Cryptography and Communications (2022) 14:271–290
https://doi.org/10.1007/s12095-021-00513-y

Deciding EA-equivalence via invariants

Nikolay Kaleyski[1]

Received: 1 December 2020 / Accepted: 3 July 2021 / Published online: 27 July 2021

Abstract
We define a family of efficiently computable invariants for (n, m)-functions under EA-equivalence, and observe that, unlike the known invariants such as the differential spectrum, algebraic degree, and extended Walsh spectrum, in the case of quadratic APN functions over $\mathbb{F}_{2^n}$ with n even, these invariants take on many different values for functions belonging to distinct equivalence classes. We show how the values of these invariants can be used constructively to implement a test for EA-equivalence of functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$; to the best of our knowledge, this is the first algorithm for deciding EA-equivalence without resorting to testing the equivalence of associated linear codes.

Keywords Vectorial Boolean functions · Almost perfect nonlinear (APN) functions · Extended affine equivalence (EAequivalence) · CCZ-equivalence

Mathematics Subject Classification 2010 11T06 · 94C10 · 94A60

1 Introduction

Let $\mathbb{F}_{2^n}$ denote the finite field with 2^n elements for some positive integer n, and let $\mathbb{F}_{2^n}^*$ denote its multiplicative group. The vector space of dimension n over $\mathbb{F}_2$ will be denoted by $\mathbb{F}_2^n$. An (n, m)-*function*, or *vectorial Boolean function*, is any mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ or, equivalently, from $\mathbb{F}_{2^n}$ to $\mathbb{F}_{2^m}$. We typically assume that $m = n$, i.e. we concentrate on functions from a finite field of characteristic two to itself; however, the approach given in the present paper can be applied to an arbitrary pair of dimensions (n, m).

In the particular case of $n = m$, we can conveniently represent (n, n)-functions by univariate polynomials over $\mathbb{F}_{2^n}$; more precisely, any (n, n)-function F can be written as $F(x) = \sum_{i=0}^{2^n-1} c_i x^i$ for some coefficients $c_i \in \mathbb{F}_{2^n}$. This form, called the *univariate representation* of F, always exists, and is unique. In the general case, any

This article belongs to the Topical Collection: *Sequences and Their Applications III*
Guest Editors: Chunlei Li, Tor Helleseth and Zhengchun Zhou

Some of the results in this paper for the particular case of $n = m$ have been partially presented at Sequences and Their Applications (SETA) 2020.

✉ Nikolay Kaleyski
nikolay.kaleyski@uib.no

[1] Department of Informatics, University of Bergen, Bergen, Norway

(n, m)-function F can be represented uniquely as a multivariate polynomial of the form $F(x) = \sum_{u \in \mathbb{F}_2^n} a_u \prod_{i=1}^n x_i^{u_i}$ for $a_u \in \mathbb{F}_2^m$, where v_i denotes the i-th component of the vector $v = (v_1, v_2, \ldots, v_n) \in \mathbb{F}_2^n$. We will not concern ourselves too deeply with the choice of representation here, as it is typically only important when defining and classifying (n, m)-functions. However, when illustrating some of the concepts of the EA-equivalence test with examples, we will mostly use (n, n)-functions and the univariate representation.

As suggested above, the finite field $\mathbb{F}_{2^n}$ can be identified with the vector space $\mathbb{F}_2^n$, and so the elements of $\mathbb{F}_{2^n}$ can be identified with binary vectors of n bits; thus, (n, m)-functions can be understood as transformations that take an n-bit sequence as input, and produce an m-bit sequence as output. Thanks to this interpretation, vectorial Boolean functions naturally find applications in the theory and practice of various areas of mathematics and computer science. In particular, vectorial Boolean functions are used in modern block ciphers in the role of so-called "S-boxes", or "substitution boxes", and typically constitute the only non-linear part of the cipher. Consequently, the security of the cipher directly depends on the properties of the underlying S-boxes, which motivates the study of vectorial Boolean functions with respect to their cryptographic properties. It is also intuitively clear that (n, n)-functions constitute one of the most practically significant cases, since in cryptography one typically wants to replace a bit sequence with a different bit sequence of the same length.

A basic property of any (n, m)-function, which also has cryptographic implications, is its algebraic degree. Given an (n, m)-function with ANF $F(x) = \sum_{u \in \mathbb{F}_2^n} u \prod_{i=1}^n x_i^{u_i}$, the *algebraic degree* of F is defined as the largest degree of any term $x_i^{u_i}$ that has a non-zero coefficient a_u. In other words, the algebraic degree of F is the multivariate degree of its ANF. Thus, for instance, $F(x) = x_1x_2x_4 + x_2x_3$ would have algebraic degree 3. In the case of an (n, n)-function given by its univariate representation, the algebraic degree also has a natural interpretation. Given a positive integer i, the *binary weight* of i is the number of non-zero entries in its binary representation; for example, 19 is written as 10011 in binary, and hence has binary weight 3. The *algebraic degree* of $F(x) = \sum_{i=0}^{2^n-1} c_i x^i$ is the largest binary weight of any exponent i with $c_i \neq 0$. Functions of algebraic degree 1, 2, and 3, are called *affine*, *quadratic*, and *cubic*, respectively. An affine function A with $A(0) = 0$ is called *linear*. An affine (n, n)-function A satisfies $A(x) + A(y) + A(z) = A(x + y + z)$ for any $x, y, z \in \mathbb{F}_{2^n}$; similarly, a linear (n, n)-function L satisfies $L(x) + L(y) = L(x + y)$ for any $x, y \in \mathbb{F}_{2^n}$. It is desirable for vectorial Boolean functions used as S-boxes to have a high algebraic degree, since the latter indicates a good resistance to higher-order differential attacks [10, 14].

Two of the most important cryptographic properties of (n, m)-functions are the differential uniformity and the nonlinearity. Suppose that F is an (n, m)-function for some positive integers n and m. Let $\delta_F(a, b)$ denote the number of solutions $x \in \mathbb{F}_2^n$ to the equation $F(x + a) + F(x) = b$ for any $0 \neq a \in \mathbb{F}_2^n$ and any $b \in \mathbb{F}_2^m$. The multiset $\{\delta_F(a, b) : 0 \neq a \in \mathbb{F}_2^n, b \in \mathbb{F}_2^m\}$ is called the *differential spectrum* of F. The largest value in the differential spectrum is denoted by δ_F and is called the *differential uniformity* of F.

The existence of a large number of solutions x to some equation of the form $F(x)+F(a+x)=b$ for some $a \neq 0$ and b makes the function F vulnerable to differential cryptanalysis [2]. The value of δ_F should thus be as low as possible. Since $x+a$ is a solution to the aforementioned equation whenever x is, the minimum possible value of δ_F is 2; the class of (n, n)-functions attaining this optimal value is called the class of almost perfect nonlinear (APN), and has been an object of intense study since its introduction by Nyberg in the 90's [16].

The *nonlinearity* $\mathcal{NL}(F)$ of F is simply the minimum Hamming distance between any component function of F and any affine $(n, 1)$-function, the *component functions* of F being

the $(n, 1)$-functions F_c of the form $F_c(x) = \mathrm{Tr}_m(cF(x))$, where $\mathrm{Tr}_m : \mathbb{F}_{2^m} \to \mathbb{F}_2$ is the *absolute trace* defined by $\mathrm{Tr}_m(x) = \sum_{i=0}^{m-1} x^{2^i}$ for any $x \in \mathbb{F}_{2^n}$; when the dimension m is clear from context, we will sometimes write just Tr instead of Tr_m. The nonlinearity should be high in order to resist linear cryptanalysis [15].

When studying "linear properties" of functions, such as their nonlinearity, it is useful to adapt some linear-algebraic notions from the vector space $\mathbb{F}_2^n$ even in the case when we are working with the finite field $\mathbb{F}_{2^n}$. The *linear span* of a set $S \subseteq \mathbb{F}_{2^n}$ is simply the set of all possible linear combinations of the elements in S, i.e. if $S = \{s_1, s_2, \ldots, s_k\}$ for some positive integer k and for $s_i \in \mathbb{F}_{2^n}$, then $\mathrm{Span}(S) = \{c_1 s_1 + c_2 s_2 + \cdots + c_k s_k : c_1, c_2, \ldots, c_k \in \mathbb{F}_2\}$; obviously, the span can be defined in the same way in the case of the vector space $\mathbb{F}_2^n$. Having formalized the linear span, it is straightforward to carry over other notions from $\mathbb{F}_2^n$, such as that of linear independence, and that of a basis of $\mathbb{F}_{2^n}$ (being a linearly independent set $B \subseteq \mathbb{F}_{2^n}$ with $\mathrm{Span}(B) = \mathbb{F}_{2^n}$).

