Deciding and reconstructing linear equivalence of uniformly distributed functions

Ivana Ivkovic and Nikolay Kaleyski

Department of Informatics, University of Bergen
(ivana.ivkovic@student.uib.no, nikolay.kaleyski@uib.no)

Abstract

We describe an efficient algorithm for testing and recovering linear equivalence between a pair of k-to-1 discrete functions with a specific structure. In particular, for $k = 3$ this applies to many APN functions over fields of even characteristic, and for $k = 2$ this applies to all known planar functions over fields of odd characteristic. Our approach is significantly faster than all known methods for testing equivalence, and allows linear equivalence to be tested in practice for dimensions much higher than what has been possible before (for instance, we can efficiently test equivalence for $n = 12$ or $n = 14$ in the case of 3-to-1 APN functions over $\mathbb{F}_{2^n}$, and for $n = 8$ or $n = 9$ in the case of 2-to-1 planar functions over $\mathbb{F}_{3^n}$ within a few minutes even in the worst case). We also develop supplementary algorithms allowing our approach to be extended to the more general case of EA-equivalence. Classifying 3-to-1 APN functions over $\mathbb{F}_{2^n}$ for dimensions as high as $n = 14$ up to EA-equivalence can be performed in a matter of minutes using the developed framework.

We introduce the notion of left and right self-equivalence orbits, explain how it can be used to reduce the computation time for testing equivalence even further, and provide an algorithm for efficiently partitioning the finite field $\mathbb{F}_{p^n}$ into orbit representatives with respect to a given function. We observe that this approach is most useful in the case of 2-to-1 planar functions, and compute the orbit partitions for representatives from all known CCZ-classes of 2-to-1 planar functions over $\mathbb{F}_{3^n}$ for $n \leq 6$. We also demonstrate that the orbit structure induced by a given function is invariant under linear equivalence, and that it can be used as an invariant to distinguish inequivalent planar functions over $\mathbb{F}_{3^n}$ more efficiently than all other known invariants.

Using the developed algorithms, we classify all known 3-to-1 quadratic APN functions over $\mathbb{F}_{2^{12}}$ up to CCZ-equivalence. Such a classification was only known for $\mathbb{F}_{2^n}$ with $n \leq 10$ before, since it was impossible to test functions over $\mathbb{F}_{2^{12}}$ for equivalence using the existing methods. Based on the computations needed to perform this classification, we provide a summary of the observed running times, showing that our approach is significantly more efficient than all previously known methods.

I. Introduction

Let p be a prime number and n be a positive integer; we denote by $\mathbb{F}_{p^n}$ the finite field with p^n elements. An (n, m, p)-function, or discrete function, is a mapping from the vector space $\mathbb{F}_p^n$ to the vector space $\mathbb{F}_p^m$, where n and m are positive integers. Since the vector space $\mathbb{F}_p^n$ can be identified with the finite field $\mathbb{F}_{p^n}$, we can also see (n, m, p)-functions as mappings from the finite field $\mathbb{F}_{p^n}$ to the finite field $\mathbb{F}_{p^m}$.

In the particular case when $p = 2$, we refer to $(n, m, 2)$-functions simply as (n, m)-functions, or as vectorial Boolean functions. When, in addition, $m = 1$, $(n, 1)$-functions are called Boolean functions. Since any element of $\mathbb{F}_2^n$ or, equivalently, $\mathbb{F}_{2^n}$, can be seen as a binary string on n bits, any operation on any kind of data can be represented as an (n, m)-function for an appropriate choice of n and m. This is why vectorial Boolean functions naturally appear in many contexts within computer science, applied mathematics, and related areas. In particular, vectorial Boolean functions are used extensively in symmetric cryptography: virtually all modern block ciphers, for instance, incorporate vectorial Boolean functions (under the name S-boxes) as fundamental building blocks in their design, and the security and efficiency of the resulting algorithm directly depends on the properties of the underlying functions. Classes of cryptographically optimal functions (such as APN functions or AB functions) also correspond to important objects in other areas of mathematics and computer science (e.g. linear codes or combinatorial designs). The study of cryptographically strong (n, m)-functions is thus not only of practical importance for the construction of secure and reliable cryptosystems, but is also of interest from the point of view of other areas and disciplines. For more details on vectorial Boolean functions and their applications in cryptography and other areas, we refer the reader to the recent monograph [13].

In the case of odd characteristic, (n, m, p)-functions are also of significant practical and theoretical interest, although their connection to cryptography is perhaps somewhat less known than in the case of Boolean functions. The connections to other structures and objects is, on the contrary, more clearly evinced in the case of (n, m, p)-functions, where the correspondence between cryptographically optimal planar functions and commutative semifields was utilized in [9] to construct the first infinite family of commutative semifields over $\mathbb{F}_{p^n}$ for any odd characteristic p since the early 50's. We refer the reader to the survey [26] for more details on planar functions.

We remark that one of the most important and well-studied cases of (n, m, p)-functions is when $n = m$, i.e. when the domain and codomain of the functions are the same. For one, this is due to the fact that replacing a sequence of digits with another sequence of the same length is arguably the most natural case in the context of cryptography; for another, in the case when $n = m$, the (n, m, p)-functions have a very convenient representation as polynomials over $\mathbb{F}_{p^n}$ that can be used to understand their properties and obtain mathematical and computational constructions of such functions.

The above considerations have resulted in a pronounced interest in finding new instances of cryptographically strong discrete functions and investigating their properties. A significant issue in these investigations is the very large number of discrete functions: it is easy to see that there are $(p^m)^{p^n}$ (n,m,p)-functions, and even for relatively small values of n, m, and p this number becomes prohibitively large. For this reason, (n,m,p)-functions are considered up to some appropriate equivalence relation that preserves the properties of interest. For instance, the class of APN functions is typically studied up to a relation called CCZ-equivalence, since CCZ-equivalence leaves the property of being APN invariant, i.e. if F and G are CCZ-equivalent discrete functions, then F is APN if and only if G is APN.

While the approach of considering discrete functions up to equivalence significantly reduces the number of instances that have to be considered and makes their study and classification manageable, it introduces other problems; namely, the issue of deciding whether two given functions are equivalent. For instance, a newly discovered APN function is only considered to be genuinely new if it is not CCZ-equivalent to any of the currently known ones. Thus, showing that two functions are inequivalent is a crucial part of any new construction of discrete functions. Conversely, equivalence relations can be used constructively to obtain simpler representations of functions, or functions possessing other desirable properties; perhaps most famously, CCZ-equivalence was used in this way to construct the only known instance of an APN permutation on an even number of bits [5]. Similarly, constructions or properties that are difficult to observe for some particular function may be more tractable for a different function that is equivalent to it. For this reason, it is often useful not only to decide whether two functions are equivalent or not, but to reconstruct the exact form of this equivalence.

Unfortunately, while the definitions of most equivalence relations used in practice are simple, testing the equivalence between two given functions turns out to be a very difficult computational problem. At the moment, the only known way of efficiently testing CCZ-equivalence for an arbitrary pair of functions F, G is via the equivalence of linear codes [19]. This approach has many disadvantages, including a high time and memory complexity, and the possibility to get false negatives.

In the case of certain classes of functions, CCZ-equivalence reduces to simpler equivalence relations, such as EA-equivalence, affine equivalence and linear equivalence. This is the case for e.g. quadratic APN functions (where it is enough to consider EA-equivalence), planar functions (where it is enough to consider affine equivalence), and quadratic planar functions (where it suffices to consider linear equivalence). Unfortunately, no efficient algorithms (besides a similar approach based on linear codes, which has the same shortcomings as the one for CCZ-equivalence) are known for testing these relations in the general case either. At present, we know of one algorithm for testing EA-equivalence between quadratic (n,m)-functions [12], and one algorithm for testing EA-equivalence between any pair of (n,m)-functions (regardless of their degree), which however is not efficient for some particular classes of functions (such as permutations and AB functions). Computational approaches for resolving some particular cases of EA-equivalence have been considered [11], [25] but these assume that we already have some information about the tentative equivalence between the given functions, or that we can restrict the form of this equivalence. Similarly, there is an algorithm for testing affine equivalence and linear equivalence [3] between permutations, but to the best of our knowledge there is no efficient way of doing this for non-bijective functions.

Recall that an (n,n,p)-function F is called k-to-1 if every non-zero element in its image set has precisely k pre-images. In this paper, we consider a subclass of k-to-1 functions that obey a particular structural property, namely that there is set of elements $U \subseteq \mathbb{F}_{p^n}^*$ such that $F(ux) = F(u'x)$ for any $u, u' \in U$ and any $x \in \mathbb{F}_{p^n}$. We will refer to such functions as *uniformly distributed*. We can readily observe that many important classes of cryptographically optimal functions have this property: in particular, most of the known infinite families of quadratic APN functions are 3-to-1 with $U = \mathbb{F}_{2^2}^*$; and all known planar functions, are 2-to-1 functions with $U = \{\pm 1\}$. Being able to efficiently decide equivalence for these two specific cases is a matter of significant practical importance.

We present an efficient algorithm for testing and reconstructing the linear equivalence between a pair of uniformly distributed functions F and G. In a certain sense, this approach can be seen as a generalization of the method from [3] for testing linear and affine equivalence between 1-to-1 functions. Our approach is significantly faster than all existing methods, and can be easily implemented in any general purpose programming language since it does not require anything more complicated than basic arithmetic. The memory consumption is negligible, which allows it to be used with (n,n,p)-functions with values of n much higher than what has been possible before. We also provide auxiliary algorithms that can be used to extend the test of linear equivalence to a test of affine equivalence and EA-equivalence, and that allow to check whether a given (n,n)-function that is not 3-to-1 is EA-equivalent to a 3-to-1 function. Together with the ortho-derivatives from [12] used as an invariant, this becomes the fastest method known to date for deciding and recovering EA-equivalence for quadratic 3-to-1 APN functions, and for quadratic planar functions.

The paper is organized as follows. In Section II, we present the background and preliminaries for the following discussion. In Section III, we present our main algorithm for testing linear equivalence between a given pair of uniformly distributed k-to-1 functions. In Section IV, we describe the auxiliary algorithms allowing us to extend Algorithm 1 to a test for EA-equivalence in the case when one of the function is not necessarily uniformly distributed, or not necessarily 3-to-1. In Section V, we introduce the notion of "orbits" corresponding to a given function, and describe how partitioning the finite field into orbits can be used to speed up equivalence tests, especially in the negative case (when the tested functions are inequivalent). We demonstrate that the structure of the orbits is invariant under linear equivalence, and can be used to distinguish between CCZ-inequivalent planar functions more efficiently than the currently known invariants. We also describe a procedure for computing the orbits, and list orbit representatives for instances from the CCZ-classes of all known planar functions over $\mathbb{F}_{3^n}$ for $3 \leq n \leq 6$. In Section VI, we describe some technical details of our implementation, and give some sample running times illustrating the efficiency of our algorithms. As an application, in Section VII, we classify all known uniformly distributed quadratic 3-to-1 APN functions over $\mathbb{F}_{2^{12}}$ up to CCZ-equivalence; without Algorithm 1, this would have been impossible to do since the previously known methods for testing equivalence are not efficient enough in terms of time and memory. Finally, in Section VIII, we give some concluding remarks and discuss possible directions for future work.

II. Preliminaries

Let $\mathbb{N} = \{1, 2, \dots\}$ be the set of natural numbers, $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$ be the set of all integers, and $\mathbb{C}$ be the field of complex numbers. Let p be a prime, and $n \in \mathbb{N}$ be a positive integer. We denote by $\mathbb{F}_{p^n}$ the finite field with p^n elements, and by $\mathbb{F}_p^n$ the vector space of degree n over the prime field $\mathbb{F}_p$. The multiplicative group of $\mathbb{F}_{p^n}$ is denoted by $\mathbb{F}_{p^n}^*$. More generally, given a set S, we will denote by S^* the subset of non-zero elements $S^* = S \setminus \{0\}$.

For any prime p and $m, n \in \mathbb{N}$ with $m \mid n$, we denote the trace from $\mathbb{F}_{p^n}$ onto $\mathbb{F}_{p^m}$ by

$$\mathrm{Tr}_m^n(x) = \sum_{i=0}^{n/m-1} x^{p^{mi}};$$

we will write Tr_n as shorthand for the absolute trace Tr_1^n, and sometimes simply Tr if the dimension is understood from the context.

For any set S, we denote by $\#S$ the cardinality of S, while $|s|$ will be used for the absolute value of a number s. In addition to sets, we will consider **multisets** which, intuitively, are sets whose elements can appear multiple times (in contrast to the ordinary notion of a set, which either contains some given element or does not). The **multiplicity** of an element s in a multiset M is the number of times that s occurs in M. We note that a multiset can formally be defined as a pair (S, μ) of a set S and a map $\mu : S \to \mathbb{N}$ giving the multiplicity of each element in S, but we do not consider it necessary to take this more formal approach here. When expressing multisets, we will write them in square brackets in order to differentiate them from ordinary sets; for instance, $[a, b, a, a, c]$ denotes the multiset containing the element a three times, the element b once and the element c once. We will also sometimes write $[a^3, b, c]$ as shorthand for this multiset (giving the multiplicities of the elements in superscript).

A. *Vectorial functions and their representations*

Let n and m be natural numbers. Any mapping from $\mathbb{F}_p^n$ to $\mathbb{F}_p^m$ is called an $(\mathbf{n}, \mathbf{m}, \mathbf{p})$**-function**, or a **discrete function**. When $p = 2$, we call a mapping from $\mathbb{F}_{2^n}$ to $\mathbb{F}_{2^m}$ an $(\mathbf{n}, \mathbf{m})$**-function**, or a **vectorial Boolean function**. When, in addition, $m = 1$, we refer to $(n, 1)$-functions as **Boolean functions**. Note that any (n, m, p)-function F can be represented as a vector $F = (f_1, f_2, \dots, f_m)$ of m $(n, 1, p)$-functions $f_1, f_2, \dots, f_m : \mathbb{F}_p^n \to \mathbb{F}_p$; the functions $f_1, f_2, \dots, f_n$ are then called the **coordinate functions** of F. The non-zero linear combinations of the coordinate functions are called the **component functions** of F. Thus, every coordinate function is a component function but not vice-versa.

The **image set** of an (n, m, p)-function F is the set $\mathrm{Im}(F) = \{F(x) : x \in \mathbb{F}_p^n\}$. The **preimage set** of $y \in \mathrm{Im}(F)$ is the set $F^{-1}(y) = \{x \in \mathbb{F}_p^n : F(x) = y\}$. We say that F is a **k-to-1 function** if $F(0) = 0$ and $\#F^{-1}(y) = k$ for every $0 \neq y \in \mathrm{Im}(F)$. We say that an (n, n, p)-function F is a **permutation** of $\mathbb{F}_p^n$ if $\#\mathrm{Im}(F) = 2^n$, i.e. if F is a 1-to-1 function.

Discrete functions are most often represented by truth tables, polynomials in n variables or univariate polynomials. The **truth table (TT)**, or **look-up table (LUT)**, is the simplest possible representation: the TT is simply a list of all the values $F(x)$ for all possible inputs $x \in \mathbb{F}_p^n$. This representation is very easy to implement, and evaluating the function (that is, computing $F(x)$ given $x \in \mathbb{F}_{p^n}$) amounts to simply indexing the array representing the TT, which is a very fast operation. The TT does have some drawbacks, however, such as a memory consumption that grows very quickly with n and m, and the difficulty of observing structural properties and patterns directly from the TT. For this reason, polynomial representations such as the ANF or the univariate representation are frequently used in the literature in lieu of the TT.

