Designs, Codes and Cryptography (2020) 88:471–486
https://doi.org/10.1007/s10623-019-00687-w

Discrete antiderivatives for functions over $\mathbb{F}_p^n$

Ana Sălăgean[1]

Received: 8 February 2019 / Revised: 12 July 2019 / Accepted: 3 October 2019 /
Published online: 12 November 2019

Abstract
In the design of cryptographic functions, the properties of their discrete derivatives have to be carefully considered, as many cryptographic attacks exploit these properties. One can therefore attempt to first construct derivatives with the desired properties and then recover the function itself. Recently Suder developed an algorithm for reconstructing a function (also called antiderivative) over the finite field $\mathbb{F}_{2^n}$ given its discrete derivatives in up to n linearly independent directions. Pasalic et al. also presented an algorithm for determining a function over $\mathbb{F}_{p^n}$ given one of its derivatives. Both algorithms involve solving a $p^n \times p^n$ system of linear equations; the functions are represented as univariate polynomials over $\mathbb{F}_{p^n}$. We show that this apparently high computational complexity is not intrinsic to the problem, but rather a consequence of the representation used. We describe a simpler algorithm, with quasilinear complexity, provided we work with a different representation of the functions. Namely they are polynomials in n variables over $\mathbb{F}_p$ in algebraic normal form (for $p > 2$, additionally, we need to use the falling factorial polynomial basis) and the directions of the derivatives are the canonical basis of $\mathbb{F}_p^n$. Algorithms for other representations (the directions of the derivatives not being the canonical basis vectors or the univariate polynomials over $\mathbb{F}_{p^n}$ mentioned above) can be obtained by combining our algorithm with converting between representations. However, the complexity of these conversions is, in the worst case, exponential. As an application, we develop a method for constructing new quadratic PN (Perfect Nonlinear) functions. We use an approach similar to the one of Suder, who used antiderivatives to give an alternative formulation of the methods of Weng et al. and Yu et al. for searching for new quadratic APN (Almost Perfect Nonlinear) functions.

Keywords Discrete derivative · Antiderivative · Algebraic normal form · Planar function · PN function · APN function

Mathematics Subject Classification 94A60 · 11T55

Communicated by P. Charpin.

✉ Ana Sălăgean
A.M.Salagean@lboro.ac.uk

[1] Department of Computer Science, Loughborough University, Loughborough, UK

1 Introduction

A function f with an n-bit input and n-bit output can be represented in various ways. In cryptographic applications, it is often represented as a univariate polynomial of degree up to $2^n - 1$ over $\mathbb{F}_{2^n}$, the finite field with 2^n elements. More generally, functions over $\mathbb{F}_{p^n}$ can be represented as univariate polynomials of degree up to $p^n - 1$ over $\mathbb{F}_{p^n}$. Another representation, which is the one we will mostly use here, is as a function $f : \mathbb{F}_p^n \to \mathbb{F}_p^m$ (which includes the more general possibility of m different from n), $f = (f_1, \ldots, f_m)$ with each component function $f_i : \mathbb{F}_p^n \to \mathbb{F}_p$ being a polynomial in n variables over $\mathbb{F}_p$, of degree at most $p - 1$ in each variable; the polynomials are usually represented in ANF (Algebraic Normal Form), i.e. as a sum of monomials, each monomial being a constant multiplied by the product of powers of some of the variables.

Many cryptographic attacks (e. g. differential attacks, higher order differential attacks) exploit properties of the discrete derivatives of the functions. The discrete derivative (also called simply derivative when no danger of confusion exists) of a function f in the direction $\mathbf{a}$ is defined as the function $\mathrm{D}_{\mathbf{a}} f(\mathbf{x}) = f(\mathbf{x} + \mathbf{a}) - f(\mathbf{x})$.

Recently Suder [4] considered a natural problem: reconstruct a function f over $\mathbb{F}_{2^n}$ from its discrete derivatives in n (or fewer) linearly independent directions. This operation can be called antiderivative, or integration, in analogy to its continuous counterpart. The proposed algorithm involves solving a $2^n \times 2^n$ system of linear equations; it is stated [4, Sect. 3.1] that the matrices are sparse and easy to compute with, but no complexity analysis is presented. Pasalic et al. [2] consider the problem of determining a function over $\mathbb{F}_{p^n}$ given one of its derivatives; similarly, their algorithm involves solving a $p^n \times p^n$ system of linear equations. Earlier, Xiong et al. [6] determined some necessary conditions for a function to be the discrete derivative of another function. In all these papers the functions are represented as univariate polynomials over $\mathbb{F}_{p^n}$.

While the representation as univariate polynomials over $\mathbb{F}_{p^n}$ is very useful for some purposes, it turns out that for computing the antiderivative the second representation above is much more convenient; when $p > 2$, additionally, we have to modify the algebraic normal form by using falling factorials instead of powers of variables, i.e. using $x^{\underline{d}} = x(x-1)\cdots(x-(d-1))$ instead of x^d; it is well known that such a representation is more convenient for discrete differentiation. When the directions of the derivatives are vectors in the canonical basis and the input polynomials are represented as the list of their monomials, we present an algorithm which is quasilinear in the size of the input polynomials (see Sect. 3). Hence the apparently exponential computational complexity of the previous algorithms is not intrinsic to the problem, but rather a consequence of the representation used. Our algorithm is similar to the one used in calculus for reconstructing a function from its gradient/partial derivatives (see for example the textbook [3, Section 15.8]). This is possible (despite obvious differences between the classical derivative from calculus and the discrete derivative) because for multivariate polynomial functions over $\mathbb{F}_p$ represented as described above, the discrete derivative behaves similarly to the formal derivative.

Algorithms for other representations can be obtained by combining our algorithm with converting between representations. When the functions are represented as multivariate polynomials but the directions of the derivatives are arbitrary, our algorithm can be combined with linear changes of coordinates (using $n \times n$ matrices) which transform those directions into the canonical basis directions; changes of coordinates can, in the worst case, cause the number of monomials in the algebraic normal form representation to increase exponentially.

If the functions are given as univariate polynomials over $\mathbb{F}_{p^n}$, as considered in the work of Suder [4] and of Pasalic et al. [2], we can obtain an alternative to their antiderivative algorithm; namely we transform the function into its multivariate polynomial representation, apply our quasilinear antiderivative algorithm, and then transform the result back to the univariate representation (see Sect. 4). Unfortunately, the conversion between representations has exponential worst case complexity.

As applications of the antiderivative construction, Pasalic et al. [2] prove results regarding the degree of planar functions, whereas Suder [4] prove results regarding the construction of APN (Almost Perfect Nonlinear) functions. Both these classes of functions are defined by the properties of their derivatives. Differential attacks on cryptographic functions exploit the situation where there is a direction $\mathbf{a}$ such that the derivative $\mathrm{D}_{\mathbf{a}} f(\mathbf{x})$ takes a particular value significantly more often than other values. To withstand such attacks, we can require that $\mathrm{D}_{\mathbf{a}} f$ is bijective (only possible when $p \geq 3$) or 2-to-1 (for $p = 2$). In the first situation the function f is called PN(Perfect Nonlinear), or planar, and in the second situation it is called APN (Almost Perfect Nonlinear).

Weng et al. [5] and of Yu et al. [7] independently presented a method of constructing quadratic APN functions by constructing $n \times n$ matrices over $\mathbb{F}_{2^n}$ with certain properties. Using this characterisation and a computer search they obtain numerous new quadratic APN functions. Suder [4] applies his method to obtain an alternative approach to the methods in [5,7]. He also suggested that antiderivatives could be used more generally, for other degrees and for other properties of the derivatives. Following these lines, we revisit and expand such applications in Sect. 5.

We give an analogue of [7, Theorem 1] for characterising quadratic APN functions in multivariate ANF representation. More interestingly, we present a similar result concerning quadratic PN functions. This leads to a method of constructing potentially new PN functions either from scratch (for example, we computed exhaustively all PN functions for $n = 3$ and $p \leq 7$ up to extended affine equivalence) or by modifying known PN functions in a manner similar to Yu et al. [7]. A more extensive investigation of this direction will be a topic of further research.

2 Preliminaries

We recall definitions and known results needed for the rest of the paper. The finite field with p^n elements will be denoted $\mathbb{F}_{p^n}$, where p is a prime. When K is any field, we denote the canonical basis of the vector space K^n by $\mathbf{e}_1, \ldots, \mathbf{e}_n$, i.e. the vector $\mathbf{e}_i$ has a 1 in position i and zeroes everywhere else; we denote by $\mathbf{0}$ the all-zero vector. In general we will denote vectors by boldface font.

