Deciding EA-equivalence via invariants

Nikolay Kaleyski
Department of Informatics
University of Bergen
Bergen, Norway
nikolay.kaleyski@uib.no

Abstract

We define a family of efficiently computable invariants under EA-equivalence, and observe that, unlike 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 over $\mathbb{F}_{2^n}$ with n even; to the best of our knowledge, this is the first algorithm for deciding EA-equivalence of any two functions.

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. An (n, m)-*function*, or *vectorial Boolean function*, is any mapping from $\mathbb{F}_{2^n}$ to $\mathbb{F}_{2^m}$. 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 is the *univariate representation* of F, which always exists, and is unique.

Vectorial Boolean functions are used in modern block ciphers in the role of "S-boxes", or "substitution boxes", and typically constitute the only non-linear part of the cipher. The security of the cipher thus directly depends on its underlying S-boxes. This motivates the study of the cryptographic properties of (n, n)-functions.

One such basic property is the algebraic degree. The *binary weight* of a positive integer i is the number of non-zero entries in its binary representation; for example, 19 is 10011 in binary, and 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*. A high algebraic degree is desirable, as it indicates good resistance to higher-order differential attacks.

In the following, we will frequently use multisets, i.e. unordered collections of elements in which (unlike in a set) the same element can occur multiple times. Formally, a multiset could be defined as a pair $M = (S, m)$ where S is an underlying set, and $m : S \to \mathbb{N}$

is a function such that $m(x)$ counts how many times an element $x \in S$ is contained in the multiset M; in other words, $m(x)$ is the *multiplicity* of x in M. We will not use this formal definition in the rest of the paper, but will frequently refer to the multiplicities of the elements in a multiset.

Two of the most important cryptographic properties of an (n, n)-function F are its differential uniformity and its nonlinearity. 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 $a, b \in \mathbb{F}_{2^n}$. The *differential spectrum* of F is the multiset $\{\delta_F(a, b) : a \in \mathbb{F}_{2^n}^*, b \in \mathbb{F}_{2^n}\}$. The largest value in the differential spectrum is denoted by δ_F, and is called the *differential uniformity* of F.

The value of δ_F should be as low as possible in order to resist differential cryptanalysis [2]. The minimum value of δ_F is clearly 2, and functions attaining this optimal value are called almost perfect nonlinear (APN). APN functions have been an object of intense study since their introduction by Nyberg in the 90's [22].

The *nonlinearity* $\mathcal{NL}(F)$ of F is 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 $F_c(x) = \mathrm{Tr}(cF(x))$, where $\mathrm{Tr} : \mathbb{F}_{2^n} \to \mathbb{F}_2$ is the *absolute trace* defined by $\mathrm{Tr}(x) = \sum_{i=0}^{n-1} x^{2^i}$ for any $x \in \mathbb{F}_{2^n}$. The nonlinearity should be high in order to resist linear cryptanalysis [21].

Since $\mathbb{F}_{2^n}$ can be identified with the vector space $\mathbb{F}_2^n$, some linear-algebraic notions can easily be adapted from $\mathbb{F}_2^n$ to the finite field $\mathbb{F}_{2^n}$. The *linear span* of $S = \{s_1, s_2, \ldots, s_k\} \subseteq \mathbb{F}_{2^n}$ (where k is a positive integer) is simply the set of all linear combinations of the elements in S, i.e. $\mathrm{Span}(S) = \{c_1 s_1 + c_2 s_2 + \cdots + c_k s_k : c_1, c_2, \ldots, c_k \in \mathbb{F}_2\}$. It is now 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 is the *Walsh transform* $W_F : \mathbb{F}_{2^n}^2 \to \mathbb{Z}$, which is the integer valued function associated with $F : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ given by the prescription $W_F(a, b) = \sum_{x \in F} \chi(bF(x) + ax)$, where $\chi(x) = (-1)^{\mathrm{Tr}(x)}$. The values of W_F are called *Walsh coefficients*, and the multiset $\{W_F(a, b) : a, b \in \mathbb{F}_{2^n}\}$ is called the *Walsh spectrum* of F. The multiset of the absolute values of F, i.e. the multiset $\{|W_F(a, b)| : a, b \in \mathbb{F}_{2^n}\}$, is called the *extended Walsh spectrum* of F.

The classification of (n, n)-functions is typically performed up to a suitable equivalence relation that preserves the properties being studied. The most general known equivalence relation preserving the differential uniformity and the nonlinearity is Carlet-Charpin-Zinoviev-equivalence, or CCZ-equivalence [10]. Two (n, n)-functions F and G are said to be *CCZ-equivalent* if there is an affine permutation $\mathcal{A}$ of $\mathbb{F}_{2^n}^2$ mapping the graph $\Gamma_F = \{(x, F(x)) : x \in \mathbb{F}_{2^n}\}$ of F to the graph Γ_G of G.

