codes Crypt. **86**(3), 665–692 (2018)
32. Dobbertin, H.: Almost perfect nonlinear power functions on $GF(2^n)$: the Niho case. Inf. Comput. **151**(1), 57–72 (1999)
33. Dobbertin, H.: Almost perfect nonlinear power functions on $GF(2^n)$: the Welch case. IEEE Trans. Inf. Theory **45**(4), 1271–1275 (1999)
34. Dobbertin, H.: Almost perfect nonlinear power functions on $GF(2^n)$: A new case for n divisible by 5. Int. Conf. Finite Fields Appl. 113–121 (2001)
35. Edel, Y.: Quadratic APN functions as subspaces of alternating bilinear forms. In: Proceedings of the Contact Forum Coding Theory and Cryptography III, vol. 2009, pp. 11–24 (2011)
36. Edel, Y., Pott, A.: A new almost perfect nonlinear function which is not quadratic. Adv. Math. Commun. **3**(1), 59–81 (2009)
37. Edel, Y., Pott, A.: On the equivalence of nonlinear functions (2009)
38. Gold, R.: Maximal recursive sequences with 3-valued recursive cross-correlation functions (corresp.) IEEE Trans. Inf. Theory **14**(1), 154–156 (1968)
39. Göloğlu, F., Pavlů, J.: On CCZ-inequivalence of some families of almost perfect nonlinear functions to permutations. Cryptogr. Commun. 1–15 (2021)
40. Janwa, H., Wilson, R.M.: Hyperplane sections of Fermat varieties in P^3 in char. 2 and some applications to cyclic codes. In: International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes, pp. 180–194. Springer (1993)
41. Joux, A.: Algorithmic cryptanalysis. CRC press (2009)
42. Kaleyski, N.: Deciding ea-equivalence via invariants. In: The 11th International Conference on Sequences and Their Applications (SETA 2020) (2020)
43. Kalgin, K., Idrisova, V.: The classification of quadratic APN functions in 7 variables (2020)
44. Kasami, T.: The weight enumerators for several classes of subcodes of the 2nd order binary Reed-Muller codes. Inf. Comput. **18**(4), 369–394 (1971)
45. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Workshop on the Theory and Application of of Cryptographic Techniques, pp. 386–397. Springer (1993)
46. Nyberg, K.: Differentially uniform mappings for cryptography. Lect. Notes Comput. Sci **765**, 55–64 (1994)
47. Nyberg, K., Knudsen, L.R.: Provable security against a differential attack. J. Cryptol. **8**(1), 27–37 (1995)
48. Özbudak, F., Sınak, A., Yayla, O.: On verification of restricted extended affine equivalence of vectorial boolean functions. In: International Workshop on the Arithmetic of Finite Fields, pp. 137–154. Springer (2014)
49. Perrin, L.: How to take a function apart with sboxu. The 5th International Workshop on Boolean Functions and their Applications (BFA 2020) (2020)
50. Sălăgean, A.: Discrete antiderivatives for functions over $\mathbb{F}_p^n$. Des. Codes Crypt. **88**(3), 471–486 (2020)
51. Sidelnikov, V.M.: On the mutual correlation of sequences. Soviet Math. Dokl. **12**, 197–201 (1971)
52. Taniguchi, H.: On some quadratic APN functions. Des. Codes Crypt. **87**, 1–11 (2019)
53. Yoshiara, S.: Equivalences of quadratic APN functions. J. Algebraic Comb. **35**(3), 461–475 (2012)
54. Yoshiara, S.: Equivalences of power APN functions with power or quadratic APN functions. J. Algebraic Comb. **44**(3), 561–585 (2016)

55. Yu, Y., Wang, M., Li, Y.: A matrix approach for constructing quadratic APN functions (2013)
56. Yuyin, Y., Wang, M., Li, Y.: A matrix approach for constructing quadratic APN functions. Des. Codes Cryptogr. **73**(2), 587–600 (2014)
57. Zhou, Y., Pott, A.: A new family of semifields with 2 parameters. Adv. Math. **234**, 43–60 (2013)

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.