Boolean functions in n variables can be viewed as functions $f : \mathbb{F}_2^n \to \mathbb{F}_2$. More generally, we can consider functions $f : \mathbb{F}_p^n \to \mathbb{F}_p$. It is well known that any such function can be uniquely represented in its ANF (Algebraic Normal Form) i.e. as a polynomial function described by a polynomial in $\mathbb{F}_p[x_1, \ldots, x_n]$ of degree at most $p-1$ in each variable, $f = \sum_t c_t t$ where $c_t \in \mathbb{F}_p$ and t ranges over all the power products $x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}$ with $i_1, \ldots, i_n \in \{0, 1, \ldots, p-1\}$. We denote by $\deg(f)$ the degree of this polynomial (also called algebraic degree of f) and by $\deg_{x_i}(f)$ the degree in the variable x_i. By convention, the degree of the zero polynomial is -1.

Recall that the *falling factorial* is defined as $x^{\underline{d}} = x(x-1)\cdots(x-(d-1))$, for $d \geq 0$ an integer (with $x^{\underline{0}} = 1$). Note that if we view this expression as a polynomial in x, then

$\deg(x^{\underline{d}}) = \deg(x^d) = d$. It is well known that polynomials (over any field) can be represented in a basis consisting of falling factorials. In our case, multivariate polynomials of degree at most $p-1$ in each variable can be represented as $f = \sum_t c_t t$ where $c_t \in \mathbb{F}_p$ and t ranges over all the terms $x_1^{\underline{i_1}} x_2^{\underline{i_2}} \cdots x_n^{\underline{i_n}}$ with $i_1, \ldots, i_n \in \{0, 1, \ldots, p-1\}$. We will call this representation the falling factorial ANF. Obviously, for $p = 2$ this is the same as the ANF.

More generally, we consider functions $f : \mathbb{F}_p^n \to \mathbb{F}_p^m$ with $f = (f_1, \ldots, f_m)$, $f_i : \mathbb{F}_p^n \to \mathbb{F}_p$, and each component f_i written in its ANF or in the falling factorial ANF, so $f_i = \sum_t c_t^{(i)} t$. For a compact notation we can write f as one multivariate polynomial $f = \sum_t \mathbf{c}_t t$, where this time the coefficients $\mathbf{c}_t \in \mathbb{F}_p^m$ are vectors $\mathbf{c}_t = (c_t^{(1)}, \ldots, c_t^{(m)})$. When we evaluate the polynomial, each variable takes values in $\mathbb{F}_p$ so the coefficients will only be multiplied by a scalar. By abuse of terminology we will call this the ANF of f, or the falling factorial ANF of f, respectively. This representation also allows us to define the degree of f as the highest degree of a term t for which $\mathbf{c}_t \neq \mathbf{0}$ (it will also equal the highest degree of the components $f_1, \ldots, f_m$).

Example 1 Let $f : \mathbb{F}_2^3 \to \mathbb{F}_2^2$ be the function given by

$$f(x_1, x_2, x_3) = (x_1x_2x_3 + x_1x_2 + x_1x_3 + x_3, x_1x_3 + x_3 + 1).$$

We can write this compactly as a polynomial

$$f(x_1, x_2, x_3) = (1, 0)x_1x_2x_3 + (1, 0)x_1x_2 + (1, 1)x_1x_3 + (1, 1)x_3 + (0, 1).$$

Evaluating at, say, $(1, 1, 0)$ we obtain $f(1, 1, 0) = (1, 0) + (0, 1) = (1, 1)$.

Alternatively, functions with n bits input and n bits output are often viewed as $f : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ represented as univariate polynomials over $\mathbb{F}_{2^n}$ of degree up to $2^n - 1$. In Sect. 4 we will consider this representation and the conversion between the representations.

We recall the definition of (discrete) derivative/ differentiation:

Definition 1 Let K be a field and $f : K^n \to K^m$ be a function in n variables $x_1, \ldots, x_n$. Let $\mathbf{a} = (a_1, \ldots, a_n) \in K^n \setminus \{\mathbf{0}\}$. The *differentiation* operator in the direction of $\mathbf{a}$ associates to each function f its *discrete derivative* $\mathrm{D}_{\mathbf{a}} f : K^n \to K^m$ defined as

$$\mathrm{D}_{\mathbf{a}} f(x_1, \ldots, x_n) = f(x_1 + a_1, \ldots, x_n + a_n) - f(x_1, \ldots, x_n).$$

Denoting $\mathbf{x} = (x_1, \ldots, x_n)$ we can also write $\mathrm{D}_{\mathbf{a}} f(\mathbf{x}) = f(\mathbf{x} + \mathbf{a}) - f(\mathbf{x})$.

Repeated differentiation in the directions $\mathbf{a}_1, \ldots, \mathbf{a}_k \in K^n$ will be denoted $\mathrm{D}^{(k)}_{\mathbf{a}_1, \ldots, \mathbf{a}_k} f = \mathrm{D}_{\mathbf{a}_1} \mathrm{D}_{\mathbf{a}_2} \ldots \mathrm{D}_{\mathbf{a}_k} f$.

The inverse of the (discrete) differentiation has been considered in [4,6]:

Definition 2 Let K be a field $\mathbf{a}_1, \ldots, \mathbf{a}_k \in K^n \setminus \{\mathbf{0}\}$ and $g_1, \ldots, g_k : K^n \to K^m$. If there is a function $f : K^n \to K^m$ such that $\mathrm{D}_{\mathbf{a}_i} f = g_i$ for $i = 1, \ldots, k$, then f is called a (discrete) *antiderivative* (or integral) of $g_1, \ldots, g_k$. The set of such functions f will be denoted

$$Antiderivative(n, k, (g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})), (\mathbf{a}_1, \ldots, \mathbf{a}_k))$$

Note that an antiderivative might not exist, and when it does exist it is not unique, e.g. the addition of a constant preserves the property. Other antiderivatives exist when $k < n$; this will be discussed later.

We recall a few useful well-known properties of the discrete derivative. They are straightforward to prove; for point (v) see [1].

Proposition 1 *Let f, f_1, $f_2 : \mathbb{F}_p^n \to \mathbb{F}_p^m$, $\mathbf{a}, \mathbf{b} \in \mathbb{F}_p^n \setminus \{\mathbf{0}\}$ and $c_1, c_2 \in \mathbb{F}_p$.*

(i) *Differentiation is a linear operator, i.e.* $\mathrm{D}_{\mathbf{a}}(c_1 f_1 + c_2 f_2) = c_1 \mathrm{D}_{\mathbf{a}} f_1 + c_2 \mathrm{D}_{\mathbf{a}} f_2$.
(ii) $\mathrm{D}_{\mathbf{e}_i} x_i^d = d x_i^{\underline{d-1}}$
(iii) $\mathrm{D}_{\mathbf{a}+\mathbf{b}} f(\mathbf{x}) = \mathrm{D}_{\mathbf{a}} f(\mathbf{x} + \mathbf{b}) + \mathrm{D}_{\mathbf{b}} f(\mathbf{x})$
(iv) *If M is an invertible $n \times n$ matrix over $\mathbb{F}_p$ and g is obtained from f by the linear invertible change of coordinates described by M, i.e. $g(\mathbf{x}) = f(M\mathbf{x})$, then* $\mathrm{D}_{\mathbf{a}} g(\mathbf{x}) = (\mathrm{D}_{M\mathbf{a}} f)(M\mathbf{x})$.
(v) $\deg(\mathrm{D}_{\mathbf{a}} f) \leq \deg(f) - 1$.
(vi) $\deg_{x_i}(\mathrm{D}_{\mathbf{e}_i} f) = \deg_{x_i}(f) - 1$.
(vii) $\mathrm{D}_{\mathbf{a}} \mathrm{D}_{\mathbf{b}} f = \mathrm{D}_{\mathbf{b}} \mathrm{D}_{\mathbf{a}} f$.
(viii) $\mathrm{D}^{(p)}_{\mathbf{a},\ldots,\mathbf{a}} f = 0$.

