Des. Codes Cryptogr.
DOI 10.1007/s10623-016-0277-5

Higher order differentiation over finite fields with applications to generalising the cube attack

Ana Sălăgean[1] · R. Winter[1] · Matei Mandache-Sălăgean[2] · Raphael C.-W. Phan[3]

Received: 1 December 2015 / Revised: 17 June 2016 / Accepted: 30 August 2016

Abstract Higher order differentiation was introduced in a cryptographic context by Lai. Several attacks can be viewed in the context of higher order differentiations, amongst them the cube attack of Dinur and Shamir and the AIDA attack of Vielhaber. All of the above have been developed for the binary case. We examine differentiation in larger fields, starting with the field GF(p) of integers modulo a prime p, and apply these techniques to generalising the cube attack to GF(p). The crucial difference is that now the degree in each variable can be higher than one, and our proposed attack will differentiate several times with respect to each variable (unlike the classical cube attack and its larger field version described by Dinur and Shamir, both of which differentiate at most once with respect to each variable). Connections to the Moebius/Reed Muller Transform over GF(p) are also examined. Finally we describe differentiation over finite fields GF(p^s) with p^s elements and show that it can be reduced to differentiation over GF(p), so a cube attack over GF(p^s) would be equivalent to cube attacks over GF(p).

Keywords Higher order differentiation · Cube attack · Higher order derivative

Communicated by J. D. Key.

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

R. Winter
R.Winter@lboro.ac.uk

Matei Mandache-Sălăgean
mfm41@cam.ac.uk

Raphael C.-W. Phan
raphael@mmu.edu.my

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

2 Trinity College, University of Cambridge, Cambridge, UK

3 Faculty of Engineering, Multimedia University, Cyberjaya, Malaysia

Published online: 21 September 2016

Mathematics Subject Classification 94A60 · 11T55

1 Introduction

The main motivation for this work was to generalise the cube attack of Dinur and Shamir [5], and the AIDA attack of Vielhaber [21], from the binary field to arbitrary finite fields. However, some of the tools and results for differentiation over finite fields could have a broader applicability in cryptography.

Higher order derivatives were introduced in a cryptographic context by Lai [16]. This notion had already been used extensively, under the name of finite difference, or discrete derivative, or difference operator, in other areas of mathematics (e.g. for the numerical approximation of the derivative or for solving difference equations).

The (discrete) derivative of a function f is defined as the function $(\Delta_{\mathbf{a}} f)(\mathbf{x}) = f(\mathbf{x} + \mathbf{a}) - f(\mathbf{x})$, for a fixed difference $\mathbf{a}$ (the domain and codomain of f are commutative groups in additive notation). Usually f is a function of n variables over a ring, so $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{a} = (a_1, \ldots, a_n)$. The operator $\Delta_{\mathbf{a}}$ which associates to each function f is derivative will be called differentiation. An important particular case is differentiation with respect to one variable, namely $\mathbf{a} = h\mathbf{e}_i$ where the difference step h is a scalar constant (equal to 1 by default) and $\mathbf{e}_i$ are the canonical basis vectors having a 1 in position i and zeroes elsewhere. Higher order differentiation means repeated application of the differentiation operator.

The functions we use here are functions in several variables over a finite field. Any such function can be represented as a polynomial function. The differentiation operator decreases the degree by at least one, so after sufficiently many applications of the operator the result will be the identically zero function. However for certain choices of differences $\mathbf{a}$, this can happen prematurely, for example over the binary field GF(2) differentiating twice using the same difference $\mathbf{a}$ will always result in the zero function, regardless of the original function f. For our applications, we need to ensure we choose the differences so that this does not happen prematurely, i.e. there is at least a function f which does not become identically zero after computing the higher order derivative w.r.t. those differences.

A number of cryptographic attacks can be reformulated using higher order differentiation. Differential cryptanalysis (introduced by Biham and Shamir [3]) has been thus reformulated by Lai [16]; the cube attack of Dinur and Shamir [5] and the related AIDA attack of Vielhaber [21] have been reformulated in Knellwolf and Meier [14], Duan and Lai [7].

In both the cube attack and the AIDA attack we have a "black box" function f in several public and secret variables and we select a set of indices of public variables $I = \{i_1, \ldots, i_k\}$. Then f is evaluated at each point of a "cube" consisting of the vectors that have all the possible combinations of 0/1 values for the variables with index in I, whereas the remaining variables are left indeterminate; the resulting values are summed and the sum will be denoted f_I. The attacks hope that for suitable choices of subsets I of public variables, the resulting f_I is linear in the secret variables, for the cube attack (or equals to one secret variable or the sum of two secret variables for the AIDA attack). This situation is particularly likely when the cardinality k of I is just marginally lower than the total degree of the function, seen that the degree decreases by k in general, and occasionally by slightly more than k. Such subsets I are found in a preprocessing phase, where the values of the secret variables can be chosen freely. In the online phase the secret variables are unknown, and by computing the f_I for the

sets I identified in the preprocessing phase, one obtains a system of linear equations in the secret variables.

It was shown (see Knellwolf and Meier [14], Duan and Lai [7]) that choosing the variable indices $I = \{i_1, \ldots, i_k\}$ and computing f_I (as described above) is equivalent to computing the k-order derivative of f with respect to the canonical basis vectors $\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_k}$, i.e. by differentiating once with respect to x_{i_1}, then w.r.t. x_{i_2} and so on, finally differentiating w.r.t. x_{i_k}.

It is known that the binary cube/AIDA attack can also be viewed in terms of the Moebius/Reed–Muller transform (which associates to a polynomial function f a new function f^M where $f^M(y_1, \ldots, y_n)$ equals the coefficient of $x_1^{y_1} \cdots x_n^{y_n}$ in f). This transform has been used in several statistical attacks before the cube attack (see for example Filiol [10] and O'Neal [20]). It has also been used for improving the efficiency of the AIDA attack (see Vielhaber [22]) and the Cube attack (see Fouque and Vannet [11]).

All the attacks above, as well as the higher order differentiation used in cryptography are over the binary field. While all cryptographic functions can be viewed as binary functions, there are a number of ciphers which make significant use of operations modulo a prime $p > 2$ in their internal processing, for example ZUC [9], IDEA [17,18], MMB [4]. It may therefore be advantageous for such ciphers to also be viewed and analysed as functions over GF(p), the field of integers modulo p. Unlike the binary case, a polynomial function can now have degree more than one in each variable, in fact it can have degree up to $p-1$ in each variable. There are yet other ciphers which use operations over Galois fields of the form GF(p^s), for example SNOW [8] and in such fields the degree of the polynomial functions can be up to $p^s - 1$ in each variable.