The CCZ-equivalence of two given functions F and G is usually decided via linear codes [17, 6]: two linear codes $\mathcal{C}_F$ and $\mathcal{C}_G$ are associated with F and G, respectively; then F and G are CCZ-equivalent if and only if $\mathcal{C}_F$ and $\mathcal{C}_G$ are equivalent. Testing the equivalence of linear codes has the advantage that it is usually already implemented in most mathematical software, such as the *Magma* programming language [4].

Unfortunately, testing CCZ-equivalence in this way can reliably be performed only over $\mathbb{F}_{2^n}$ with $n \leq 9$; 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: if the equivalence test fails, we have no way of determining whether this is due to insufficient memory, or due to a successfully completed exhaustive search.

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. Unfortunately, the Walsh spectra and differential spectra of all known APN functions are the same (with some rare exceptions), rendering these invariants nearly useless in practice. Other invariants, such as the Γ-rank and Δ-rank have been introduced [16], that can take different values for distinct CCZ-classes of functions. The drawback of these invariants is that they require significant computational resources, and take a long time to be computed: computing a single Γ-rank over $\mathbb{F}_{2^{10}}$ takes around 10 days on our server; on the other hand, computing Γ-ranks over $\mathbb{F}_{2^n}$ with $n > 10$ is impossible due to insufficient memory. For example, our server has around 500 gigabytes of memory available, which is insufficient for computing Gamma ranks for $n \geq 11$.

A special case of CCZ-equivalence is extended affine equivalence, or EA-equivalence. 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 such that

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

with A_1 and A_2 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 [8], but in certain cases, such as for quadratic and monomial functions, checking whether two functions (or, potentially, one function and the inverse of the other) are EA-equivalent is enough to decide CCZ-equivalence: two quadratic APN functions are CCZ-equivalent if and only if they are EA-equivalent [24]; and two power functions are CCZ-equivalent if and only if they are cyclotomic equivalent [12]. Recall that $F(x) = x^d$ and $G(x) = x^e$ over $\mathbb{F}_{2^n}$ are cyclotomic equivalent if $e \equiv 2^k d \mod (2^n - 1)$ or $e^{-1} \equiv 2^k d \mod (2^n - 1)$ for some non-negative integer k; 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 8000 distinct CCZ-equivalence classes) are CCZ-equivalent to a monomial or quadratic function, with only a single exception in dimension $n = 6$ [16]. Thus, being able to test EA-equivalence is virtually as useful as being able to test CCZ-equivalence, as far as the classification of APN functions is concerned.

The only currently known algorithm for computationally testing EA-equivalence also reduces the problem to an equivalence test of associated linear codes [17]. It is similar in principle to the CCZ-equivalence test described above: linear codes $\mathcal{C}'(F)$ and $\mathcal{C}'(G)$ are associated with the functions F and G; then F is EA-equivalent to G if and only if $\mathcal{C}'(F)$ is equivalent to $\mathcal{C}'(G)$. Checking the equivalence of $\mathcal{C}'(F)$ and $\mathcal{C}'(G)$ can take less time than checking the equivalence of $\mathcal{C}(F)$ and $\mathcal{C}(G)$; for instance, deciding that x^3 and x^{17} are EA-inequivalent over $\mathbb{F}_{2^9}$ takes around 26 seconds, as opposed to the approximately

118 seconds needed by the CCZ-equivalence test. Nonetheless, testing EA-equivalence via linear codes has some of the same drawbacks as the CCZ-equivalence test: in particular, it requires an existing implementation of an algorithm for testing the equivalence of linear codes; and while a positive result is essentially constructive, and implies the equivalence of the tested functions, it is not easy to conclusively decide whether a negative result is legitimate, or is caused by a lack of memory as in the case of the CCZ-equivalence test. Algorithms for testing EA-equivalence in some other special cases (grouped under the umbrella term "restricted EA-equivalence") have previously been studied in [3], [9], and [23].

Since EA-equivalence is a special case of CCZ-equivalence, any CCZ-invariant is also an EA-invariant. It is natural to expect, however, to have properties that are EA-invariant but not CCZ-invariant. One such invariant is the algebraic degree. Unfortunately, it is not particularly useful for classifying APN functions either, since nearly all known instances of APN functions are quadratic (or CCZ-equivalent to a quadratic function).

In this paper, we present an approach for computationally testing the EA-equivalence of two (n, n)-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'. When the dimension n of $\mathbb{F}_{2^n}$ is even, our approach allows the set of possible affine permutations A_1 to be drastically reduced (as compared 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, and is thus easily parallelizable. Unlike the CCZ-equivalence and EA-equivalence tests that 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 individual step of the algorithm has a concrete input and output that can be monitored and verified. This precludes the possibility of false negatives (or positives) as in the case of the CCZ-equivalence test.