The **algebraic normal form (ANF)** of an (n, m, p)-function F is the polynomial in n variables over $\mathbb{F}_p^m$

$$F(x_1, x_2, \dots, x_n) = \sum_{I \subseteq \mathcal{P}(\{1,2,\dots,n\})} a_I \prod_{i \in I} x_i,$$

where $\mathcal{P}(\{1, 2, \dots, n\})$ is the power set of $\{1, 2, \dots, n\}$, and $a_I \in \mathbb{F}_p^m$ for all $I \subseteq \mathcal{P}(\{1, 2, \dots, n\})$. The ANF of any (n, m, p)-function always exists, and is uniquely defined. While the ANF can have up to p^n non-zero coefficients, and so does not have to be more compact than the TT in general, it does allow many (n, m, p)-functions to be represented in a more concise way when the majority of the coefficients are equal to zero (and so do not have to be explicitly written). This comes at the cost of performance, since computing $F(x)$ for some given $x \in \mathbb{F}_p^n$ now requires a series of arithmetic operations to be performed. Nonetheless, this trade-off is usually desirable, especially in the case of (n, m, p)-functions for large values of n and m, where storing the entire TT of the function in memory would be problematic. Perhaps more importantly, the ANF allows us to define structural properties such as the algebraic degree. The **algebraic degree** $\deg(F)$ of a function F is defined as the degree of the ANF of F as a multivariate polynomial. The algebraic degree is significant both in the cryptographic sense, as a property indicating resilience to higher-order differential attacks [18], [23], as well as a structural property defining classes of discrete functions. In particular, we define the **affine** functions as the (n,m,p)-functions F with $\deg(F) \leq 1$. The **linear** functions are those affine functions F for which $F(0) = 0$, i.e. that have a zero constant term. Much as the name suggests, an affine function A satisfies $A(x+y+z) = A(x)+A(y)+A(z)$ for any $x,y,z \in \mathbb{F}_p^n$, while a linear function L satisfies $L(x+y) = L(x)+L(y)$ for any $x,y \in \mathbb{F}_p^n$. Other important classes of functions include the **quadratic** and **cubic** functions that are defined as those or which $\deg(F) = 2$ and $\deg(F) = 3$, respectively.

When $n = m$, it is possible to represent (n,m,p)-functions as polynomials over $\mathbb{F}_{p^n}$. The **univariate representation** of $F : \mathbb{F}_{p^n} \to \mathbb{F}_{p^n}$ is the polynomial

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

where $c_i \in \mathbb{F}_{p^n}$ for $i = 1, 2, \dots, 2^n-1$. Any (n,n,p)-function has a unique univariate representation. We note that the univariate representation can be generalized to (n,m,p)-functions with $m \mid n$, although in this case some additional restrictions need to be imposed on the polynomial in order to guarantee uniqueness.

There are also other representations of discrete functions, especially vectorial Boolean functions [29], [31], [32], [28] that are of limited interest to the present study. We refer the reader to [13] for a detailed discussion.

B. *Derivatives of vectorial Boolean functions*

The **derivative** of an (n,m,p)-function F in direction $a \in \mathbb{F}_{p^n}$ is the function $D_aF(x) = F(a+x) - F(a)$. A related function is $\Delta_aF(x) = F(a+x) - F(x) - F(a) + F(0)$, which is referred to as a symplectic form when F is quadratic. Cryptographic properties such as the differential uniformity of F, as well as it being APN or planar, can be equivalently defined using D_aF and Δ_aF. The latter is sometimes more convenient since it always has zero constant term, and since it is symmetric in a and x. In particular, if F is quadratic (which is by far the most frequently encountered case in the study of i.a. planar and APN functions), then Δ_aF is linear.

Let $\delta_F(a,b)$ denote the number of solutions $x \in \mathbb{F}_p^n$ to the equation $D_aF(x) = b$ for some $a \in \mathbb{F}_2^n, b \in \mathbb{F}_2^m$, i.e.

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

The **differential uniformity** of F is then defined as

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

The multiset $[\delta_F(a,b) : a,b \in \mathbb{F}_{p^n}, a \neq 0]$ of all values of $\delta_F(a,b)$ is called the **differential spectrum** of F.

The larger the value of δ_F, the more vulnerable F is to differential cryptanalysis [2], which is one of the most efficient cryptanalytic attacks known to date against block ciphers. Cryptographically strong functions should therefore have a value of δ_F that is as low as possible. The functions possessing the optimal value $\delta_F = 1$ are called **perfect nonlinear (PN)**, or **planar**. Unfortunately, PN functions exist only in the case of odd p since when $p = 2$ we have $D_aF(x) = D_aF(a+x)$ for any $a, x \in \mathbb{F}_{2^n}$, and so the differential uniformity is always even. If $\delta_F = 2$, we say that F is **almost perfect nonlinear (APN)**, which is the optimal case for even characteristic.

The **differential set** H_aF of an (n,m)-function F in direction $a \in \mathbb{F}_2^n$ is simply the image set of the derivative D_aF, that is

$$H_aF = \operatorname{Im}(D_aF) = \{D_aF(x) : x \in \mathbb{F}_p^n\}.$$

We can easily see that F is PN if and only if $H_aF = \mathbb{F}_{p^n}$ for every $a \in \mathbb{F}_{p^n}^*$; and that F is APN if and only if $\#H_aF = 2^{n-1}$ for every $a \in \mathbb{F}_{2^n}^*$. In particular, a function is PN if and only if all of its derivatives D_aF for $a \in \mathbb{F}_{p^n}^*$ are permutations.

C. *The Walsh transform*

The Walsh transform of an (n,m,p)-function F is the function $W_F : \mathbb{F}_p^n \times \mathbb{F}_p^m \to \mathbb{C}$ defined by

$$W_F(a,b) = \sum_{x \in \mathbb{F}_2^n} \zeta^{b \cdot F(x) + a \cdot x},$$

where "$\cdot$" is a scalar product[1] on $\mathbb{F}_p^m$ and $\mathbb{F}_p^n$, respectively (the dimension being understood from the context), and $\zeta \in \mathbb{C}$ is a p-th root of unity. When $n = m$, the Walsh transform $W_F : \mathbb{F}_{p^n}^2 \to \mathbb{C}$ of an (n,n,p)-function F can equivalently be written as

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

[1]A scalar product on $\mathbb{F}_p^n$ is a symmetric bivariate function on $\mathbb{F}_p^n$ such that $x \mapsto a \cdot x$ is a non-zero linear form for any $0 \neq a \in \mathbb{F}_p^n$. Using the identification of the vector space $\mathbb{F}_p^n$ with the finite field $\mathbb{F}_{p^n}$, this is typically defined as $x \cdot y = \operatorname{Tr}(xy)$, with the product xy being computed in the finite field $\mathbb{F}_{p^n}$, and then mapped to $\mathbb{F}_p$ via the absolute trace function.

where $\chi : \mathbb{F}_{p^n} \to \mathbb{C}$ is the canonical additive character of $\mathbb{F}_{p^n}$ defined by $\chi(x) = \zeta^{\mathrm{Tr}(x)}$. The values of the Walsh transform W_F are called the **Walsh coefficients** of F. The multiset of all Walsh coefficients is called the **Walsh spectrum** of F; and the multiset of their absolute values is called the **extended Walsh spectrum** of F and denoted by $\mathcal{W}_F$; symbolically:

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

The Walsh transform is known to be invertible, i.e. knowing the values $W_F(a,b)$ for all a and b allows us to uniquely reconstruct the function F. In this way, W_F can be seen as yet another possible representation of (n,m,p)-functions. Furthermore, many important properties of discrete functions, including their differential uniformity, can be characterized using the values of their Walsh transform. The extended Walsh spectrum is also a well-known invariant under CCZ-equivalence, and can be used to distinguish between inequivalent functions; the approach of computing the differential spectra of two quadratic APN functions has been shown to be particularly effective for demonstrating their inequivalence [12].

D. Equivalence relations

CCZ-equivalence [14], or Carlet-Charpin-Zinoviev equivalence, is the most general known equivalence relation that preserves the differential uniformity. For this reason, APN functions and planar functions, among others, are classified up to CCZ-equivalence.

The graph Γ_F of an (n,m,p)-function F is the set $\Gamma_F = \{(x, F(x)) : x \in \mathbb{F}_p^n\} \subseteq \mathbb{F}_p^n \times \mathbb{F}_p^m$. Note that the set $\mathbb{F}_p^n \times \mathbb{F}_p^m$ can be naturally identified with $\mathbb{F}_p^{n+m}$, and so the set of pairs Γ_F can be seen as a set of elements from $\mathbb{F}_p^{n+m}$. If F and G are two (n,m,p)-functions, we say that they are **CCZ-equivalent** if there exists an affine permutation A of $\mathbb{F}_p^{n+m}$ mapping Γ_F to Γ_G, i.e. such that $A(\Gamma_F) = \Gamma_G$.

Two (n,m,p)-functions F and G are called **EA-equivalent** if there exist affine permutations A_1 and A_2 of $\mathbb{F}_p^m$ and $\mathbb{F}_p^n$, respectively, and an affine (n,m,p)-function A, such that

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

Any two functions that are EA-equivalent are also CCZ-equivalent. CCZ-equivalence, however, is strictly more general than EA-equivalence and taking inverses of permutations [7]. On the other hand, if two quadratic APN functions are CCZ-equivalent, then they are also necessarily EA-equivalent [30]; and the same is true for planar functions [9] (in fact, for planar functions, CCZ-equivalence coincides with even less general equivalence relations; we discuss this in more detail below). We also stress that the vast majority of the known planar and APN functions are quadratic, and so in practice the case of testing CCZ-equivalence more often than not reduces to that of the testing EA-equivalence.

Further equivalence relations can be obtained by imposing additional constraints on the functions A_1, A_2 and A from the definition of EA-equivalence. We say that F and G are **affine equivalent** if $A = 0$ in (1); and we say that F and G are **linear equivalent** if $A = 0$ and A_1 and A_2 are linear. We note that two planar functions are CCZ-equivalent if and only if they are affine equivalent; and that two quadratic planar functions are CCZ-equivalent if and only if they are linearly equivalent) [9]. Testing linear and affine equivalence, while seemingly more specialized, is at the core of our approach to testing EA-equivalence.

One more notion of equivalence that we will need is the special case of EA-equivalence when both A_1 and A_2 are identity permutations, in which case (1) becomes simply $F + A = G$. This very specialized equivalence relation has, to the best of our knowledge, no common name in the literature; we will call it additive equivalence, and will then say that F and G are **additive equivalent**.

In the case of quadratic planar functions, there is a further equivalence relation that is strictly more general than CCZ-equivalence in some cases. This is known as "isotopic equivalence", and is defined in terms of structures called commutative semifields that can be associated with quadratic planar functions. We only mention this for the sake of completeness, but do not go into further details on this relation since it is not relevant to our study. We refer the reader to e.g. [16] for more details.

E. Uniformly distributed functions

A function F is called **k-to-1** for some $k \in \mathbb{N}$ if for any $y \in \mathrm{Im}(F)^*$, the preimage $F^{-1}(y)$ has size k. In particular, permutations of $\mathbb{F}_{p^n}$ are 1-to-1 functions, and the derivatives of APN functions are 2-to-1 functions. We note that functions of this form are of independent interest, especially in the case of permutations, and have been studied extensively in the literature; we refer the reader to [20] for a survey on permutation polynomials, and to e.g. [24], [10], [21] for examples of studies of 2-to-1 and 3-to-1 functions.

Our main interest in functions with uniform preimage sets stems from their relation to the cryptographically optimal APN and PN functions. It is known that any quadratic 3-to-1 function over $\mathbb{F}_{2^n}$ is APN, and that any quadratic 2-to-1 function over $\mathbb{F}_{p^n}$ for odd p is PN. Furthermore, we can easily see that, all of the known PN functions are (or are equivalent to) 2-to-1 functions (see e.g. [26] for a summary of all known planar functions). In the case of APN functions, the situation is a bit more varied, since there are many examples of APN functions that are not 3-to-1 (even up to equivalence). However, the majority of the functions originating from the known infinite families of APN functions are 3-to-1 (see [10] for a detailed survey) and thus constitute an important subclass of APN functions. Furthermore, searching for e.g. quadratic 3-to-1 functions is significantly faster than searching for quadratic APN functions in general, since verifying that a given function $F : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ is APN is a quadratic operation in 2^n, while verifying that it is 3-to-1 is merely linear; this allows computational searches to be conducted much faster and for larger dimensions n than in the general case. It would thus be natural to expect many new instances of 2-to-1 and 3-to-1 functions to be found via computer searches, and having an efficient and reliable way of classifying them up to CCZ-equivalence is then an important practical consideration.

We will mostly concentrate on a subclass of k-to-1 functions whose outputs are distributed in a particularly well structured way. More precisely, we will say that an (n, m, p)-function F is **uniformly distributed** if there exists a set of elements $U \subseteq \mathbb{F}_{p^n}^*$ such that for any $x \in \mathbb{F}_{p^n}$ we have $F(xu) = F(xu')$ for any $u, u' \in U$. Such a function is then k-to-1 for $k = \#U$. We will call the set U the **multiplicative kernel** of F. As we have observed in [10], many of the known infinite families of APN functions are (or are equivalent to) uniformly distributed 3-to-1 functions with $U = \mathbb{F}_{2^2}^*$; functions of this form are called *canonical 3-to-1 functions* in [10] to differentiate them from 3-to-1 functions that are not necessarily uniformly distributed. Similarly, all of the known planar functions are uniformly distributed 2-to-1 functions with $U = \{\pm 1\}$ (see e.g. [26] for a survey of the known planar functions).

In fact, $U = \mathbb{F}_{2^2}^*$, resp. $U = \{\pm 1\}$ are the only possible choices of U in the case of 3-to-1, resp. 2-to-1 uniformly distributed functions over $\mathbb{F}_{2^n}$, resp. $\mathbb{F}_{p^n}$ for p odd. Indeed, suppose that $F : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ is 3-to-1, $U = \{u, v, w\}$ and $F(ux) = F(vx) = F(wx)$ for every $x \in \mathbb{F}_{2^n}$. By substituting u for x, we get $F(u^2) = F(uv) = F(wu)$; by substituting v for x, we get $F(uv) = F(v^2) = F(vw)$, and hence $F(uv) = F(v^2) = F(vw) = F(u^2) = F(wu)$. Since F is 3-to-1, and assuming that all 3 elements of U are distinct, we must have $v^2 = wu$, and hence $v^3 = uvw$. Similarly, we can derive $u^3 = v^3 = w^3 = uvw$. Thus $(u/v)^3 = 1$, implying that $2 \mid n$ and $(u/v) \in \mathbb{F}_4 \setminus \mathbb{F}_2$. Thus, the triple (u, v, w) is of the form $(u, \beta u, \beta^2 u)$ where β is a primitive element of $\mathbb{F}_4$. If $U = \{u, v, w\}$ is a triple such that $F(ux) = F(vx) = F(wx)$ for every $x \in \mathbb{F}_{2^n}$, then the same is true for $U = \{cu, cv, cw\}$ for any $c \in \mathbb{F}_{2^n}^*$; by multiplying U with the inverse of u, we obtain that U has the form $(1, \beta, \beta^2)$ up to multiplication by a non-zero constant.

In the case of $U = \{a, b\}$ for 2-to-1 functions over $\mathbb{F}_{p^n}$, the proof is even simpler: we obtain $a^2 = b^2$ due to $F(a^2) = F(ab) = F(b^2)$, which leads to $(a/b)^2 = 1$ and hence $a/b = -1$, assuming $a \neq b$. Multiplying with the inverse of a then yields $U = \{-1, 1\}$.

It is easy to find other instances of uniformly distributed functions; for instance, taking U to be the multiplicative group of any subfield $\mathbb{F}_{2^m}$ of $\mathbb{F}_{2^n}$, we can obtain $(2^m - 1)$-to-1 (n, n)-functions that are uniformly distributed. We note that using an approach similar to the above, we can conclude that U is always a multiplicative group of order k up to multiplication by a non-zero constant. Indeed, we can assume $1 \in U$ as outlined above (otherwise, we multiply all elements of U by a constant), and if $U = \{1, u_2, u_3, \ldots, u_k\}$, then we have e.g. $F(u_2 u_3) = F(u_3^2) = F(u_3) = F(u)$ for any $u \in U$, so that $u_2 u_3 \in U$; therefore, U is closed under multiplication, and hence forms a multiplicative subgroup of $\mathbb{F}_{p^n}^*$. We formulate our algorithm for testing linear equivalence between k-to-1 functions (which is at the core of the equivalence test proposed in our paper) in the case of general k; however, in the sequel we mostly concentrate on the specific cases $k = 2$ and $k = 3$ which are by far the most significant cases in practice.