A first generalisation of the cube attack to GF(p^s) was sketched by Dinur and Shamir in [5, p. 284] and also developed more explicitly by Agnese and Pedicini [1]. We show that, like in the binary case, their approach can be viewed as k-order differentiation, where we differentiate once with respect to each of the variables $x_{i_1}, \ldots, x_{i_k}$. However we argue that their generalisation, while correct, has very low chances to lead to a successful attack because we cannot differentiate sufficiently many times. Namely, on one hand, like in the binary case, the best chances of success are when the function is differentiated a number of times just marginally lower than its total degree; on the other hand in their proposed scheme the number of times that the function is differentiated is upper bounded by the number of variables, which (unlike the binary case) can be significantly lower then the degree of the function (see Remark 3).

Our proposed generalisation of the cube attack to GF(p) improves the chances of success by differentiating several times with respect to each of the chosen variables. Thus there is no intrinsic limit on the number of differentiations and therefore this number can be as close as we want to the degree of the polynomial in the public variables (only limited by the computing power available).

We first examine higher order differentiation in GF(p) (Sect. 4.1). For repeated differentiation with respect to the same variable, we can use any non-zero difference steps and there will be functions whose degree in that variable will decrease by exactly one for each differentiation. Choosing all the steps equal to one gives a compact and efficient formula for evaluating the higher order derivative for a "black box" function. While evaluating a higher order derivative we can obtain, at negligible extra cost, several derivatives of smaller order, just like one can use an efficient Moebius transform algorithm to compute all the subcubes of a large cube (see [11]).

We then show, in Sect. 4.2 that the main result of the classical cube attack, [5, Theorem 1], no longer holds when we differentiate repeatedly with respect to the same variable in

GF(p) (see Example 3 for a counterexample). However, we show that a modified version of this theorem does hold, see Theorem 6. In Sect. 4.3 we show that this Theorem can also be viewed as describing the relation between differentiation and the Moebius Transform over GF(p).

The proposed algorithm generalising the cube attack to GF(p) is sketched in Sect. 4.4. Now we not only choose variables for the "cube" but we also choose the number of times we are going to differentiate with respect to each variable. For computational efficiency, choosing only one variable (or a small number of variables) and differentiating a large number of times with respect to that variable is preferable. In GF(p) probabilistic linearity testing has a smaller expected number of tests than in GF(2), see [13]. We implemented this algorithm and give some concrete examples of functions where this algorithm is much more successful than a binary cube attack. Unfortunately we were unsuccessful so far in breaking actual ciphers (or reduced round versions thereof). We suspect this is due to the fact that the ciphers that use operations in GF(p) do combine them with operations in other fields/rings (e.g. ZUC includes operations modulo $2^{31} - 1$ but also modulo 2^{32}, IDEA includes operations modulo $2^{16} - 1$ and modulo 2^{16}), thus increasing the degree of the polynomials over GF(p) beyond reach.

While this paper concentrates on generalising the cube attack, it is likely that other attacks that use differentiation could also be generalised to GF(p) using our technique, for example cube testers (see [2]) can be generalised using Theorem 7, which shows that the values of the higher order derivative are uniformly distributed.

Finally, for completeness, we deal with generalisations to finite fields of the form GF(p^s) in Sect. 5.2. Here, for functions such as x^d with $p \mid d$, differentiation with respect to x decreases the degree by more than one regardless of the difference step. We give a more precise expression of the decrease in degree for higher order differentiation depending on the representation of the degree as an integer in base p. Any function can be differentiated at most $s(p-1)$ times before it becomes identically zero. We describe a choice of difference steps for which there are functions which do not become zero for anything less than $s(p-1)$ differentiations. Due to the fact that differentiation only uses the additive group of GF(p^s), which is isomorphic to GF(p)^s, differentiation over GF(p^s) can in fact be reduced to differentiation over GF(p) in each component of the projection of the function f. Therefore, we feel that developing a cube attack in GF(p^s), while possible, does not bring any additional advantages compared to a cube attack in GF(p).

2 Preliminaries

Throughout this paper GF(p^s) denotes the finite field with p^s elements where p is prime. R denotes an arbitrary commutative ring with identity and no zero divisors (i.e. an integral domain); the typical examples we have in mind are polynomials rings over GF(p^s) or over other fields. Note that since R is a domain, its characteristic is either zero or a prime number.

We denote by $\mathbf{e}_i = (0, \ldots, 0, 1, 0, \ldots, 0) \in R^n$ the vector which has a 1 in position i and zeroes elsewhere, i.e. $\mathbf{e}_1, \ldots, \mathbf{e}_n$ is the canonical basis of the vector space R^n.

We recall the definition of discrete derivative/discrete differentiation here, which will be called simply derivative and differentiation for the rest of this paper.

Definition 1 Let $f : R^n \to R^k$ be a function in n variables $x_1, \ldots, x_n$. Let $\mathbf{a} = (a_1, \ldots, a_n) \in R^n \setminus \{\mathbf{0}\}$. The *differentiation* operator with respect to a vector $\mathbf{a}$ associates to each function f its *derivative* $\Delta_{\mathbf{a}} f$ defined as

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

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

For the particular case of $a = h\mathbf{e_i}$ for some $1 \leq i \leq n$ and $h \in R \setminus \{0\}$, we will call $\Delta_{h\mathbf{e_i}}$ the differentiation operator with respect to the variable x_i with step h, or simply differentiation with respect to the variable x_i if $h = 1$ or if h is clear from the context. We will use the abbreviation "w.r.t. x_i" for "with respect to x_i".

Remark 1 Note that the discrete derivative with respect to a variable x should not be confused with the formal derivative with respect to x. For example, for the function $f : \mathrm{GF}(5) \to \mathrm{GF}(5)$, $f(x) = x^3$, the discrete derivative is $3x^2 + 3x + 1$ whereas the formal derivative is $3x^2$. While the degree and leading coefficient are the same for both the discrete derivative and formal derivative, in the context of "black box" functions it is preferable to work with the discrete derivative: to evaluate the discrete derivative at one point we only need two evaluations of the original function (i.e. two calls to the black box), whereas to evaluate the formal derivative we typically require evaluations of the original function at most points in its domain, which would be as costly as a full interpolation of the "black box" function. The two notions of discrete derivative and formal derivative coincide for polynomials of degree at most one in x, so in particular for all functions over GF(2), as they can always be represented as polynomial functions of degree at most one in each variable.

Remark 2 For defining the (discrete) differentiation operator, we do not actually need to work over a ring R, a commutative group (using additive notation for convenience) is sufficient. Here we used a ring due to our application to finite fields, and also due to some of the techniques involving polynomials.