A useful tool for analyzing vectorial Boolean functions is the *Walsh transform*, which is an integer valued function $W_F : \mathbb{F}_2^n \times \mathbb{F}_2^m \to \mathbb{Z}$ associated with $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$, and given by the prescription

$$W_F(a, b) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\mathrm{Tr}_m(bF(x)) + \mathrm{Tr}_n(ax)}$$

for $a \in \mathbb{F}_2^n$, $b \in \mathbb{F}_2^m$. In the case of (n, n)-functions, we can more succinctly write

$$W_F(a, b) = \sum_{x \in \mathbb{F}_{2^n}} \chi(bF(x) + ax),$$

where $\chi(x) = (-1)^{\mathrm{Tr}(x)}$. The values of W_F are called *Walsh coefficients*, and the multiset $\{W_F(a, b) : a \in \mathbb{F}_2^n, b \in \mathbb{F}_2^m\}$ is called the *Walsh spectrum* of F. The multiset of the absolute values of F, i.e. the multiset $\{|W_F(a, b)| : a \in \mathbb{F}_2^n, b \in \mathbb{F}_2^m\}$, is called the *extended Walsh spectrum* of F.

Due to the huge number of (n, m)-functions even for small values of n and m, their classification is typically only performed up to some equivalence relation that preserves the properties being studied. In the case of cryptographically optimal vectorial Boolean functions, the most general equivalence relation preserving both the differential uniformity and the nonlinearity is the so-called Carlet-Charpin-Zinoviev-equivalence, or CCZ-equivalence [9]. Two (n, m)-functions F and G are said to be *CCZ-equivalent* if there is an affine permutation $\mathcal{A}$ of $\mathbb{F}_2^n \times \mathbb{F}_2^m$ mapping the graph $\Gamma_F = \{(x, F(x)) : x \in \mathbb{F}_2^n\}$ of F to the graph Γ_G of G.

Testing whether two given functions F and G are CCZ-equivalent is usually done by means of the equivalence of linear codes [5, 13]. More precisely, a particular linear code $\mathcal{C}_F$ is associated with F, and a particular linear code $\mathcal{C}_G$ is associated with G; F and G are then CCZ-equivalent if and only if $\mathcal{C}_F$ and $\mathcal{C}_G$ are equivalent. Testing whether two given linear codes are equivalent has the advantage that it is usually already implemented in most mathematical software, such as the *Magma* programming language that we use for most of our computations [4].

Unfortunately, computationally testing CCZ-equivalence in this way can reliably be performed only when the dimensions m and n are relatively small; in the case of (n, n)-functions, this means $n \leq 9$, since for higher values of n, the memory consumption becomes overwhelming, and the test cannot be performed in a lot of cases. Furthermore, the current implementation of *Magma* can give false negatives due to insufficient memory; in other words, if the equivalence test outputs "false", we have no reliable way of determining

whether this is due to insufficient memory, or due to a successfully completed exhaustive search proving the inequivalence of the linear codes (and hence, vectorial Boolean functions) in question.

Ruling out e.g. CCZ-equivalence can be facilitated by means of invariants, i.e. properties or statistics that are preserved under CCZ-equivalence. The differential spectrum and the extended Walsh spectrum are invariant under CCZ-equivalence, i.e. if two (n, m)-functions F and G are CCZ-equivalent, then their differential spectra and extended Walsh spectra are the same. Unfortunately, the Walsh spectra and differential spectra of all known APN functions are the same (with some rare exceptions in the case of the Walsh spectrum), rendering these invariants nearly useless in practice. Other invariants, such as the Γ-rank and Δ-rank have been introduced [12], that can take different values for distinct CCZ-classes of functions, and can therefore be used to rule out CCZ-equivalence in some cases. The major drawback of these invariants is that they require significant computational resources, meaning that, on the one hand, their calculation takes a long time, e.g. around ten days for a single Γ-rank over $\mathbb{F}_{2^{10}}$, and, on the other hand, computing these invariants for $\mathbb{F}_{2^n}$ with $n > 10$ is impossible at the moment due to overwhelming memory requirements.

A special case of CCZ-equivalence is extended affine equivalence, or EA-equivalence. Two (n, m)-functions F and G are said to be *EA-equivalent* if there exist affine functions $A_1 : \mathbb{F}_2^m \to \mathbb{F}_2^m$, $A_2 : \mathbb{F}_2^n \to \mathbb{F}_2^n$, and $A : \mathbb{F}_2^n \to \mathbb{F}_2^m$ such that

$$A_1 \circ F \circ A_2 + A = G, \tag{1}$$

with A_1 and A_2 being bijective. We will refer to A_1 as the *outer permutation* and to A_2 as the *inner permutation* throughout the paper. CCZ-equivalence is strictly more general than EA-equivalence combined with taking inverses [7], but in certain cases, such as for quadratic and monomial functions, checking whether two functions (or, potentially, their inverses) are EA-equivalent is enough to decided CCZ-equivalence: two quadratic APN functions are CCZ-equivalent if and only if they are EA-equivalent [18]; and two power functions are CCZ-equivalent if and only if they are cyclotomic equivalent [11]. Recall that two power functions $F(x) = x^d$ and $G(x) = x^e$ over $\mathbb{F}_{2^n}$ are said to be cyclotomic equivalent if $e \equiv 2^k d \mod (2^n - 1)$ or $e^{-1} \equiv 2^k d \mod (2^n - 1)$; furthermore, cyclotomic equivalence is a special case of EA-equivalence and taking inverses. This is particularly interesting when one takes into account that all known APN functions (which fall into more than 20000 distinct CCZ-equivalence classes) are CCZ-equivalent to a monomial or quadratic function, with only a single exception in dimension $n = 6$. Thus, from a practical point of view, being able to test functions for EA-equivalence is virtually as useful as being able to test them for CCZ-equivalence, at least as far as the classification of APN functions is concerned.

Surprisingly, despite its simple definition, the only known algorithm to date for computationally testing the EA-equivalence of two given functions is one by means of associated linear codes, much like in the case of CCZ-equivalence [13]; the associated codes used in this approach are of a somewhat more complicated form than the ones used in the CCZ-equivalence test, and so this approach is even more restrictive with respect to computational resources and memory requirements. Indeed, testing EA-equivalence for quadratic functions (which coincides with CCZ-equivalence) is typically done by testing them for CCZ-equivalence. Algorithms for testing the EA-equivalence of two functions in some other special cases (grouped under the umbrella term "restricted EA-equivalence") have previously been studied in [3], [8], and [17].

Since EA-equivalence is a special case of CCZ-equivalence, any CCZ-invariant is also an EA-invariant. As mentioned above, the extended Walsh spectrum and differential spectrum are practically useless in the case of APN functions, as they almost always take the same

value in the APN case, while the Γ- and Δ-rank involve somewhat laborious computations. EA-equivalence being less general than CCZ-equivalence, it is natural to expect to have properties that are EA-invariant but not CCZ-invariant. One such property is the algebraic degree, which is preserved by EA-equivalence, but not by CCZ-equivalence. Unfortunately, this is not terribly useful for classifying APN function either since, as mentioned above, nearly all known instances of APN functions are quadratic.

In this paper, we present an approach for computationally testing the EA-equivalence of two (n, m)-functions by first guessing the outer permutation A_1, applying its inverse to (1) to obtain a relation of the form $F \circ A_2 + A' = G'$, and then solving the latter for A_2 and A'. In the case of (n, n)-functions with n even, our approach allows the set of possible affine permutations A_1 to be drastically reduced (as opposed to exhaustive search), which makes the entire procedure computationally feasible. Our approach has the advantage that it can be broken down into a multitude of small independent steps, which makes the resulting algorithm easily parallelizable. Unlike the CCZ-equivalence test and EA-equivalence test described in [13], which rely on testing the equivalence of a pair of linear codes (and therefore require specialized and rather complex algorithms), our approach uses only basic arithmetics and linear algebra, and can be easily implemented in any general-purpose programming language, and ran on any computer. Furthermore, each of the individual steps comprising the algorithm has a concrete and meaningful input and output that can be monitored and verified. This precludes the possibility of false positives or negatives as in the case of the current CCZ-equivalence test.