In [10], it is observed that the pre-image sets of a 3-to-1 uniformly distributed function F have the "summation property" (as it is referred to there); that is, if $\{x_1, x_2, x_3\}$ and $\{y_1, y_2, y_3\}$ are two pre-images of F (in other words, cosets of $U = \mathbb{F}_{2^2}^*$), then either $\{x_1 + y_1, x_2 + y_2, x_3 + y_3\}$ or $\{x_1 + y_1, x_2 + y_3, x_3 + y_2\}$ is a pre-image set; and either $\{x_1 + y_2, x_2 + y_1, x_3 + y_3\}$ or $\{x_1 + y_2, x_2 + y_3, x_3 + y_1\}$ is a pre-image set; and either $\{x_1 + y_3, x_2 + y_2, x_3 + y_1\}$ or $\{x_1 + y_3, x_2 + y_1, x_3 + y_2\}$ is a pre-image set. It is also observed in the same paper that this property is preserved under linear equivalence. Generalizing this to the case of k-to-1 functions with arbitrary k, we say that a k-to-1 function F has the **pre-image summation property** if for any two pre-image sets $F^{-1}(x) = \{a_1, a_2, \ldots, a_k\}$ and $F^{-1}(y) = \{b_1, b_2, \ldots, b_k\}$ of F, there exists some permutation π of the indices $1, 2, \ldots, k$ such that $\{a_1 + b_{\pi(1)}, a_2 + b_{\pi(2)}, \ldots, a_k + b_{\pi(k)}\}$ is a pre-image set of F as well. The fact that the pre-image summation property is invariant under linear equivalence can be easily seen in the same way as in [10].

III. Testing linear equivalence of uniformly distributed functions

Suppose that F and G are uniformly distributed with multiplicative kernel U. Suppose that they are linear-equivalent via $L_1 \circ F \circ L_2 = G$ and we wish to reconstruct L_1 and L_2. The basic idea of the algorithm is to iteratively guess values of L_1 and L_2 on a fixed basis of $\mathbb{F}_{p^n}$ until a contradiction is encountered (in which case we backtrack to a previous guess) or all values of L_1 and L_2 on the basis are known (in which case one possible equivalence between F and G has been recovered). Knowledge of a value of L_2 can be used to derive information about L_1, and vice-versa. The principle is somewhat similar to the one developed for bijective F and G in [3], except that in our more general setting, some additional guesses and deductions have to be made due to the pre-images under F and G having a more complicated structure. An important structural observation that underlies the strategy of the algorithm is that L_2 maps cosets of U to cosets of U; that is, if $L_2(x) = y$ for some $x, y \in \mathbb{F}_{p^n}$, then for any $u_1 \in U$ there exists a $u_2 \in U$ such that $L_2(u_1 x) = u_2 y$; more succinctly, we can express this as $L_2(Ux) = Uy$.

The following general types of derivation are possible:

- Knowledge of values of L_2 allows us to deduce values of L_1. For example, suppose that we have guessed $L_2(1) = c$. Since we know F and G, we obtain $L_1(F(c)) = G(1)$, and so we can deduce that the value of L_1 on $F(c)$ must be precisely $G(1)$. This can lead to contradiction (if $L_1(F(c))$ has been assigned some other value previously).
- Knowledge of values of L_1 allows us to restrict the values of L_2. For example, suppose that we know that $L_1(A) = B$. If $A \in \mathrm{Im}(F)$ but $B \notin \mathrm{Im}(G)$ or $A \notin \mathrm{Im}(F)$ and $B \in \mathrm{Im}(G)$; then we can immediately derive a contradiction. Otherwise, if $A \in \mathrm{Im}(F)$ and $B \in \mathrm{Im}(G)$, we must have $F(L_2(x)) = A$ and $G(x) = B$, so that for any $x \in G^{-1}(B)$, we have $L_2(x) \in F^{-1}(A)$. As discussed above, L_2 maps cosets of U to cosets of U.
- The linearity of L_1 and L_2 allows us to derive further values of L_1 and L_2 (for instance, knowing $L_1(A)$ and $L_1(B)$ allows us to derive $L_1(A+B)$); these values can lead to a contradiction (as described above), and can otherwise be used to derive further values of L_1 and L_2 in turn.

While knowing values of L_2 immediately allows us to deduce the corresponding values of L_1, knowing values of L_1 only restricts the values of L_2 since we get $L_2(x) \in F^{-1}(A)$ for all $x \in G^{-1}(B)$. In other words, we have $L_2(G^{-1}(B)) = F^{-1}(A)$; that is, knowing values of L_1 allows us to recover values of L_2 only "up to cosets" of U. This makes it necessary to guess *how* exactly L_2 maps $G^{-1}(B)$ to $F^{-1}(A)$. Let $G^{-1}(B) = Ua$ and $F^{-1}(A) = Ub$ for some $a, b \in \mathbb{F}_{2^n}$. In total, there are $\#U!$ mappings between aU and bU, although in reality we have to check much less than that due to L_2 being linear. We will refer to each such mapping of Ua onto Ub as a **configuration**, and to the process of guessing the form of the mapping as **configuring** the preimages Ua and Ub. In our current design, we simply try all possible configurations, and expect that if a wrong guess is made, it will lead to a contradiction and the search will backtrack sooner rather than later. We recall that the most important use cases of the algorithm are for 3-to-1 APN functions (with $U = \mathbb{F}_{2^2}^*$) and for 2-to-1 planar functions (with $U = \{-1, +1\}$). Even when brute-forcing the configurations, the number of possible guesses is very small ($3! = 6$ in the case of APN functions, and $2! = 2$ in the case of planar functions) and does not significantly increase the running time of the algorithm. As we observe in our computational results, the running times are very fast even in high dimensions, and so we do not consider further optimizations to be necessary at present. In the case of uniform functions with a much larger set U, it may be beneficial to investigate more sophisticated ways of configuring the preimages; however, since we are currently not aware of any practically relevant use cases for larger values of k, we leave this as a potential problem for future work.

A "chain" of derivations can be started by first guessing either a value of L_1 or a value of L_2. In our implementation for 3-to-1 APN functions, we begin by guessing a value of L_1, and then guessing the configurations of the corresponding preimages. In principle, a variation of the algorithm in which a value of L_2 is guessed first could have been implemented, although we do not expect that there would be any significant differences in the computational efficiency. In the case of our implementation for testing equivalence of 2-to-1 functions, we guess a value of L_2 first in order to further speed up the computation using the orbit partitions from Table I as described in Section V.

The different phases of guesses and derivations used in Algorithm 1 are visualized in Figure III. Pseudocode describing the general algorithm is given under Algorithm 1. At any given moment of its execution, we will have guessed some values of L_1 and L_2, while others will still be unknown; we denote by $\mathrm{Dom}(L_i)$ the **partial domain** of L_i, i.e. the elements $x \in \mathbb{F}_{p^n}$ for which we currently know $L_i(x)$; and, similarly, the **partial image** $\mathrm{Im}(L_i)$ of L_i is the set $\mathrm{Im}(L_i) = \{L_i(x) : x \in \mathrm{Dom}(L_i)\}$ of all currently known images of L_i.

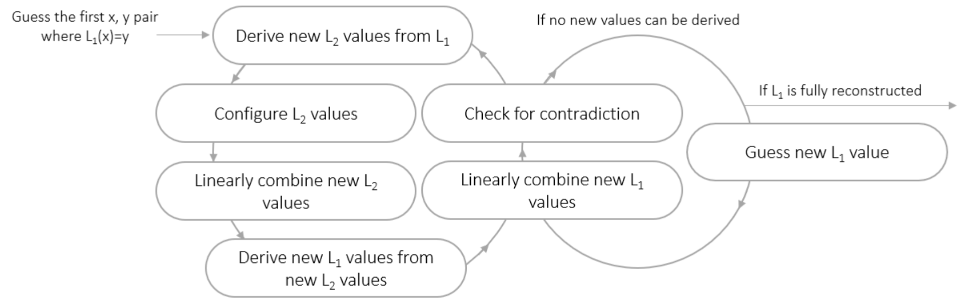
Guess the first x, y pair where L1(x)=y
Derive new L2 values from L1
If no new values can be derived
Configure L2 values
Check for contradiction
If L1 is fully reconstructed
Guess new L1 value
Linearly combine new L2 values
Linearly combine new L1 values
Derive new L1 values from new L2 values

Fig. 1. General operation of Algorithm 1

As indicated above, the search begins by guessing a single value of say L_1, e.g. guessing that $L_1(x)$ is equal to y for some $x, y \in \mathbb{F}_{2^n}$. Every such guess leads to a "chain reaction" of derivations: further values of L_1 can be derived by linearity, and may potentially lead to a contradiction, indicating that one of the previously made guesses is wrong. If no contradiction is encountered, the chain of derivations can stop for two reasons: either the values of L_1 and L_2 on all basis elements are already known (in which case an equivalence between F and G has been successfully found), or additional guesses are needed (in which case we guess another value of L_1 and start another "chain reaction"). According to our observations, very few guesses need to be made in practice, and the search will find an equivalence between F and G extremely quickly if one exists.

We remark that in the case of quadratic 2-to-1 planar functions over fields $\mathbb{F}_{p^n}$ of odd characteristic p, Algorithm 1 is sufficient to decide CCZ-equivalence since the latter coincides with linear equivalence [8]. Since it is hypothetically possible

Algorithm 1: Testing linear equivalence of two uniformly distributed functions

```
Data: Two uniformly distributed $(n,n)$-functions $F, G$ with multiplicative kernel $U$
Result: A pair of linear $(n,n)$-permutations $L_1, L_2$ such that $L_1 \circ F \circ L_2 = G$, or failure
begin
    # list of pairs $(x,y)$ representing input-output pairs of $L_1$ that can still be used to derive new values by linearity
    UnProcessed $\leftarrow \emptyset$
    # list of pairs $(x,y)$ representing input-output pairs of $L_1$ that can still be used to derive new configurations of $L_2$
    UnConfigured $\leftarrow \emptyset$
    # initiates a recursive chain of guesses and derivations which returns either the reconstructed $L_1$ or failure
    $L_1 \leftarrow$ Guess_New_Value()
    if $L_1 \neq$ failure then
        reconstruct $L_2$ via $G = L_1 \circ F \circ L_2$
        return $L_1, L_2$
    return failure
Function Guess_New_Value():
    if all values of $L_1$ are defined then
        return $L_1$
    # otherwise make a new guess
    $x \leftarrow$ first $x$ from the basis for which $L_1$ is undefined
    for $y \in \mathrm{Im}(G) \setminus \mathrm{Im}(L_1)$ do
        $L_1(x) \leftarrow y$
        push $(x,y)$ to UnProcessed
        backup $L_1, L_2$, UnProcessed, UnConfigured
        # We now derive all possible information from this new guess; if no contradiction is encountered, we recursively
          call Guess_New_Value() which will either reconstruct $L_1$, or make more guesses if needed
        if Process_L1() $\neq$ contradiction then
            $Res \leftarrow$ Guess_New_Value()
            if $Res \neq$ failure then
                return $Res$
        else
            restore $L_1, L_2$, UnProcessed, UnConfigured
    return failure
...
```

that two 3-to-1 APN functions over a field $\mathbb{F}_{2^n}$ of even characteristic are EA-equivalent but not linear-equivalent, it could be necessary to check whether a given uniformly distributed 3-to-1 function is additively equivalent to another 3-to-1 function. According to our experimental results however, this is never necessary in practice, and computing the differential spectrum of the ortho-derivatives combined with Algorithm 1 is always sufficient to decide the EA-equivalence between any pair of quadratic 3-to-1 APN functions. More precisely, in all cases where the differential spectra of the orthoderivatives of two uniformly distributed 3-to-1 functions matched, we were able to verify that they are linearly equivalent using Algorithm 1.

IV. Auxiliary algorithms

In the following, we formulate some auxiliary algorithms that can be used to extend Algorithm 1 to the general case of EA-equivalence, to k-to-1 functions that are not necessarily uniformly distributed, and to functions that are EA-equivalent to k-to-1 functions but are not k-to-1 themselves. Note that in the case of quadratic 2-to-1 planar functions, CCZ-equivalence coincides with linear-equivalence [8], so such an extension is unnecessary; while k-to-1 functions with $k > 3$ do not appear to be of immediate interest from the point of view of cryptography; and so we restrict ourselves to the case of quadratic 3-to-1 APN functions for the sake of simplicity. Nonetheless, most of the principles naturally generalize to the case of $k > 3$.

A visual summary of how Algorithm 1 and the auxiliary algorithms can be used to test equivalence of a given quadratic APN function F to a 3-to-1 function is given in Figure 2. If F is not 3-to-1, we first try to find a 3-to-1 function equivalent to it via Algorithms 3 and 4; if F is 3-to-1 but not uniformly distributed, we find a uniformly distributed function equivalent to it via Algorithm 2; finally, we use Algorithm 1 to compare a uniformly distributed 3-to-1 F for equivalence against all known uniformly distributed 3-to-1 representatives.

Algorithm 1: Testing linear equivalence of two uniformly distributed functions (continued)

```
# Derives further values of $L_1$ by linearity
Function Process_L1():
    while $UnProcessed \neq \emptyset$ do
        pop $(x, y)$ from UnProcessed
        for $x' \in \mathrm{Dom}(L_1)$ do
            $y' \leftarrow L_1(x')$
            $new\_x \leftarrow x + x'$
            $new\_y \leftarrow y + y'$
            if $(new\_x \in \mathrm{Im}(F)$ and $new\_y \notin \mathrm{Im}(G))$ or $(new\_x \notin \mathrm{Im}(F)$ and $new\_y \in \mathrm{Im}(G))$ then
                return contradiction
            $L_1(new\_x) \leftarrow new\_y$
            push $(new\_x, new\_y)$ onto UnConfigured
            # We immediately try to guess the configuration of $L_2$ following from the new value of $L_1$ since it can
              already lead to contradiction even before further values of $L_1$ are derived
            if Configure() == contradiction then
                return contradiction
    return success
# Guesses the configuration corresponding to $L_1(A) = B$ and derives values of $L_2$ from the guess
Function Configure():
    while $UnConfigured \neq \emptyset$ do
        pop $(A, B)$ from UnConfigured
        $x_1, x_2, \dots, x_k \leftarrow F^{-1}(A)$
        $y_1, y_2, \dots, y_k \leftarrow G^{-1}(B)$
        # here $Sym(\{1, 2, \dots, k\})$ is the symmetric group on $\{1, 2, \dots, k\}$
        for all possible configurations $\pi \in Sym(\{1, 2, \dots, k\})$ do
            # if e.g. $\pi(1) = 2$, then $L_2(x_1) = y_{\pi(1)} = y_2$ etc.
            for $x' \in \mathrm{Dom}(L_2)$ do
                $y' \leftarrow L_2(x')$
                $new\_x \leftarrow x' + x_1$
                $new\_y \leftarrow y' + y_{\pi(1)}$
                $L_2(new\_x) \leftarrow new\_y$
                # we know $L_1(F(L_2(new\_x))) = L_1(F(new\_y)) = G(new\_x)$
                $new\_l_1\_x \leftarrow F(L_2(new\_x))$
                $new\_l_1\_y \leftarrow G(new\_x)$
                $L_1(new\_l_1\_x) \leftarrow new\_l_1\_y$
                push $(new\_l_1\_x, new\_l_1\_y)$ onto UnProcessed
                # We proceed to derive further information from the knowledge of $L_1(new\_l_1\_x) = new\_l_1\_y$
                if Process_L1() == contradiction then
                    return contradiction
    return success
```

A. Testing whether a 3-to-1 function is linear-equivalent to a uniformly distributed 3-to-1 function

Our main algorithm, i.e. Algorithm 1, can only be applied to two (n, n, p)-functions F and G if both of them are uniformly distributed; for instance, if both F and G are 3-to-1 APN functions with $U = \mathbb{F}_{2^2}^*$. As observed in [10], some of the known families of APN functions contain quadratic functions that are 3-to-1 but not uniformly distributed; however, they are linear-equivalent to uniformly distributed 3-to-1 functions. Indeed, the existence of quadratic 3-to-1 APN functions EA-inequivalent to uniformly distributed ones is left as an open problem in [10].