The differentiation operator is a linear operator; composition of such operators is commutative and associative. Repeated application of the operator (also called higher order differentiation or higher order derivative in [16]) will be denoted by

$$\Delta^{(m)}_{\mathbf{a_1},\ldots,\mathbf{a_m}} f = \Delta_{\mathbf{a_1}} \Delta_{\mathbf{a_2}} \ldots \Delta_{\mathbf{a_m}} f \tag{1}$$

where $\mathbf{a_1}, \ldots, \mathbf{a_m} \in R^n$ are not necessarily distinct. An explicit formula can be obtained easily from Definition 1 by induction (see [16, Proposition 3]):

Proposition 1 *Let $f : R^n \to R^k$ be a function in n variables $x_1, \ldots, x_n$. Let $\mathbf{a_1}, \ldots, \mathbf{a}_m \in R^n \setminus \{\mathbf{0}\}$ not necessarily distinct. Then*

$$\begin{aligned}\Delta^{(m)}_{\mathbf{a_1},\ldots,\mathbf{a_m}} f(\mathbf{x}) &= \sum_{j=0}^{m} (-1)^{m-j} \sum_{\{i_1,\ldots,i_j\} \subseteq \{1,\ldots,m\}} f\left(\mathbf{x} + \mathbf{a_{i_1}} + \cdots + \mathbf{a_{i_j}}\right) \\ &= \sum_{\mathbf{b} \in \{0,1\}^m} (-1)^{m-\mathrm{w}(\mathbf{b})} f\left(\mathbf{x} + b_1\mathbf{a_1} + \cdots + \mathbf{x} + b_m\mathbf{a}_m\right)\end{aligned}$$

where w( ) *denotes the Hamming weight, i.e. the number of non-zero entries of a vector.*

In particular, when we differentiate a function w.r.t. one variable repeatedly, with all differentiation steps equal to 1 we obtain the following formula, well known in the context of finite differences:

$$\Delta^{(m)}_{\mathbf{e_1},\ldots,\mathbf{e_1}} f(x_1, \ldots, x_n) = \sum_{i=0}^{m} (-1)^{m-i} \binom{m}{i} f(x_1 + i, x_2, \ldots, x_n). \tag{2}$$

While the derivative can be defined for any function, in the sequel we will concentrate on polynomial functions. We will denote by $\deg_{x_i}(f)$ the degree of f in the variable x_i. The total degree will be denoted $\deg(f)$, with the usual convention of $\deg(0) = -\infty$.

Proposition 2 ([16, Proposition 2]) *Let $f : R^n \to R$ be a polynomial function in n variables and $\mathbf{a} \in R^n \setminus \{\mathbf{0}\}$. Then* $\deg(\Delta_{\mathbf{a}} f) \leq \deg(f) - 1$.

Differentiating with respect to one variable decreases the degree in that variable by at least one:

Proposition 3 *Let $f : R^n \to R$ be a polynomial function. Let $h, h_1, \ldots, h_m \in R \setminus \{0\}$ and $i \in \{1, \ldots, n\}$. If $\deg_{x_i}(f) = 0$ then $\Delta_{h\mathbf{e_i}} f(x_1, \ldots, x_n) \equiv 0$. If $\deg_{x_i}(f) > 0$ then $\deg_{x_i}(\Delta_{h\mathbf{e_i}} f) \leq \deg_{x_i}(f) - 1$. Consequently $\deg_{x_i}(\Delta^{(m)}_{h_1\mathbf{e_i},\ldots,h_m\mathbf{e_i}} f) \leq \deg_{x_i}(f) - m$.*

Recall that for integers $d, k_1, k_2, \ldots, k_m$ such that $\sum_{i=1}^m k_i = d$ and $k_i \geq 0$ the multinomial coefficient is defined as:

$$\binom{d}{k_1, k_2, \ldots, k_m} = \frac{d!}{k_1! k_2! \cdots k_m!},$$

generalising binomials coefficients, as $\binom{d}{k} = \binom{d}{k, d-k}$.

Next we examine the effect of higher order differentiation on univariate monomials. This result is straightforward and probably known but we could not find a reference so we included a proof for completeness; the general formula for univariate polynomials can be obtained using the linearity of the Δ operator.

Theorem 1 *Let $f : R \to R$ defined by $f(x) = x^d$ and let $h_1, \ldots, h_m \in R \setminus \{0\}$. Then*

$$\Delta^{(m)}_{h_1\mathbf{e_1},\ldots,h_m\mathbf{e_1}} x^d = \sum_{j=m}^{d} \binom{d}{j} \left(\sum_{\substack{(k_1,\ldots,k_m)\in\{1,2,\ldots,j-m+1\}^m \\ k_1+\ldots+k_m=j}} \binom{j}{k_1, \ldots, k_m} h_1^{k_1} \cdots h_m^{k_m} \right) x^{d-j}$$

Proof Induction on m. For $m = 1$ we have $\Delta_{h_1\mathbf{e_1}} x^d = (x+h_1)^d - x^d = \sum_{j=1}^d \binom{d}{j} h_1^j x^{d-j} = \sum_{j=1}^d \binom{d}{j, d-j} h_1^j x^{d-j}$ and the statement is verified.

Now let us assume the statement holds for a given m and we prove it for $m+1$.

$$\begin{aligned}
\Delta^{(m+1)}_{h_1\mathbf{e_1},\ldots,h_{m+1}\mathbf{e_1}} x^d &= \Delta_{h_{m+1}\mathbf{e_1}} \Delta^{(m)}_{h_1\mathbf{e_1},\ldots,h_m\mathbf{e_1}} x^d \\
&= \sum_{j=m}^{d} \binom{d}{j} \left(\sum_{\substack{(k_1,\ldots,k_m)\in\{1,2,\ldots,j-m+1\}^m \\ k_1+\cdots+k_m=j}} \binom{j}{k_1, \ldots, k_m} h_1^{k_1} \cdots h_m^{k_m} \right) \left((x + h_{m+1})^{d-j} - x^{d-j} \right) \\
&= \sum_{j=m}^{d} \binom{d}{j} \left(\sum_{\substack{(k_1,\ldots,k_m)\in\{1,2,\ldots,j-m+1\}^m \\ k_1+\cdots+k_m=j}} \binom{j}{k_1, \ldots, k_m} h_1^{k_1} \cdots h_m^{k_m} \right) \\
&\quad \times \left(\sum_{k_{m+1}=1}^{d-j} \binom{d-j}{k_{m+1}} h_{m+1}^{k_{m+1}} x^{d-j-k_{m+1}} \right) \\
&= \sum_{j'=m+1}^{d} \binom{d}{j'} \left(\sum_{\substack{(k_1,\ldots,k_{m+1})\in\{1,2,\ldots,j'-m\}^{m+1} \\ k_1+\cdots+k_{m+1}=j'}} \binom{j'}{k_1, \ldots, k_{m+1}} h_1^{k_1} \cdots h_{m+1}^{k_{m+1}} \right) x^{d-j'}.
\end{aligned}$$

In the last line we denoted $j' = j + k_{m+1}$ and used the identity:

$$\binom{d}{j}\binom{j}{k_1, \ldots, k_m}\binom{d-j}{k_{m+1}} = \binom{d}{j'}\binom{j'}{k_1, \ldots, k_{m+1}}.$$

□

Recall that in a finite field $\mathrm{GF}(p^s)$ we have $a^{p^s} = a$ for all elements a. Hence, while (formal) polynomials over $\mathrm{GF}(p^s)$ could have any degree in each variable, when we talk about the associated *polynomial function*, there will always be a unique polynomial of degree at most $p^s - 1$ in each variable which defines the same polynomial function. In other words we are working in the quotient ring $\mathrm{GF}(p^s)[x_1, \ldots, x_n]/\langle x_1^{p^s} - x_1, \ldots x_n^{p^s} - x_n\rangle$, and we use as representative of each class the unique polynomial which has degree at most $p^s - 1$ in each variable.