2 A family of EA-invariants

Let m, n, k be positive integers, and t be an element of $\mathbb{F}_2^n$. We denote by $\mathcal{T}_k(t)$ the set of all k-tuples of elements from $\mathbb{F}_2^n$ that add up to t, i.e.

$$\mathcal{T}_k(t) = \{(x_1, x_2, \ldots, x_k) \in (\mathbb{F}_2^n)^k \mid \sum_{i=1}^{k} x_i = t\}.$$

If A is an affine (n, n)-permutation, then the image of any k-tuple $(x_1, x_2, \ldots, x_k)$ from $\mathcal{T}_k(t)$ is a k-tuple $(A(x_1), A(x_2), \ldots, A(x_k))$, the sum of whose elements is

$$A(x_1) + A(x_2) + \cdots + A(x_k) = \begin{cases} A(x_1 + x_2 + \cdots + x_k) & k \text{ odd;} \\ A(x_1 + x_2 + \cdots + x_k) + A(0) & k \text{ even.} \end{cases}$$

Equivalently, A is a one-to-one mapping from $\mathcal{T}_k(t)$ to $\mathcal{T}_k(t')$, where $t' = A(t)$ when k is odd, and $t' = A(t) + A(0)$ when k is even. In particular, a linear A always permutes $\mathcal{T}_k(0)$.

For any (n, m)-function F, let $\Sigma_k^F(t)$ denote the multiset of all sums of the form $F(x_1) + F(x_2) + \cdots + F(x_k)$ for all k-tuples $(x_1, x_2, \ldots, x_k) \in \mathcal{T}_k(t)$. Symbolically:

$$\Sigma_k^F(t) = \left\{ \sum_{i=1}^{k} F(x_i) : (x_1, x_2, \ldots, x_k) \in \mathcal{T}_k(t) \right\}.$$

The multiplicities of $\Sigma_k^F(0)$ are then an EA-invariant for any even value of k.

Proposition 1 *Let F and G be (n, m)-functions with $A_1 \circ F \circ A_2 + A = G$ for some affine functions $A_1 : \mathbb{F}_2^m \to \mathbb{F}_2^m$, $A_2 : \mathbb{F}_2^n \to \mathbb{F}_2^n$, $A : \mathbb{F}_2^n \to \mathbb{F}_2^m$ with A_1, A_2 bijective. Let k be a*

positive integer. Then

$$\Sigma_k^G(0) = \begin{cases} \{A_1(s) + A(0) : s \in \Sigma_k^F(A_2(0))\} & k \text{ odd}; \\ \{A_1(s) + A_1(0) : s \in \Sigma_k^F(0)\} & k \text{ even}. \end{cases}$$

In particular, the multiplicities of $\Sigma_k^F(0)$ *and* $\Sigma_k^G(0)$ *(that is, the number of times that each element occurs in each multiset) are the same when k is even.*

Proof Consider a k-tuple $(x_1, x_2, \ldots, x_k)$ with $x_1+x_2+\cdots+x_k = 0$. The sum of the images of x_i under A_2 is $A_2(x_1) + A_2(x_2) + \cdots + A_2(x_k)$, which becomes $A_2(x_1 + x_2 + \cdots + x_k)$ for odd values of k, and $A_2(x_1 + x_2 + \cdots + x_k) + A_2(0)$ for even values of k. Since $x_1 + x_2 + \cdots + x_k = 0$ by assumption, the sum $A_2(x_1) + \cdots + A_2(x_k)$ is then $A_2(0)$ for odd k, and 0 for even k. Thus, computing all sums of the form $F \circ A_2(x_1) + F \circ A_2(x_2) + \cdots + F \circ A_2(x_k)$ for $(x_1, x_2, \ldots, x_k) \in \mathcal{T}_k(0)$ is equivalent to computing all sums of the form $F(x_1) + F(x_2) + \cdots + F(x_k)$ for $(x_1, x_2, \ldots, x_k) \in \mathcal{T}_k(q)$, with $q = 0$, resp. $q = A_2(0)$ for k even, resp. k odd. Thanks to the affinity of A_1, computing the sums of values of $A_1 \circ F \circ A_2$ amounts to computing the corresponding sums of values of $F \circ A_2$, and then taking their image under A_1, up to the addition of the constant $A_1(0)$ in the case of even k. Finally, the sum $A(x_1) + A(x_2) + \cdots + A(x_k)$ of the images of $x_1, \ldots, x_k$ under A is equal to $A(0)$ for k odd, and is equal to 0 for k even. Computing the sums of values of $G = A_1 \circ F \circ A_2 + A$ is thus the same as computing the sums of values of $A_1 \circ F \circ A_2$, up to the addition of the constant $A(0)$ in the case of odd k. The particular statement follows immediately from the above by observing that the elements of $\Sigma_k^G(0)$ are simply the images of the elements in $\Sigma_k^F(0)$ under the linear part of the permutation A_1. □

Recall that the majority of known APN functions are quadratic, and that testing the equivalence of quadratic (n, n)-functions represents the case of highest practical interest. One very useful observation that we can make in the quadratic case is that we can assume $A_1(0) = A_2(0) = 0$, which (as we see later), greatly simplifies the complexity of the entire EA-equivalence test; and, in particular, means that the multiplicities of $\Sigma_k^F(0)$ are an EA-invariant for quadratic functions in the case of odd values of k as well.

Proposition 2 *Let* F, G *be quadratic* (n, n)*-functions for some positive integer n, and suppose that* $A_1 \circ F \circ A_2 + A = G$ *for some affine* (n, n)*-functions* A_1, A_2, A *with* A_1*,* A_2 *bijective. Furthermore, let* $c_1 = A_1(0)$*,* $c_2 = A_2(0)$*, and* $L_1(x) = A_1(x) + c_1$*,* $L_2(x) = A_2(x) + c_2$ *so that* L_1 *and* L_2 *are linear. Then there exists an affine* (n, n)*-function* A' *such that*

$$L_1 \circ F \circ L_2 + A' = G.$$

Proof We can assume that F is purely quadratic, i.e. of the form $F(x) = \sum_{0 \le i < j < n} c_{ij} x^{2^i+2^j}$ for some coefficients $c_{ij} \in \mathbb{F}_{2^n}$. The composition $F \circ A_2$ expands to

$$\begin{aligned} F(A_2(x)) &= F(L_2(x) + c_2) = \sum_{ij} c_{ij} (L_2(x) + c_2)^{2^i+2^j} \\ &= \sum_{ij} c_{ij} L_2(x)^{2^i+2^j} + \sum_{ij} c_{ij} (c_2^{2^j} L_2(x)^{2^i} + c_2^{2^i} L_2(x)^{2^j} + c_2^{2^i+2^j}) \\ &= F(L_2(x)) + A''(x), \end{aligned}$$

where $A''(x) = \sum_{ij} c_{ij}(c_2^{2^j} L_2(x)^{2^i} + c_2^{2^i} L_2(x)^{2^j} + c_2^{2^i+2^j})$ is an affine function. Then the composition $A_1 \circ F \circ A_2$ becomes

$$A_1(F(A_2(x))) = L_1((F \circ L_2)(x) + A''(x)) + c_1 = L_1 \circ F \circ L_2(x) + AA'''(x),$$

where $AA'''(x) = L_1(A''(x)) + c_1$. Finally, taking $A' = AA'''(x) + A(x)$, we have

$$L_1 \circ F \circ L_2 + A' = G$$

as desired. □

As suggested above, Proposition 2 implies that the multiplicities of $\Sigma_k^F(0)$ are an invariant for both odd and even values of k in the quadratic case. Note that the condition of the function being quadratic is necessary, as witnessed by e.g. $F(x) = x^{15}$ and $G(x) = (x+\alpha)^{15}$ over $\mathbb{F}_2six$, where α is a primitive element of $\mathbb{F}_2six$: the elements of the finite field in question fall into three distinct classes based on their multiplicities in $\Sigma_3^F(0)$, but into five distinct classes based on their multiplicities in $\Sigma_3^G(0)$.

Corollary 3 *Following the notation and hypothesis of Proposition 1, if F and G are in addition quadratic, then the multiplicities of $\Sigma_k^F(0)$ and $\Sigma_k^G(0)$ are the same for any value of k.*

The complexity of computing the multiplicities of $\Sigma_k^F(t)$ for an (n,m)-function F increases exponentially with each increment of k. Fortunately, computing the multiplicities via the Walsh transform of F results in a complexity that does not depend on the value of k.