For any $c \in \mathbb{F}_p$, applying inductively Proposition 1(iii) for $\mathbf{b} = i\mathbf{a}$, we obtain

$$\mathrm{D}_{c\mathbf{a}} f(\mathbf{x}) = \sum_{j=0}^{c-1} \mathrm{D}_{\mathbf{a}} f(\mathbf{x} + j\mathbf{a}).$$

Furthermore, we can express the derivative in an arbitrary direction in terms of the derivatives in the canonical directions:

Proposition 2 *Let $f : \mathbb{F}_p^n \to \mathbb{F}_p^m$ and $\mathbf{a} \in \mathbb{F}_p^n \setminus \{\mathbf{0}\}$. Assume $\mathbf{a} = \sum_{i=1}^k c_i \mathbf{b}_i$ for some $c_1, \ldots, c_k \in \mathbb{F}_p$ and some $\mathbf{b}_1, \ldots, \mathbf{b}_k \in \mathbb{F}_p^n \setminus \{\mathbf{0}\}$. Then*

$$\mathrm{D}_{\mathbf{a}} f(\mathbf{x}) = \sum_{i=1}^{k} \sum_{j=0}^{c_i - 1} (\mathrm{D}_{\mathbf{b}_i} f) \left(\mathbf{x} + j\mathbf{b}_i + \sum_{s=i+1}^{k} c_s \mathbf{b}_s \right). \tag{1}$$

In particular for $\mathbf{a} = (a_1, \ldots, a_n) = \sum_{i=1}^n a_i \mathbf{e}_i$ we have

$$\mathrm{D}_{\mathbf{a}} f(\mathbf{x}) = \sum_{i=1}^{n} \sum_{j=0}^{a_i - 1} (\mathrm{D}_{\mathbf{e}_i} f) \left(\mathbf{x} + j\mathbf{e}_i + \sum_{s=i+1}^{n} a_s \mathbf{e}_s \right). \tag{2}$$

For $p = 2$ the two equations above become:

$$\mathrm{D}_{\mathbf{a}} f(\mathbf{x}) = \sum_{i=1}^{k} (c_i \mathrm{D}_{\mathbf{b}_i} f) \left(\mathbf{x} + \sum_{s=i+1}^{k} c_s \mathbf{b}_s \right).$$

and

$$\mathrm{D}_{\mathbf{a}} f(\mathbf{x}) = \sum_{i=1}^{n} (a_i \mathrm{D}_{\mathbf{e}_i} f) \left(\mathbf{x} + \sum_{s=i+1}^{n} a_s \mathbf{e}_s \right).$$

3 Antiderivatives for functions over $\mathbb{F}_p^n$

We consider the problem of retrieving a function $f : \mathbb{F}_p^n \to \mathbb{F}_p^m$ when we know its derivatives $g_1, \ldots, g_k$ in $k \leq n$ linearly independent directions $\mathbf{a}_1, \ldots, \mathbf{a}_k$, i.e. we want to compute the antiderivative. Note that it suffices to consider directions which are linearly independent; if we were given, additionally, the derivative g in a direction $\mathbf{a}$ which is a linear combination of $\mathbf{a}_1, \ldots, \mathbf{a}_k$, that would not provide any additional information about f, as g could be computed from $g_1, \ldots, g_k$ using Proposition 2(1).

In order for a solution to exist, the g_i have to satisfy certain compatibility conditions (see [4,6])). The first set of conditions is

$$\mathrm{D}^{(p-1)}_{\mathbf{a}_i,\ldots,\mathbf{a}_i} g_i = 0 \text{ for all } i = 1,\ldots,k \tag{3}$$

which is necessary in a field of characteristic p as $g_i = \mathrm{D}_{\mathbf{a}_i} f$ (see Proposition 1(viii)). The second set of conditions is

$$\mathrm{D}_{\mathbf{a}_i} g_j = \mathrm{D}_{\mathbf{a}_j} g_i \text{ for all } i, j = 1,\ldots,k \tag{4}$$

as both sides should be equal to $\mathrm{D}_{\mathbf{a}_i}\mathrm{D}_{\mathbf{a}_j} f = \mathrm{D}_{\mathbf{a}_j}\mathrm{D}_{\mathbf{a}_i} f$, see Proposition 1(vii). The fact that these conditions are not only necessary, but also sufficient, will follow from the construction of f below.

3.1 Antiderivatives in the directions of the canonical basis

First we consider the case when the directions are vectors in the canonical basis $\mathbf{a}_1 = \mathbf{e}_{\ell_1}, \ldots, \mathbf{a}_k = \mathbf{e}_{\ell_k}$. In this case the compatibility condition (3) means that $\deg_{x_{\ell_i}}(g_i) \leq p-2$. (see Proposition 1(vi)). Algorithm 1 constructs the antiderivative when $p = 2$ and Algorithm 2 for arbitrary p. These algorithms are similar to the method used in calculus for reconstructing a function from its gradient (see for example the textbook [3]).

Algorithm 1 AD2$(n, k, (g_1, \ldots, g_k), (\mathbf{e}_{\ell_1}, \ldots, \mathbf{e}_{\ell_k}))$

Input: $n, k, (g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})), (\mathbf{e}_{\ell_1}, \ldots, \mathbf{e}_{\ell_k})$with $g_i : \mathbb{F}_2^n \to \mathbb{F}_2^m$, $\deg_{x_{\ell_i}} g_i \leq 0$ and $\mathrm{D}_{\mathbf{e}_{\ell_i}} g_j = \mathrm{D}_{\mathbf{e}_{\ell_j}} g_i$ for all $i \neq j$

Output: $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ such that $\mathrm{D}_{\mathbf{e}_{\ell_i}} f = g_i$ for all $i = 1, \ldots, k$

```
f ← 0
for i = 1 to k do
    h ← g_i − D_{e_{ℓ_i}} f
    f ← f + x_{ℓ_i} h
end for
return f
```

Algorithm 2 AD$(n, k, (g_1, \ldots, g_k), (\mathbf{e}_{\ell_1}, \ldots, \mathbf{e}_{\ell_k}))$

Input: n, k, $(g_1(\mathbf{x}), \ldots, g_k(\mathbf{x}))$, $(\mathbf{e}_{\ell_1}, \ldots, \mathbf{e}_{\ell_k})$ with $g_i : \mathbb{F}_p^n \to \mathbb{F}_p^m$ represented in falling factorial ANF, $\deg_{x_{\ell_i}} g_i \leq p-2$ and $\mathrm{D}_{\mathbf{e}_{\ell_i}} g_j = \mathrm{D}_{\mathbf{e}_{\ell_j}} g_i$ for all $i \neq j$

Output: $f : \mathbb{F}_p^n \to \mathbb{F}_p^m$ such that $\mathrm{D}_{\mathbf{e}_{\ell_i}} f = g_i$ for all $i = 1, \ldots, k$

```
f ← 0
for i = 1 to k do
    h ← g_i − D_{e_{ℓ_i}} f
    for all ct monomial in h do
        d ← deg_{x_{ℓ_i}}(t)
        f ← f + c(d + 1)^{−1}(x_{ℓ_i} − d)t
    end for
end for
return f
```

Theorem 1 *Algorithms 1 and 2 are correct. If f is the output of the algorithm, then*

$$\begin{aligned}&Antiderivative(n, k, (g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})), (\mathbf{e}_{\ell_1}, \ldots, \mathbf{e}_{\ell_k}))\\ &= \{f + u | u \text{ a polynomial function which does not depend on } x_{\ell_1}, \ldots, x_{\ell_k}\}.\end{aligned}$$

Proof For ease of notation, and without loss of generality, let us assume $\ell_1 = 1, \ldots, \ell_k = k$. Note that for $p = 2$ Algorithm 2 becomes Algorithm 1, so it suffices to prove the correctness of Algorithm 2.

For convenience we will use the following notation $A_i(x_1^{\underline{d_1}} \cdots x_n^{\underline{d_n}}) = (d_i + 1)^{-1} x_1^{\underline{d_1}} \cdots x_i^{\underline{d_i+1}} \cdots x_n^{\underline{d_n}}$ and we extend A_i by linearity to any polynomial expressed in falling factorial ANF. Note that for any polynomial g we have $\mathrm{D}_{\mathbf{e}_i}(A_i(g)) = g$ and for any $j \neq i$ we have $\mathrm{D}_{\mathbf{e}_j}(A_i(g)) = A_i(\mathrm{D}_{\mathbf{e}_j}(g))$. To prove both these facts it suffices to verify for monomials (which is left as an exercise, using Proposition 1(ii)), since both operators A_i and $\mathrm{D}_{\mathbf{e}_j}$ are linear.