Moreover, all functions in n variables over a finite field can be written in Algebraic Normal Form (ANF) i.e. as polynomial functions of n variables. To summarise, each function in n variables over $\mathrm{GF}(p^s)$ can be uniquely expressed as a polynomial function defined by a polynomial in $\mathrm{GF}(p^s)[x_1, \ldots, x_n]$ of degree at most $p^s - 1$ in each variable.

The transform that associates to each function $f : \mathrm{GF}(p)^n \to \mathrm{GF}(p)$ the coefficients of its ANF is called the Moebius transform or Reed–Muller transform. More precisely, using the usual representation of $\mathrm{GF}(p)$ as $\{0, 1, \ldots, p-1\}$, the transform of f is a function $f^M : \mathrm{GF}(p)^n \to \mathrm{GF}(p)$ with $f^M(y_1, \ldots, y_n)$ defined as being equal to the coefficient of $x_1^{y_1} \cdots x_n^{y_n}$ in the ANF of f. We define a partial order on $\mathrm{GF}(p)^n$ as $\mathbf{a} \preceq \mathbf{b}$ iff $a_i \leq b_i$ for all $i = 1, \ldots, n$ (the order $\leq$ on $\mathrm{GF}(p) = \{0, 1, \ldots, p-1\}$ being the usual order on the integers). In the binary case, the Moebius transform of f can be obtained from the truth table of f using the following result (see for example [19, Chapter 13, Theorem 1]):

Theorem 2 *Let $f : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$. With the notations above,*

$$f^M(\mathbf{y}) = \sum_{\mathbf{a} \preceq \mathbf{y}} f(\mathbf{a}). \tag{3}$$

Note that using the Theorem above and Proposition 1 we also have that $f^M(\mathbf{y}) = \Delta^{(m)}_{\mathbf{e_{i_1}}, \ldots, \mathbf{e_{i_m}}} f(\mathbf{0})$ where $i_1, \ldots i_m$ are the positions of the non-zero elements of $\mathbf{y}$. An efficient algorithm for computing the truth table of f^M from the truth table of f is given for example in [12, Algorithm 9.6]. The algorithm has $\mathcal{O}(n2^n)$ complexity and works in-place (i.e. no significant additional memory needed). For computing f^M over $\mathrm{GF}(p)$ with $p > 2$, Theorem 2 does not hold; however f^M can be computed by the an in-place algorithm given in [12, Algorithm 9.9] with $np^{n-1}(p-1)^2$ operations, hence $\mathcal{O}(np^{n+1})$ complexity.

3 The cube/AIDA attack

3.1 Binary cube/AIDA attack

We recall the classical cube/AIDA attacks (see [5] and [21]) and their interpretation in the framework of higher order derivatives (see [7,14]). Throughout this subsection we work in the binary field $\mathrm{GF}(2)$.

The next definitions are taken from [5]: "Any subset I of size k defines a k dimensional Boolean cube C_I of 2^k vectors in which we assign all the possible combinations of 0/1 values to variables in I and leave all the other variables undetermined. Any vector $\mathbf{v} \in C_I$

defines a new derived polynomial $f_{|\mathbf{v}}$ with $n-k$ variables (whose degree may be the same or lower than the degree of the original polynomial). Summing these derived polynomials over all the 2^k possible vectors in C_I we end up with a new polynomial which is denoted by $f_I = \sum_{\mathbf{v}\in C_I} f_{|\mathbf{v}}$." Note that the computation of f_I requires 2^k calls to the "black box" function f. On the other hand denoting by t_I the product of the variables with indices in $I = \{i_1, \ldots, i_k\}$, i.e. $t_I = x_{i_1} \cdots x_{i_k}$, we can factor the common subterm t_I out of some of the terms in f and write f as

$$f(x_1, \ldots, x_n) = t_I f_{S(I)} + r(x_1, \ldots, x_n).$$

where each of the terms of $r(x_1, \ldots, x_n)$ misses at least one of the variables with index in I. Note that $f_{S(I)}$ is a polynomial in the variables with indices in $\{1, 2, \ldots, n\} \setminus I$.

The cube attack is based on the following main result:

Theorem 3 ([5, Theorem 1]) *For any polynomial f and subset of variables I, $f_I \equiv f_{S(I)}$* (mod 2).

For the cube attack we are particularly interested in the situation when $f_{S(I)}$ (and therefore f_I) has degree exactly one, i.e. it is linear but not constant (the corresponding term t_I is then called maxterm in [5]). Let d be the total degree of f. Then I having $d-1$ elements is a sufficient (but not necessary) condition for $f_{S(I)}$ to have degree at most one, i.e. to be linear or constant.

Several authors ([7,14]) showed that f_I can also be viewed as a higher order derivative:

Proposition 4 *For any polynomial f and subset of variables $I = \{i_1, \ldots, i_k\}$, we have $f_I = \Delta^{(k)}_{\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_k}} f$.*

We can also express f_I using the Moebius transform. Denote by g the polynomial function f viewed as a polynomial function in the variables $x_{i_1}, \ldots, x_{i_k}$ with the coefficients being polynomials in the remaining variables. By the definition of the Moebius transform and of $f_{S(I)}$ we have $f_{S(I)} = g^M(1, \ldots, 1)$. On the other hand, by Theorem 2, $g^M(1, \ldots, 1) = f_I$. This gives an alternative proof of Theorem 3. Note that using Proposition 1 and Theorem 2 we also have that $g^M(1, \ldots, 1) = \Delta^{(k)}_{\mathbf{e_{i_1}}, \ldots, \mathbf{e_{i_k}}} f$.

3.2 Existing generalisation of the cube attack to GF(p^s)

A generalisation of the cube attack from the binary field to GF(p^s) was sketched in [5]: "Over a general field GF(p^s) with $p > 2$, the correct way to apply cube attacks is to alternately add and subtract the outputs of the master polynomial with public inputs that range only over the two values 0 and 1 (and not over all their possible values of $0, 1, \ldots, p$), where the sign is determined by the sum (modulo 2) of the vector of assigned values."

We make this idea more precise; this was also done in [1] but we will follow a simpler approach for the proof of the main result. Let f be again a function of n variables $x_1, \ldots, x_n$, but this time over an arbitrary finite field GF(p^s). Note that now f can have degree up to $p^s - 1$ in each variable.