Proposition 4 *Let F be an (n,m)-function, k be a positive integer, $t \in \mathbb{F}_2^n$ and $s \in \mathbb{F}_2^m$. Let $M_k^F(t,s)$ denote the number of k-tuples $(x_1, x_2, \ldots, x_k)$ such that $x_1 + x_2 + \cdots + x_k = t$ and $F(x_1) + F(x_2) + \cdots + F(x_k) = s$. Then*

$$2^{m+n} M_k^F(t,s) = \sum_{a \in \mathbb{F}_2^n} (-1)^{Tr_n(at)} \sum_{b \in \mathbb{F}_2^m} (-1)^{Tr_m(bs)} W_F^k(a,b). \tag{2}$$

Proof From the definition of the Walsh transform, the expression

$$\sum_{a \in \mathbb{F}_2^n} (-1)^{\mathrm{Tr}_n(at)} \sum_{b \in \mathbb{F}_2^m} (-1)^{\mathrm{Tr}_m(bs)} W_F^k(a,b)$$

expands to

$$\sum_{a \in \mathbb{F}_2^n, b \in \mathbb{F}_2^m} \sum_{x_1, \ldots, x_k \in \mathbb{F}_2^n} (-1)^{\mathrm{Tr}_m(b(s + \sum_{i=1}^k F(x_i))) + \mathrm{Tr}_n(a(t + \sum_{i=1}^k x_i))}.$$

By changing the order of summation, this becomes

$$\sum_{b\in\mathbb{F}_2^m}\sum_{x_1,\ldots,x_k\in\mathbb{F}_2^n}(-1)^{\mathrm{Tr}_m(b(s+\sum_{i=1}^k F(x_i)))}\sum_{a\in\mathbb{F}_2^n}(-1)^{\mathrm{Tr}_n(a(t+\sum_{i=1}^k x_i))}.$$

The statement then follows by recalling that $\sum_{a\in\mathbb{F}_2^n}(-1)^{\mathrm{Tr}_n(ax)}$ evaluates to 0 for any $0\neq x\in\mathbb{F}_2^n$, and evaluates to 2^n for $x=0$. □

Finding the multiplicity of a given element $s\in\mathbb{F}_2^m$ in $\Sigma_k^F(t)$ now amounts to computing the Walsh coefficients $W_F(a,b)$ of F, raising them to the power k, and combining them according to (2). We note that for the purposes of testing EA-equivalence, we always assume $t=0$, and hence (2) simplifies to $2^{m+n}M_k^F(0,s)=\sum_{a\in\mathbb{F}_2^n,b\in\mathbb{F}_2^m}(-1)^{\mathrm{Tr}_m(bs)}W_F^k(a,b)$. Furthermore, the Walsh coefficients $W_F(a,b)$ can be precomputed for all F from a known set of EA-representatives, allowing the computations to be sped up at the cost of storing the precomputed result.

Remark 5 We note that, in the case of APN functions, the multiset of the multiplicities of $\Sigma_3^F(0)$ is essentially the same as the multiset Π_F^0 studied in [6]. The latter, given an (n,n)-function F, is defined as the multiset

$$\Pi_F^0=\{\#\{a\in\mathbb{F}_{2^n}\mid(\exists x\in\mathbb{F}_{2^n})F(x)+F(a+x)+F(a)=b\}:b\in\mathbb{F}_{2^n}\}.$$

The equation $F(x)+F(a+x)+F(a)=b$ has either 0 or 2 solutions for any $0\neq a\in\mathbb{F}_{2^n}$ and any $b\in\mathbb{F}_{2^n}$ if F is APN. Thus, an equivalent invariant would be the multiset

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

and it is easy to see that this can equivalently be rewritten as

$$\{F(x_1)+F(x_2)+F(x_3):(x_1,x_2,x_3)\in\mathbb{F}_{2^n}^3\mid x_1+x_2+x_3=0\},$$

which is essentially the same as $\Sigma_3^F(0)$. As pointed out in [6], the multiset Π_F^0 is a CCZ-invariant for quadratic APN functions.

3 Guessing the outer permutation

Suppose that we are given two EA-equivalent functions F and G from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ for some positive integers n,m, so that $A_1\circ F\circ A_2+A=G$ for some affine functions $A_1:\mathbb{F}_2^m\to\mathbb{F}_2^m$, $A_2:\mathbb{F}_2^n\to\mathbb{F}_2^n$ and $A:\mathbb{F}_2^n\to\mathbb{F}_2^m$, with A_1 and A_2 bijective. Let $c_1=A_1(0)$ and $c=A(0)$ so that $L_1(x)=A_1(x)+c_1$ and $L(x)=A(x)+c$ are the linear parts of A_1 and A, respectively. We note that if $A_1\circ F\circ A_2+A=G$, then also $A_1'\circ F\circ A_2+A'=G$ where $A_1'(x)=A_1(x)+\Delta$ and $A'(x)=A(x)+\Delta$ for any $\Delta\in\mathbb{F}_2^m$. In particular, we can always assume that $c_1=0$ without loss of generality, so that A_1 is linear. In the following, we will write simply $G=L_1\circ F\circ A_2+A$.

By Proposition 1, for even values of k, we have

$$\Sigma_k^G(0)=\{L_1(a):a\in\Sigma_k^F(0)\}.$$

Besides justifying that the multiset of multiplicities of $\Sigma_k^F(0)$ is an EA-invariant for any positive even integer k, the above relation gives us some information about L_1; namely, it

implies that if $L_1(x) = y$ for some $x, y \in \mathbb{F}_2^m$, then the multiplicity of x in $\Sigma_k^F(0)$ should be the same as that of y in $\Sigma_k^G(0)$. If the elements of $\mathbb{F}_2^m$ are partitioned according to the multiplicities of F as

$$\mathbb{F}_2^m = K_1 \oplus K_2 \oplus \cdots \oplus K_s$$

for some positive integer s, so that all elements in K_i for every $1 \leq i \leq s$ have the same multiplicity in $\Sigma_k^F(0)$, and elements in K_i and K_j have distinct multiplicities for $i \neq j$; and, similarly, as

$$\mathbb{F}_2^m = C_1 \oplus C_2 \oplus \cdots \oplus C_s$$

according to the multiplicities of G, then we must have

$$L_1(K_i) = C_i$$

for all $1 \leq i \leq s$. We will say that any permutation L_1 satisfying $L_1(K_i) = C_i$ for all i *respects* the two partitions of $\mathbb{F}_2^m$. Consequently, we obtain conditions that can be used to restrict the possible choices for L_1. Intuitively, the larger the number of classes s in the partition of $\mathbb{F}_2^m$, the fewer linear permutations L_1 can satisfy the conditions thus obtained. In particular, if all elements of $\mathbb{F}_2^m$ occur with the same multiplicity, we do not obtain any information on L_1. This is clearly the case when F is a permutation. Furthermore, the same appears to be true for all APN (n, n)-functions with odd n (regardless of whether they are permutations or not), which is why we concentrate on fields of even extension degree in our work.

All linear permutations L respecting the partitions $\mathbb{F}_2^m = K_1 \oplus \cdots \oplus K_s$ and $\mathbb{F}_2^m = C_1 \oplus \cdots \oplus C_s$ can now be found by trying to guess the values of L on a basis of $\mathbb{F}_2^m$, and backtracking whenever some assignment violates these partitions. An algorithmic description of this procedure is provided below under Algorithm 1 and Algorithm 2. The former presents the general framework for partitioning $\mathbb{F}_2^m$ according to the multiplicities of sums of values of F and G, while the latter describes the process of reconstructing all linear permutations that respect the constructed partitions. We remark that the algorithm is described for the particular case of $k = 4$ (which is what we have mostly used in practice for our computational experiments), but the principle trivially generalizes to any value of k. We also note that computing the number $M_k^F(0, s)$ of k-tuples whose values under F add up to a given $s \in \mathbb{F}_2^m$ can be done via the values of the Walsh transform as described in Proposition 4; this is particularly useful if the selected value of k is large, or if a precomputed table of the Walsh coefficients for one (or both) of the tested functions is available.

Let us take a closer look at Algorithm 1. We first fix an even value of k, for instance $k = 4$. Given two (n, n)-functions, F and G, that we would like to test for equivalence, we begin by computing the multiplicities of the elements in the multisets $\Sigma_k^F(0)$ and $\Sigma_k^G(0)$. The number of times that the element $s \in \mathbb{F}_2^m$ appears in $\Sigma_k^F(0)$ is denoted by $M_k^F(0, s)$ (this means that $M_k^F(0, s)$ k-tuples $(x_1, x_2, \ldots x_k)$ with $x_1 + x_2 + \cdots + x_k = 0$ satisfy $F(x_1) + F(x_2) + \cdots + F(x_k) = s$).