Suppose that we have a concrete 3-to-1 function T, and we want to check whether it is linear-equivalent to a uniformly distributed 3-to-1 function C. We can observe that any composition $L_1 \circ C = C'$ of a uniformly distributed C with a linear permutation L_1 results in a uniformly distributed C'. Thus, to decide the linear equivalence of a given 3-to-1 function T to a uniformly distributed 3-to-1 function C, it suffices to find a linear permutation L_2 such that $T \circ L_2 = C$. Observing that the

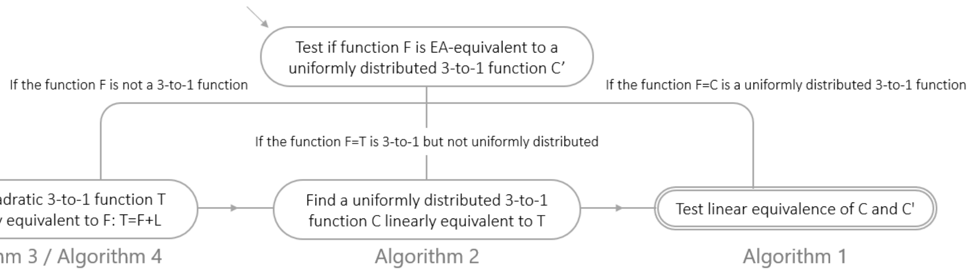
Test if function F is EA-equivalent to a uniformly distributed 3-to-1 function C'
If the function F is not a 3-to-1 function
If the function F=T is 3-to-1 but not uniformly distributed
If the function F=C is a uniformly distributed 3-to-1 function
Find a quadratic 3-to-1 function T additively equivalent to F: T=F+L
Find a uniformly distributed 3-to-1 function C linearly equivalent to T
Test linear equivalence of C and C'
Algorithm 3 / Algorithm 4
Algorithm 2
Algorithm 1

Fig. 2. Testing EA-equivalence of a given quadratic APN function F to a uniformly distributed 3-to-1 function using the algorithms in this paper

linear permutation L_2 maps pre-images of C to pre-images of T without affecting the image set (so that $\mathrm{Im}(C) = \mathrm{Im}(T)$), Algorithm 2 seeks to find such a function L_2 by first guessing pairs of triples mapping to the same image, and subsequently guessing their configurations. Once we know (or have guessed) the pairing of two triples and their configurations under L_2, e.g. $L_2 : (c_{11}, c_{12}, c_{13}) \mapsto (t_{11}, t_{12}, t_{13})$ and $L_2 : (c_{21}, c_{22}, c_{23}) \mapsto (t_{21}, t_{22}, t_{23})$, we can derive more information using the linearity of L_2. More precisely, we can see that e.g. $L_2(c_{11}) + L_2(c_{21}) = L_2(c_{11} + c_{21})$.

Recall that the pre-image summation property is invariant under linear equivalence, and that any uniformly distributed 3-to-1 function has this property. This means that if T is linear-equivalent to a uniformly distributed 3-to-1 function, then it also has the pre-image summation property; we use this in Algorithm 2 in the `Combine()` and `Check()` functions to arrive at possible contradictions. More precisely, we know that if $\{a_i, b_i, c_i\}$ and $\{a_j, b_j, c_j\}$ are two pre-image sets of T, then by the triple summation property e.g. one of $\{a_i+a_j, b_i+b_j, c_i+c_j\}$ or $\{a_i+a_j, b_i+c_j, c_i+b_j\}$ must be a pre-image set as well. Note that the pre-image summation property by itself only requires that one of $\{a_i+a_j, b_i+b_j, c_i+c_j\}$ or $\{a_i+a_j, b_i+c_j, c_j+b_i\}$ be a pre-image set; however, since we know that a_i, b_i, c_i are the images of three elements of the form $x, \beta x, \beta^2 x$ (where β is primitive in $\mathbb{F}_{2^2}$) for some x, and a_j, b_j, c_j are the images of $y, \beta y, \beta^2 y$ for some y, it is clear that the second option cannot possibly happen. For this reason, in `Combine()`, we only test three combinations instead of six.

Further algorithm details are implementation specific, but for the sake of clarity, we will briefly mention the most important points. Algorithm 2 is organized as a recursive function with the root of the recursion starting with the first guess; in this case, what we try to guess is which pre-image set of T has the same image as a pre-image set of C (recall that we must necessarily have $\mathrm{Im}(T) = \mathrm{Im}(C)$). After the first guess is made, the recursive function `Configure()` is called to guess the configuration of the elements in the pre-image sets. If the chosen configuration does not lead to a contradiction when calling `Combine()` and `Check()`, the algorithm proceeds with another guess; otherwise, the next configuration is tested. The `Configure()` function backtracks if no configuration is viable, and successfully terminates when it has made enough guesses to derive all the values of L_2 via linear combinations. New guesses are taken from the set of pre-images that have not yet been used or derived via the pre-image summation property.

In the algorithm on line 4, we need to order the pre-images of C is some way, and so we consider some arbitrary order on the elements of $\mathbb{F}_{2^n}$; this can be, for example, the lexicographic order of their coordinate vectors with respect to the standard basis (or any other well-defined order on $\mathbb{F}_{2^n}$); the concrete choice affects neither the correctness of the algorithm, nor its running time.

In the `Check()` function, the notion of linear span is generalized to pre-image sets, so that $\mathrm{Span}(B_T) = \mathrm{Span}(\bigcup B_T)$ for any set of pre-images B_T. This generalisation is used to verify that B_T is one of the sets with the minimum number of elements whose union contains a basis of $\mathrm{Dom}(T)$. This property is then used for reconstructing the linear function L_2 by mapping the basis contained in B_C to the one contained in B_T.

B. Testing EA-equivalence to a k-to-1 function

In the prequel, we have assumed that we are working with k-to-1 functions (either uniformly distributed, or not). While this is by far the most significant case in practice (for instance, computational searches exploiting the structure of 2-to-1 or 3-to-1 functions are typically designed in such a way that they only produce uniformly distributed k-to-1 functions), it is clearly possible to find examples of functions that are EA-equivalent to, but are not themselves, k-to-1. It might thus be useful in certain situations to check whether a given (n, m, p)-function F is EA-equivalent to a k-to-1 function. If this is so, then Algorithm 2 can then be used to find a uniformly distributed function that is linearly equivalent to the k-to-1 function. We make use of this procedure when classifying the functions from family C3 in Section VII, since these functions are EA-equivalent to 3-to-1 functions while not being 3-to-1 themselves. In order to apply Algorithms 1 and 2 to them, we thus first need to find a 3-to-1 representation.

Affine equivalence clearly preserves the property of a function being k-to-1. Thus, if F is EA-equivalent to a k-to-1 function T (but not k-to-1 itself), then there must exist an affine (n, n)-function L such that $F + L = T$. Since $T(0) = 0$ by definition, we can assume that $F(0) = L(0)$ as well without loss of generality, so that F and T are additive equivalent (otherwise we

Algorithm 2: Testing linear equivalence of a 3-to-1 function to a uniformly distributed 3-to-1 function

```
Data: A 3-to-1 function $T : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$
Result: A linear $(n,n)$-function $L_2$ and a uniformly distributed 3-to-1 function $C : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ such that $T \circ L_2 = C$, or
        failure
1  begin
2      # find the pre-image sets corresponding to $T$ and to $C$
3      $A_T \leftarrow \{\{t_1,t_2,t_3\} : t_1,t_2,t_3 \in \mathbb{F}_{2^n}, T(t_1) = T(t_2) = T(t_3)$ and $t_1+t_2+t_3 = 0\}$
4      $A_C \leftarrow \{(c_1,c_2,c_3) : c_1,c_2,c_3 \in \mathbb{F}_{2^n}, c_2 = \beta c_1, c_3 = \beta^2 c_1, c_1 < c_2, c_1 < c_3\}$ # $\beta$ is primitive in $\mathbb{F}_{2^2}$
5      let $B_C$ be a minimal set of pre-image sets from $A_C$ such that $\bigcup B_C$ contains a basis of $\mathrm{Dom}(C)$
6      let $B_T \leftarrow \emptyset$ be a set of ordered pre-image sets from $A_T$
7      for all $\{t_1,t_2,t_3\} \in A_T$ do
8          if Configure($\{t_1,t_2,t_3\}$) $\neq \emptyset$ then
9              reconstruct $L_2$ using elements of basis contained in $B_C$ and $B_T$
10             reconstruct $C = T \circ L_2$
11             return $L_2, C$
12     return failure
13 Function Configure($\{b_1,b_2,b_3\}$):
14     for all possible configurations $\pi \in Sym(\{1,2,3\})$ do
15         # e.g. $\pi(1) = 2, \pi(2) = 3, \pi(3) = 1$
16         $B_T \leftarrow B_T \cup \{(b_{\pi(1)}, b_{\pi(2)}, b_{\pi(3)})\}$
17         if Combine($B_T$)=True then
18             # if it is possible to reconstruct the whole function $L_2$
19             if #$B_T$ = #$B_C$ then
20                 return $B_T$
21             # otherwise make a new guess
22             for all $\{t_1,t_2,t_3\} \in A_T$ not already guessed or derived by linear combinations do
23                 backup $B_T$
24                 $B_T \leftarrow$ Configure($\{t_1,t_2,t_3\}$)
25                 if $B_T \neq \emptyset$ then
26                     return $B_T$
27                 restore $B_T$
28         $B_T \leftarrow B_T \setminus \{(b_{\pi(1)}, b_{\pi(2)}, b_{\pi(3)})\}$
29     return $\emptyset$
30 ...
```

must have $L(0) = F(0)$, and so we can replace L by $L' = L + L(0)$ and F by $F' = F + L(0)$). For a given function F, we thus want to find all k-to-1 functions that are additive equivalent to it. We note that this approach can also be applied to a function that is already k-to-1 since, it is hypothetically possible that two k-to-1 functions F and G are EA-equivalent but not linear-equivalent; resolving the EA-equivalence between F and G then reduces to applying the test for linear equivalence to F' and G for each k-to-1 function F' that is additively equivalent to F. As already remarked above, in all our computations and experiments on quadratic 3-to-1 functions, we have never encountered this case in practice, and whether EA-equivalence of k-to-1 functions implies linear equivalence in general is an open problem.

Algorithm 1 in [10] allows one to test whether a given (n,n)-function F is additive-equivalent to a triplicate function T; we note that triplicate functions were defined in [10] as a generalization of 3-to-1 functions[2]. In particular, if F is additive-equivalent to a 3-to-1 function, this algorithm can be used to find this equivalence. Unfortunately, the time complexity of the procedure grows exponentially with the dimension n, and it is not usable for values of n beyond $n = 10$. This is not surprising, as that algorithm handles a much more general problem (equivalence to a triplicate function) and does not exploit the specific structural properties of 3-to-1 functions.

In this section, we propose a significantly more efficient approach for testing equivalence of a given (n,n)-function F to a 3-to-1 function T. Somewhat similarly to Algorithm 1 of [10], the basic idea consists of guessing the values of the linear function L on a basis $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$ of $\mathbb{F}_{2^n}$, and backtracking upon discovery of a violation. In this case, we assume that the function T is 3-to-1, so that every non-zero element in the image of T has precisely 3 pre-images. In addition, we

[2]More precisely, a triplicate function is any function F such that for any $y \in \mathrm{Im}(F)$, the size of the pre-image $F^{-1}(y)$ is a multiple of 3

Algorithm 2: Testing linear equivalence of a 3-to-1 function to a uniformly distributed 3-to-1 function (continued)

```
Function Combine($B_T$):
    # linearly combine elements of $B_T$
    for all $i, j$ between $1$ and #$B_T$ with $i \neq j$ do
        $(a_i, b_i, c_i) \leftarrow x_i \in B_T$
        $(a_j, b_j, c_j) \leftarrow x_j \in B_T$
        if Check($\{a_i + a_j, b_i + b_j, c_i + c_j\}, x_j$)=False then
            return False
        if Check($\{a_i + b_j, b_i + c_j, c_i + a_j\}, x_j$)=False then
            return False
        if Check($\{a_i + c_j, b_i + a_j, c_i + b_j\}, x_j$)=False then
            return False
    return true
Function Check($\{t_1, t_2, t_3\}, x$):
    # check conditions for contradiction
    $S \leftarrow \mathrm{Span}(B_T \setminus x)$
    # check that $t_1, t_2, t_3$ have the zero-sum property
    if $(t_1 + t_2 + t_3 \neq 0)$ then
        return False
    # check that $t_1, t_2, t_3$ belong to a triple
    if $\{t_1, t_2, t_3\} \notin A_T$ then
        return False
    # check that $t_1, t_2, t_3$ triple is unique
    if $t_1 \in S$ or $t_2 \in S$ or $t_3 \in S$ then
        return False
    return True
```

assume that any pre-image set of T adds up to 0; we note that this is true for any quadratic uniformly distributed 3-to-1 function, and for any function that is affine equivalent to one; the existence of 3-to-1 APN functions that do not satisfy this "zero-sum property" is left as an open problem in [10], where it is observed that all the known 3-to-1 APN functions have it. A violation can thus occur, for instance, if we end up with more than 3 elements x mapping to the same image via the partially reconstructed T; or if the size of the image set of T exceeds $(2^n - 1)/3 + 1$ elements.

We can observe that for any two (n, n)-functions F and G, and for any linear (n, n)-function L such that $F = L + G$, we have

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

since L vanishes on $\{x, y, x + y\}$; more generally, this is true for the sum of F and G on any set of elements that add up to 0. Assuming that the function G is known, this means that knowledge of the values $F(x)$ and $F(y)$ of F at some two elements $x, y \in \mathbb{F}_{2^n}$ is sufficient to derive its value $F(x + y)$ at $x + y$. In this way, knowledge of the values of F on a set of elements $S \subseteq \mathbb{F}_{2^n}$ allows us to uniquely reconstruct its values on the linear span of S.

Furthermore, given a function F that is additive-equivalent to a 3-to-1 function T with the zero-sum property, we can predict the exact image set of T by the multiplicities of the multiset $M_F = [F(x) + F(y) + F(x + y) : x, y \in \mathbb{F}_{2^n}]$ which is shown to be invariant under additive equivalence in Corollary 4 of [10]. According to the latter, for a quadratic 3-to-1 function T, the non-zero elements in $\mathrm{Im}(T)$ can be distinguished from those in $\mathbb{F}_{2^n}^* \setminus \mathrm{Im}(T)$ according to their multiplicities in M_T; and since M_T is invariant under additive equivalence, we can assume that we know the image set of T a priori. If we want to test equivalence to a 3-to-1 function that is not necessarily quadratic, we can modify the algorithm by replacing this derivation of $\mathrm{Im}(T)$ with a condition that backtracks if the image set of the partially constructed function T exceeds $(2^n - 1)/3 + 1$ elements. Since by far the most important use case at the moment involves quadratic APN functions, we do not discuss this modification in further detail, and assume that $M_F = M_T$.

In practice, this means that if the multiplicities of M_F do not split $\mathbb{F}_{2^n}^*$ into precisely two sets of size, respectively, $(2^n - 1)/3$ and $2(2^n - 1)/3$, then F can not be additive-equivalent to a 3-to-1 function. Otherwise, we know the exact image set of any 3-to-1 function T that is additive equivalent to F, and there is no need to keep track of the size of the image set of T in the algorithm; instead, it suffices to verify that every value $T(x)$ of T for any $x \in \mathbb{F}_{2^n}^*$ that we guess or derive in the course of the search belongs to $\mathrm{Im}(T)$. This simplifies the implementation of the algorithm, and improves its running time since it allows for more incorrect branches of the search tree to be eliminated.

This correspondence between the image sets of the two tested functions allows us to obtain some conditions on which pairs of elements $x, y \in \mathbb{F}_{2^n}$ can satisfy $T(x) = T(y)$. To be more precise, we know that if $T(x) = T(y)$, then also $T(x) = T(y) = T(x+y)$ by the zero-sum property; and so $T(x) = T(y) = T(x+y) = F(x) + F(y) + F(x+y)$. If $F(x) + F(y) + F(x+y) \notin \mathrm{Im}(T)$, then x and y (as well as x and $x+y$, and y and $x+y$) cannot belong to the same triple. We can thus define a set

$$\mathcal{A}_F = \{\{x, y\} \subseteq \mathbb{F}_{2^n}^* : F(x) + F(y) + F(x+y) \in \mathrm{Im}(T)\}$$

of admissible pairs $\{x, y\}$. If at any point in the computation we obtain $T(x) = T(y)$ for some pair $\{x, y\}$ that is not in $\mathcal{A}_F$, then we can immediately backtrack.