We note that, due to space limitations, some of the proofs have been omitted.

2 A family of EA-invariants

Let n, k be positive integers, and $t \in \mathbb{F}_{2^n}$. Let us denote by $\mathcal{T}_k(t)$ the set $\mathcal{T}_k(t) = \{\{x_1, x_2, \ldots, x_k\} \subseteq \mathbb{F}_{2^n} \mid \sum_{i=1}^{k} x_i = t\}$. If A is an affine (n, n)-permutation, then the image of any k-set $\{x_1, x_2, \ldots, x_k\}$ from $\mathcal{T}_k(t)$ is a k-set $\{A(x_1), A(x_2), \ldots, A(x_k)\}$, the sum of whose elements is $A(x_1)+\cdots+A(x_k) = A(x_1+\cdots+x_k)$ for k odd, and $A(x_1)+\cdots+A(x_k) = A(x_1 + \cdots + x_k) + A(0)$ for k even. Furthermore, suppose that F is an (n, n)-function, and let $\Sigma_k^F(t)$ be the multiset $\Sigma_k^F(t) = \left\{\sum_{i=1}^{k} F(x_i) : \{x_1, x_2, \ldots, x_k\} \in \mathcal{T}_k(t)\right\}$. The multiplicities of the elements in $\Sigma_k^F(0)$ are then an EA-invariant for any even value of k.

Proposition 1. *Let F and G be (n, n)-functions with $A_1 \circ F \circ A_2 + A = G$ for some affine*

(n,n)-functions A_1, A_2, A with A_1, A_2 bijective. 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 $\Sigma_k^G(0)$ are the same when k is even.

Proof. Consider a k-set $\{x_1, x_2, \ldots, x_k\}$ 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 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 bijective function A_1. □

If the multiplicities of the elements in $\Sigma_k^F(t)$ are computed via the Walsh transform as shown below, the complexity of the computation does not depend on k.

Proposition 2. *Let F be an (n,n)-function, k be a positive integer, and t, s be elements in $\mathbb{F}_{2^n}$. Let $M_k^F(t,s)$ denote the multiplicity of s in $\Sigma_k^F(t)$, i.e. the number of k-sets $\{x_1, x_2, \ldots, x_k\} \subseteq \mathbb{F}_{2^n}$ such that $x_1+x_2+\cdots+x_k = t$ and $F(x_1)+F(x_2)+\cdots+F(x_k) = s$. Then*

$$2^{2n} M_k^F(t,s) = \sum_{a \in \mathbb{F}_{2^n}} \chi(at) \sum_{b \in \mathbb{F}_{2^n}} \chi(bs) W_F^k(a,b). \tag{2}$$

Proof. From the definition of the Walsh transform, we have

$$\sum_{a \in \mathbb{F}_{2^n}} \chi(at) \sum_{b \in \mathbb{F}_{2^n}} \chi(bs) W_F^k(a,b) =$$

$$\sum_{a,b \in \mathbb{F}_{2^n}} \sum_{x_1,\ldots,x_k \in \mathbb{F}_{2^n}} \chi(b(F(x_1) + F(x_2) + \cdots + F(x_k) + s) + a(x_1 + x_2 + \cdots + x_k + t)) =$$

$$\sum_{b \in \mathbb{F}_{2^n}} \sum_{x_1,\ldots,x_k \in \mathbb{F}_{2^n}} \chi(b(F(x_1) + F(x_2) + \cdots + F(x_k) + s)) \sum_{a \in \mathbb{F}_{2^n}} \chi(a(x_1 + x_2 + \cdots + x_k + t)).$$

The statement follows by recalling that $\sum_{a \in \mathbb{F}_{2^n}} \chi(ax)$ evaluates to 0 for any $0 \neq x \in \mathbb{F}_{2^n}$, and evaluates to 2^n for $x = 0$. □

Remark 3. We note that, in the case of APN functions, the multiset of the multiplicities of the elements in $\Sigma_3^F(0)$ is essentially the same as the multiset Π_F^0 studied in [7], up to replacing ordered triples of elements with unordered ones. The latter, given an (n,n)-function F, is defined as $\Pi_F^0 = \{\#\{a \in \mathbb{F}_{2^n} \mid (\exists x \in \mathbb{F}_{2^n})F(x) + F(a+x) + F(a) = b\} : b \in \mathbb{F}_{2^n}\}$, and is a CCZ-invariant for quadratic APN functions.