Using these multiplicities, we partition $\mathbb{F}_2^m$ in two ways: using the multiplicities $\{M_k^F(0, s) : s \in \mathbb{F}_2^m\}$, and using the multiplicities $\{M_k^G(0, s) : s \in \mathbb{F}_2^m\}$. More precisely, we write $\mathbb{F}_2^m$ as $\mathbb{F}_2^m = K_1 \oplus K_2 \oplus \cdots \oplus K_s$, with $K_1, K_2, \ldots, K_s$ being disjoint sets of elements; two elements s_1 and s_2 are in the same block K_i of the partition if and only if $M_k^F(0, s_1) = M_k^F(0, s_2)$, i.e. if s_1 and s_2 occur with the same multiplicity in $\Sigma_k^F(0)$. Equiv-

alently, we could say that the multiplicities $M_k^F(0,s)$ induce an equivalence relation, in which two elements $s_1, s_2 \in \mathbb{F}_2^m$ are equivalent precisely when $M_k^F(0,s_1) = M_k^F(0,s_2)$; the blocks $K_1, K_2, \ldots, K_s$ are then the equivalence classes of this equivalence relation. In the same way that $K_1, K_2, \ldots, K_s$ is the partition induced by $\Sigma_k^F(0)$, $C_1, C_2, \ldots, C_{s'}$ is the partition induced by $\Sigma_k^G(0)$.

If F and G are EA-equivalent, then the number of blocks in both partitions must be the same, and the individual blocks must have the same sizes. Thus, if $s \neq s'$, or if the multiset $\{\#K_i : i = 1, 2, \ldots, s\}$ is not equal to $\{\#C_i : i = 1, 2, \ldots, s\}$, we can immediately conclude that F and G are not EA-equivalent. Otherwise, we can rearrange the blocks $K_1, K_2, \ldots, K_s$ and $C_1, C_2, \ldots, C_s$ in such a way that $\#K_i = \#C_i$ for $i = 1, 2, \ldots, s$. At this point, we know that if F and G are equivalent via $L_1 \circ F \circ A_2 + A = G$, then L_1 must map K_i to C_i for $i = 1, 2, \ldots s$. This additional information allows us to significantly reduce the number of linear permutations L_1 that needs to be considered.

The set of all linear permutations preserving the partitions can be found using Algorithm 2. The latter is essentially an exhaustive search that tries to guess the values of L_1 on a basis $B = \{b_1, b_2, \ldots, b_m\}$ of $\mathbb{F}_2^m$. After we have guessed the values of L_1 on $b_1, b_2, \ldots, b_i$ for some $i \leq m$, we know the values of L_1 on all elements of $\mathbb{F}_2^m$ generated by $\{b_1, b_2, \ldots, b_i\}$. For any such element x, we can find the indices j, j' such that $x \in K_j$ and $L_1(x) \in C_{j'}$. If $j \neq j'$, then L_1 does not respect the partitions, and so we backtrack, attempting a different guess for b_i. If we do not find any contradiction of this type, we proceed to guessing the value of b_{i+1}. We continue in this manner until we have exhausted all possibilities.

Algorithm 1 General framework for reconstructing the outer permutation.

Input : Two (n,m)-functions F and G
Output: All linear permutations L_1 of $\mathbb{F}_2^m$ respecting the partitions induced by F and G
for $s \in \mathbb{F}_2^m$ **do**
  compute the number $M_4^F(0,s)$ of $(x_1, x_2, x_3, x_1+x_2+x_3) \in \mathcal{T}_4(0)$ such that $F(x_1)+F(x_2)+F(x_3)+F(x_1+x_2+x_3) = s$;
  compute the number $M_4^G(0,s)$ of $(x_1, x_2, x_3, x_1+x_2+x_3) \in \mathcal{T}_4(0)$ such that $G(x_1)+G(x_2)+G(x_3)+G(x_1+x_2+x_3) = s$;
end
partition $\mathbb{F}_2^m = K_1 \oplus K_2 \oplus \cdots \oplus K_s$ so that $M_4^F(0,s_1) = M_4^F(0,s_2)$ for $s_1 \in K_i$ and $s_2 \in K_j$ if and only if $i = j$;
partition $\mathbb{F}_2^m = C_1 \oplus C_2 \oplus \cdots \oplus C_{s'}$ so that $M_4^G(0,s_1) = M_4^G(0,s_2)$ for $s_1 \in C_i$ and $s_2 \in C_j$ if and only if $i = j$;
if $s \neq s'$ **then**
  return $\emptyset$
end
rearrange $C_1, C_2, \ldots, C_s$ if necessary so that $M_4^F(0,s_1) = M_4^G(0,s_2)$ where $s_1 \in K_i$, $s_2 \in C_i$ for any $1 \leq i \leq s$;
if $\#C_i \neq \#K_i$ *for some* i *in* $1 \leq i \leq s$ **then**
  return $\emptyset$
end
select $\mathcal{I} \subseteq \{1, \ldots, s\}$ such that $U = \bigcup_{i \in \mathcal{I}} K_i$ contains a basis $B = \{b_1, \ldots, b_m\}$ of $\mathbb{F}_2^m$ and $\#U$ is as small as possible ;
return *all linear permutations* L_1 *of* $\mathbb{F}_{2^n}$ *mapping* K_i *to* C_i *for* $1 \leq i \leq s$ *as per Algorithm 2*

Algorithm 2 Finding all linear permutations respecting a pair of partitions.

Input : Two partitions $\mathbb{F}_2^m = K_1 \oplus K_2 \oplus \cdots \oplus K_s$ and $\mathbb{F}_2^m = C_1 \oplus C_2 \oplus \cdots \oplus C_s$ of the vector space $\mathbb{F}_2^m$, a basis $B = \{b_1, b_2, \cdots, b_m\}$ of $\mathbb{F}_2^m$, and a set U of possible values for the images of B

Output: All linear permutations L_1 of $\mathbb{F}_2^m$ such that $L_1(K_i) = C_i$ for $1 \leq i \leq s$

```
Set L_1(0) ← 0 ;
return assign(1)

procedure assign(i) ;
if i = m + 1 then
    return {L_1} ;
end
Results ← ∅ ;
for c_i ∈ U do
    partitionPreserved ← true ;
    for x ∈ Span({b_1, ··· , b_{i−1}}) do
        L_1(x + b_i) ← L_1(x) + c_i ;
        find j such that x + b_i ∈ K_j ;
        if L_1(x + b_i) ∉ L_j then
            partitionPreserved ← false ;
            break ;
        end
    end
    if partitionPreserved then
        Results ← Results ∪ assign(i + 1) ;
    end
end
return Results
```

The partitions $\mathbb{F}_2^m = K_1 \oplus K_2 \oplus \cdots \oplus K_s$ can be precomputed for representatives from e.g. all known EA-classes of APN functions; in particular, we refer to our computational results described in Section 5 where we describe how we provide such pre-computed results for all currently known APN functions over $\mathbb{F}_{2^n}$ up to dimension $n = 10$. When using Algorithms 1 and 2 to find all possibilities for the outer permutation L_1 in $L_1 \circ F \circ A_2 + A = G$, however, we need to know the partitions according to both F and G, which makes the precomputation of the permutations L_1 impossible.

Nonetheless, we can observe that the set of linear permutations L_1 mapping K_i to C_i for every $1 \leq i \leq s$ is simply a coset in the symmetric group of $\mathbb{F}_2^m$ of the subgroup of linear permutations mapping K_i to K_i for $1 \leq i \leq s$. The latter can be precomputed for known EA-representatives, and hence finding a single linear permutation mapping every K_i to C_i with $1 \leq i \leq s$ allows us to reconstruct all such permutations by composing it with the precomputed ones. This can be formalized as follows.

Proposition 6 *Let n be a positive integer, and $\mathbb{F}_2^n = K_1 \oplus K_2 \oplus \cdots \oplus K_s$ and $\mathbb{F}_2^n = C_1 \oplus C_2 \oplus \cdots \oplus C_s$ be two partitions of the elements of $\mathbb{F}_2^n$ such that $\#K_i = \#C_i$ for every*

$1 \leq i \leq s$. Let $\mathcal{K}$ be the set of all linear permutations L of $\mathbb{F}_2^n$ such that $L(K_i) = K_i$ for all $1 \leq i \leq s$, and let $\mathcal{P}$ be the set of all linear permutations L of $\mathbb{F}_2^n$ such that $L(K_i) = C_i$ for $1 \leq i \leq s$. Then $\mathcal{K}$ is a subgroup of the symmetric group of $\mathbb{F}_2^n$, and $\mathcal{P}$ is a coset of $\mathcal{K}$.

Proof The composition of two linear permutations is clearly a linear permutation itself, and so is the inverse of a linear permutation. Furthermore, if L_1 and L_2 are linear permutations that permute some set $K_i \subseteq \mathbb{F}_2^n$, then their composition and their inverses do so as well. Thus, $\mathcal{K}$ is closed under composition and taking inverses, and is a subgroup of the symmetric group of $\mathbb{F}_2^n$.