As before, we select a subset of k indices $I = \{i_1, \ldots, i_k\} \subseteq \{1, 2, \ldots, n\}$ and consider a "cube" C_I consisting of the n-tuples which have all combinations of the values 0/1 for the variables with indices in I, while the other variables remain indeterminate. The function f is evaluated at the points in the cube and these values are summed with alternating + and − signs obtaining a value

$$f_I = \sum_{\mathbf{v}\in C_I} (-1)^{k-\mathrm{w}_I(\mathbf{v})} f_{|\mathbf{v}}$$

where $w_I(\mathbf{v})$ denotes the Hamming weight of the vector $\mathbf{v}$ restricted to the positions with indices in I.

On the other hand denoting by t_I the product of the variables with indices in I, we can factor the common subterm t_I out of some of the terms in f and write f as

$$f(x_1, \ldots, x_n) = t_I f_{S(I)}(x_1, \ldots, x_n) + r(x_1, \ldots, x_n).$$

where each of the terms of $r(x_1, \ldots, x_n)$ misses at least one of the variables with index in I. Note that, unlike the binary case, now $f_{S(I)}$ can contain variables with indices in I.

Now we can prove an analogue of Theorem 3 and Proposition 4. (A similar theorem appears in [1, Theorem 6], but both the statement and the proof are more complicated, involving a term $t = x_{i_1}^{r_1} \cdots x_{i_k}^{r_k}$ instead of $x_{i_1} \cdots x_{i_k}$ and consequently when factoring out t and writing $f = t f_{S(t)} + q$, having to treat separately the terms of q which contain all the variables with indices in I, and the terms of q which do not.)

Theorem 4 *Let $f : \mathrm{GF}(p^s)^n \to \mathrm{GF}(p^s)$ be a polynomial function and I a subset of variable indices. Denote by $\mathbf{u}$ the n-tuple having values of 0 in the positions with indices in I and indeterminates in the other positions, and by $\mathbf{v}$ the n-tuple having values of 1 in the positions in I and indeterminates in the other positions. Then:*

$$f_I = (\Delta^{(k)}_{\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_k}} f)(\mathbf{u}) = f_{S(I)}(\mathbf{v})$$

Proof The fact that $f_I = (\Delta^{(k)}_{\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_k}} f)(\mathbf{u})$ follows from Proposition 1.

It suffices to show $(\Delta^{(k)}_{\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_k}} f)(\mathbf{u}) = f_{S(I)}(\mathbf{v})$ for the case when f is a monomial. The rest follows from the linearity of the operators, as $(f + g)_{S(I)} = f_{S(I)} + g_{S(I)}$ and Δ is a linear operator.

If f is a monomial not divisible by t_I, then both $f_{S(I)}$ and $\Delta^{(k)}_{\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_k}} f$ are identically zero, the latter using Proposition 3.

Now assume $f = t_I f_{S(I)}$ for some monomial $f_{S(I)}$. Like in the proof of [5, Theorem 1], we note that in the sum in the definition of f_I (or, equivalently in the sum given by Proposition 1 for $(\Delta^{(k)}_{\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_k}} f)(\mathbf{u})$) only one term is non-zero; namely t_I evaluates to a non-zero value iff $x_{i_1} = \ldots = x_{i_k} = 1$; hence $f_I = f_{S(I)}(\mathbf{v})$.

□

Remark 3 A cube attack based on Theorem 4 above would again search for sets I for which f_I is linear in the variables whose indices are not in I. If the total degree of f is d, the total degree of f_I can have any value up to and including $d - k$. If $k = d - 1$ we can guarantee that f_I is linear or a constant. More generally, the closer k gets to $d - 1$ (while still having $k \leq d - 1$), the higher the probability of linearity (assuming f is a random polynomial of degree d, in the sense that each term of degree up to d appears in f with probability 0.5).

However, unlike the binary case where $d \leq n$, now d can have any value up to $(p^s - 1)n$. Hence the (unknown) degree d of f could well be considerably higher than the number of variables n. In such a case, $k \leq n - 1$ is considerably lower than $d - 1$, and the probability of f_I being linear are very small. In other words, since we differentiate at most once w.r.t. each variable, so a total of at most $n - 1$ times, the degree decreases by $n - 1$ in general (with low probability, it will decrease by more than $n - 1$), and the resulting function can still have quite a high degree. Therefore, while a cube attack based on this result would be correct, it would have extremely low probability of success.

Our proposed generalisation of this attack, presented in the next section, would increase the success probability by differentiating several times with respect to each variable. This

will result in a greater decrease of the degree, thus improving the probability of reaching a linear result.

4 Generalisations to GF(p)

4.1 Differentiation in GF(p)

Differentiation with respect to a variable decreases the degree in that variable by at least one. In the binary case, the degree of a polynomial function in each variable is at most one, so we can only differentiate once w.r.t. each variable; a second differentiation will trivially produce the zero function. In GF(p) the degree in each variable is up to $p-1$. We can therefore consider differentiating several times (and possibly using different difference steps) with respect to each variable. We first show that any polynomial of degree d_i in a variable x_i can be differentiated m_i times w.r.t. x_i, for any $m_i \leq d_i$ (and using any collection of non-zero difference steps) and the degree decreases by exactly m_i. Hence we can differentiate d_i times without the result becoming identically zero.

Proposition 5 *Let* $f : \mathrm{GF}(p)^n \to \mathrm{GF}(p)$ *and* $1 \leq m_1 \leq \deg_{x_1}(f) < p$. *Let* $h_1, \ldots, h_{m_1} \in \mathrm{GF}(p) \setminus \{0\}$. *Then*

$$\deg_{x_1}\left(\Delta^{(m_1)}_{h_1\mathbf{e_1},\ldots,h_{m_1}\mathbf{e_1}} f\right) = \deg_{x_1}(f) - m_1.$$

Proof We first assume $f = x_1^{d_1}$, and we use Theorem 1. The coefficient of $x_1^{d_1-m_1}$ equals

$$\binom{d_1}{m_1}\binom{m_1}{1,1,\ldots,1} h_1 \cdots h_{m_1} = \frac{d_1!}{(d_1-m_1)!} h_1 \cdots h_{m_1}.$$

Since $d_1 = \deg_{x_1}(f) < p$, we have that $\dfrac{d_1!}{(d_1-m_1)!}$, as an integer, is not divisible by p, so it is non-zero in GF(p).

When f is an arbitrary polynomial, we can write it as $f = \sum_{i=0}^{d_1} g_i x_1^i$ with $g_i \in \mathrm{GF}(p)[x_2, \ldots, x_n]$. We have $\deg_{x_1}(\Delta^{(m_1)}_{h_1\mathbf{e_1},\ldots,h_{m_1}\mathbf{e_1}} g_i x_1^i) = i - m_1$ and since the derivative of each $g_i x_1^i$ has a different degree in x_1 their leading monomials cannot cancel out so $\deg_{x_1}(\Delta^{(m_1)}_{h_1\mathbf{e_1},\ldots,h_{m_1}\mathbf{e_1}} g_i x_1^i) = d_1 - m_1$.