In fact, this suggests another useful trick, which is that instead of trying to guess the values of T (or, equivalently, L) on $\mathcal{B}$, we can instead try to guess which elements belong to the same triples. For instance, in the first iteration of the search procedure, we would consider e.g. $b_1 \in \mathcal{B}$, and attempt to guess for which element y we have $F(b_1) = F(y)$. We know that we can restrict the guesses to only the elements from the set $\{y \in \mathbb{F}_{2^n} : \{b_1, y\} \in \mathcal{A}_F\}$; and, once we have guessed y, we know that the remaining element from the triple is $b_1 + y$, and so we can recover $T(b_1) = T(y) = T(b_1+y) = F(b_1) + F(y) + F(b_1+y)$. We would then try to guess which element y' belongs to the same triple as b_2 (provided, of course, that $b_2 \notin \{y, b_1 + y\}$; otherwise, we would take some other "indeterminate" element instead of b_2), and derive the values of T at b_2, y', and $b_2 + y'$. We would then derive the values of T on the linear span of $\{b_1, b_2, y, y'\}$; at every step, we would check whether any conditions have been violated, and backtrack if so.

A pseudocode description of the procedure is given under Algorithm 3. While our C implementation follows the same conceptual ideas, the pseudocode omits a few technical details that a real implementation would have to take into account. For instance, one would have to maintain data structures representing the pre-images of T and the set of admissible pairs; the iterative guesses would most likely be represented by a recursive function rather than a while loop; the aforementioned structures would have to be copied before a recursive call, and then restored to their previous state before backtracking; and so forth. Such details are bound to be language- and implementation-specific, and do not affect the principal logic of the algorithm, so we omit discussing them here; in addition, such a discussion would be overly technical, and we believe it would distract from, rather than clarify, the principles underlying the algorithm. Furthermore, we note that some of the conditions, such as the test on line 36 of whether some element is assigned more than 3 pre-images, can be performed already during the derivation of values on line 26; in this way, one would be able to backtrack immediately upon encountering such a contradiction, instead of having to wait until all possible new values of T have been derived. We have chosen to separate this condition into a stand-alone function since in the next subsection we discuss a modified version of this algorithm, in which the only difference is that we expand the `CheckConditions()` function by adding additional conditions.

As before, we denote by $\mathrm{Dom}(T)$ and $\mathrm{Im}(T)$ the partial domain and partial image set of T; indeterminate entries are denoted by a star symbol "*".

As observed in [10], besides the zero-sum property, any quadratic 3-to-1 function has the pre-image summation property. If we want to check whether a given function F is additive-equivalent to a 3-to-1 function T having the pre-image summation property, we can modify Algorithm 3 by adding more conditions that must be checked after every derivation. While this does not allow us to derive any further values of T and L as compared to Algorithm 3 (since as soon as we know e.g. $T(x)$ and $T(u)$, we can immediately derive $T(x+u)$ as $T(x+u) = T(x) + T(u) + F(x) + F(u) + F(x+u)$), it does introduce one more condition that might be violated and allow us to backtrack early. This algorithm is essentially the same as Algorithm 3, except that the `CheckConditions()` function is modified by having it check that any two pre-image sets $\{x_1, x_2, x_3\}$ and $\{z_1, z_2, z_3\}$ such that $T(x_1) = T(x_2) = T(x_3)$ and $T(z_1) = T(z_2) = T(z_3)$ can be "summed" to obtain another pre-image set under T. Note that as soon as we know the values of T on x_i and z_i for $i = 1, 2, 3$, we can uniquely reconstruct its values on all elements of the form $x_i + z_j$ for $i, j = 1, 2, 3$, and so the condition can always be verified. Another consideration that we need to take into account is that we do not know in what order the elements of the two pre-image sets have to be summed together in order to produce a third pre-image set. For this reason, we explicitly check three cases in the algorithm.

A pseudocode description of this expanded version of the `CheckConditions()` function is given in Algorithm 4. In it, we keep a list of all pre-image sets $\{x_1, x_2, x_3\}$ that we currently know of for which $T(x_1) = T(x_2) = T(x_3)$. These are stored in the set P. Upon processing a new pre-image set of this form (and verifying that $x_1 + x_2 + x_3 = 0$ so that it satisfies the zero-sum property), we check whether it is compatible (in the sense of the pre-image summation property) with every other pre-image set that we have already observed. Once again, the approach given in Algorithm 4 describes the main idea, and we do not claim that it is the optimal way for implementing the function in practice; one could, for instance, keep a constantly updated list of known pre-images, and then only verify the conditions starting on line 10 of Algorithm 4 for those triples for which they have not already been verified. This improves the running times of the algorithm, as can be seen from the computational results in Table II.

Algorithm 3: Testing additive-equivalence to a quadratic 3-to-1 function with the zero-sum property

```
Data: A function $F : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ with $F(0) = 0$
Result: A linear $(n,n)$-function $L$ such that $F + L$ is a 3-to-1 function with the zero-sum property, or failure
1  begin
2      # Compute the multiset $M_F$ and predict the image set $I$ of $T$
3      $M_F \leftarrow [F(x) + F(y) + F(x+y) : x, y \in \mathbb{F}_{2^n}]$
4      partition the elements of $\mathbb{F}_{2^n}^* = C_1 \cup C_2 \cup \cdots \cup C_l$ according to their multiplicities in $M_F$
5      if $l \neq 2$ or $(2^n - 1)/3 \notin \{\#C_1, \#C_2\}$ then
6          return failure
7      $I \leftarrow C_i$ with $i \in \{1, 2\}$ such that $\#C_i = (2^n - 1)/3$
8      # Compute the set of admissible pairs
9      $\mathcal{A}_F \leftarrow \{\{x, y\} : x, y \in \mathbb{F}_{2^n}, F(x) + F(y) + F(x+y) \in I\}$
10     # Initialize a partial truth table for $T$, with $T(0) = 0$ as the only determinate element
11     $T(0) \leftarrow 0, T(x) \leftarrow *$ for $x \in \mathbb{F}_{2^n}^*$
12     while $\#\mathrm{Dom}(T) < 2^n$ do
13         # Get an element $x$ that has not been assigned to a triple yet
14         find $x \in \mathbb{F}_{2^n}$ such that $T^{-1}(x) \leq 1$
15         # Guess another element in the triple containing $x$
16         for $y \in \{y \in \mathbb{F}_{2^n} : \{x, y\}, \{x, y + x\} \in \mathcal{A}_F\}$ do
17             $v \leftarrow F(x) + F(y) + F(x+y)$
18             if $v \notin I$ then
19                 go to next $y$
20             $T(x), T(y), T(x+y) \leftarrow v$
21             # Derive new values of $T$
22             for $z \in \{x, y, x+y\}$, $x' \in \mathrm{Dom}(T) \setminus \{x, y, x+y\}$ do
23                 $v' \leftarrow F(x') + F(z) + F(z + x') + T(z) + T(x')$
24                 if $v' \notin I$ or $z + x' \in \mathrm{Dom}(T), T(z + x') \neq v'$ then
25                     go to next $y$
26                 $T(z + x') \leftarrow v'$
27             if CheckConditions(T) = false then
28                 go to next $y$
29         # If no conditions are violated, but there are still indeterminate positions in $T$, we proceed to make another guess
             in the following iteration of the while loop
30     if $\#\mathrm{Dom}(T) = 2^n$ then
31         return $T$
32     else
33         return failure
34 Function CheckConditions($T$):
35     for $y \in \mathrm{Im}(T)$ do
36         if $\#T^{-1}(y) > 3$ or $\#T^{-1}(y) = 3, \sum T^{-1}(y) \neq 0$ then
37             return false
38     return true
```

V. Pre-computation of self-equivalence orbits

The core concept at the heart of Algorithm 1 for reconstructing the linear equivalence (L_1, L_2) in $L_1 \circ F \circ L_2 = G$ for some given uniformly distributed F and G is the simple idea of guessing a value of L_1 (or, alternatively, of L_2), and then deriving as many other values as possible based on this guess. The complexity of the algorithm clearly depends on the number of guesses that we have to make, and on the number of possibilities that we have for each guess. For the very first guess that we make in the algorithm, we have practically no information about the values that L_1 and L_2 can take, and so we have to try out all possible non-zero elements of $\mathbb{F}_{p^n}$ for this first value.g. L_1.

There is a way to significantly restrict the number of choices for this first guess if we perform some pre-computation on the function F. Suppose without loss of generality that we begin the search by guessing a value of L_2, and that this first value is

Algorithm 4: Testing conditions for EA-equivalence to a quadratic 3-to-1 function with the pre-image summation property

```
begin
    Function CheckConditions2($T$):
        # Keep a list of processed elements $y$ with a complete pre-image set
        $P \leftarrow \emptyset$
        for $y \in \mathrm{Im}(T)$ do
            if $\#T^{-1}(y) > 3$ then
                return false
            if $\#T^{-1}(y) = 3$ then
                let $T^{-1} = \{x_1, x_2, x_3\}$
                if $x_1 + x_2 + x_3 \neq 0$ then
                    return false
                for $\{z_1, z_2, z_3\} \in P$ do
                    if $(T(x_1 + z_1) \neq T(x_2 + z_2)$ or $T(x_1 + z_1) \neq T(x_3 + z_3))$ and $(T(x_1 + z_1) \neq T(x_2 + z_3)$ or $T(x_1 + z_1) \neq T(x_3 + z_2)$ then
                        return false
                    if $(T(x_1 + z_2) \neq T(x_2 + z_1)$ or $T(x_1 + z_2) \neq T(x_3 + z_3))$ and $(T(x_1 + z_2) \neq T(x_2 + z_3)$ or $T(x_1 + z_2) \neq T(x_3 + z_1)$ then
                        return false
                    if $(T(x_1 + z_3) \neq T(x_2 + z_1)$ or $T(x_1 + z_3) \neq T(x_3 + z_2))$ and $(T(x_1 + z_3) \neq T(x_2 + z_2)$ or $T(x_1 + z_3) \neq T(x_3 + z_1)$ then
                        return false
                $P \leftarrow P \cup \{\{x_1, x_2, x_3\}\}$
    return true
```

$L_2(1)$. In this section, we will show how the elements of $\mathbb{F}_{p^n}^*$ can be divided into "orbits" according to the self-equivalences of $F : \mathbb{F}_{p^n} \to \mathbb{F}_{p^n}$ so that when guessing the value v of $L_2(1)$, we only have to consider one element v from each orbit.

Recall that a **linear self-equivalence** for a function $F : \mathbb{F}_{p^n} \to \mathbb{F}_{p^n}$ is a pair of linear permutations (L_1, L_2) of $\mathbb{F}_{p^n}$ such that $L_1 \circ F \circ L_2 = F$; in other words, a linear equivalence of the function F "with itself". To simplify notation, let

$$\mathrm{EQ}(F, G) = \{(L_1, L_2) : L_1, L_2 : \mathbb{F}_{p^n} \to \mathbb{F}_{p^n} \text{ bijective}, L_1 \circ F \circ L_2 = G\}$$

be the set of all linear equivalences between two given functions F and G. The self-equivalences $\mathrm{EQ}(F, F)$ clearly form a group under the composition $(L_1, L_2) \circ (L_3, L_4) = (L_1 \circ L_3, L_4 \circ L_2)$. Furthermore, we can see that $\mathrm{EQ}(F, G)$ is a coset (under composition) of $\mathrm{EQ}(F, F)$, and so if we know the group of self-equivalences of F, and manage to reconstruct at least one linear equivalence (L_1, L_2) between F and G (for instance, using Algorithm 1), we can obtain all possible linear equivalences between F and G by composing (L_1, L_2) with all the self-equivalences in $\mathrm{EQ}(F, F)$. The set of all linear self-equivalences of a given F can be computed using Algorithm 1 by taking $G = F$, and letting the algorithm run until it has exhausted all possibilities (instead of terminating the search as soon as the first self-equivalence is found). While this process can be somehat lengthy since the entire search tree has to be traversed, this is a precomputation that only needs to be performed once per function. Nonetheless, we will show that in order to compute the "orbits" that we allude to above, it is not necessary to know the group $\mathrm{EQ}(F, F)$ of self-equivalences, and a much less laborious computation suffices. Furthermore, since the group of self-equivalences of a function is of interest in its own right, it is possible that in practice it has already been computed for a given function, and so no further computations are needed.

Consider a function F, and a self-equivalence $(L_1, L_2) \in \mathrm{EQ}(F, F)$. If $L_1(x) = y$ for some $x, y \in \mathbb{F}_{p^n}$, we say that x and y lie on the same orbit (with respect to L_1). More precisely, we will define the **left orbit** of $x \in \mathbb{F}_{p^n}$ under F as the set

$$\{y \in \mathbb{F}_{p^n} : (\exists (L_1, L_2) \in \mathrm{EQ}(F, F)) L_1(x) = y\}.$$

Similarly, the **right orbit** of x under F is

$$\{y \in \mathbb{F}_{p^n} : (\exists (L_1, L_2) \in \mathrm{EQ}(F, F)) L_2(x) = y\}.$$

We now argue that when using Algorithm 1, it is enough to consider a single element from each right orbit when guessing the value of $L_2(1)$. Indeed, suppose that we are testing $F, G : \mathbb{F}_{p^n} \to \mathbb{F}_{p^n}$ for equivalence. Suppose that we have guessed $L_2(1) = v$

for some $v \in \mathbb{F}_{p^n}$, and that v' lies on the same right orbit as v. Then there exists a self-equivalence $(L_3, L_4) \in \mathrm{EQ}(F, F)$ such that $L_4(v) = v'$; and so, if F and G are linear-equivalent via $L_1 \circ F \circ L_2 = G$ with $L_2(1) = v$, then substituting the self-equivalence into $L_1 \circ F \circ L_2 = G$ yields

$$(L_1 \circ L_3) \circ F \circ (L_4 \circ L_2) = G,$$

with $(L_4 \circ L_2)(1) = L_4(v) = v'$. Thus, there is a linear equivalence (L_1, L_2) of F and G with $L_2(1) = v$ if and only if there is a linear equivalence (L_1', L_2') between F and G with $L_2'(1) = v'$. Consequently, when guessing the value of $L_2(1)$ in Algorithm 1, only one of these two values has to be considered.

In the case of the left orbits, the situation is a bit more complicated. Suppose once again that F and G are linearly equivalent via $L_1 \circ F \circ L_2 = G$, and that $(L_3, L_4) \in \mathrm{EQ}(F, F)$. Substituting $L_3 \circ F \circ L_4$ for F in the linear equivalence of F and G, we have $(L_1 \circ L_3) \circ F \circ (L_4 \circ L_2) = G$ as before. If, say, $L_3(1) = v$ so that 1 and v lie on the same left orbit under F, then we have $(L_1 \circ L_3)(1) = L_1(v)$. In this way, the left orbits allow us to change the *input* to L_1 (in other words, if we had fixed $L_1(1) = c$, we would not need to consider the case $L_1(v) = c$) but not the outputs like in the case of the right orbits. Since in our implementation of Algorithm 1 we guess values of L_1 or L_2 for a fixed set of inputs (namely, a basis of $\mathbb{F}_{p^n}$ over $\mathbb{F}_p$), it does not seem like the left orbits can be immediately applied to improve the efficiency of our approach. For this reason, in the following we mostly concentrate on the case of the right orbits.

Furthermore, we can show that the number and sizes of the left and right orbits are invariant under linear equivalence. This follows in a fairly straightforward way by observing that there is a one-to-one correspondence between the self-equivalence groups $\mathrm{EQ}(F, F)$ and $\mathrm{EQ}(G, G)$ of any two linearly equivalent discrete functions F and G. The latter fact has already been observe in e.g. [1]; for the sake of clarity, we give a self-contained proof in the following proposition. Furthermore, we give the exact form the correspondence between $\mathrm{EQ}(F, F)$ and $\mathrm{EQ}(G, G)$, and use it to show that the structure of the left and right orbits of F and G is the same.