3 Guessing the outer permutation

Suppose F and G are EA-equivalent via $A_1 \circ F \circ A_2 + A = G$. 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 can assume $c_1 = 0$ 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 k we have $\Sigma_k^G(0) = \{L_1(a) : a \in \Sigma_k^F(0)\}$. This implies that if $L_1(x) = y$ for some $x, y \in \mathbb{F}_{2^n}$, then the multiplicity of x in $\Sigma_k^F(0)$ is the same as that of y in $\Sigma_k^G(0)$. In the following, let $A \oplus B$ denote the disjoint union of $A, B \subseteq \mathbb{F}_{2^n}$. If the elements of $\mathbb{F}_{2^n}$ are partitioned according to the multiplicities of F as $\mathbb{F}_{2^n} = K_1 \oplus K_2 \oplus \cdots \oplus K_s$ for some positive integer s, so that two elements belong to the same class K_i precisely when they have the same multiplicities in $\Sigma_k^F(0)$; and, similarly, as $\mathbb{F}_{2^n} = C_1 \oplus C_2 \oplus \cdots \oplus C_s$ according to the multiplicities of G (so that elements in K_i and C_i correspond to the same multiplicities for $1 \leq i \leq s$), then we must have $L_1(K_i) = C_i$ for all $1 \leq i \leq s$. We will say that such a permutation *respects* the two partitions of $\mathbb{F}_{2^n}$. This can be used to restrict the possible choices for L_1. Intuitively, the larger the number of classes s in the partition of $\mathbb{F}_{2^n}$, the more we restrict the choice of L_1. In particular, if all elements of $\mathbb{F}_{2^n}$ occur with the same multiplicity, no restriction occurs. This is clearly the case when F is a permutation; and, according to our experimental results, the same seems to be true for all known APN (n,n)-functions with odd n, regardless of whether they are bijective or not. We thus concentrate on fields $\mathbb{F}_{2^n}$ with even n.

All linear permutations L respecting the partitions $\mathbb{F}_{2^n} = K_1 \oplus \cdots \oplus K_s$ and $\mathbb{F}_{2^n} = C_1 \oplus \cdots \oplus C_s$ can be found by fixing a basis $B = \{b_1, b_2, \ldots, b_n\}$ of $\mathbb{F}_{2^n}$, and, for each linear combination $x = \sum_{i=1}^n c_i b_i$ for $c_i \in \mathbb{F}_2$, recording the index i of the class K_i containing x. The set of possible permutations L_1 is constructed by exhausting all possibilities for the images $L_1(b_i)$ of the elements b_i in the basis B, where $1 \leq i \leq n$. After assigning $b_1, \ldots b_j$ for some $1 \leq j \leq s$, we compute all elements $y = \sum_{i=1}^j c_i L_1(b_i)$ that can be expressed using the elements of the basis to which we have already assigned an image, and find the index i such that C_i contains y. If the corresponding preimage $x = \sum_{i=1}^j c_i b_i$ does not belong to K_i, we can immediately backtrack. A precise description of how to reconstruct all linear permutations that respect two given partitions is given in Algorithm 1. Note that a basis B and a set U of possible values for the images of B under L_1 are given as inputs; for example, if we can find a basis B with $B \subseteq K_i$ for some i, then we can take $U = C_i$. If no single K_i contains a basis, U can be taken as a union of several C_i.

The partitions $\mathbb{F}_{2^n} = K_1 \oplus K_2 \oplus \cdots \oplus K_s$ can be precomputed for representatives from e.g. all known EA-classes of APN functions. Precomputing the set of possible permutations L_1 is impossible, since both partitions (the one induced by F, and the one

Algorithm 1: Finding all linear permutations respecting a pair of partitions

Input : Two partitions $\mathbb{F}_{2^n} = K_1 \oplus K_2 \oplus \cdots \oplus K_s$ and $\mathbb{F}_{2^n} = C_1 \oplus C_2 \oplus \cdots \oplus C_s$ of the finite field $\mathbb{F}_{2^n}$, a basis $B = \{b_1, b_2, \ldots, b_n\}$ of $\mathbb{F}_{2^n}$, and a set U of possible values for the images of B