□

Example 1 Let $f(x_1, x_2, x_3, x_4) = x_1^5 x_2 + x_1^4 x_3 x_4 + x_4^6$ be a polynomial with coefficients in GF(31). We choose the variable x_1 and differentiate repeatedly w.r.t. x_1, always with difference step equal to one. Differentiating once w.r.t. x_1 we obtain:

$$\begin{aligned}\Delta_{\mathbf{e_1}} f(x_1, x_2, x_3, x_4) &= x_1^4(5x_2) + x_1^3(10x_2 + 4x_3x_4) + x_1^2(10x_2 + 6x_3x_4) \\ &\quad + x_1(5x_2 + 4x_3x_4) + x_2 + x_3x_4\end{aligned}$$

Differentiating again w.r.t. x_1 we obtain:

$$\begin{aligned}\Delta^{(2)}_{\mathbf{e_1},\mathbf{e_1}} f(x_1, x_2, x_3, x_4) &= x_1^3(20x_2) + x_1^2(29x_2 + 12x_3x_4) \\ &\quad + x_1(8x_2 + 24x_3x_4) + 30x_2 + 14x_3x_4\end{aligned}$$

Finally, if we differentiate a total of 5 times we obtain:

$$\Delta^{(5)}_{\mathbf{e_1},\ldots,\mathbf{e_1}} f(x_1, x_2, x_3, x_4) = 5! x_2 = 27x_2.$$

Evaluating $\Delta^{(m)}_{h_1\mathbf{e_1},\ldots,h_m\mathbf{e_1}} f$ for a "black box" function f takes, in general, 2^m evaluations of f, according to Proposition 1. However, depending on the values of the h_i, some of the evaluation points might appear repeatedly:

Example 2 Consider a function in one variable, $f : \mathrm{GF}(31) \to \mathrm{GF}(31)$, and differentiation of order 3.

First consider $h_1 = 1, h_2 = 3, h_3 = 5$. We have

$$\begin{aligned}\Delta^{(3)}_{h_1\mathbf{e_1},h_2\mathbf{e_1},h_3\mathbf{e_1}} f(x) &= f(x+9) - f(x+4) - f(x+6) - f(x+8) + f(x+1)\\ &\quad + f(x+3) + f(x+5) - f(x),\end{aligned}$$

so we need to evaluate f at 8 distinct arguments.

Next consider $h_1 = h_2 = h_3 = 1$. We have

$$\Delta^{(3)}_{h_1\mathbf{e_1},h_2\mathbf{e_1},h_3\mathbf{e_1}} f(x) = f(x+3) - 3f(x+2) + 3f(x+1) - f(x),$$

so we need to evaluate f at only 4 distinct arguments.

We determine how to obtain the minimum number of evaluations:

Proposition 6 *Let $f : \mathrm{GF}(p)^n \to \mathrm{GF}(p)$. Let $1 \le m < p$ and let $h_1, \ldots, h_m \in \mathrm{GF}(p)\backslash\{0\}$. For one evaluation of $\Delta^{(m)}_{h_1\mathbf{e_1},\ldots,h_m\mathbf{e_1}} f$ (using the formula in Proposition 1) we need to evaluate f at least $m+1$ times. If $h_1 = \ldots = h_m$, this minimum is achieved.*

Proof Let $S_m = \{\sum_{j\in J} h_j | J \subseteq \{1, \ldots, m\}\}$. By Proposition 1 we need to evaluate f at the points $x + s$ with $s \in S_m$, so we need to determine the minimum cardinality of S_m. We prove by induction that $|S_m| \ge m + 1$. The $m = 1$ base case: $S_1 = \{0, h_1\}$ has 2 elements.

We have $S_m = S_{m-1} \cup \{s + h_m | s \in S_{m-1}\}$, so obviously $|S_m| \ge |S_{m-1}|$. For the induction step we assume $|S_{m-1}| \ge m$ and we prove $|S_m| \ge m + 1$. If $|S_{m-1}| \ge m + 1$ we are done. Assume $|S_{m-1}| = m$. We know $|S_m| \ge |S_{m-1}| = m$, so assume for a contradiction that $|S_m| = m$. This means $S_m = S_{m-1}$; hence for any $s \in S_{m-1}$ we have $s + h_m \in S_{m-1}$. Starting from a value $s \in S_{m-1}$ and applying this argument repeatedly we obtain that $s + h_m, s + 2h_m, \ldots, s + mh_m \in S_{m-1}$. This is a list of m distinct elements, since $m < p$, so all elements of S_{m-1} appear in this list. Hence there is $1 \le i \le m$ such that $s = s + ih_m$. But this means $ih_m = 0$, contradiction.

When $h_1 = \ldots = h_m$ we have $S_m = \{ih_1 | i = 0, \ldots, m\}$, so $|S_m| = m + 1$. Since $0 \le i \le m < p$ and p is prime, $\binom{m}{i}$ cannot be divisible by p, so it is non-zero in a field of characteristic p. Hence the evaluation of the derivative needs exactly $m + 1$ evaluations of f in this case. □

Note that decreasing the number of evaluations from the maximum, 2^m, to the minimum, $m + 1$ is a considerable saving, so this is what we will do for the remainder of this section: we set all the difference steps h_i equal to each other, and for simplicity, equal to one. For convenience we will introduce some more notation. We pick a subset of k variable indices $I = \{i_1, \ldots, i_k\}$ and we also pick multiplicities for each variable, $m_1, \ldots, m_k$. Denote by t the term $t = x_{i_1}^{m_1} \cdots x_{i_k}^{m_k}$. We will apply the differentiation operator m_1 times w.r.t. the variable x_{i_1}, and m_2 times w.r.t. the variable x_{i_2} etc. always with difference step equal to one. More precisely we define:

$$f_t(x_1, \ldots, x_n) = \Delta^{(m_1)}_{\mathbf{e_{i_1}},\ldots,\mathbf{e_{i_1}}} \ldots \Delta^{(m_k)}_{\mathbf{e_{i_k}},\ldots,\mathbf{e_{i_k}}} f(x_1, \ldots, x_n)$$

Since we have set all differentiation steps equal to one, Theorem 1 becomes simpler; we will also generalise it to differentiating a monomial w.r.t. several variables:

Theorem 5 *Let* $f : \mathrm{GF}(p)^n \to \mathrm{GF}(p)$.