Let us denote by $f^{(0)}$ the initial value of f and by $f^{(i)}$ the value of f at the end of the ith run of the outer **for** loop. We prove by induction on i that $\mathrm{D}_{\mathbf{e}_j} f^{(i)} = g_j$ for all $j \leq i$. Looking at the inner **for** loop we see that it computes $f^{(i)} = f^{(i-1)} + A_i(h)$. For the base case, $i = 1$, we have $\mathrm{D}_{\mathbf{e}_1} f^{(1)} = \mathrm{D}_{\mathbf{e}_1}(A_1(h)) = h = g_1$. Now assume the statement holds for $i - 1$ and we prove it for i. We have:

$$\mathrm{D}_{\mathbf{e}_i} f^{(i)} = \mathrm{D}_{\mathbf{e}_i} f^{(i-1)} + \mathrm{D}_{\mathbf{e}_i}(A_i(h)) = \mathrm{D}_{\mathbf{e}_i} f^{(i-1)} + h = g_i.$$

For any $j < i$, using the induction hypothesis and the compatibility conditions $\mathrm{D}_{\mathbf{e}_j} g_i = \mathrm{D}_{\mathbf{e}_i} g_j$ we have:

$$\begin{aligned}\mathrm{D}_{\mathbf{e}_j} f^{(i)} &= \mathrm{D}_{\mathbf{e}_j} f^{(i-1)} + \mathrm{D}_{\mathbf{e}_j}(A_i(h))\\ &= g_j + \mathrm{D}_{\mathbf{e}_j}(A_i(g_i)) - \mathrm{D}_{\mathbf{e}_j}(A_i(\mathrm{D}_{\mathbf{e}_i}(f^{(i-1)})))\\ &= g_j + A_i(\mathrm{D}_{\mathbf{e}_j}(g_i)) - A_i(\mathrm{D}_{\mathbf{e}_i}\mathrm{D}_{\mathbf{e}_j} f^{(i-1)})\\ &= g_j + A_i(\mathrm{D}_{\mathbf{e}_j} g_i) - A_i(\mathrm{D}_{\mathbf{e}_i} g_j)\\ &= g_j.\end{aligned}$$

Let F be another function F such that $\mathrm{D}_{\mathbf{e}_i} F = g_i$ for all $i = 1, \ldots, k$. Therefore $\mathrm{D}_{\mathbf{e}_i}(F - f) = 0$ for all i. This happens if and only if $F - f$ has degree zero in each of the variables $x_1, \ldots, x_k$ (see Proposition 1(vi)), i.e. it does not depend on those variables. □

Remark 1 We treated the case $p = 2$ separately in Algorithm 1 no only because the formulation becomes simpler, but also because, unlike Algorithm 2, the polynomials for Algorithm 1 do not even need to be given in ANF. They could be given as polynomials which are not in normal form, or even as black box functions (such functions are often considered in attacks on symmetric stream ciphers; they model well the situation when the ANF is too large to be written explicitly).

For $k = n$ we can use Algorithm 1 to obtain by induction an explicit formula for f:

$$\begin{aligned}f &= \sum_{i=1}^{n} x_i g_i - \sum_{1 \leq i_1 < i_2 \leq n} x_{i_1} x_{i_2} \mathrm{D}_{\mathbf{e}_{i_1}} g_{i_2} + \sum_{1 \leq i_1 < i_2 < i_3 \leq n} x_{i_1} x_{i_2} x_{i_3} \mathrm{D}^{(2)}_{\mathbf{e}_{i_1}, \mathbf{e}_{i_2}} g_{i_3} + \cdots\\ &= \sum_{s=1}^{n} (-1)^{s-1} \sum_{1 \leq i_1 < i_2 < \ldots < i_s \leq n} x_{i_1} \ldots x_{i_s} \mathrm{D}^{(s-1)}_{\mathbf{e}_{i_1}, \ldots \mathbf{e}_{i_{s-1}}} g_{i_s}. \qquad (5)\end{aligned}$$

Keeping in mind that f has to satisfy $g_i = \mathrm{D}_{\mathbf{e}_i} f$, this is equivalent to the following known result (see [1, Proposition 1]):

$$f = \sum_{i=1}^{n} x_i \mathrm{D}_{\mathbf{e}_i} f - \sum_{1 \le i_1 < i_2 \le n} x_{i_1} x_{i_2} \mathrm{D}^{(2)}_{\mathbf{e}_{i_1}, \mathbf{e}_{i_2}} f + \sum_{1 \le i_1 < i_2 < i_3 \le n} x_{i_1} x_{i_2} x_{i_3} \mathrm{D}^{(3)}_{\mathbf{e}_{i_1}, \mathbf{e}_{i_2} \mathbf{e}_{i_3}} f + \cdots$$
$$= \sum_{s=1}^{n} (-1)^{s-1} \sum_{1 \le i_1 < i_2 < \ldots < i_s \le n} x_{i_1} \ldots x_{i_s} \mathrm{D}^{(s)}_{\mathbf{e}_{i_1} \ldots \mathbf{e}_{i_s}} f.$$

The complexity of Algorithms 1 and 2 depends on how we represent the polynomial functions. If we represent them in falling factorials ANF, with each polynomial represented as the set of its monomials, the algorithm becomes even simpler, see Algorithm 3.

Algorithm 3 AD-set$(n, k, (g_1(\mathbf{x}), \ldots, g_k(g_1(\mathbf{x})), (\mathbf{e}_{\ell_1}, \ldots, \mathbf{e}_{\ell_k}))$

Input: $n, k, g_1, \ldots, g_k, \mathbf{e}_{\ell_1}, \ldots, \mathbf{e}_{\ell_k}$, with $g_i : \mathbb{F}_p^n \to \mathbb{F}_p^m$, $\deg_{x_{\ell_i}} g_i \le p-2$ and $\mathrm{D}_{\mathbf{e}_{\ell_i}} g_j = \mathrm{D}_{\mathbf{e}_{\ell_j}} g_i$ for all $i, j = 1, \ldots, k$. Polynomials are represented as the set of monomials that appear in their falling factorial ANF

Output: $f : \mathbb{F}_p^n \to \mathbb{F}_p^m$ such that $\mathrm{D}_{\mathbf{e}_{\ell_i}} f = g_i$ for all $i = 1, \ldots, k$

1: **for** $i = 1$ **to** k **do**
2: $\quad G_i \leftarrow \{\mathbf{c}(d+1)^{-1}(x_{\ell_i} - d)t \mid \mathbf{c}t$ monomial, $\mathbf{c}t \in g_{\ell_i}, \deg_{x_{\ell_i}}(t) = d, \}$
3: **end for**
4: $f \leftarrow \bigcup_{i=1}^{k} G_i$
5: **return** f

Theorem 2 *Algorithm 3 is correct. Assume each polynomial is represented as the list of monomials* $\mathbf{c}x_1^{\underline{d_1}} \cdots x_n^{\underline{d_n}}$ *appearing in its falling factorial ANF, ordered in lexicographical order of* $(d_1, \ldots, d_n)$ *(or any other admissible monomial order). Then the time complexity of the algorithm is quasilinear in the size of its input. Namely if the total size of the* k *input polynomials* $g_1, \ldots, g_k$ *is* s *bits, the algorithm has complexity* $\mathcal{O}(s \log k)$.

Proof For ease of notation, and without loss of generality, let us assume $\ell_1 = 1, \ldots, \ell_k = k$.

It can be easily checked, using Proposition 1(ii) that $\mathrm{D}_{\mathbf{e}_i} G_i = g_i$. By the construction of G_i, we know that all monomials of G_i do contain x_i.

Consider a term $t = x_1^{\underline{d_1}} \cdots x_n^{\underline{d_n}}$ which appears in at least one of the polynomials $G_1, \ldots, G_k$. We prove that t appears in all the polynomials G_i for which i has the property $d_i \ge 1$, and moreover it appears with the same coefficient. Without loss of generality, assume that t appears in G_1 and that $G_2, \ldots, G_s$ are the other polynomials G_i for which i has the property $d_i \ge 1$.

Let $\mathbf{c}_i$ be the coefficient of t in G_i, for $i = 1, \ldots, s$ (at this moment we only know $\mathbf{c}_1 \ne \mathbf{0}$, the other could be zero).

Let $j \in \{2, \ldots, s\}$. The term $x_1^{\underline{d_1 - 1}} x_2^{\underline{d_2}} \cdots x_j^{\underline{d_j - 1}} \cdots x_n^{\underline{d_n}}$ appears in $\mathrm{D}_{\mathbf{e}_1} \mathrm{D}_{\mathbf{e}_j} G_j$ with coefficient $d_1 d_j \mathbf{c}_j$, whereas in $\mathrm{D}_{\mathbf{e}_1} \mathrm{D}_{\mathbf{e}_j} G_1$ it appears with coefficient $d_1 d_j \mathbf{c}_1$. On the other hand, $\mathrm{D}_{\mathbf{e}_1} \mathrm{D}_{\mathbf{e}_j} G_j = \mathrm{D}_{\mathbf{e}_1} g_j$ and $\mathrm{D}_{\mathbf{e}_1} \mathrm{D}_{\mathbf{e}_j} G_1 = \mathrm{D}_{\mathbf{e}_j} g_1$, so using the compatibility condition $\mathrm{D}_{\mathbf{e}_j} g_1 = \mathrm{D}_{\mathbf{e}_1} g_j$ we have that $\mathbf{c}_j = \mathbf{c}_1$.

We have therefore proved that taking the union of the sets $\bigcup_{i=1}^{k} G_i$ maintains the condition that all monomials in the set have different power products.

For the correctness, we need to prove $\mathrm{D}_{\mathbf{e}_i} f = g_i$ for $i = 1, \ldots, k$; since we represent polynomials as the sets of their monomials, we will prove equality of sets. Fix an i. Let $\mathbf{c}t \in f$

with $t = x_1^{\underline{d_1}} \cdots x_n^{\underline{d_n}}$. If $d_i = 0$ then $\mathrm{D}_{\mathbf{e}_i} t = 0$. If $d_i \geq 1$ then, as shown above, $\mathbf{c}t \in G_i$, so $\mathrm{D}_{\mathbf{e}_i} \mathbf{c}t \in \mathrm{D}_{\mathbf{e}_i} G_i = g_i$. Conversely, let $\mathbf{c}t \in g_i$. We therefore have $\mathbf{c}(d_i+1)^{-1}(x_i - d)t \in G_i \subseteq f$, hence $\mathrm{D}_{\mathbf{e}_i}(\mathbf{c}(d_i+1)^{-1}(x_i-d)t) = \mathbf{c}t \in \mathrm{D}_{\mathbf{e}_i} f$.