Output: All linear permutations L_1 of $\mathbb{F}_{2^n}$ 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 = n + 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
```

induced by G) need to be known. 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 $\mathbb{F}_{2^n}$ 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 for all $1 \leq i \leq s$ allows us to immediately reconstruct the rest.

Proposition 4. *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 $\mathbb{F}_{2^n}$ 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 if $\mathcal{P}$ is not empty, then it is a coset of $\mathcal{K}$.*

Among other things, Proposition 4 allows us to estimate the complexity of testing EA-equivalence between a known EA-representative F and a function G inducing a partition

n	ID	Partition	Time	Permutations	ID	Partition	Time	Permutations
6	1.1	2.28	$1, 21, 42$	1008	2.6	1.27	$1, 1, 1, 5, 10, 20, 26$	8
	1.2	1.69	$1, 21, 42$	336	2.7	1.30	$1, 1, 1, 5, 10, 20, 26$	10
	2.1	1.40	$1, 1, 1, 5, 10, 20, 26$	10	2.8	1.32	$1, 1, 1, 5, 10, 20, 26$	8
	2.2	1.63	$1, 21, 42$	336	2.9	1.31	$1, 2, 6, 9, 22, 24$	7
	2.3	2.21	$1, 21, 42$	1008	2.10	1.31	$1, 2, 6, 9, 22, 24$	8
	2.4	1.26	$1, 1, 1, 5, 10, 20, 26$	8	2.11	1.29	$1, 1, 1, 5, 10, 20, 26$	8
	2.5	1.35	$1, 6, 6, 10, 10, 15, 16$	60	2.12	2.48	$1, 1, 3, 4, 6, 7, 10, 32$	48
8	1.1	98.00	$1, 85, 170$	680	1.13	132.85	$1, 85, 170$	2
	1.2	293.4	$1, 85, 170$	680	1.14	127.11	1, 3, 4, 8, 11, 25, 28 36, 42, 46, 52	2
	1.3	316.74	$1, 85, 170$	8	1.15	144.52	$1, 85, 170$	1
	1.4	153.24	$1, 85, 170$	8	1.16	131.01	$1, 85, 170$	2
	1.5	165.88	$1, 85, 170$	4	1.17	190.65	$1, 85, 170$	2
	1.6	169.5	$1, 85, 170$	4	2.1	121.72	$1, 85, 170$	360
	1.7	131.34	$1, 85, 170$	1	3.1	170.47	$1, 85, 170$	4
	1.8	148.41	$1, 85, 170$	4	4.1	110.11	1, 1, 10, 12, 16, 20, 40, 44, 48, 64	16
	1.9	162.10	$1, 85, 170$	4	5.1	96.67	$1, 3, 8, 8, 8, 16, 20, 48, 48, 48, 48$	8
	1.10	131.67	$1, 85, 170$	2	6.1	105.65	1, 1, 16, 16, 18, 32, 32, 36, 48, 56	8
	1.11	173.12	$1, 85, 170$	4	7.1	142.68	$1, 85, 170$	680
	1.12	167.76	$1, 85, 170$	4				

Table 1: Computational experiments for finding the outer permutation

of $\mathbb{F}_{2^n}$ compatible with the one induced by F.

We have computed the groups $\mathcal{K}$ for all switching classes of APN functions over $\mathbb{F}_{2^n}$ with $n \in \{6, 8\}$ [16] in Table 1. The dimension n of $\mathbb{F}_{2^n}$ is given first. Functions are indexed in the second column in the same way as in [16]. Next, the time in seconds is given for partitioning $\mathbb{F}_{2^n}$ (either directly or using the Walsh transform, with the shorter time being taken into account) and for finding all linear permutations preserving the partition. The following column gives the sizes of the classes K_i in the partition, and the number of the permutations is given in the last column. For comparison, there are 27998208 linear permutations of $\mathbb{F}_{2^6}$, and 132640470466560 linear permutations of $\mathbb{F}_{2^8}$. Computing the Γ-rank of x^3 over $\mathbb{F}_{2^n}$ takes around 1.620 seconds for $n = 6$, and around 138 seconds for $n = 8$; the Δ-rank takes around 1.860 seconds for $n = 6$ and around 327 seconds for $n = 8$.

4 Guessing the inner permutation

Applying the inverse L_1^{-1} of L_1 to both sides of (1), we obtain

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

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

Let $c = A'(0)$ and $c_2 = A_2(0)$, and write $L_2 = A_2 + c_2$ and $A = L + c$ for the linear parts of A_2 and A, respectively. Substituting 0 for x in (3) yields $F(c_2) + c = G'(0)$. Thus, 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'). Having guessed c_2 and c', we rewrite (3) as

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

where $G''(x) = G'(x+c_2)+c$. Deciding EA-equivalence now amounts to checking whether a pair of linear functions L_2 and L' satisfying (4) exists.

To guess L_2, we observe that, given some k-set $\{x_1, x_2, \ldots, x_k\} \in \mathcal{T}_k(0)$, 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-set whose sum under G is t, then its image $L_2(x_i)$ under L_2 must be part of some k-set whose sum under F is also t.

Proposition 5. *Let F and G be (n,n)-functions for some positive integer n 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 $t \in \mathbb{F}_{2^n}$, denote $O_k^F(t) = \{\{x_1, x_2, \ldots, x_k\} \in \mathcal{T}_k(0) \mid F(x_1) + F(x_2) + \cdots + F(x_k) = t\}$. Then*

$$O_k^F(t) = \{\{L_2(x_1), \ldots, L_2(x_k)\} : \{x_1, \ldots, x_k\} \in O_k^G(t)\}. \tag{5}$$

Using (5) for $k = 3$, we can significantly reduce the *domains* of $L_2(x)$ for $x \in \mathbb{F}_{2^n}$, i.e. the sets of possible values that $L_2(x)$ can take. Many domains end up with only three elements. Guessing L_2 amounts to guessing its values on a basis B of $\mathbb{F}_{2^n}$; the elements of B can be chosen so that the Cartesian product of the respective domains is small. In most cases, we are thus left with only 3^n possibilities for L_2. Here, the sets $O_3^F(t)$ can be precomputed for known representatives F.