Now, suppose that L is a linear permutation of $\mathbb{F}_2^n$ mapping some subset $K_i \subseteq \mathbb{F}_2^n$ onto some $C_i \subseteq \mathbb{F}_2^n$. Then $K \circ L$ is also a linear permutation mapping K_i onto C_i for any $K \in \mathcal{K}$. Thus, $K \mapsto K \circ L$ maps $\mathcal{K}$ to $\mathcal{P}$, and is clearly invertible since L is a permutation. Consequently, $\mathcal{P}$ is a coset of $\mathcal{K}$ represented by L. □

Besides delegating a large portion of the work in constructing $\mathcal{P}$ to the precomputation of $\mathcal{K}$, Proposition 6 allows us to estimate the complexity of testing EA-equivalence between a function F (which we can assume is a known EA-representative) and another function G inducing a partition of $\mathbb{F}_2^m$ compatible with the one induced by F.

Furthermore, it is clear that the size of the group $\mathcal{K}$ of linear permutations that preserve the partition $\mathbb{F}_2^m = K_1 \oplus \cdots \oplus K_s$ induced by the multiplicities in $\Sigma_F^k(0)$ is an EA-invariant. What makes this interesting, is that it is more discriminating than the sizes of the partition classes: for instance, the APN functions $F(x) = x^3$ and $G(x) = x^3 + \alpha^{11}x^6 + \alpha x^9$ over $\mathbb{F}_2six$ (where α is primitive in $\mathbb{F}_2six$) both partition $\mathbb{F}_2six$ into three classes of size 1, 21, and 42, respectively; but the group of linear permutations preserving the partition of $F(x)$ contains 1008 elements, while the group of linear permutations preserving the partition of G has 336 elements. Thus, precomputing the groups $\mathcal{K}$ of linear permutations preserving the partition for representatives from the known classes of APN functions has the additional advantage that it allows us to rule out equivalence in more cases (using a stronger invariant). We note that the actual elements of the group $\mathcal{K}$ are not, in general, invariant under EA-equivalence.

To give some basic idea of how efficient these processes are, we have computed the groups $\mathcal{K}$ for representatives from all switching classes of APN functions over $\mathbb{F}_{2^n}$ with $n \in \{6, 8\}$ [12]. The results are presented in Table 1 below. The first column gives the dimension n of $\mathbb{F}_{2^n}$. The functions are indexed in the second column in the same way as in [12]. The next two columns give the time in seconds for computing the partition of $\mathbb{F}_{2^n}$ according to the quadruple sums of F directly and using the Walsh transform, respectively (including the time in seconds for precomputing the Walsh coefficients). The following column gives the time for computing all linear permutations preserving the corresponding partition. The last column gives the number of linear permutations found in each case, which is a direct measure of the complexity of an EA-equivalence test by our method, as the approach for guessing the inner permutation (described in the following Section 4) has to be applied to every possible choice of the outer permutation.

We note that the running times are highly dependent on the programming language, implementation, and computational equipment used, and the ones presented in the paper are given only for illustrative purposes.

For comparison, there are 27998208 linear permutations of $\mathbb{F}_{2^6}$, and 132640470466560 linear permutations of $\mathbb{F}_{2^8}$.

Table 1 Computational experiments for finding the outer permutation

n	ID	Sums	Walsh	Time	Permutations
6	1.1	1.650	1.250	1.030	1008
	1.2	1.510	1.390	0.300	336
	2.1	1.390	1.450	0.010	10
	2.2	1.250	1.250	0.380	336
	2.3	1.240	1.450	0.970	1008
	2.4	1.260	1.250	0.010	8
	2.5	1.300	1.310	0.050	60
	2.6	1.260	1.290	0.010	8
	2.7	1.310	1.290	0.010	10
	2.8	1.310	1.310	0.010	8
	2.9	1.300	1.310	0.010	7
	2.10	1.580	1.300	0.010	8
	2.11	1.290	1.290	0.000	8
	2.12	2.450	2.470	0.030	48
8	1.1	103.580	74.910	23.090	680
	1.2	92.140	86.570	206.830	680
	1.3	244.540	238.560	78.180	8
	1.4	146.520	140.710	12.530	8
	1.5	112.860	107.580	58.300	4
	1.6	111.810	106.920	62.580	4
	1.7	127.330	121.320	10.020	1
	1.8	126.210	121.740	26.670	4
	1.9	127.250	121.730	40.370	4
	1.10	127.090	121.270	10.400	2
	1.11	127.410	122.560	50.560	4
	1.12	127.950	121.240	46.520	4
	1.13	127.850	122.320	10.530	2
	1.14	132.900	127.100	0.010	2
	1.15	126.410	121.940	22.580	1
	1.16	127.020	121.040	9.970	2
	1.17	126.860	120.790	69.860	2
	2.1	99.690	94.340	27.380	360
	3.1	118.870	112.990	57.480	4
	4.1	115.700	110.040	0.070	16
	5.1	102.470	96.640	0.030	8
	6.1	110.940	105.610	0.040	8
	7.1	98.650	93.330	49.350	680

4 Guessing the inner permutation

If, in addition to the (n, m)-functions F and G, we know the linear permutation L_1 in the relation $L_1 \circ F \circ A_2 + A = G$, we can apply its inverse, L_1^{-1} to both sides, obtaining

$$F \circ A_2 + A' = G', \tag{3}$$

where $A' = L_1^{-1} \circ A$ and $G' = L_1^{-1} \circ G$. A pair of affine (n, m)-functions A_2, A' satisfying the above relation then exists if and only if F is EA-equivalent to G.

Once again, let us write $c = A'(0)$ and $c_2 = A_2(0)$, and $L_2 = A_2 + c_2$ and $A = L + c$ for the linear parts of A_2 and A, respectively. Substituting 0 for x in (3) yields $F(c_2) + c = G'(0)$. Since we know G', and hence also $G'(0)$, this means that any choice of c_2 uniquely determines c. It is thus enough to loop over all possible choices of $c_2 \in \mathbb{F}_2^n$ and take $c = F(c_2) + G'(0)$ in order to exhaust all possibilities for (c_2, c'). As observed in Proposition 2, if F and G are quadratic, then we can assume that $c_2 = 0$ and $c = G'(0)$ without loss of generality; for functions of higher algebraic degree, we have to consider all possible values of c_2. In the following, we assume that we have guessed the constants c_2 and c, and rewrite (3) as

$$F \circ L_2 + L' = G'', \tag{4}$$

where $G''(x) = G'(x+c_2)+c$. It now remains to look for a pair of functions $L_2 : \mathbb{F}_2^m \to \mathbb{F}_2^m$ and $L : \mathbb{F}_2^n \to \mathbb{F}_2^m$ satisfying (4).

To guess the permutation L_2, we observe that, given some k-tuple $(x_1, x_2, \cdots, x_k) \in \mathcal{T}_k(0)$, by Proposition 1, we have

$$G''(x_1) + \cdots + G''(x_k) = F(L_2(x_1)) + \cdots + F(L_2(x_k)),$$

and thus, if some element $x_i \in \mathbb{F}_2^n$ is part of a k-tuple whose sum under G is t, then its image $L_2(x_i)$ under L_2 must be part of some k-tuple whose sum under F is t. We state this formally as follows.

Proposition 7 *Let F and G be (n, m)-functions for some positive integers n, m such that $F \circ L_2 + L = G$ for L_2, L linear and L_2 bijective. Let k be a positive integer, and, for any $t \in \mathbb{F}_2^m$, denote*

$$O_k^F(0, t) = \{(x_1, x_2, \cdots, x_k) \in \mathcal{T}_k(0) \mid F(x_1) + F(x_2) + \cdots + F(x_k) = t\}.$$

Then, if $(x_1, x_2, \cdots x_k) \in \mathcal{T}_k(0)$ with $G(x_1) + \cdots + G(x_k) = t$, we must have $L_2(x_i) \in$

$M_k^F(0,t)$ *for all* $1 \le i \le k$. *Consequently,*

$$L_2(x) \in \bigcap_{t \in \sum_k^G(0,x)} \left(\bigcup O_k^F(0,t) \right), \tag{5}$$

where

$$\Sigma_k^G(0,x) = \{G(x_1) + \cdots + G(x_k) : (x_1, \ldots, x_k) \in \mathcal{T}_k(0) \mid x \in (x_1, \ldots x_k)\}.$$

Using (5) for $k = 3$, we can significantly reduce the *domains* of $L_2(x)$ for $x \in \mathbb{F}_2^m$, i.e. the ranges of possible values that $L_2(x)$ can take. A large number of the domains end up consisting of three elements (although we do obtain larger domains in some cases). Since guessing L_2 amounts to guessing its values on a basis of $\mathbb{F}_2^m$, the elements of the basis can be chosen in such a way that the Cartesian product of the respective domains is small. In most cases, we can indeed choose the basis elements in such a way that all domains consist of three elements, and thus end up with only 3^n possibilities for L_2.