Proposition 1. Let F and G be linearly equivalent (n, m, p)-functions via $L_1 \circ F \circ L_2 = G$. Then:

1) there is a one-to-one correspondence φ between the groups $\mathrm{EQ}(F, F)$ and $\mathrm{EQ}(G, G)$ given by
$$\varphi : (L_3, L_4) \mapsto (L_1 \circ L_3 \circ L_1^{-1}, L_2^{-1} \circ L_4 \circ L_2);$$
2) if $x, y \in \mathbb{F}_{p^n}$ lie on the same right orbit under F, then $L_2^{-1}(x)$ and $L_2^{-1}(y)$ lie on the same right orbit under G; in particular, there is a one-to-one correspondence between the right orbits of F and the right orbits of G;
3) similarly, if x and y belong to the same left orbit under F, then $L_1(x)$ and $L_1(y)$ belong to the same left orbit under G, inducing an analogical one-to-one correspondence between the left orbits of F and the left orbits of G.

Proof. Suppose $(L_3, L_4) \in \mathrm{EQ}(F, F)$ so that $L_3 \circ F \circ L_4 = G$. Composing $L_1 \circ F \circ L_2 = G$ with the inverses of L_1 and L_2, we get $F = L_1^{-1} \circ G \circ L_2^{-1}$. Substituting this for F in the self-equivalence with L_3 and L_4, we get

$$L_3 \circ L_1^{-1} \circ G \circ L_2^{-1} \circ L_4 = L_1^{-1} \circ G \circ L_2^{-1}.$$

Composing both sides of the above with L_1 and L_2, we finally get

$$(L_1 \circ L_3 \circ L_1^{-1}) \circ G \circ (L_2^{-1} \circ L_4 \circ L_2) = G,$$

so that $(L_1 \circ L_3 \circ L_1^{-1}, L_2^{-1} \circ L_4 \circ L_2) \in \mathrm{EQ}(G, G)$. Since this transformation is clearly invertible, we obtain the one-to-one correspondence described in the first item of the hypothesis.

Suppose now that we have two elements x, y that lie on the same right orbit with respect to F, i.e. there exists some $(L_3, L_4) \in \mathrm{EQ}(F, F)$ such that $L_4(x) = y$. Let $a = L_2^{-1}(x)$ and $b = L_2^{-1}(y)$ be the pre-images of x and y under L_2. Then we have that

$$(L_2^{-1} \circ L_4 \circ L_2)(a) = (L_2^{-1} \circ L_4)(x) = L_2^{-1}(y) = b,$$

and since $L_2^{-1} \circ L_4 \circ L_2 \in \mathrm{EQ}(G, G)$ as shown above, we have that a and b belong to the same right orbit under G.

The proof in the case of the left orbits is similar and we omit it here for the sake of brevity. $\square$

To the best of our knowledge, the only two useful invariants for distinguishing between CCZ-inequivalent planar functions are the sizes of their nuclei [16] and the size of the automorphism group of an associated linear code [27]. Since the exact definitions of these invariants are not immediately relevant to our work, we omit their definitions here; details can be found in the aforementioned articles. Computing the rank of the automorphism group appears to only be feasible over $\mathbb{F}_{3^n}$ for $n \leq 6$; for higher dimensions, the memory is insufficient. The nuclei, on the other hand, can be computed for dimensions greater than 6, but it is easy to see that the number of right orbits (used as an invariant) is strictly more discriminating than the sizes of the nuclei. For instance, it can be easily verified that all known quadratic planar functions over $\mathbb{F}_{3^5}$ have the same nuclei, while in Table I, we can clearly see that there are three distinct cases for the number of right orbits. Although we do not have a classification for dimensions higher than 6 at the moment, we can observe that e.g. over $\mathbb{F}_{3^8}$, the sporadic planar instance

$$\alpha^{3608}x^{1458} + \alpha^{3608}x^{738} + \alpha^{3810}x^{486} + \alpha^{3810}x^{246} + \alpha^{3413}x^{162} + \alpha^{3413}x^{82} + \alpha^{3608}x^{18} + \alpha^{3810}x^{6} + \alpha^{2565}x^{2}$$

from [17], and the planar instance

$$\alpha^{3608}x^{1458} + \alpha^{3608}x^{738} + \alpha^{3810}x^{486} + \alpha^{3810}x^{246} + \alpha^{3413}x^{162}$$

from the family defined in [15] (where α is a primitive element of $\mathbb{F}_{3^8}$) have the same nuclei, but the former has 410 right orbits, while the latter has only 12 right orbits. In this way, we can immediately establish their CCZ-inequivalence.

This leaves us with the question of how to compute the left and right orbits of the elements in $\mathbb{F}_{p^n}$ under a given function F. Clearly, if we have computed the entire group $\mathrm{EQ}(F, F)$, we can simply go through all $(L_1, L_2) \in \mathrm{EQ}(F, F)$ and compute the orbit of any $x \in \mathbb{F}_{p^n}$ from the definition. However, if we only need to compute the orbits, this is not necessary. We are able to test whether two given elements $x, y \in \mathbb{F}_{p^n}$ belong to the same orbit much faster as follows.

Suppose $x, y \in \mathbb{F}_{p^n}$ are given, and we want to check whether there exists a self-equivalence $(L_1, L_2) \in \mathrm{EQ}(F, F)$ such that $L_2(x) = y$, i.e. such that x and y belong to the same right orbit. In order to do this, we run Algorithm 1 for $G = F$, but fix $L_2(x) = y$ as the first guess for L_2. If the algorithm terminates with success (that is, if it finds at least one pair (L_1, L_2)), we can conclude that x and y do belong to the same orbit. Conversely, if Algorithm 1 terminates with failure, then x and y must belong to different orbits.

The partitioning of $\mathbb{F}_{p^n}$ into orbits can be further simplified as follows. If at any point we have determined that some elements x, y belong to the same orbit, then we must have found a self-equivalence $(L_1, L_2) \in \mathrm{EQ}(F, F)$ mapping x to y as described above. We can then apply the L_2 from this self-equivalence to all elements of $\mathbb{F}_{p^n}$ (not just x) in order to deduce further pairs of elements that belong to a common orbit. If we keep track of L_2 from all self-equivalences (L_1, L_2) that we have encountered so far, we can also consider compositions of these L_2; that is, if we have encountered L_2 and L_2', we can also take e.g. $L_2 \circ L_2'$, $L_2' \circ L_2$, $L_2 \circ L_2$ and $L_2' \circ L_2'$ (which must be the right-hand part of some self-equivalence since $\mathrm{EQ}(F, F)$ is a group under functional composition as described above), and we can apply these functions to the elements of $\mathbb{F}_{p^n}$ to deduce further information about the orbits. Finally, since belonging to a given orbit is clearly an equivalence relation, if we know that e.g. $S_1 \subseteq \mathbb{F}_{p^n}$ and $S_2 \subseteq \mathbb{F}_{p^n}$ are sets of elements such that all elements in S_1 belong to the same orbit, and all elements in S_2 belong to the same orbit, then we only need to test whether s_1, s_2 belong to the same orbit for one pair of elements s_1, s_2 such that $s_1 \in S_1$, $s_2 \in S_2$; the orbits containing S_1 and S_2 are either the same, or completely disjoint.

Pseudocode for partitioning $\mathbb{F}_{p^n}$ into orbits with respect to a given function F is given below in Algorithm 5. In this procedure, we gradually "build up" the orbits of $\mathbb{F}_{p^n}$ by initializing every element to belong to its own orbit in the beginning, and then merging orbits whenever we discover that two elements lie on the same orbit.

We keep track of the set $\mathcal{L}$ of all linear permutations L_2 that we have discovered that are part of self-equivalences. By $\mathrm{Span}(\mathcal{L})$, we refer to the set of all linear permutations that can be obtained by composing two or more permutations from $\mathcal{L}$ (with repetitions allowed). This "compositional span" is, in fact, the set of all linear permutations that we know belong to the group of self-equivalences at any given moment. Every time we discover a new L_2, we add it to $\mathcal{L}$, and attempt to merge as many of the existing orbits as possible using the new linear permutations "spanned" by $\mathcal{L}$ and L_2. This is done using the `Merge()` auxiliary function. The latter simply goes through all permutations L in $\mathrm{Span}(\mathcal{L})$, and for each L, keeps merging orbits (O_1, O_2) until no more mergers are possible. At this point, it proceeds to the next $L \in \mathrm{Span}(\mathcal{L})$. Once all liner permutations in the "span" of $\mathcal{L}$ have been exhausted, `Merge()` returns control to the main procedure.

Every time the main procedure manages to merge two (or more, thanks to `Merge()`) orbits, it goes to the beginning of the *while* loop on line 8. Eventually, an orbit O will be selected on line 9 that cannot be merged with anything more; at this point, lines 23-24 remove this orbit from the list Orbits, and add it to $\mathrm{Complete}$. Once all orbits have been transferred to $\mathrm{Complete}$, the algorithm terminates.

The intention of this method is to pre-compute the orbit representatives with respect to F for a list of known functions F that need to be tested for equivalence against others, and then use these orbit representatives to restrict the number of guesses that need to be performed by Algorithm 1. In some cases, it is not even necessary to pre-compute the orbit representatives, since the orbits of the function can be determined mathematically. This is the case, for instance, when working with monomials. As we see in the following proposition, for a monomial function $F(x)$, all the non-zero elements of $\mathbb{F}_{p^n}$ lie on a single right orbit.

Proposition 2. Let $F(x) = x^d$ be an (n, n, p)-function for some natural numbers n, p, d with p prime. Then for any $a, b \in \mathbb{F}_{p^n}^*$ there exists a pair of linear (n, n, p)-permutations (L_1, L_2) such that $L_1 \circ F \circ L_2 = F$ and $L_2(a) = b$. Consequently, all the non-zero elements of $\mathbb{F}_{p^n}$ lie on the same right orbit.

Proof. Let $L_2(x) = x\frac{b}{a}$ and $L_1(x) = x\frac{a^d}{b^d}$. Then $L_1 \circ F \circ L_2 = F$, and $L_2(a) = b$. $\square$

Besides monomials, another case where we have some a priori information about the orbits, are functions over $\mathbb{F}_{p^n}$ represented by polynomials $F(x) = \sum c_i x^i$ with all coefficients c_i belonging to some subfield $\mathbb{F}_{p^m}$ of $\mathbb{F}_{p^n}$. Then we can observe that by taking $L_1(x) = x^{p^{km}}$ and $L_2(x) = x^{p^{n-km}}$ for any k, we have $L_1 \circ F \circ L_2 = F$ since $c_i^{p^{km}} = c_i$ for all coefficients c_i. Consequently, we have the following observation. We note that many of the known 3-to-1 APN functions have coefficients in the subfield $\mathbb{F}_{2^2}$, and almost all of the known planar functions over $\mathbb{F}_{3^n}$ have coefficients in the prime field $\mathbb{F}_3$. In this way,

Algorithm 5: Partitioning $\mathbb{F}_{p^n}$ into right orbits with respect to a given $F : \mathbb{F}_{p^n} \to \mathbb{F}_{p^n}$

```
Data: An $(n, m, p)$ function $F$
Result: A partition of the finite field $\mathbb{F}_{p^n}$ into right orbits with respect to $F$
1  begin
2      # Initially, place every element $x$ in its own orbit $\{x\}$
3      $\mathrm{Orbits} \leftarrow \{\{x\} : x \in \mathbb{F}_{p^n}\}$
4      # A list of orbits that are complete (cannot be merged with anything more)
5      $\mathrm{Complete} \leftarrow \emptyset$
6      # A list of $L_2$ we have found so far
7      $\mathcal{L} \leftarrow \emptyset$
8      while $\mathrm{Orbits} \neq \emptyset$ do
9          $O \leftarrow \mathrm{Random}(\mathrm{Orbits})$
10         for $O' \in \mathrm{Random}(\mathrm{Orbits}), O' \neq O$ do
11             $o \leftarrow \mathrm{Random}(O)$
12             $o' \leftarrow \mathrm{Random}(O')$
13             Use Algorithm 1 to search for $(L_1, L_2) \in \mathrm{EQ}(F, F)$ with $L_2(o) = o'$
14             if such $(L_1, L_2)$ exists then
15                 # Replace $O$ with the union of $O$ and $O'$
16                 $\mathrm{Orbits} \leftarrow \mathrm{Orbits} \setminus \{O, O'\}$
17                 $O \leftarrow O \cup O'$
18                 $\mathrm{Orbits} \leftarrow \mathrm{Orbits} \cup \{O\}$
19                 if $L_2 \notin \mathrm{Span}(\mathcal{L})$ then
20                     $\mathcal{L} \leftarrow \mathcal{L} \cup \{L_2\}$
21                     $Merge(\mathrm{Orbits}, \mathcal{L})$
22                 continue while loop
23         # If $O$ cannot be merged with any other orbit, then it is complete
24         $\mathrm{Orbits} \leftarrow \mathrm{Orbits} \setminus \{O\}$
25         $\mathrm{Complete} \leftarrow \mathrm{Complete} \cup \{O\}$
26 Merge(Orbits, $\mathcal{L}$):
27     # Merge as many orbits as possible using the given set of functions
28     for $L \in \mathrm{Span}(\mathcal{L})$ do
29         $Change \leftarrow \mathrm{false}$
30         for $O_1 \in \mathrm{Orbits}$ do
31             $o_1 \leftarrow \mathrm{Random}(O_1)$
32             $o_2 \leftarrow L(o_1)$
33             let $O_2 \in \mathrm{Orbits}$ be such that $o_2 \in O_2$
34             if $O_1 \neq O_2$ then
35                 $\mathrm{Orbits} \leftarrow \mathrm{Orbits} \setminus \{O_1, O_2\} \cup \{O_1 \cup O_2\}$
36                 $Change \leftarrow \mathrm{true}$
37         if $Change = \mathrm{true}$ then
38             repeat for loop with the same $L$
```

Observation 1 simplifies the partitioning of $\mathbb{F}_{p^n}$ into orbits further, since we already know that many combinations of elements belong to the same orbit.

Furthermore, if U is the multiplicative kernel of F, then we can see that $F \circ L_2 = F$ for any $L_2(x) = ux$ with $u \in U$. In the case of 3-to-1 uniformly distributed functions, this means that any triple $\{x, \beta x, \beta^2 x\}$ of elements (where β is primitive in $\mathbb{F}_{2^2}$) lies on the same orbit; while in the case of uniformly distributed 2-to-1 functions over fields of odd characteristic, it means that x and $-x$ always belong to the same orbit. As we will observe later from the computational results in Table I, these observations cannot be extended further, since there exist functions with coefficients from the prime field whose right orbits are precisely all sets of the form $\{ux^{pk} : 0 \leq k \leq n-1\}$; for example, the planar trinomials $x^{10} \pm x^6 - x^2$ over $\mathbb{F}_{3^5}$ and $\mathbb{F}_{3^7}$ (and, quite likely, all odd dimensions n) are examples of functions having this kind of "minimal" right orbit partition.