Algorithm 2 describes the approach for reconstructing L_2 from (3) suggested by the above considerations. We note that it returns all affine permutations A_2 for which $A = F \circ A_2 + G$ is affine. We can, however, terminate as soon as the first such permutation is found, since this already witnesses the EA-equivalence of F and G.

We ran 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 [16], we generated a random affine permutation A_2 and a random affine function A, and used Algorithm 2 to attempt to reconstruct A_2 and A from F and $G \circ A_2 + A$. We stop as soon as a single pair (A_2, A) solving $F \circ A_2 + A = G$ is found. For each combination of F and G, we generated 10 pairs of (A_2, A), and recorded the average running time.

Due to limited space, we omit a full table of the computational results; instead, we briefly describe the observed running times for $n = 8$ (in the case of $n = 6$, all tests take less than 0.2 seconds). When both F and G are from the same class, the running time is quite short, taking slightly more than 1 second in most cases, and around 0.8 seconds on average. The only exception is x^{57}, where this takes 9.18 seconds. When F and G are from different classes (so that no pair (A_2, A) exists), the running times are more uniform: the minimum is 7.3 seconds, the maximum is 11.3 seconds, and the average is 7.5 seconds. For comparison, the CCZ-equivalence test over $\mathbb{F}_{2^8}$ takes around 0.2 seconds, while the EA-equivalence test using linear codes takes around 0.8 seconds.

5 Further notes on the implementation

By combining the approaches from Sections 3 and 4, it is possible to test any pair (F, G) of (n,n)-functions for EA-equivalence. The method from Section 3 potentially allows the number of choices for A_1 to be significantly reduced with respect to exhaustive search.

The magnitude of the reduction depends on the image set of F. In the case of APN functions for n odd, there is no reduction at all; in the case of even n, the reduction is significant (in one case for $n = 8$, only a single choice is left for A_1). For each choice of A_1, the approach from Section 4 can be used to find the corresponding A_2 and A. This is done for every possible value of A_1, and hence the magnitude of the reduction of choices for A_1 directly affects the running time of the entire EA-equivalence test. For this reason, it is difficult to give a meaningful theoretical measure of the complexity of the entire EA-equivalence test. The worst-case complexity is the same as for exhaustive search, which is what we see for all APN functions that we have tested over $\mathbb{F}_{2^n}$ with odd n. In the case of even n, the reduction in the number of choices for A_1 is quite significant, varying between 1 and 1008 for representatives from the known switching classes of APN functions over $\mathbb{F}_{2^n}$ with $n \in \{6, 8\}$. The sizes of the classes in the partitions induced by F and G are often different, so that EA-equivalence can be ruled out without doing any further calculations: the 14 switching class representatives over $\mathbb{F}_{2^6}$, as well as the 23 switching class representatives over $\mathbb{F}_{2^8}$, fall into 5 distinct classes according to the sizes of the classes in the partitions that they induce. For comparison, the Γ-ranks allow the 14 representatives over $\mathbb{F}_{2^6}$ to be partitioned into 9 classes, and the 23 representation over $\mathbb{F}_{2^8}$ to be partitioned into 17 classes; and the Δ-ranks allow them to be partitioned into 3 and 9 classes over $\mathbb{F}_{2^6}$ and $\mathbb{F}_{2^8}$, respectively. Based on some limited experimental observations, it seems that there is no correlation between the sizes of the classes in the partition, and the values of the Γ- and Δ-rank: for instance, switching class representatives 2.5 and 2.6 (using the same indexing as in [16]) have the same Γ- and Δ-rank, but have different sizes of the partition classes; conversely, representatives 1.1 and 1.2 have the same partition sizes, but distinct Γ-ranks.

When the classes in the partitions do have the same sizes, we can estimate the running time for the EA-equivalence test based on the experimental results described above. The longest time occurs when F and G are inequivalent, since in this case we have to go over all possible triples (A_1, A_2, A); in the case of EA-equivalent functions, we stop immediately after finding the first triple.