(i) *If* $f(x_1, \ldots, x_n) = x_1^{d_1}$, $1 \le m_1 \le d_1 \le p-1$ *and* $t = x_1^{m_1}$ *then*

$$f_t(x_1, \ldots, x_n) = m_1! \sum_{j=m_1}^{d_1} \binom{d_1}{j} S(j, m_1) x_1^{d_1 - j}$$

where $S(.,.)$ *denotes the Stirling numbers of the second kind. The degree of* f_t *in* x_1 *equals* $d_1 - m_1$ *and the leading coefficient is* $\frac{d_1!}{(d_1-m_1)!}$. *The free term of* f_t *is* $f_t(\mathbf{0}) = m_1! S(d_1, m_1)$.

(ii) *If* $f(x_1, \ldots, x_n) = x_{i_1}^{d_1} \cdots x_{i_k}^{d_k}$ *and* $t = x_{i_1}^{m_1} \cdots x_{i_k}^{m_k}$ *for some* $k \le n$, $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$ *and* $1 \le m_\ell \le d_\ell \le p-1$ *for* $\ell = 1, \ldots, k$, *then:*

$$f_t = \left(\prod_{\ell=1}^{k} m_\ell! \right) \sum_{j_1=m_1}^{d_1} \cdots \sum_{j_k=m_k}^{d_k} \left(\prod_{\ell=1}^{k} \binom{d_\ell}{j_\ell} S(j_\ell, m_\ell) \right) x_{i_1}^{d_1 - j_1} \ldots x_{i_k}^{d_k - j_k}.$$

The total degree of f_t *is* $\sum_{\ell=1}^{k} (d_\ell - m_\ell)$ *and the free term is* $\prod_{\ell=1}^{k} m_\ell! S(d_\ell, m_\ell)$.

Proof For (i), we use Theorem 1 with all $h_i = 1$. The easiest way to see that

$$\sum_{\substack{(k_1, \ldots, k_m) \in \{1,2,\ldots, j-m+1\}^m \\ k_1 + \ldots + k_m = j}} \binom{j}{k_1, \ldots, k_m} = m! S(j, m)$$

is to see that both sides of this equation count the number of ways of distributing j labelled objects into m labelled boxes so that there is at least one object in each box. For (ii) we use (i) for each variable. □

We examine the generalisation of Eq. (2) to differentiation in several variables:

Proposition 7 *Let* $f : \mathrm{GF}(p)^n \to \mathrm{GF}(p)$ *and* $t = \prod_{\ell=1}^{k} x_{i_\ell}^{m_\ell}$. *Then*

$$f_t(\mathbf{x}) = \sum_{j_1=0}^{m_1} \cdots \sum_{j_k=0}^{m_k} (-1)^{\sum_{\ell=1}^{k} (m_\ell - j_\ell)} \binom{m_1}{j_1} \cdots \binom{m_k}{j_k} f(\ldots, x_{i_1} + j_1, \ldots, x_{i_k} + j_k, \ldots). \tag{4}$$

All the coefficients $\binom{m_1}{j_1} \cdots \binom{m_k}{j_k}$ *in the sum above are non-zero. If* f *is a "black box" function, one evaluation of* f_t *needs* $\prod_{\ell=1}^{k} (m_\ell + 1)$ *evaluations of* f. *In particular evaluating* f_t *for* $x_{i_\ell} = 0$, $\ell = 1, \ldots, k$ *we obtain:*

$$\sum_{j_1=0}^{m_1} \cdots \sum_{j_k=0}^{m_k} (-1)^{\sum_{\ell=1}^{k} (m_\ell - j_\ell)} \binom{m_1}{j_1} \cdots \binom{m_k}{j_k} f(\ldots, j_1, \ldots, j_k, \ldots) \tag{5}$$

with $j_1, \ldots, j_k$ *in positions* $i_1, \ldots, i_k$ *respectively.*

We note that in terms of the time complexity of evaluating f_t, it is now not only the total degree of t that matters (as in the binary case), but also the exponents of each variable. For a given number m of differentiations (i.e. t of total degree m), the smallest time complexity is achieved when t contains only one variable, i.e. $t = x_{i_1}^m$. Among all t of total degree m that contain k variables, the best time complexity is achieved when t has degree one in each but one of its variables, e.g. $t = x_{i_1}^{m-k+1} x_{i_2} \ldots x_{i_k}$.

Remark 4 We saw that in the binary case, differentiating once w.r.t. each of the variables $x_{i_1}, \ldots x_{i_k}$ is equivalent to summing f evaluated over a "cube" consisting of all the 0/1 combinations for the variables $x_{i_1}, \ldots x_{i_k}$. According to Proposition 7, the analogue of the "cube" will now be a k-dimensional grid/mesh with sides of "length" $m_1, m_2, \ldots, m_k$. Namely each variable x_{i_ℓ} w.r.t. which we differentiate will have increments of $0, 1, \ldots, m_\ell$ and each term in the sum has alternating signs as well as being multiplied by binomial coefficients.

Instead of evaluating $f_t(\mathbf{0})$ for a "black box" function f using the formula in Proposition 7 we can, at a small extra cost, compute $f_{t'}(\mathbf{0})$ for all terms t' that divide t, using the following algorithm that generalises the Moebius transform algorithm from the binary case, see [12, Algorithm 9.6] (but note our generalisation does not compute the Moebius transform over GF(p), as this is no longer the same as the derivative, see further discussion in Sect. 4.3). The algorithm is quite straightforward, using the recursive definition of higher order derivatives, see (1). Firstly, Algorithm 1 deals with higher order differentiation w.r.t. one variable. This algorithm performs $m + (m-1) + \cdots + 1 = m(m+1)/2$ operations so its complexity is $\mathcal{O}(m^2)$ operations in the ring GF(p). For the correctness we can prove by induction that at the end of each run of the outer **for** loop we have $A[i] = f_{x^i}(0)$ for $i = 1, \ldots, j$ and $A[i] = f_{x^j}(i-j)$ for $i = j+1, \ldots, m$.

Algorithm 1: HigherDerivativesUnivariate(A, m)

Input : An array A indexed from 0 to m so that $A[i] = f(i)$ where $f : \mathrm{GF}(p) \to \mathrm{GF}(p)$
Output: The array A is modified in-place so that now $A[i] = f_{x^i}(0)$

```
1 begin
2     for j = 1 to m do
3         for i = m downto j do
4             A[i] = A[i] − A[i − 1]
5         end
6     end
7     return(A);
8 end
```

Algorithm 2 deals with differentiation w.r.t. several variables and uses Algorithm 1. For ease of notation, in this algorithm we allow $m_i = 0$ i.e. differentiating zero times (i.e. no differentiation) w.r.t. some of the variables x_i. The number of operations is $\sum_{j=1}^{k}\left(\frac{m_j(m_j+1)}{2}\prod_{\ell\neq j}(m_\ell+1)\right) = \frac{1}{2}(\sum_{j=1}^{k} m_j)(\prod_{j=1}^{n}(m_j+1))$. Comparing to Proposition 7, this is only higher by a factor of $\frac{1}{2}(\sum_{j=1}^{k} m_j)$ than computing only $f_t(\mathbf{0})$. Since all $m_i < p$, the overall complexity is $\mathcal{O}(kp^{k+1})$ operations in GF(p), compared to $\mathcal{O}(p^k)$ for computing only $f_t(\mathbf{0})$ (where k is the number of variables w.r.t. which we actually differentiate i.e. $m_i > 0$). For the correctness we can prove by induction that at the end of each run of the outer **for** loop we have $A[k_1, \ldots, k_n] = f_{x_1^{k_1}\ldots x_j^{k_j}}(0, \ldots, 0, k_{j+1}, \ldots, k_n)$ for all $(k_1, \ldots, k_n)$.