Observation 1. Let $F(x) = \sum_{i=0}^{p^n-1} c_i x^i$ represent an (n, n, p)-function with multiplicative kernel U for some natural number

TABLE I
RIGHT ORBITS FOR ALL KNOWN PLANAR FUNCTIONS OVER $\mathbb{F}_{3^n}$ UP TO CCZ-EQUIVALENCE WITH $3 \leq n \leq 6$

n	ID	$F(x)$	# orbits	Orbit representatives
3	3-1	x^2	1	1
	3-2	x^4	1	1
4	4-1	x^2	1	1
	4-2	$x^4 + x^{10} - x^{36}$	2	$1, \alpha$
	4-3	x^{14}	1	1
5	5-1	x^2	1	1
	5-2	x^4	1	1
	5-3	x^{10}	1	1
	5-4	$x^{10} + x^6 - x^2$	25	$1, \alpha, \alpha^2, \alpha^4, \alpha^5, \alpha^7, \alpha^8, \alpha^{10}, \alpha^{11}, \alpha^{13}, \alpha^{16}, \alpha^{17}, \alpha^{19}, \alpha^{20}, \alpha^{22}, \alpha^{25}, \alpha^{26}, \alpha^{31}, \alpha^{34}, \alpha^{35}, \alpha^{38}, \alpha^{40}, \alpha^{61}, \alpha^{67}, \alpha^{76}$
	5-5	$x^{10} - x^6 - x^2$	25	(same as 5-4)
	5-6	$x^2 + x^{90}$	3	$1, a, a^2$
	5-7	x^{14}	1	1
	5-8	$x^{162} + x^{108} - x^{84} + x^2$	25	(same as 5-4)
6	6-1	x^2	1	1
	6-2	x^{10}	1	1
	6-3	$x^{162} + x^{84} + \alpha^{58}x^{54} + \alpha^{58}x^{28} + x^6 + \alpha^{531}x^2$	7	$1, \alpha, \alpha^2, \alpha^3, \alpha^6, \alpha^7, \alpha^{44}$
	6-4	$\alpha^{205}x^{82} + \alpha^{75}x^{30} + 2x^{28}$	3	$1, \alpha, \alpha^2$
	6-5	$2x^{270} + x^{246} + 2x^{90} + x^{82} + x^{54} + 2x^{30} + x^{10} + x^2$	4	$1, \alpha, \alpha^2, \alpha^7$
	6-6	$x^{270} + 2x^{244} + \alpha^{449}x^{162} + \alpha^{449}x^{84} + \alpha^{534}x^{54} + 2x^{36}$	12	$1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^8, \alpha^{10}, \alpha^{15}, \alpha^{17}, \alpha^{20}$
	6-7	$x^{486} + x^{252} + \alpha^{561}x^{162} + \alpha^{561}x^{84} + \alpha^{183}x^{54} + \alpha^{183}x^{28}$	33	$1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{12}, \alpha^{13}, \alpha^{16}, \alpha^{17}, \alpha^{19}, \alpha^{22}, \alpha^{23}, \alpha^{31}, \alpha^{34}, \alpha^{35}, \alpha^{36}, \alpha^{38}, \alpha^{39}, \alpha^{44}, \alpha^{45}, \alpha^{47}, \alpha^{48}, \alpha^{50}, \alpha^{54}, \alpha^{66}, \alpha^{72}, \alpha^{90}$
	6-8	x^{122}	1	1
	6-9	-	7	$1, \alpha, \alpha^2, \alpha^6, \alpha^8, \alpha^{13}, \alpha^{15}$

n and some prime number p, and suppose that all coefficients c_i belong to the subfield $\mathbb{F}_{p^m}$ for some $m \mid n$. Let k be any natural number. Then any two elements $x, y \in \mathbb{F}_{p^n}$ with $x = y^{p^{mk}}$ belong to the same left orbit under F; and any two elements $x, y \in \mathbb{F}_{p^n}$ with $x = uy^{p^{n-mk}}$ for some $u \in U$ belong to the same right orbit under F.

According to our computational results, the benefit of restricting the first guess for L_1 or L_2 to left or right orbit representatives, respectively, is most pronounced in the case when the tested functions are not equivalent. If the functions F and G are equivalent, we can intuitively see why this is so, particularly in the extremal case when all the elemnets of $\mathbb{F}_{p^n}^*$ fall into one orbit under F. In this case, any guess of the value of $L_2(1)$ leads to an equivalence (L_1, L_2) between F and G, and so restricting this first guess through the orbit representatives essentially has no effect. On the other hand, if F and G are not equivalent, then we would only have to consider one guess for the value of $L_2(1)$ instead of $p^n - 1$ guesses. A similar situation occurs for functions that partition $\mathbb{F}_{p^n}$ into a small number of orbits.

Consequently, computing the orbits is mostly useful in practice when dealing with planar, as opposed to APN, functions: in the case of APN functions, we can use the differential and extended Walsh spectra of the ortho-derivatives to distinguish between EA-inequivalent quadratic APN functions; Algorithm 1 only has to be applied when the orthoderivatives of the functions F and G under consideration have the same differential spectrum, in which case F and G are practically guaranteed to be EA-equivalent. In the case of planar functions, however, we are not aware of any efficient invariants that can be used to distinguish distinct EA-equivalence classes, and Algorithm 1 appears to be the most efficient method at the moment for deciding both the positive and the negative case.

For this reason, we have computed the orbits for all known planar functions (up to CCZ-equivalence) over $\mathbb{F}_{3^n}$ for $n \leq 6$. A list of CCZ-inequivalent representatives of the known planar functions is given in [27]. For each representative, we compute the right orbits, and give the results in Table I. The functions are indexed according to the order given in [27], e.g. function “5-3” in Table I is the same as function “3” in Table 4 of [27]. The only exception two exceptions are the function designated “5-8” corresponding to a sporadic instance from [17], and the function 6-9 originating from the Zhou-Pott family [33]. These functions were not yet published at the time of writing of [27] which is why they were not included in their classification. Since the univariate represenation of the latter is rather complicated, we do not list in the table; it is

$$\alpha^{438}x^{486} + \alpha^{180}x^{324} + \alpha^{458}x^{270} + \alpha^{672}x^{252} + \alpha^{622}x^{246} + \alpha^{94}x^{244} + \alpha^{650}x^{162} + \alpha^{441}x^{108} + \alpha^{50}x^{90} + x^{84} + \alpha^{77}x^{82} +$$
$$a^{328}x^{36} + a^{583}x^{30} + a^{407}x^{28} + a^{178}x^{18} + a^{492}x^{12} + a^{692}x^{10} + a^{78}x^6 + a^{219}x^4 + a^{69}x^2,$$

where α is a primitive element of $\mathbb{F}_{3^6}$.

A natural question that arises from the above discussion is whether the self-equivalence of F can be used to restrict more than one value of L_2, e.g. whether it is possible to restrict both $L_2(1)$ and $L_2(\alpha)$ (for some $0, 1 \neq \alpha \in \mathbb{F}_{p^n}$) at the same time. This would involve extending the concept of orbits to pairs of elements; in other words, we would say that two pairs of elements (x_1, x_2) and (y_1, y_2) belong to the same e.g. right orbit if there exists $(L_1, L_2) \in \mathrm{EQ}(F, F)$ such that $L_2(x_1) = y_1$

and $L_2(x_2) = y_2$. Then only one pair from each orbit would have to be considered when guessing the images of the first two basis elements under L_1. According to our preliminary observations, very few pairs can be eliminated in this way, and so this approach is likely not worth the effort for precomputation. We leave the closer investigation of these "orbits of pairs" as a potential problem for future work.

VI. Implementation and experimental results

We have implemented Algorithm 1 for two of the most important classes of cryptographically optimal functions, i.e. for 3-to-1 functions over $\mathbb{F}_{2^n}$, and for 2-to-1 functions over $\mathbb{F}_{3^n}$. The implementation of the former is done in C since the elements of $\mathbb{F}_{2^n}$ can be represented as sequences of bits, and C allows these to be manipulated very efficiently. In the case of the latter, there is no natural way to represent the elements of $\mathbb{F}_{3^n}$ in C, and so we have implemented the algorithm in the *Magma* algebra system instead [4]. We have also implemented the auxiliary algorithms for the case of 3-to-1 functions over $\mathbb{F}_{2^n}$ in C.

To show the efficiency of the discussed algorithms in the case of 3-to-1 functions, we present the average running times from the classification and computational experiments in dimensions 10, 12 and 14 in Table II. For lower dimensions, both Algorithm 1 and Algorithm 2 provide practically instantaneous results; the average running time is below the measurable threshold. The running times for Algorithm 4 come from the classification of the C3 infinite family in dimension 12 described in Section VII. The running times in Table II are given for the positive case only (that is, when the tested F and G are linear-equivalent). In the case when the functions are not linear-equivalent, this can virtually always be shown in practice by computing the differential spectra of their ortho-derivatives, which has a running time comparable to that of Algorithm 1. We stress that in all cases that we encountered in our experiments and classifications where two functions had the same differential spectrum of the orthoderivative, they ended up being linear equivalent.

TABLE II
Sample running times (in seconds) for the 3-to-1 algorithms described in this paper

n	6	8	10	12	14
Algorithm 1	-	-	0.039	71.008	88.999
Algorithm 2	-	-	-	0.011	?
Algorithm 3	-	-	1.241	?	?
Algorithm 4	-	-	0.946	45.644	?

In order to compare the efficiency and running times of Algorithm 1 with other known algorithms for testing equivalence, we perform the following experiment: for a given function F (APN in the case of the 3-to-1 implementation, and planar in the case of the 2-to-1 implementation), we generate a function G equivalent to it at random (for instance, in the case of linear equivalence we pick L_1 and L_2 at random and set $G = L_1 \circ F \circ L_2$), and then run the corresponding algorithm on F and G in order to recover the equivalence. We repeat this experiment several times, and give the average running time of all experiments. Table III gives a comparison of the running times necessary to verify and reconstruct linear equivalence between x^3 and randomly generated equivalent 3-to-1 functions as described above using various algorithms from the literature. The code isomorphism approach described in [19] does not work for dimensions 12 and above due to insufficient memory (this was tested on our department server with around 500 GB of memory available). The algorithm of Kaleyski from [22] was only tested for dimensions up to 10 since the results make it clear that the running time becomes prohibitively long for larger dimensions. In addition, there is an algorithm due to Canteaut et al. [12] for which we could not measure the running time since we do not have an implementation available. In the original paper [12], it is indicated that testing equivalence for functions equivalent to x^3 in dimension 8 takes around 89 seconds; we take this as a good indication that the running times of Algorithm 1 will be significantly faster than those of [12] as well.

Figure 3 presents a detailed breakdown of the running times of Algorithm 1 that we have observed throughout all of our computational experiments and classifications for dimensions $n = 10, 12, 14$. The bars visualize the proportion of functions that we have observed for which the running time is not greater than a certain bound; for instance, the first bar (labeled "$<$ 0,01") gives the percentage of functions for which the running time is less that one hundredth of a second, while the last column (labeled "$<$ 1000") gives the percentage of functions for which the running time is less than 1000 seconds. As we can see, there are some "outliers" for which the running time can be longer than in the average case, but for the majority of functions, the computation time is quite short, even in higher dimensions like $n = 14$.

In the case of quadratic planar functions over $\mathbb{F}_{3^n}$, we perform a similar experiment using our *Magma* implementation. We consider the planar monomial $F(x) = x^2$, and construct a random $G(x)$ linear-equivalent to it via $G = L_1 \circ F \circ L_2$ for randomly selected linear permutations L_1 and L_2. Note that in the case of quadratic 2-to-1 planar functions, G is always uniformly distributed whenever F is, due to $L(-x) = -L(x)$ for any $x \in \mathbb{F}_{p^n}$ and any linear (n, n, p)-function L. To the best of our knowledge, the only algorithm for testing linear equivalence between planar functions is the code isomorphism test of [19]. The observed running times are given in Table IV. The code isomorphism test cannot be used in $\mathbb{F}_{3^n}$ with $n > 7$ due to insufficient memory (even with over 500 GB available on our server). Thus, Algorithm 1 is currently the only known approach for testing CCZ-equivalence between quadratic planar functions in dimensions greater than 7.

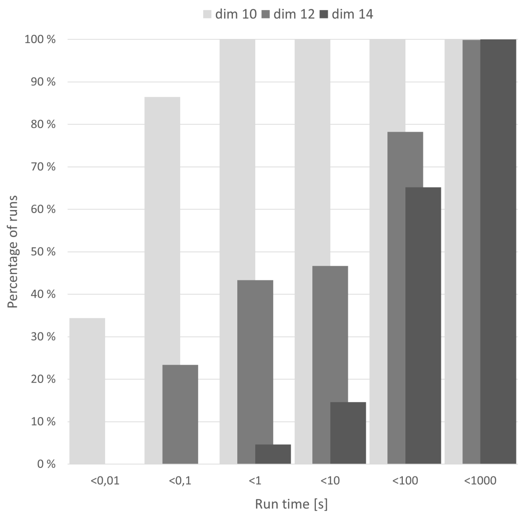

Fig. 3. Comparison of running times for Algorithm 1 for 3-to-1 uniformly distributed APN functions for dimensions $n = 10, 12, 14$

TABLE III
Comparison of running times (in seconds) for 3-to-1 APN functions

n	6	8	10	12	14	16	18
Algorithm 1	0	0	0	0.7	0.425	1.85	27.26
Kaleyski	0.07	9.26	17225.140	?	?	?	?
Code isomorphism	0.01	0.18	14.96	550.680	N/A	N/A	N/A

VII. Classification of 3-to-1 APN functions over for dimension 12

As an application, we compute a classification of all known quadratic 3-to-1 APN functions over $\mathbb{F}_{2^{12}}$ up to CCZ-equivalence. We note that a classification of these functions has already been given in [10] but up to the differential spectrum of the orthoderivative. We recall that the differential spectrum of the orthoderivative is an invariant under CCZ-equivalence in the case of quadratic APN functions; and while it can be used to show that two functions are inequivalent if they have distinct values of this differential spectrum, it does not give us any information about their equivalence or inequivalence in the case when the differential spectrum is the same. It is thus possible that a classification up to the differential spectra of the orthoderivatives is incomplete in the sense that two functions belonging to the same class (having the same differential spectrum of the orthoderivative) are, in fact, CCZ-inequivalent. The classifications for $n \leq 10$ given in [10] have been verified to be complete in this sense, since the code isomorphism test can be used to verify that the functions within every class are equivalent to each other. The classification given in Table V of [10] however, could not be previously verified, since the code isomorphism test requires too much memory and can not be used in dimensions higher than 10.

The function with index 15 in Table V does not originate from an infinite family, but rather, from a series of computational searches that we carried out over $\mathbb{F}_{2^{12}}$ using the principle of "polynomial expansion", as described for instance in [10]. In total, we found 16 polynomials with the orthoderivative differential spectrum given in the table for function 15. Since this spectrum was distinct from those of representatives from all known infinite families and known instances, we could conclude that these 16 functions represent at least one new CCZ-equivalence class. Using Algorithm 1, we verified that all of these 16 functions are pairwise equivalent, and so they represent precisely one CCZ-equivalence class. Function 15 in Table V is thus currently the only known sporadic APN instance over $\mathbb{F}_{2^{12}}$.

Out of all the known infinite families of APN functions, family C3 is by far the most problematic to classify due to the large number of functions that it contains. This family is given by the formula

$$C(x) = sx^{q+1} + x^{2^i+1} + x^{q(2^i+1)} + cx^{2^iq+1} + c^q x^{2^i+q}, \tag{2}$$

TABLE IV
Comparison of running times (in seconds) for 2-to-1 planar functions

n	4	5	6	7	8
Algorithm 1	0.06	0.06	5.551	53.792	585.26
Code isomorphism	0.09	4.8	108.427	10812.63	N/A

where $n = 2m$, $q = 2^m$, $\gcd(i, m) = 1$, $c \in \mathbb{F}_{2^n}$ and $s \in \mathbb{F}_{2^n} \setminus \mathbb{F}_q$ such that $x^{2^i+1} + cx^{2^i} + c^q x + 1$ has no solution x such that $x^{q+1} = 1$ [6]. A distinct function can be obtained for any choice of c and s, where c can be any element of $\mathbb{F}_{2^n}$, and s can be almost any element of $\mathbb{F}_{2^n}$ (with only the elements in the subfield $\mathbb{F}_q$ excluded). Even with the added restriction of $x^{2^i+1} + cx^{2^i} + c^q x + 1$ not having solutions with $x^{q+1} = 1$, this leaves us with approximately $(2^n)^2 = 2^{2n}$ functions, which is a very large number as soon as $n > 6$. In fact, just generating the list of all these functions for $n = 8$ takes close to an hour, and for $n > 8$ it is unrealistic to generate all functions, let alone classify them up to equivalence.

In order to reduce the number of functions that we need to work with, we can apply some simple tricks in order to eliminate functions that we a priori know will be equivalent to others. For instance, let $v \in \mathbb{F}_q^*$, and consider the function $C(vx)/v^{2^i+1}$ which is clearly linear-equivalent to $C(x)$. We have

$$C(vx) = sv^2 x^{q+1} + v^{2^i+1} x^{2^i+1} + v^{2^i+1} x^{q(2^i+1)} + cv^{2^i+1} x^{2^i q+1} + c^q v^{2^i+1} x^{2^i+q},$$

and so

$$C(vx)/v^{2^i+1} = sv^{1-2^i} + x^{2^i+1} + x^{q(2^i+1)} + cx^{2^i q+1} + c^q x^{2^i+q},$$

which is also a function of the form (2), except that the coefficient s has been replaced by sv^{1-2^i}. Since $\gcd(i, m) = 1$ by assumption, we have that $x \mapsto x^{1-^i}$ permutes $\mathbb{F}_q^*$. Thus, replacing the coefficient s in a function of the form (2) with sw for any $w \in \mathbb{F}_q^*$ yields a function EA-equivalent to the original function. Thus, it is enough to consider one coefficient s from each coset of $\mathbb{F}_q^*$.