Recall that the average time for reconstructing the inner permutation is around 7.5 seconds. From Table 1, the largest number of choices for A_1 varies between 1 and 680; we would thus expect the longest running time to be around $680 \times 7.5 = 5100$ seconds, i.e. slightly less that 1.5 hours. On the other hand, 18 out of the 23 classes leave us with no more than 8 choices for A_1; we would thus expect to spend around $8 \times 7.5 = 60$ seconds, i.e. around 1 minutes for testing EA-equivalence for these classes. For comparison, the linear code test for CCZ-equivalence takes slightly less than a second on average for disproving equivalence between two classes.

For any two choices of A_1, the inner permutation is reconstructed independently; this makes our approach naturally parallelizable. Another advantage is that many computational steps can be precomputed. Furthermore, each step produces concrete output that can be verified, stored, and processed at any time. This allows the equivalence test to be broken down into small, independent steps, allowing for a very flexible implementation.

6 Conclusion

We have introduced a family of invariants under EA-equivalence that can be efficiently computed via the Walsh transform. We have experimentally observed that these invariants partition quadratic APN (n, n)-functions into small subclasses, thereby significantly facilitating their classification up to EA- and CCZ-equivalence. We have demonstrated how these invariants can be used to restrict the values of A_1 in $A_1 \circ F \circ A_2 + A = G$ for given (n, n)-functions F and G, and have ran experiments to measure how much this approach reduces the search space. We have described how the values of the images of $\mathbb{F}_{2^n}$ under the inner permutation A_2 can be restricted, and have thus obtained a computational test for efficiently testing EA-equivalence of (n, n)-functions with even n. Our approach is slower than the CCZ-equivalence test via linear codes, but 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. We remark that the bottleneck for testing CCZ-equivalence via linear codes is the memory consumption, rather than the running time; since the algorithm proposed in this paper does not require significant memory, it can potentially be used to decide equivalence in dimensions higher than e.g. 10; an efficient implementation of the algorithm in a general purpose programming language, and detailed experimental results as to its limitations and capabilities remain a good subject for future work. To the best of our knowledge, this is the first efficient algorithm for directly testing EA-equivalence; the algorithm described in [17] for testing EA-equivalence via linear codes seems to be the only approach that predates ours.

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. It would also be interesting to investigate the functions for which a large number of choices for A_1 remain, and see whether some of these choices can be ruled out using some other criterion; this would directly impact the efficiency of the EA-equivalence test for these functions as a whole.

We have implemented the algorithms from Sections 3 and 4 in the *Magma* programming language [4] due to 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.

We expect that the algorithm will prove useful in the construction of new infinite families of quadratic APN functions: in order to show that a constructed family is truly "new", one typically wants to find at least one instance of the family (for some concrete dimension) that is inequivalent to any of the currently known families. This is very hard to do theoretically, and is usually accomplished computationally. Furthermore, the family is considered to be new even if its instances for some (but not all) dimensions are equivalent to those of other families. In particular, this means that there is currently no conclusive method of showing that a family is new, if the smallest dimension n for which its instances

are inequivalent to known ones is say $n \geq 11$: computing invariants such as the Γ- and Δ-rank is computationally infeasible, while the equivalence test via linear codes can give false negatives. We hope that the proposed algorithm would allow inequivalence to be established for higher dimensions than is currently possible.

Acknowledgements

This research is supported by the Trond Mohn foundation.

References

[1] Thomas Beth and Cunsheng Ding. "On almost perfect nonlinear permutations." Workshop on the Theory and Application of of Cryptographic Techniques. Springer, Berlin, Heidelberg, 1993.

[2] Eli Biham and Adi Shamir. "Differential cryptanalysis of DES-like cryptosystems." Journal of CRYPTOLOGY 4.1 (1991): 3-72.

[3] Alex Biryukov, Christophe De Cannière, An Braeken and Bart Preneel. "A toolbox for cryptanalysis: Linear and affine equivalence algorithms." International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2003.