For the complexity of the algorithm, note, firstly, that the sorted list of monomials in G_i is obtained in linear time from the sorted list of monomials in g_i, because all exponents are changed by increasing the exponent of x_i by one, which preserves the lexicographical order. Secondly, computing the union of k sets, when each set is a sorted list, can be achieved by an algorithm similar to the merging phase of Mergesort, by $\log k$ rounds of merging pairs of lists (ensuring entries which appear in both lists are introduced only once in the merged list), and each merging round having linear complexity in the sizes of the merged lists. □

Example 2 Let $p = 2, n = k = 3, m = 2$ and

$$\begin{aligned} g_1(x_1, x_2, x_3) &= (1,0)x_2x_3 + (1,0)x_2 + (1,1)x_3 \\ g_2(x_1, x_2, x_3) &= (1,0)x_1x_3 + (1,0)x_1 \\ g_3(x_1, x_2, x_3) &= (1,0)x_1x_2 + (1,1)x_1 + (1,1). \end{aligned}$$

One can verify that they satisfy the compatibility conditions. Applying Algorithm 3 we compute $G_i = x_i g_i$ for $i = 1, 2, 3$:

$$\begin{aligned} x_1 g_1(x_1, x_2, x_3) &= (1,0)x_1x_2x_3 + (1,0)x_1x_2 + (1,1)x_1x_3 \\ x_2 g_2(x_1, x_2, x_3) &= (1,0)x_1x_2x_3 + (1,0)x_1x_2 \\ x_3 g_3(x_1, x_2, x_3) &= (1,0)x_1x_2x_3 + (1,1)x_1x_3 + (1,1)x_3. \end{aligned}$$

Finally we collect all the monomials in $x_1 g_2, x_2 g_2, x_3 g_3$ to obtain f:

$$f(x_1, x_2, x_3) = (1,0)x_1x_2x_3 + (1,0)x_1x_2 + (1,1)x_1x_3 + (1,1)x_3$$

Example 3 Let $p = 7, n = k = 2, m = 1$ and

$$\begin{aligned} g_1(x_1, x_2) &= 3x_1^{\underline{2}}x_2^{\underline{5}} + 6x_1^{\underline{5}} + 2x_2^{\underline{4}} \\ g_2(x_1, x_2) &= 5x_1^{\underline{3}}x_2^{\underline{4}} + x_1x_2^{\underline{3}} + 2x_2^{\underline{2}} + 3x_2. \end{aligned}$$

One can verify that they satisfy the compatibility conditions. Using Algorithm 3 we compute:

$$\begin{aligned} G_1(x_1, x_2) &= x_1^{\underline{3}}x_2^{\underline{5}} + x_1^{\underline{6}} + 2x_1x_2^{\underline{4}} \\ G_2(x_1, x_2) &= x_1^{\underline{3}}x_2^{\underline{5}} + 2x_1x_2^{\underline{4}} + 3x_2^{\underline{3}} + 5x_2^{\underline{2}}. \end{aligned}$$

Finally computing the union of G_1 and G_2 we obtain f:

$$f(x_1, x_2) = x_1^{\underline{3}}x_2^{\underline{5}} + 2x_1x_2^{\underline{4}} + x_1^{\underline{6}} + 3x_2^{\underline{3}} + 5x_2^{\underline{2}}.$$

3.2 Antiderivatives in arbitrary directions

Next let us consider the general case where we are given the discrete derivatives $g_1, \ldots, g_k$ in $k \leq n$ arbitrary linearly independent directions $\mathbf{a}_1, \ldots, \mathbf{a}_k$.

The following Theorem presents two possible approaches to computing the antiderivative.

Theorem 3 *Let $\mathbf{a}_1, \ldots, \mathbf{a}_k \in \mathbb{F}_p^n$ with $k \leq n$ be linearly independent vectors and let $g_1, \ldots, g_k : \mathbb{F}_p^n \to \mathbb{F}_p^m$ be such that $\mathrm{D}^{(p-1)}_{\mathbf{a}_i, \ldots, \mathbf{a}_i} g_i = 0$ and $\mathrm{D}_{\mathbf{a}_i} g_j = \mathrm{D}_{\mathbf{a}_j} g_i$ for all $i, j = 1, \ldots, k$.*

(i) *Let A be an invertible $n \times n$ matrix over $\mathbb{F}_p$ having the first k columns equal to $\mathbf{a}_1, \ldots, \mathbf{a}_k$. Then*

$$Antiderivative(n, k, (g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})), (\mathbf{a}_1, \ldots, \mathbf{a}_k)) = (Antiderivative(n, k, (g_1(A\mathbf{x}), \ldots, g_k(A\mathbf{x})), (\mathbf{e}_1, \ldots, \mathbf{e}_k)))(A^{-1}\mathbf{x}) \quad (6)$$

and the right hand side Antiderivative can be computed using one of Algorithms 1–3 and Theorem 1.

(ii) *If $k = n$, write $\mathbf{e}_1, \ldots, \mathbf{e}_n$ in the basis $\mathbf{a}_1, \ldots, \mathbf{a}_n$ as $\mathbf{e}_i = \sum_{j=1}^{n} b_{ij}\mathbf{a}_j$. Denoting*

$$h_i(\mathbf{x}) = \sum_{j=1}^{n} \sum_{v=0}^{b_{ij}-1} \left(g_j \left(\mathbf{x} + v\mathbf{a}_j + \sum_{s=j+1}^{n} b_{is}\mathbf{a}_s \right) \right) \quad (7)$$

we have

$$Antiderivative(n, n, (g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})), (\mathbf{a}_1, \ldots, \mathbf{a}_k)) = Antiderivative(n, n, (h_1(\mathbf{x}), \ldots, h_n(\mathbf{x})), (\mathbf{e}_1, \ldots, \mathbf{e}_n)) \quad (8)$$

and the right hand side Antiderivative can be computed using one of Algorithms 1-3 and Theorem 1.

Proof (i) This first approach involves a change of coordinates. Obviously $A\mathbf{e}_i = \mathbf{a}_i$ and $A^{-1}\mathbf{a}_i = \mathbf{e}_i$ for all i. Let f be an element of the set on the right hand side of Eq. (6), i.e. $f(\mathbf{x}) = g(A^{-1}\mathbf{x})$ for some $g \in Antiderivative(n, k, (g_1(A\mathbf{x}), \ldots, g_k(A\mathbf{x})), (\mathbf{e}_1, \ldots, \mathbf{e}_k))$. Proposition 1(iv) gives for all i

$$(\mathrm{D}_{\mathbf{a}_i} f)(\mathbf{x}) = (\mathrm{D}_{A^{-1}\mathbf{a}_i} g)(A^{-1}\mathbf{x}) = (\mathrm{D}_{\mathbf{e}_i} g)(A^{-1}\mathbf{x}) = g_i(AA^{-1}\mathbf{x}) = g_i(\mathbf{x})$$

hence f does belong to the set on the left side of the Eq. (6).

Conversely, let f be an element of the set on the left hand side of Eq. (6). Let $g(\mathbf{x}) = f(A\mathbf{x})$. Again, Proposition 1(iv) gives for all i

$$(\mathrm{D}_{\mathbf{e}_i} g)(\mathbf{x}) = (\mathrm{D}_{A\mathbf{e}_i} f)(A\mathbf{x}) = (\mathrm{D}_{\mathbf{a}_i} f)(A\mathbf{x}) = g_i(A\mathbf{x})$$

hence $g \in (Antiderivative(n, k, (g_1(A\mathbf{x}), \ldots, g_k(A\mathbf{x})), (\mathbf{e}_1, \ldots, \mathbf{e}_k)))$, so f does belong to the set on the right side of the Eq. (6).