Similarly, consider the function

$$C(x^{2^{n-1}})^2 = s^2 x^{q+1} + x^{2^i+1} + x^{q(2^i+1)} + c^2 x^{2^i q+1} + (c^2)^q x^{2^i+q}.$$

Again, this function is linearly equivalent to $C(x)$, with the only difference being that the coefficients (c, s) are replaced with (c^2, s^2). Provided we consider the functions (2) for all choices of s, it is thus enough to consider one c from every cyclotomic coset $\{c^{2^k} : k\}$ in order to generate all functions up to EA-equivalence.

In conclusion, if we denote by $C_{s,c}(x)$ the function from (2) parametrized by s and c, then for any $s' \in \{(sw)^{2^k} : k \in \{0, 1, \dots, n-1\}, w \in \mathbb{F}_q^*\}$, there exists a $c' \in \mathbb{F}_{2^n}$ such that $C_{s,c}$ is EA-equivalent to $C_{s',c'}$. Consequently, we only need to consider one representative s from every set of the form $\{(sw)^{2^k} : k, w\}$. This significantly reduces the number of functions that we have to consider, and makes classification possible for $n = 12$.

Since the functions from C3 are not 3-to-1 functions as defined in (2), we use Algorithms 2 and 4 to find equivalent 3-to-1 uniformly distributed representations, and then classify them using Algorithm 1.

Using Algorithm 1, we verify that the classes given in Table V of [10] cannot be "broken down" further, i.e. any two known 3-to-1 quadratic APN functions over $\mathbb{F}_{2^{12}}$ having the same differential spectrum of the orthoderivative are, in fact, CCZ-equivalent. For the convenience of the reader, we give the table with the resulting classification below.

TABLE V
CLASSIFICATION OF ALL KNOWN QUADRATIC 3-TO-1 FUNCTIONS OVER $\mathbb{F}_{2^{12}}$ UP TO EA-EQUIVALENCE

ID	Families	Representative	Ortho-derivative differential spectrum
1	Gold	x^3	$0^{9832095}, 2^{6220305}, 6^{716625}, 8^{4095}$
2	Gold	x^{33}	$0^{10077795}, 2^{5225220}, 4^{1253070}, 6^{212940}, 8^{4095}$
3	C1	$x^3 + a^{15}x^{528}$	$0^{10118010}, 2^{5186790}, 4^{1238265}, 6^{200130}, 8^{26775}, 10^{3150}$
4	C1	$x^{33} + a^{15}x^{768}$	$0^{10149615}, 2^{5124105}, 4^{1267560}, 6^{201285}, 8^{29295}, 10^{1260}$
5	C2	$x^3 + a^7x^{528}$	$0^{10241910}, 2^{5003460}, 4^{1263465}, 6^{219555}, 8^{34335}, 10^{5670}, 12^{3150}, 14^{945}, 20^{630}$
6	C2	$x^{33} + a^7x^{768}$	$0^{10171350}, 2^{5118120}, 4^{1234485}, 6^{211050}, 8^{30555}, 10^{4410}, 12^{1890}, 14^{630}, 20^{630}$
7	C3, C10	$a^{1031}x^{256} + x^{192} + ax^{130} + ax^{129} + a^{515}x^{128} + a^{64}x^{66} + x^{65} + a^{401}x^4 + x^3 + a^{200}x^2$	$0^{10278072}, 2^{4954194}, 4^{1252503}, 6^{237384}, 8^{42462}, 10^{7938}, 20^{567}$
8	C4	$x^3 + \mathrm{Tr}(x^9)$	$0^{10137531}, 2^{5156403}, 4^{1240776}, 6^{208725}, 8^{26694}, 10^{2466}, 12^{360}, 14^{156}, 16^9$
9	C4	$x^3 + a^{-1}\mathrm{Tr}(a^3x^9)$	$0^{10146186}, 2^{5159556}, 4^{1213488}, 6^{219798}, 8^{30204}, 10^{3537}, 12^{324}, 14^{27}$
10	C5	$x^3 + \mathrm{Tr}_3^{12}(x^9 + x^{18})$	$0^{10171467}, 2^{5096757}, 4^{1259532}, 6^{215145}, 8^{26904}, 10^{2664}, 12^{396}, 14^{144}, 16^{63}, 18^{36}, 26^{12}$
11	C5	$x^3 + a^{-1}\mathrm{Tr}_3^{12}(a^3x^9 + a^6x^{18})$	$0^{10164375}, 2^{5105754}, 4^{1262763}, 6^{209601}, 8^{27261}, 10^{2835}, 12^{459}, 14^{72}$
12	C6	$x^3 + \mathrm{Tr}_3^{12}(x^{18} + x^{36})$	$0^{10156995}, 2^{5113977}, 4^{1268280}, 6^{203805}, 8^{26442}, 10^{3060}, 12^{432}, 14^{120}, 24^9$
13	C6	$x^3 + a^{-1}\mathrm{Tr}_3^{12}(a^6x^{18} + a^{12}x^{36})$	$0^{10173339}, 2^{5094351}, 4^{1259388}, 6^{215262}, 8^{27090}, 10^{2943}, 12^{657}, 14^{90}$
14	C10, C12	$a^{833}x^{768} + a^{581}x^{516} + a^{329}x^{264} + x^{192} + x^{129} + a^{65}x^{128} + x^{66} + a^{2211}x^{65} + a^{77}x^{12} + x^3 + a^2x^2$	$0^{10120950}, 2^{5169087}, 4^{1263276}, 6^{191835}, 8^{25452}, 10^{2268}, 12^{189}, 36^{63}$
15	sporadic [10]	$x^3 + \delta^{42}x^{66} + \delta^{21}x^{129} + \delta^{14}x^{1536}$	$0^{10231011}, 2^{5093109}, 4^{1162917}, 6^{228501}, 8^{42462}, 10^{2268}, 12^{6615}, 16^{1134}, 20^{3969}, 22^{1134}$

We note that we only had to apply Algorithm 1 in order to verify that any pair of tested functions is EA-equivalent. In particular, it was never necessary to search for uniformly distributed 3-to-1 functions additively equivalent to other uniformly distributed 3-to-1 functions.

As a further illustration of the efficiency of our algorithm, Table VI gives the number of functions that we had to classify in each case, and the average running time for verifying the equivalence per function.

TABLE VI
RUNNING TIMES (IN SECONDS) FOR VERIFYING THE CLASSIFICATION OF THE KNOWN QUADRATIC 3-TO-1 APN FUNCTIONS IN $\mathbb{F}_{2^{12}}$

ID	Number	Average	Minimum	Maximum
1	1	-	-	-
2	1	-	-	-
3	2	0.46	0.46	0.46
4	2	0.89	0.89	0.89
5	2	0.77	0.77	0.77
6	2	0.90	0.90	0.90
7	32256	7.20	0.01	65.43
8	1365	64.46	0.03	358.22
9	2730	113.53	0.03	575.23
10	1365	122.82	0.03	700.07
11	2730	213.79	0.03	1103.44
12	1365	71.90	0.03	399.20
13	2730	204.20	0.02	1006.48
14	168	0.03	0.02	0.04
15	16	0.36	0.01	0.66

VIII. CONCLUSION AND FUTURE WORK

We have developed and implemented an algorithm for efficiently testing linear equivalence between k-to-1 functions with a particular multiplicative structure, and have demonstrated that many of the known APN and planar functions exhibit this structure and can therefore be classified up to linear equivalence using this algorithm. In the case of planar functions, linear equivalence coincides with the more general notion of CCZ-equivalence, under which such functions are typically classified. In the case of 3-to-1 APN functions, we describe several auxiliary algorithms that can be used to extend the test of linear equivalence to one of EA-equivalence; for quadratic 3-to-1 APN functions (which are by far the most frequently encountered case), EA-equivalence coincides with CCZ-equivalence, and so we can classify them up to this more general equivalence relation using the framework developed in our paper as well.

We have introduced the notion of left and right self-equivalence orbits, provided an algorithm for partitioning the finite field $\mathbb{F}_{p^n}$ into orbits with respect to a given function, and explained how these orbits can be used to reduce the computation time for testing equivalence even further. We show that the number and sizes of the orbits are invariant under linear equivalence, and can be used to distinguish between CCZ-inequivalent planar functions more efficiently than using the previously known invariants. We give tables of the orbit representatives for all known planar functions over $\mathbb{F}_{3^n}$ with $n \leq 6$, and give some observations on the structure of the orbits of particular classes of functions (monomials, and functions whose univariate representation only has coefficients in a subfield).

We use the developed algorithms to classify all known quadratic 3-to-1 APN functions over $\mathbb{F}_{2^{12}}$, and present detailed running times illustrating the efficiency of our approach.

A straightforward direction for future work would be to run computational searches for e.g. 2-to-1 planar functions and 3-to-1 APN functions in high dimensions (where classification has not been possible before) and use the proposed suite of algorithms for classifying them up to equivalence. This could provide us with new sporadic instances of functions that may lead to new infinite families and constructions.

Another aspect to consider would be whether a similar approach can be used to test equivalence between functions that are not necessarily uniformly distributed, or even k-to-1, or whether the proposed methods can be adapted to other notions of equivalence, such as CCZ-equivalence. We note that for quadratic planar and APN functions, CCZ-equivalence coincides with linear equivalence and EA-equivalence, respectively, and so can be tested using the proposed methods. For functions that are not necessarily planar or APN, or for planar and APN functions of higher algebraic degree, EA-equivalence can be less general than CCZ-equivalence, and so adapting the approach to the more general case of CCZ-equivalence would be beneficial. At the moment, non-quadratic planar and APN instances are quite rare, but constructing such functions is an important open problem, and we might expect to see more of them in the future.

It would be interesting to identify other classes of uniformly distributed functions (besides 2-to-1 planar and 3-to-1 APN functions) that are of interest, and to test the efficiency of the proposed algorithms for deciding their equivalence. If such functions are k-to-1 for large enough k, finding an efficient way to guess the "configurations" of k-tuples (as described in the commentary on Algorithm 1) would be interesting.

Finally, mathematically computing the left and right orbits of various classes of functions would be an interesting exercise, and prove useful especially for classifying planar functions in higher dimensions, where the computation of the orbits may take a very long time.

ACKNOWLEDGMENTS

This research is partly sponsored by the Trond Monh foundation.

References

[1] Christof Beierle, Marcus Brinkmann, and Gregor Leander. Linearly self-equivalent APN permutations in small dimension. *IEEE Transactions on Information Theory*, 2021.

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

[3] Alex Biryukov, Christophe De Canniere, An Braeken, and Bart Preneel. A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In *International conference on the theory and applications of cryptographic techniques*, pages 33–50. Springer, 2003.

[4] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system I: The user language. *Journal of Symbolic Computation*, 24(3-4):235–265, 1997.

[5] KA Browning, JF Dillon, MT McQuistan, and AJ Wolfe. An APN permutation in dimension six. *Finite Fields: theory and applications*, 518:33–42, 2010.

[6] Lilya Budaghyan and Claude Carlet. Classes of quadratic APN trinomials and hexanomials and related structures. *IEEE Transactions on Information Theory*, 54(5):2354–2357, 2008.

[7] Lilya Budaghyan, Claude Carlet, and Alexander Pott. New classes of almost bent and almost perfect nonlinear polynomials. *IEEE Transactions on Information Theory*, 52(3):1141–1152, 2006.

[8] Lilya Budaghyan and Tor Helleseth. New perfect nonlinear multinomials over $\mathbb{F}_{p^{2k}}$ for any odd prime p. In *International Conference on Sequences and Their Applications*, pages 403–414. Springer, 2008.

[9] Lilya Budaghyan and Tor Helleseth. New commutative semifields defined by new pn multinomials. *Cryptography and communications*, 3(1):1–16, 2011.

[10] Lilya Budaghyan, Ivana Ivkovic, and Nikolay Kaleyski. Triplicate functions. *arXiv preprint arXiv:2021/1387*, 2021.

[11] Lilya Budaghyan and Oleksandr Kazymyrov. Verification of restricted EA-equivalence for vectorial boolean functions. In *International Workshop on the Arithmetic of Finite Fields*, pages 108–118. Springer, 2012.

[12] Anne Canteaut, Alain Couvreur, and Léo Perrin. Recovering or testing extended-affine equivalence. *arXiv preprint arXiv:2103.00078*, 2021.

[13] Claude Carlet. *Boolean functions for cryptography and coding theory*. Cambridge University Press, 2021.

[14] Claude Carlet, Pascale Charpin, and Victor A. Zinoviev. Codes, bent functions and permutations suitable for DES-like cryptosystems. *Designs, Codes and Cryptography*, 15(2):125–156, 1998.

[15] Stephen D Cohen and Michael J Ganley. Commutative semifields, two dimensional over their middle nuclei. *Journal of Algebra*, 75(2):373–385, 1982.

[16] Robert S Coulter and Marie Henderson. Commutative presemifields and semifields. *Advances in Mathematics*, 217(1):282–304, 2008.

[17] Robert S Coulter and Pamela Kosick. Commutative semifields of order 243 and 3125. *Finite Fields: Theory and Applications, in: Contemp. Math*, 518:129–136, 2010.

[18] Itai Dinur and Adi Shamir. Breaking Grain-128 with dynamic cube attacks. In *International Workshop on Fast Software Encryption*, pages 167–187. Springer, 2011.

[19] Daniel Edel and Alexander Pott. On the equivalence of nonlinear functions. In *Enhancing Cryptographic Primitives with Techniques from Error Correcting Codes*, volume 23, pages 87–103. IOS Press, 2009.

[20] Xiang-dong Hou. Permutation polynomials over finite fieldsa survey of recent advances. *Finite Fields and Their Applications*, 32:82–119, 2015.

[21] Valeriya Idrisova. On an algorithm generating 2-to-1 apn functions and its applications to the big apn problem. *Cryptography and Communications*, 11(1):21–39, 2019.

[22] Nikolay Kaleyski. Deciding ea-equivalence via invariants. *Cryptography and Communications*, 14(2):271–290, 2022.

[23] Lars R Knudsen. Truncated and higher order differentials. In *International Workshop on Fast Software Encryption*, pages 196–211. Springer, 1994.

[24] Sihem Mesnager and Longjiang Qu. On two-to-one mappings over finite fields. *IEEE Transactions on Information Theory*, 65(12):7884–7895, 2019.

[25] Ferruh Özbudak, Ahmet Sınak, and Oğuz Yayla. On verification of restricted extended affine equivalence of vectorial boolean functions. In *International Workshop on the Arithmetic of Finite Fields*, pages 137–154. Springer, 2014.

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

[27] Alexander Pott and Yue Zhou. Switching construction of planar functions on finite fields. In *International Workshop on the Arithmetic of Finite Fields*, pages 135–150. Springer, 2010.

[28] Ana Salagean. Discrete antiderivatives for functions over fpn. 2019.

[29] Guobiao Weng, Yin Tan, and Guang Gong. On quadratic almost perfect nonlinear functions and their related algebraic object. In *Workshop on Coding and Cryptography*, pages 57–68. Citeseer, 2013.

[30] Satoshi Yoshiara. Equivalences of quadratic APN functions. *Journal of Algebraic Combinatorics*, 35(3):461–475, 2012.

[31] Yuyin Yu, Mingsheng Wang, and Yongqiang Li. A matrix approach for constructing quadratic APN functions.

[32] Yuyin Yu, Mingsheng Wang, and Yongqiang Li. A matrix approach for constructing quadratic APN functions. *Designs, codes and cryptography*, 73(2):587–600, 2014.

[33] Yue Zhou and Alexander Pott. A new family of semifields with 2 parameters. *Advances in Mathematics*, 234:43–60, 2013.