[4] Wieb Bosma, John Cannon and Catherine Playoust. "The Magma algebra system I: The user language." Journal of Symbolic Computation 24.3-4 (1997): 235-265.

[5] Marcus Brinkmann and Gregor Leander. "On the classification of APN functions up to dimension five." Designs, Codes and Cryptography 49.1-3 (2008): 273-288.

[6] K. Browning, J. Dillon, R. Kibler and M. McQuistan. "APN polynomials and related codes." Special volume of Journal of Combinatorics, Information and System Sciences 34 (2009): 135-159.

[7] Lilya Budaghyan, Claude Carlet, Tor Helleseth, and Nikolay Kaleyski. "On the distance between APN functions." IEEE Transactions on Information Theory (2020).

[8] Lilya Budaghyan, Claude Carlet, and Alexander Pott. "New classes of almost bent and almost perfect nonlinear polynomials." IEEE Transactions on Information Theory 52.3 (2006): 1141-1152.

[9] Lilya Budaghyan and Oleksandr Kazymyrov. "Verification of restricted EA-equivalence for vectorial boolean functions." International Workshop on the Arithmetic of Finite Fields. Springer, Berlin, Heidelberg, 2012.

[10] Claude Carlet, Pascale Charpin, and Victor Zinoviev. "Codes, bent functions and permutations suitable for DES-like cryptosystems." Designs, Codes and Cryptography 15.2 (1998): 125-156.

[11] Joan Daemen and Vincent Rijmen. "The design of Rijndael: AES-the advanced encryption standard." Springer Science & Business Media, 2013.

[12] Ulrich Dempwolff. "CCZ equivalence of power functions." Designs, Codes and Cryptography 86.3 (2018): 665-692.

[13] Hans Dobbertin. "Almost perfect nonlinear power functions on GF (2/sup n/): the Welch case." IEEE Transactions on Information Theory 45.4 (1999): 1271-1275.

[14] Hans Dobbertin. "Almost perfect nonlinear power functions on GF (2n): the Niho case." Information and Computation 151.1-2 (1999): 57-72.

[15] Hans Dobbertin. "Almost perfect nonlinear power functions on GF (2 n): a new case for n divisible by 5." Finite Fields and Applications. Springer, Berlin, Heidelberg, 2001. 113-121.

[16] Yves Edel and Alexander Pott. "A new almost perfect nonlinear function which is not quadratic." Adv. in Math. of Comm. 3.1 (2009): 59-81.

[17] Yves Edel and Alexander Pott. "On the equivalence of nonlinear functions." Enhancing cryptographic primitives with techniques from error correcting codes. Vol. 23. NATO Sci. Peace Secur. Ser. D. Inf. Commun. Secur. Amsterdam: IOS (2009): 87-103.

[18] Robert Gold. "Maximal recursive sequences with 3-valued recursive cross-correlation functions (Corresp.)." IEEE transactions on Information Theory 14.1 (1968): 154-156.

[19] Heeralal Janwa and Richard M. Wilson. "Hyperplane sections of Fermat varieties in P 3 in char. 2 and some applications to cyclic codes." International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes. Springer, Berlin, Heidelberg, 1993.

[20] Tadao Kasami. "The weight enumerators for several classes of subcodes of the 2nd order binary Reed-Muller codes." Information and Control 18.4 (1971): 369-394.

[21] Mitsuru Matsui. "Linear cryptanalysis method for DES cipher." Workshop on the Theory and Application of of Cryptographic Techniques. Springer, Berlin, Heidelberg, 1993.

[22] Kaisa Nyberg. "Differentially uniform mappings for cryptography." Workshop on the Theory and Application of of Cryptographic Techniques. Springer, Berlin, Heidelberg, 1993.

[23] Ferruh Özbudak, Ahmet Sınak, and Oğuz Yayla. "On verification of restricted extended affine equivalence of vectorial boolean functions." International Workshop on the Arithmetic of Finite Fields. Springer, Cham, 2014.

[24] Satoshi Yoshiara. "Equivalences of quadratic APN functions." Journal of Algebraic Combinatorics 35.3 (2012): 461-475.

Algorithm 2: Reconstructing the inner permutation A_2

Input : Two (n, n)-functions F and G with $F(0) = G(0) = 0$
Output: All affine permutations A_2 of $\mathbb{F}_{2^n}$ 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 $\{x_1, x_2\} \in \mathbb{F}_{2^n}^2$ **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 $\{x_1, x_2\} \in \mathbb{F}_{2^n}^2$ **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 $x \in \mathbb{F}_{2^n}$ into $x_1, x_2, \ldots, x_{2^n}$, so that
$i < j \implies \#\mathcal{D}(x_i) \leq \#\mathcal{D}(x_j)$;
$B \leftarrow \emptyset$ (basis) ;
Results $\leftarrow \emptyset$;
for $i = 1, 2, \ldots, 2^n$ **do**
  if $x_i \notin \mathrm{Span}(B)$ **then**
    $B \leftarrow B \cup \{x_i\}$;
    if $\#B = n$ **then**
      break ;
    end
  end
end
for $c_2 \in \mathbb{F}_{2^n}$ **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 $B = (b_1, \ldots, b_n)$;
    $A_2 \leftarrow L_2 + c_2$;
    $A \leftarrow F \circ A_2 + G$;
    if $\deg(A) \leq 1$ **then**
      Results $\leftarrow$ *Results* $\cup \{(A_2, A + F(c_2))\}$;
    end
  end
end
return *Results*