(ii) This second approach consists of computing the derivatives in the canonical directions from the derivatives in arbitrary directions. Let f be an element of the set on the left hand side of Eq. (8). Therefore, $\mathrm{D}_{\mathbf{a}_i} f(\mathbf{x}) = g_i(\mathbf{x})$ for all i. Using Proposition 2(1) and the fact that $\mathbf{e}_i = \sum_{j=1}^{n} b_{ij}\mathbf{a}_j$: we have that $\mathrm{D}_{\mathbf{e}_i} f = h_i$ with h_i as computed in Eq. (7). Hence f is indeed in the set on the right side of Eq. (8). Any other element f_1 in the set on the right hand side of Eq. (8) differs from f by a constant $\mathbf{c} \in \mathbb{F}_p^m$, see Algorithm 2 and Theorem 1. Therefore $\mathrm{D}_{\mathbf{a}_i} f_1 = \mathrm{D}_{\mathbf{a}_i}(f + \mathbf{c}) = \mathrm{D}_{\mathbf{a}_i} f = g_1$, so f_1 belongs to the set on the left hand side of Eq. (8). □

Note that in both approaches in the Theorem above, the complexity of computing the intermediate polynomials $g_1(A\mathbf{x}), \ldots, g_k(A\mathbf{x})$ or $h_1, \ldots, h_n$ may be exponential compared to the size of the input polynomials $g_1, \ldots, g_k$. (For the first approach, note that a linear change of coordinates applied to a monomial of degree d can result in up to n^d monomials; a similar problem appears when we compute h_i in the second approach.) If the size of the output f is exponential compared to the size of the input, any algorithm for computing f in

ANF would take exponential time. When the size of f is polynomial in the size of the input, it may still be possible that the computation of intermediate polynomials is exponential and therefore the whole algorithm is exponential.

In [4, Definition 8], an equivalence relation on the class of functions is defined, whereby for a fixed vector space V, two functions f_1, f_2 are called *differentially equivalent* with respect to V if $\mathrm{D}_{\mathbf{v}} f_1 = \mathrm{D}_{\mathbf{v}} f_2$ for all $\mathbf{v} \in V$. Obviously, is suffices to check whether $\mathrm{D}_{\mathbf{a}_i} f_1 = \mathrm{D}_{\mathbf{a}_i} f_2$ for some basis $\mathbf{a}_1, \ldots, \mathbf{a}_k$ of V. (see Proposition 2(1)). A characterisation of differential equivalence similar to [4, Proposition 2 and Eq. (6)] follows:

Corollary 1 *Let $\mathbf{a}_1, \ldots, \mathbf{a}_k \in \mathbb{F}_p^n$ with $k \leq n$ be linearly independent vectors. Two functions $f_1, f_2 : \mathbb{F}_p^n \to \mathbb{F}_p^m$ have the same derivatives in the directions $\mathbf{a}_1, \ldots, \mathbf{a}_k$, i.e. $\mathrm{D}_{\mathbf{a}_i} f_1 = \mathrm{D}_{\mathbf{a}_i} f_2$ for $i = 1, \ldots, k$, if and only if $f_1(\mathbf{x}) - f_2(\mathbf{x}) = h(A^{-1}\mathbf{x})$ for some function $h : \mathbb{F}_p^n \to \mathbb{F}_p^m$ which does not depend on the variables $x_1, \ldots, x_k$ and some invertible matrix A having the first k columns equal to $\mathbf{a}_1, \ldots, \mathbf{a}_k$.*

Proof Since $\mathrm{D}_{\mathbf{a}_i}(f_1 - f_2) = 0$ for $i = 1, \ldots, k$, by Theorem 3(i) we have

$$\begin{aligned} f_1 - f_2 &\in \mathit{Antiderivative}(n, k, (0, \ldots, 0), (\mathbf{a}_1, \ldots, \mathbf{a}_k)) \\ &= (\mathit{Antiderivative}(n, k, (0, \ldots, 0), (\mathbf{e}_1, \ldots, \mathbf{e}_k)))(A^{-1}\mathbf{x}). \end{aligned}$$

We then use Theorem 1. □

4 Antiderivatives of functions over $\mathbb{F}_{p^n}$

4.1 Converting between representations

We will first recall how to convert a function between different representations. For the conversion from univariate to multivariate representation we will work along the lines of [7]. Let $g : \mathbb{F}_{p^n} \to \mathbb{F}_{p^n}$ be represented as a univariate polynomial over $\mathbb{F}_{p^n}$ of degree at most $p^n - 1$. Fix a basis $\alpha = \{\alpha_1, \ldots, \alpha_n\}$ for $\mathbb{F}_{p^n}$, viewed as a vector space of dimension n over $\mathbb{F}_p$. The isomorphism $\varphi_\alpha : \mathbb{F}_{p^n} \to \mathbb{F}_p^n$ associates to each element in $\mathbb{F}_{p^n}$ its coordinates in the basis α. We then consider n new variables $x_1, \ldots, x_n$ representing the coordinates of x in the basis α; note that these new variables are thought of as taking values in $\mathbb{F}_p$, whereas x was taking values in $\mathbb{F}_{p^n}$. Substituting x by $\alpha_1 x_1 + \ldots + \alpha_n x_n$ and keeping in mind that $x_i^p = x_i$ we convert $g(x)$ into a multivariate polynomial in $x_1, \ldots, x_n$, with coefficients in $\mathbb{F}_{p^n}$. By further applying φ_α to each coefficient we obtain the multivariate polynomial representation described in Sect. 2; we will denote it by $\varphi_\alpha(g)$. We can then represent this polynomial in ANF or falling factorial ANF.

To convert the representations in the other direction, let $f : \mathbb{F}_p^n \to \mathbb{F}_p^n$ viewed as a polynomial in n variables with coefficients in $\mathbb{F}_p^n$. First we convert to coefficients to $\mathbb{F}_{p^n}$ by applying φ_α^{-1}. We then consider a new variable $x = \alpha_1 x_1 + \ldots + \alpha_n x_n$. Keeping in mind that $x_i \in \mathbb{F}_p$ means $x_i^p = x_i$, we have the equations $x^{p^i} = \alpha_1^{p^i} x_1 + \ldots + \alpha_n^{p^i} x_n$ for $i = 0, \ldots, n-1$. Defining M_α as the $n \times n$ matrix having $\alpha_j^{p^{i-1}}$ in position i, j, this becomes

$$\begin{pmatrix} x \\ x^p \\ \vdots \\ x^{p^{n-1}} \end{pmatrix} = M_\alpha \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

We therefore have

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = M_\alpha^{-1} \begin{pmatrix} x \\ x^p \\ \vdots \\ x^{p^{n-1}} \end{pmatrix}$$

which gives n equations expressing each x_i as a linear combination of $x, x^p, \ldots, x^{p^{n-1}}$ with coefficients in $\mathbb{F}_{p^n}$. Substituting all x_i in f using these equations we obtain a polynomial g in x. One can verify that $\varphi_\alpha(g) = f$; we denote therefore $g = \varphi_\alpha^{-1}(f)$. For computing M_α^{-1} it is convenient to have a basis β which is dual to α; we then have $M_\beta^T = M_\alpha^{-1}$ (see [7, Section 2.1]).

Once we fix a basis of $\mathbb{F}_{p^n}$, differentiation does not depend on the representation, as differentiation only uses the additive group, and $(\mathbb{F}_{p^n}, +)$ and $(\mathbb{F}_p^n, +)$ are isomorphic. More precisely, we have:

Lemma 1 *Let $\alpha = \{\alpha_1, \ldots, \alpha_n\}$ be a basis of $\mathbb{F}_{p^n}$ as a vector space over $\mathbb{F}_p$. Let $f : \mathbb{F}_{p^n} \to \mathbb{F}_{p^n}$ be a univariate polynomial function and $\mathbf{a} \in \mathbb{F}_{p^n} \setminus \{\mathbf{0}\}$. Then*

$$\varphi_\alpha(\mathrm{D}_{\mathbf{a}} f) = \mathrm{D}_{\varphi_\alpha(\mathbf{a})} \varphi_\alpha(f).$$

4.2 Algorithm for antiderivatives of functions over $\mathbb{F}_{p^n}$

We can now give an alternative to the antiderivative algorithms of Suder [4] and of Pasalic et al. [2] for functions over $\mathbb{F}_{p^n}$. Using the changes of representation discussed in Sect. 4.1 and Lemma 1 we have:

Theorem 4 *Let $a_1, \ldots, a_k \in \mathbb{F}_{p^n}$ be linearly independent over $\mathbb{F}_p$ and let $g_1, \ldots, g_k : \mathbb{F}_{p^n} \to \mathbb{F}_{p^n}$. There exists a function $f : \mathbb{F}_{p^n} \to \mathbb{F}_{p^n}$ such that $\mathrm{D}_{\mathbf{a}_i} f = g_i$ for all $i = 1, \ldots, k$ iff the following conditions are satisfied:*

(i) $\mathrm{D}_{a_i, \ldots, a_i}^{(p-1)} g_i = 0$ *for all* $i = 1, \ldots, k$.
(ii) $\mathrm{D}_{a_i} g_j = \mathrm{D}_{a_j} g_i$ *for all* $i, j = 1, \ldots, k$.

If these conditions are satisfied then the set of such functions f can be computed as

$$\varphi_\alpha^{-1}(Antiderivative(n, k, (\varphi_\alpha(g_1(x)), \ldots, \varphi_\alpha(g_k(x))), (\mathbf{e}_1, \ldots, \mathbf{e}_k)))$$

where the basis $\alpha = \{\alpha_1, \ldots, \alpha_n\}$ is chosen so that $\alpha_1 = a_1, \ldots, \alpha_k = a_k$ and the function Antiderivative is computed using one of Algorithms 1–3.