In addition, assuming that e.g. F is a known representative, some precomputations are possible; namely, the sets $O_k^F(0,t)$ can be precomputed for $k = 3$ and all values of t. Alternatively, the roles of F and G in (4) can be swapped by composing both sides with the inverse of L_2 from the right, in which case the sets $\Sigma_k^F(0,x)$ can be precomputed for $k = 3$ and for all $x \in \mathbb{F}_2^m$.

We note that for values of k greater than 3, it seems to always be possible to express all elements in $\mathbb{F}_{2^n}$ as the sum of four values of F for an APN function F, in which case $\sum_k^G(0,x) = \mathbb{F}_2^m$ for all $x \in \mathbb{F}_2^m$, and consequently the domains of all $x \in \mathbb{F}_2^m$ end up being the entire field $\mathbb{F}_2^m$. Since the definitions of $\mathcal{T}_k(0)$ and $\sum_k^F(0)$ only make sense for values of k greater than 2, $k = 3$ remains the only practically useful choice for the value of k (at least in the case of APN functions).

Algorithm 3 describes the approach for reconstructing L_2 from (3) suggested by the above considerations. The first part of the algorithm computes the domains $\mathcal{D}(x)$ for all elements $x \in \mathbb{F}_2^n$; we then know that for any $x \in \mathbb{F}_2^n$ we must have $L_2(x) \in \mathcal{D}(x)$. All domains are initially set to $\mathbb{F}_2^n$, i.e. no restrictions on the value of $L_2(x)$ is made. We then compute the sets $O_3^F(t)$ of all triples (x_1, x_2, x_1+x_2) with $F(x_1)+F(x_2)+F(x_1+x_2) = t$. For any element y_1 belonging to a triple (y_1, y_2, y_1+y_2) with $G(y_1)+G(y_2)+G(y_1+y_2) = t$, we know that $L_2(y_1)$ must belong to $O_3^F(t)$; we use this to reduce the domain $\mathcal{D}(y_1)$ of y_1.

Having computed the domains, the second part of the algorithm consists of finding a basis $B = \{b_1, b_2, \ldots, b_n\}$ of $\mathbb{F}_2^n$, and constructing all linear permutations L_2 for which $L_2(b_i) \in \mathcal{D}(b_i)$ for $i = 1, 2, \ldots, n$. Since we assume that $F \circ A_2 + A = G$ (with $A_2 = L_2 + c_2$, where we also guess the value of c_2 by going through all possibilities), if the choice of L_2 is correct, then $A = F \circ A_2 + G$ must be affine. For every possible of choice of L_2 and c_2, we thus compute A and check whether its algebraic degree is at most 1; if so, then we have found the equivalence between F and G.

Algorithm 3 Reconstructing the inner permutation A_2.

Input : Two (n, m)-functions F and G with $F(0) = G(0) = 0$
Output: All affine permutations A_2 of $\mathbb{F}_2^m$ such that $F \circ A_2 + G$ is affine
for $x \in \mathbb{F}_{2^n}$ **do**
  $\mathcal{D}(x) \leftarrow \mathbb{F}_{2^n}$ (initialize domains) ;
  $O_3^F(x) \leftarrow \emptyset$;
end
for $(x_1, x_2) \in (\mathbb{F}_2^m)^2$ **do**
  $t \leftarrow F(x_1) + F(x_2) + F(x_1 + x_2)$;
  $O_3^F(t) \leftarrow O_3^F(t) \cup (x_1, x_2, x_1 + x_2)$;
end
for $(x_1, x_2) \in (\mathbb{F}_2^m)^2$ **do**
  $t \leftarrow G(x_1) + G(x_2) + G(x_1 + x_2)$;
  $\mathcal{D}(x_1) \leftarrow \mathcal{D}(x_1) \cap O_3^F(t)$;
  $\mathcal{D}(x_2) \leftarrow \mathcal{D}(x_2) \cap O_3^F(t)$;
  $\mathcal{D}(x_1 + x_2) \leftarrow \mathcal{D}(x_1 + x_2) \cap O_3^F(t)$;
end
Order the elements $x \in \mathbb{F}_2^m$ into x_i for $1 \le i \le 2^n$, so that
$i < j \implies \#\mathcal{D}(x_i) \le \#\mathcal{D}(x_j)$;
$B \leftarrow \emptyset$ (basis) ;
$Results \leftarrow \emptyset$;
for $i = 1, 2, \ldots, 2^m$ **do**
  if $x_i \notin Span(B)$ **then**
    $B \leftarrow B \cup \{x_i\}$;
    if $\#B = m$ **then**
      break ;
    end
  end
end
for $c_2 \in \mathbb{F}_2^m$ **do**
  for $(v_1, v_2, \ldots, v_n) \in \prod_{x \in B} \mathcal{D}(x)$ **do**
    Let L_2 be linear with $L_2(b_i) = v_i$ for $B = (b_1, \ldots, b_m)$;
    $A_2 \leftarrow L_2 + c_2$;
    $A \leftarrow F \circ A_2 + G$;
    if $\deg(A) \le 1$ **then**
      $Results \leftarrow Results \cup \{(A_2, A + F(c_2))\}$;
    end
  end
end
return $Results$

Recall that by Proposition 2 we can assume that $c_2 = 0$ if the functions being tested for equivalence are quadratic, which significantly reduces the computation time.

We note that Algorithm 3 will return all affine permutations A_2 for which $A = F \circ A_2 + G$ is affine. If our goal is to check whether such a permutation exists (which is all that we need for the purposes of the EA-equivalence test), we can immediately terminate as soon as a single such permutation is found. Furthermore, we remark that if (3) is obtained by applying

the inverse L_1^{-1} of the outer permutation, and a solution (A_2, A) of (3) is found, then this already witnesses that F and G are EA-equivalent.

In order to get an idea of the efficiency of this method, we once again run a number of experiments on representatives from the known APN functions for $n = 6$ and $n = 8$. For every pair (F, G) of representatives from the switching classes in [12], we generate a random affine permutation A_2 and a random affine function A, and use Algorithm 3 to attempt to reconstruct A_2 and A from F and G. In the cases when F and G are not EA-equivalent this, of course, will fail; in the remaining cases (when F and G do belong to the same EA-equivalence class), we stop as soon as we find the first pair of affine functions (A_2, A) solving $F \circ A_2 + A = G$. For each combination of F and G, we generate 10 pairs of (A_2, A). Table 2 gives the average running time for solving $F \circ A_2 + A = G$ for dimensions $n = 8$. There are 23 switching APN representatives in $\mathbb{F}_{2^8}$, and we index them from 1 to 23 in Table 2 in the same order that they are listed in [12]. In the case of $n = 6$, the running time does not exceed 0.2 seconds in the worst case; we omit a detailed table of the running times for the sake of brevity. The running times are given in seconds, multiplied by a factor of 100; e.g. deciding that $F \circ A_2 + A = G$ is unsolvable when F is 1.1 and G is 1.2 from [12] takes 7.51 seconds.

5 Computational results

A recent paper [1] introduces 12 923 new APN functions over $\mathbb{F}_{2^8}$, in addition to the more than 8000 instances previously found and documented in [19]. For the purposes of measuring how efficient the multiplicities of the elements in $\Sigma_k^F(0)$ are as an invariant, and for speeding-up potential EA-equivalence tests, we have computed the exact partitions for $k = 4$ for all of these functions. We also perform similar computations for all known APN functions up to dimension $n = 10$. A complete list of these partitions is available online at https://boolean.h.uib.no/mediawiki. Here, we give a summary of the computed data.

In total, we have computed the partition induced by $\Sigma_k^F(0)$ for 21105 CCZ-inequivalent functions F. From these, we have obtained 19300 distinct partitions. Of these, the "Gold-like" partition (which splits the field into three partition classes, of size 1, 70, and 185, respectively) is the most frequently occurring, and is induced by 21 functions including, of course, the Gold function x^3. The number of partitions that occur only once is 18103; and the remaining partitions occur between two and eleven times.

Most of the partitions contain a large number of classes: indeed, only the "Gold-like" partition described above has three classes, while all other observed partitions have at least 6 classes; the vast majority of functions induce a partition having between 12 and 16 classes, while the largest number of classes, 22, is achieved by only two functions. We recall that a large number of classes intuitively corresponds to a small number of linear permutations respecting the corresponding partition, and consequently to a faster test for EA-equivalence.