Note that although Algorithm 3 is quasilinear, the transformations φ_α and φ_α^{-1} have exponential worst case complexity, so the total complexity of computing the antiderivative for functions given in their univariate representation using Theorem 4 is exponential.

In Theorem 4 we could choose an arbitrary basis α, in which case the set would be:

$$\varphi_\alpha^{-1}(Antiderivative(n, k, (\varphi_\alpha(g_1(x)), \ldots, \varphi_\alpha(g_k(x))), (\varphi_\alpha(a_1), \ldots, \varphi_\alpha(a_k)))).$$

and we would need to apply Theorem 3 in addition to Algorithms 1-3 to compute *Antiderivative*.

5 Applications to PN and APN functions

We recall the definitions of PN and APN functions:

Definition 3 A function $f : \mathbb{F}_p^n \to \mathbb{F}_p^n$ is called *PN (Perfect Nonlinear) or planar* if for all $\mathbf{a} \in \mathbb{F}_p^n \setminus \{\mathbf{0}\}$ its derivative $\mathrm{D}_\mathbf{a} f$ in direction $\mathbf{a}$ is bijective.

Note that PN functions only exist when $p > 2$. For $p = 2$ only a weaker condition can be achieved:

Definition 4 A function $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is called APN (Almost Perfect Nonlinear) if for all $\mathbf{a} \in \mathbb{F}_2^n \setminus \{\mathbf{0}\}$ its derivative $\mathrm{D}_\mathbf{a} f$ in direction $\mathbf{a}$ is 2-to-1, i.e. for any $\mathbf{b} \in \mathbb{F}_2^n$ the equation $\mathrm{D}_\mathbf{a} f(\mathbf{x}) = \mathbf{b}$ has either two or zero solutions.

Both the PN and APN properties are invariant to extended affine equivalence (EA-equivalence); two functions $f, g : \mathbb{F}_p^n \to \mathbb{F}_p^n$ are called EA-equivalent if $f = L_1 \circ g \circ L_2 + H$ for some affine functions (i.e. of algebraic degree at most 1) $L_1, L_2, H : \mathbb{F}_p^n \to \mathbb{F}_p^n$ with L_1, L_2 being invertible.

Suder [4] pointed out that antiderivatives can be used in the construction of quadratic APN functions (reinterpreting [5,7]) and also suggested they can be used for other degrees and other properties. We expand this direction.

Assume that we want to construct a function $f : \mathbb{F}_p^n \to \mathbb{F}_p^n$, in ANF representation, such that all its derivatives satisfy some property P (being bijective in the case of PN functions or being 2-to-1 in the case of APN functions; other properties can also be considered). It suffices to construct n functions $g_1, \ldots, g_n$ such that $\deg_{x_i}(g_i) \leq p - 2$ and $\mathrm{D}_{\mathbf{e}_i} g_j = \mathrm{D}_{\mathbf{e}_j} g_i$ for all $i, j = 1, \ldots, n$ and, moreover each g_i has property P and also for all $(\mathbf{a}_1, \ldots, \mathbf{a}_n) \in \mathbb{F}_p^n \setminus \{\mathbf{0}\}$ the function $\sum_{i=1}^n \sum_{j=0}^{a_i - 1} ((\mathrm{D}_{\mathbf{e}_i} f)(\mathbf{x} + j\mathbf{e}_i + \sum_{k=i+1}^n a_k \mathbf{e}_k))$ has property P. Once such functions $g_1, \ldots, g_n$ have been found, f can be constructed efficiently by computing $Antiderivative(n, n, (g_1(\mathbf{x}), \ldots, g_n(\mathbf{x})), (\mathbf{e}_1, \ldots, \mathbf{e}_n))$ using one of the Algorithms 1-3. Using Proposition 2(2) we see that indeed all the derivatives of f have property P.

One important case is the one of quadratic functions. The construction above becomes so much simpler that we do not even need to use antiderivatives explicitly. We give the multivariate ANF equivalent of the characterisation of quadratic APN functions [7, Theorem 1] (in fact Yu et al. [7] use implicitly the transformation from univariate polynomials to multivariate ANF):

Theorem 5 *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a quadratic function in ANF, $f = \sum_{1 \leq i < j \leq n} \mathbf{b}_{ij} x_i x_j$. We have that f is APN if and only if the $n \times n$ matrix $C = (\mathbf{c}_{ij})_{i,j=1,\ldots,n}$ with entries in $\mathbb{F}_2^n$ defined as $\mathbf{c}_{ji} = \mathbf{c}_{ij} = \mathbf{b}_{ij}$ for $i < j$ and $\mathbf{c}_{ii} = \mathbf{0}$ has the properties:*

(i) *Each row of C, consists of n elements of $\mathbb{F}_2^n$ that span a space of dimension $n - 1$.*
(ii) *Each linear combination (with coefficients in $\mathbb{F}_2$) of the rows in C consists of n vectors in $\mathbb{F}_2^n$ that span a space of dimension $n - 1$ (condition (i) is subsumed by (ii) but for convenience we wrote them separately).*

Remark 2 Note that the Theorem above could also be formulated for differential uniformity. Recall that a function f over $\mathbb{F}_2^n$ has differential δ-uniformity if for any $\mathbf{a}, \mathbf{b} \in \mathbb{F}_2^n$, $\mathbf{a} \neq \mathbf{0}$ the equation $\mathrm{D}_\mathbf{a} f(\mathbf{x}) = \mathbf{b}$ has at most δ solutions. The Theorem above still holds if we replace APN by 2^k-uniformity and "dimension $n - 1$" by "dimension at least $n - k$".

We present a similar characterisation for quadratic PN (planar) functions:

Theorem 6 *Let $f : \mathbb{F}_p^n \to \mathbb{F}_p^n$, with $p > 2$, be a quadratic function in ANF, $f = \sum_{i=1}^n \mathbf{b}_{ii} x_i^2 + \sum_{1 \le i < j \le n} \mathbf{b}_{ij} x_i x_j$. We have that f is a PN function if and only if the $n \times n$ matrix $C = (\mathbf{c}_{ij})_{i,j=1,\ldots,n}$ with entries in $\mathbb{F}_p^n$ defined as $\mathbf{c}_{ji} = \mathbf{c}_{ij} = \mathbf{b}_{ij}$ for $i < j$ and $\mathbf{c}_{ii} = 2\mathbf{b}_{ii}$ has the properties:*

(i) *Each row of C consists of n elements of $\mathbb{F}_p^n$ which are linearly independent over $\mathbb{F}_p$.*
(ii) *Each linear combination (with coefficients in $\mathbb{F}_p$) of the rows in C consists of n vectors in $\mathbb{F}_p^n$ which are linearly independent over $\mathbb{F}_p$ (condition (i) is subsumed by (ii) but for convenience we wrote them separately).*

Proof (for both Theorems 6 and 5) Here we can use either the APN form or the falling factorial APN form, as x_i^2 and ${x_i}^{\underline{2}}$ differ by a polynomial of algebraic degree one, so they are EA-equivalent. For any $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{F}_p^n \setminus \{\mathbf{0}\}$ using Proposition 2(2) and the fact that $\mathrm{D}_{\mathbf{e}_i} f$ is affine (since f is quadratic) we have, for some constant $\mathbf{c} \in \mathbb{F}_2^n$:

$$\begin{aligned}
\mathrm{D}_{\mathbf{a}} f(\mathbf{x}) &= \sum_{i=1}^{n} \sum_{j=0}^{a_i - 1} \left((\mathrm{D}_{\mathbf{e}_i} f) \left(\mathbf{x} + j\mathbf{e}_i + \sum_{k=i+1}^{n} a_k \mathbf{e}_k \right) \right) \\
&= \sum_{i=1}^{n} (a_i \mathrm{D}_{\mathbf{e}_i} f)(\mathbf{x}) + \mathbf{c} \\
&= \sum_{i=1}^{n} \left(a_i \sum_{j=1}^{n} \mathbf{c}_{ij} x_j \right) + \mathbf{c} \\
&= \sum_{j=1}^{n} \left(\sum_{i=1}^{n} a_i \mathbf{c}_{ij} \right) x_j + \mathbf{c}.
\end{aligned}$$

Finally, we use the fact that a linear function $g : \mathbb{F}_p^n \to \mathbb{F}_p^n$, $g = \mathbf{c}_1 x_1 + \cdots + \mathbf{c}_n x_n$ is bijective iff $\mathbf{c}_1, \ldots, \mathbf{c}_n$ are linearly independent; for $p = 2$, g is 2-to-1 if and only if $\mathbf{c}_1, \ldots, \mathbf{c}_n$ span a space of dimension $n - 1$. □

The theorem above can be used for constructing PN functions from scratch, as shown in Example 4 below. It can also be used similarly to the construction in Yu et al. [7], namely we can start with a known quadratic PN function (obtained by some other technique), construct the associated matrix C and then attempt to modify only a few of the entries of C while preserving the property required by Theorem 6. This approach will be the subject of further work.

Example 4 Let $n = 3$. Due to invariance under EA-equivalence, we can assume that the first row (and column) of C consists of the canonical basis vectors:

$$C = \begin{pmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ \mathbf{e}_2 & \mathbf{u} & \mathbf{v} \\ \mathbf{e}_3 & \mathbf{v} & \mathbf{w} \end{pmatrix}$$

For small p we can find all values of $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$ which satisfy Theorem 6 using an exhaustive computer search. For $p = 3$ there are 288 solutions; one example solution is $\mathbf{u} = \mathbf{e}_3$, $\mathbf{v} = \mathbf{e}_1 + \mathbf{e}_2$, $\mathbf{w} = \mathbf{e}_2 + \mathbf{e}_3$. The corresponding quadratic PN function can be constructed explicitly as

$$\begin{aligned}
f &= 2\mathbf{e}_1 x_1^2 + \mathbf{e}_2 x_1 x_2 + \mathbf{e}_3 x_1 x_3 + 2\mathbf{e}_3 x_2^2 + (\mathbf{e}_1 + \mathbf{e}_2) x_2 x_3 + 2(\mathbf{e}_2 + \mathbf{e}_3) x_3^2 \\
&= (2x_1^2 + x_2 x_3, 2x_3^2 + x_1 x_2 + x_2 x_3, 2x_2^2 + 2x_3^2 + x_1 x_3).
\end{aligned}$$

For $p = 5$ there are 8000 solutions and for $p = 7$ there are 65856 solutions. One solution, for both $p = 5$ and $p = 7$, is $\mathbf{u} = \mathbf{e}_3, \mathbf{v} = \mathbf{e}_1 + \mathbf{e}_3, \mathbf{w} = \mathbf{e}_1 + \mathbf{e}_2 + \mathbf{e}_3$ which gives the PN function

$$f(x_1, x_2, x_3) = (2^{-1}x_1^2 + 2^{-1}x_3^2 + x_2x_3, 2^{-1}x_3^2 + x_1x_2, 2^{-1}x_2^2 + 2^{-1}x_3^2 + x_1x_3 + x_2x_3).$$

For $p = 11$ an example solution is $\mathbf{u} = \mathbf{e}_3, \mathbf{v} = \mathbf{e}_1 + 4\mathbf{e}_3, \mathbf{w} = 5\mathbf{e}_1 + 3\mathbf{e}_2$.

We also look at examples that use Theorem 5 for constructing APN functions.

Example 5 Let $p = 2$. For constructing APN functions, due to invariance to EA-equivalence we can assume that the first row (and column) of C consists of vectors in the canonical basis. For $n = 3$ we can assume therefore that

$$C = \begin{pmatrix} 0 & \mathbf{e}_2 & \mathbf{e}_3 \\ \mathbf{e}_2 & 0 & \mathbf{u} \\ \mathbf{e}_3 & \mathbf{u} & 0 \end{pmatrix}$$

It is easy to check that for $\mathbf{u} = \mathbf{e}_1$ this matrix does satisfy the conditions of Theorem 5, so it corresponds to an APN function, which can be explicitly constructed as $f = \mathbf{e}_2x_1x_2 + \mathbf{e}_3x_1x_3 + \mathbf{e}_1x_2x_3 = (x_2x_3, x_1x_2, x_1x_3)$. If $\mathbf{u}$ is in the vector space generated by $\mathbf{e}_2, \mathbf{e}_3$ then one can easily see that C does not satisfy Theorem 5. Therefore, up to EA-equivalence, the function above is the only APN function in 3 variables.

Note that in Suder [4, Section5], the equivalent of Theorem 5 above reads: "Furthermore, if the linear combinations of these affine derivatives are also 2-to-1, i.e. the columns...span a subspace of dimensions $n - 1$ over $\mathbb{F}_{2^n}$, then we have a quadratic APN function." While the first part of the statement is correct, the second part can be misleading, as it seems to imply that it is sufficient that each column of C (which, due to symmetry is the same as a each row of C) spans a space of dimension $n - 1$, in other words, only condition (i) from Theorem 5 need to be satisfied, and not (ii). Condition (i) is necessary, but not sufficient, as the following counterexample demonstrates:

Example 6 For $p = 2$ and $n = 4$ consider the matrix:

$$C = \begin{pmatrix} 0 & \mathbf{e}_2 & \mathbf{e}_3 & \mathbf{e}_4 \\ \mathbf{e}_2 & 0 & \mathbf{u} & \mathbf{v} \\ \mathbf{e}_3 & \mathbf{u} & 0 & \mathbf{w} \\ \mathbf{e}_4 & \mathbf{v} & \mathbf{w} & 0 \end{pmatrix}$$

If $\mathbf{u} = \mathbf{e}_4, \mathbf{v} = \mathbf{e}_1$ and $\mathbf{w} = \mathbf{e}_2$ then the elements of each row span a space of dimension 3 so condition (i) of Theorem 5 is satisfied. One can also verify that for any sum of two or three rows, the resulting elements also span a space of dimension 3. However, the sum of all four rows consists of the vectors $\mathbf{e}_2 + \mathbf{e}_3 + \mathbf{e}_4, \mathbf{e}_1 + \mathbf{e}_2 + \mathbf{e}_4, \mathbf{e}_2 + \mathbf{e}_3 + \mathbf{e}_4, \mathbf{e}_1 + \mathbf{e}_2 + \mathbf{e}_4$ which span a space of dimension only 2. So the function $f = (x_2x_4, x_1x_2 + x_3x_4, x_1x_3, x_1x_4 + x_2x_3)$ corresponding to this matrix does not satisfy condition (ii) of Theorem 5, therefore it is *not* APN. Indeed the derivative of f in the direction $(1, 1, 1, 1)$ is not 2-to-1.

One choice which does give an APN function is $\mathbf{u} = \mathbf{e}_4, \mathbf{v} = \mathbf{e}_3 + \mathbf{e}_4$ and $\mathbf{w} = \mathbf{e}_1$, corresponding to $f = (x_3x_4, x_1x_2, x_1x_3 + x_2x_4, x_1x_4 + x_2x_3 + x_2x_4)$.

6 Conclusion

For computing the antiderivative (i.e. retrieving a function from its derivatives) for functions with n bits input and n bits output (or more generally integers modulo a prime p instead of bits), it turns out that representing the function as n multivariate polynomial functions in algebraic normal form over $\mathbb{F}_p$ is particularly convenient. For this representation we developed a simple algorithm, which is in general more efficient (quasilinear if the directions of the derivatives are in the canonical basis) compared to previous algorithms which represent the functions as univariate polynomial functions over $\mathbb{F}_{p^n}$ and involve solving a system of linear equations of size $p^n \times p^n$.

The connection pointed out by Suder [4] between antiderivatives and the methods of of Yu et al. [7] and of Weng et al. [5] for constructing new quadratic APN (Almost Perfectly Nonlinear) functions is preserved and simplified by our approach to antiderivatives. Moreover, we develop a similar technique which can be used to construct new quadratic PN (Perfect Nonlinear) functions.

Acknowledgements We would like to thank Ferruh Özbudak for useful discussions. We also thank the anonymous referees for several suggestions that led to improvements to the paper.

References

1. Lai X.: Higher order derivatives and differential cryptanalysis. In: Blahut, R.E., Costello, D.J. Jr., Maurer, U., Mittelholzer, T. (eds) Communications and Cryptography. Springer International Series in Engineering and Computer Science, vol. 276, pp. 227–233. Springer, New York (1994).
2. Pasalic E., Muratović-Ribić A., Hodzić S., Gangopadhyay S.: On derivatives of polynomials over finite fields through integration. Discret. Appl. Math. **217**(2), 294–303 (2017).
3. Salas S.L., Etgen G.J., Hille E.: Calculus: One and Several Variables. Wiley, New York (2007).
4. Suder V.: Antiderivative functions over $\mathbb{F}_{2^n}$. Des. Codes Cryptogr. **82**(1), 435–447 (2017).
5. Weng G., Tan Y., Gong G.: On quadratic almost perfect nonlinear functions and their related algebraic object. In: Workshop on Coding and Cryptography, pp. 57–68 (2013).
6. Xiong H., Qu L., Li C., Li Y.: Some results on the differential functions over finite fields. Appl. Algebra Eng. Commun. Comput. **25**(3), 189–195 (2014).
7. Yu Y., Wang M., Li Y.: A matrix approach for constructing quadratic APN functions. Des. Codes Cryptogr. **73**(2), 587–600 (2014).