In the case of $n = 10$, we only observe the "Gold-like" partition for all the ten known representatives from the infinite families. However, among the five new functions given in the dataset accompanying [1], we find three that have different (and pairwise distinct) spectra.

For odd dimensions ($n = 7$ and $n = 9$), we also compute the partitions induced by the known APN representatives, but these always yield a trivial partition of $\mathbb{F}_{2^n}$ into a zero and non-zero elements (even when we take into account the newly discovered APN classes from [1]).

Table 2 Computation time for reconstructing the inner permutation for $n = 8$

	1	2	3	4	5	6	7	8	9	10	11	12
1	60	751	749	751	751	751	751	751	750	751	753	751
2	739	59	744	743	742	743	743	743	745	745	742	742
3	809	809	106	808	808	808	810	829	818	807	814	832
4	775	776	778	77	778	785	786	784	809	782	789	778
5	766	766	766	767	66	766	766	766	766	788	769	769
6	775	773	769	769	768	66	769	773	768	769	769	769
7	779	778	778	777	779	778	73	778	778	778	777	778
8	778	779	779	779	778	779	778	73	835	771	772	774
9	777	776	776	776	776	776	776	776	73	776	776	776
10	773	774	774	775	776	774	780	781	776	73	775	776
11	778	775	775	775	777	774	774	774	773	774	73	774
12	782	776	776	776	776	777	777	776	776	776	777	73
13	774	775	776	773	771	769	769	770	770	769	770	769
14	782	783	783	783	783	786	781	785	786	784	783	785
15	778	773	781	779	773	775	775	774	775	775	774	774
16	775	775	775	775	775	775	775	776	776	775	775	776
17	778	778	778	778	778	778	778	778	778	778	778	777
18	766	766	766	767	766	766	779	782	772	766	766	768
19	767	767	766	767	767	767	767	767	767	766	767	767
20	779	779	779	778	779	779	778	779	778	778	778	778
21	770	770	770	770	770	770	770	770	770	770	770	770
22	769	769	769	769	769	769	769	769	769	768	769	769
23	753	753	753	754	754	753	753	754	754	753	754	753
	13	14	15	16	17	18	19	20	21	22	23	
1	751	751	752	752	751	752	751	751	751	752	916	
2	741	743	743	741	742	743	743	743	743	743	909	
3	812	810	813	815	809	809	809	809	809	809	971	
4	779	779	779	780	776	775	776	776	776	776	941	
5	773	772	770	772	769	770	769	769	772	771	942	
6	772	770	772	771	771	917	786	791	814	797	941	
7	777	777	776	779	778	778	779	778	776	779	972	
8	823	775	833	772	772	772	771	771	772	771	937	
9	776	775	776	783	793	780	776	778	774	774	939	
10	775	776	776	776	776	774	775	776	775	775	942	
11	778	775	775	775	775	775	775	775	775	775	941	
12	776	776	778	776	776	776	776	776	776	776	952	
13	72	770	769	770	769	769	769	780	781	787	940	
14	789	75	781	789	781	785	785	784	781	782	949	
15	775	774	73	773	773	775	774	777	773	777	942	
16	776	777	773	72	773	773	773	775	773	773	943	
17	777	777	777	778	73	777	778	777	779	775	942	
18	766	765	770	766	766	63	765	763	766	764	931	

Table 2 (continued)

	1	2	3	4	5	6	7	8	9	10	11	12
19	767	767	767	767	767	769	68	767	767	775	936	
20	778	778	778	779	779	778	779	67	780	774	940	
21	770	769	770	770	770	769	770	770	63	770	937	
22	769	769	769	769	769	768	769	781	769	65	934	
23	753	753	753	753	753	753	753	753	752	753	918	

6 Conclusion

We have introduced a family of invariants under EA-equivalence, and have shown how their values can be efficiently computed using the Walsh transform. We have experimentally observed that over $\mathbb{F}_{2^n}$ with even n, these invariants can be used to partition quadratic APN functions into small subclasses, thereby significantly facilitating their classification up to EA- and CCZ-equivalence. We have demonstrated how the values of these invariants can be used to restrict the values of the outer permutation A_1 in the relation $A_1 \circ F \circ A_2 + A = G$ for two given (n, m)-functions F and G, and have ran experiments in order to measure how much this approach reduces the search space. We have described how a variation of the same invariants can be used to restrict the values of the images of $\mathbb{F}_{2^n}$ under the inner permutation, A_2, and have combined the above into a computational test for deciding the EA-equivalence of any two (n, m)-functions F and G. Although slower than the standard test for CCZ-equivalence via the permutation equivalence of linear codes, our approach has the advantage that it is easily implementable on any programming language, and can be separated into a multitude of small, independent steps with concrete output, the majority of which can be naturally parallelized and run in different processes or on different computers. Furthermore, this is, to the best of our knowledge, the first efficient algorithm for directly testing the EA-equivalence of two given functions.

One direction for future work would be to investigate the invariants described in Section 2 more closely, and see whether they can be modified in order to provide more efficient restrictions. In the same vein, it would be interesting to investigate the functions for which our experimental results show a large number of choices for the outer permutation A_1 following the restriction described in Section 3, and to see whether some of these choices can be ruled out using some other criterion; this would directly impact the efficiency of the entire EA-equivalence test for these functions.

So far, we have implemented the algorithms described in Sections 3 and 4 in the *Magma* programming language [4] due to the ease of implementation. As pointed out above, our approach is quite simple, and does not depend on anything more complicated than computing linear combinations of binary vectors, and so it should be readily implementable in any general-purpose programming language. We expect that a careful implementation in an efficient language would further reduce the computational time needed for testing EA-equivalence, and make the method even more useful in practice.

Acknowledgements This research is supported by the Trond Mohn foundation. The author would like to thank the anonymous reviewers for their careful proofreading and helpful remarks.

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

References

1. Beierle, C., Leander, G.: New Instances of Quadratic APN Functions. arXiv:2009.07204 (2020)
2. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. J. Cryptol. **4.1**, 3–72 (1991)
3. Biryukov, A., De Cannière, C., An, B., Preneel, B.: A toolbox for cryptanalysis Linear and affine equivalence algorithms. International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin (2003)
4. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system I: The user language. J. Symbol. Comput. **24.3-4**, 235–265 (1997)
5. Browning, K., Dillon, J., Kibler, R., McQuistan, M.: APN Polynomials and related codes. Special volume of Journal of Combinatorics. Inf. Syst. Sci. **34**, 135–159 (2009)
6. Budaghyan, L., Carlet, C., Helleseth, T., Kaleyski, N.: On the distance between APN functions. IEEE Transactions on Information Theory (2020)
7. 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)
8. Budaghyan, L., Kazymyrov, O.: Verification of restricted EA-equivalence for vectorial boolean functions. International Workshop on the Arithmetic of Finite Fields. Springer, Berlin (2012)
9. Carlet, C., Charpin, P., Zinoviev, V.: Codes, bent functions and permutations suitable for DES-like cryptosystems. Des. Codes Cryptogr. **15.2**, 125–156 (1998)
10. Dinur, I., Shamir, A.: Breaking Grain-128 with dynamic cube attacks. International Workshop on Fast Software Encryption. Springer, Berlin (2011)
11. Dempwolff, U.: CCZ equivalence of power functions. Des. Codes Cryptogr. **86.3**, 665–692 (2018)
12. Edel, Y., Pott, A.: A new almost perfect nonlinear function which is not quadratic. Adv. Math. Comm. **3.1**, 59–81 (2009)
13. Edel, Y., Pott, A.: On the equivalence of nonlinear functions. Enhancibng cryptographic primitives with techniques from error correcting codes, vol. 23, pp. 87–103. NATO Sci. Peace Secur. Ser. D. Inf. Commun. Secur. Amsterdam: IOS (2009)
14. Knudsen, L.R.: Truncated and higher order differentials. International Workshop on Fast Software Encryption. Springer, Berlin
15. Matsui, M.: Linear cryptanalysis method for DES cipher. Workshop on the Theory and Application of of Cryptographic Techniques. Springer, Berlin (1993)
16. Nyberg, K.: Differentially uniform mappings for cryptography. Workshop on the Theory and Application of of Cryptographic Techniques. Springer, Berlin (1993)
17. Özbudak, F., Sınak, A., Yayla, O.: On verification of restricted extended affine equivalence of vectorial boolean functions. International Workshop on the Arithmetic of Finite Fields. Springer, Cham (2014)
18. Yoshiara, S.: Equivalences of quadratic APN functions. J. Algebraic Comb. **35.3**, 461–475 (2012)
19. Yu, Y., Wang, M., Li, Y.: A Matrix Approach for Constructing Quadratic APN Functions., Cryptology ePrint Archive: Report 2013/731