Remark 5 All the results in this section also hold with GF(p) replaced by an arbitrary integral domain R provided that either R is of zero characteristic, or if the characteristic of R is a prime p then all the degrees d_i and all the m_i are less than p.

Algorithm 2: HigherDerivatives(A, n, $(m_1, \ldots, m_n)$)

Input : An n-dimensional array A of size $(m_1 + 1) \times \cdots \times (m_n + 1)$, with all indices starting from 0, such that $A[k_1, \ldots, k_n] = f(k_1, \ldots, k_n)$ where $f : \mathrm{GF}(p)^n \to \mathrm{GF}(p)$

Output: The array A is modified in-place so that now $A[k_1, \ldots, k_n] = f_{x_1^{k_1} \ldots x_n^{k_n}}(\mathbf{0})$

```
1 begin
2   for j = 1 to n do
3     for all (k_1, ..., k_{j-1}, k_{j+1}, ..., k_n) ∈
      {0, ..., m_1} × ··· × {0, ..., m_{j-1}} × {0, ..., m_{j+1}} × ··· , ×{0, ..., m_n} do
4       Apply the algorithm HigherDerivativesUnivariate to the array
        (A[k_1, ..., k_{j-1}, i, k_{j+1}, ..., k_n], i = 0, ..., m_j)
5     end
6   end
7   return(A);
8 end
```

4.2 Fundamental theorem of the cube attack generalised to GF(p)

Proposition 2, stating that the degree decreases by at least one with each differentiation, is already sufficient for a cube attack. Namely if we differentiate a number of times equal to $\deg(f) - 1$ the result is guaranteed to have degree at most one. Depending on f, we might obtain degree one earlier (i.e. for a lower number of differentiations). However, we will give a more refined result shortly, in order to give an analogue of the main theorem of the classical cube attack (see Theorems 3, 4 and Proposition 4).

We will use the notation $f_t = \Delta^{(m_1)}_{\mathbf{e_{i_1}}, \ldots, \mathbf{e_{i_1}}} \ldots \Delta^{(m_k)}_{\mathbf{e_{i_k}}, \ldots, \mathbf{e_{i_k}}} f$ with $t = x_{i_1}^{m_1} \cdots x_{i_k}^{m_k}$ as in the previous section. Factoring out t, we can write f as

$$f(x_1, \ldots, x_n) = t f_{S(t)}(x_1, \ldots, x_n) + r(x_1, \ldots, x_n)$$

where $f_{S(t)}$ and r are unique such as none of the terms in r is divisible by t.

At first sight we might expect Theorem 4 to hold here too, namely we might expect that $f_t(x_1, \ldots, x_n)$ evaluated at $x_{i_j} = 0$ for $j = 1, \ldots, k$ equals $f_{S(t)}(x_1, \ldots, x_n)$ evaluated at $x_{i_j} = 1$ for $j = 1, \ldots, k$. However this is not true in general, as the following counterexample shows:

Example 3 We continue Example 1 for $f(x_1, x_2, x_3, x_4) = x_1^5 x_2 + x_1^4 x_3 x_4 + x_4^6$.

We computed $f_{x_1}(x_1, x_2, x_3, x_4)$ in Example 1. Evaluating at $x_1 = 0$ we obtain $f_{x_1}(0, x_2, x_3, x_4) = x_2 + x_3 x_4$. On the other hand $f_{S(x_1)}(x_1, x_2, x_3, x_4) = x_1^4 x_2 + x_1^3 x_3 x_4$. Evaluating at $x_1 = 1$ we obtain $f_{S(x_1)}(1, x_2, x_3, x_4) = x_2 + x_3 x_4$. Hence we verified that $f_{x_1}(0, x_2, x_3, x_4) = f_{S(x_1)}(1, x_2, x_3, x_4)$ as expected by Theorem 4.

Differentiating again w.r.t. x_1 and evaluating $f_{x_1^2}(x_1, x_2, x_3, x_4)$ at $x_1 = 0$ gives $f_{x_1^2}(0, x_2, x_3, x_4) = 30x_2 + 14x_3 x_4$. On the other hand $f_{S(x_1^2)}(x_1, x_2, x_3, x_4) = x_1^3 x_2 + x_1^2 x_3 x_4$, which evaluated at $x_1 = 1$ gives $f_{S(x_1^2)}(1, x_2, x_3, x_4) = x_2 + x_3 x_4$.

Hence $f_{x_1^2}(0, x_2, x_3, x_4) \neq f_{S(x_1^2)}(1, x_2, x_3, x_4)$, so Theorem 4 cannot be extended in its current form to the case when we differentiate more than once w.r.t. one variable. However, note that the two quantities computed here do contain the same monomials.

Finally, if we differentiate 5 times with respect to x_1 we obtain: $f_{x_1^5}(x_1, x_2, x_3, x_4) = 27x_2$, whereas $f_{S(x_1^5)}(x_1, x_2, x_3, x_4) = x_2$ so again the two polynomials do not coincide; however they only differ by multiplication by a constant.

The correct generalisation of the main theorem of the classical cube attack is the following:

Theorem 6 *Let $f : \mathrm{GF}(p)^n \to \mathrm{GF}(p)$ be a polynomial function and $t = \prod_{j=1}^{k} x_{i_j}^{m_j}$. Denote*

$$f_t(x_1, \ldots, x_n) = \Delta^{(m_1)}_{\mathbf{e_{i_1}}, \ldots, \mathbf{e_{i_1}}} \ldots \Delta^{(m_k)}_{\mathbf{e_{i_k}}, \ldots, \mathbf{e_{i_k}}} f(x_1, \ldots, x_n).$$

Write f as

$$f(\mathbf{x}) = t f_{S(t)}(\mathbf{x}) + r(\mathbf{x})$$

so that none of the monomials in r are divisible by t.

Denote by $\mathbf{u}$ the n-tuple having values of 0 in the positions $i_1, \ldots, i_k$ and indeterminates elsewhere, and by $\mathbf{v}$ the n-tuple having values of 1 in the positions $i_1, \ldots, i_k$ and indeterminates elsewhere.

Write $f_{S(t)} = t_1 g_1 + \cdots + t_j g_j$, where $g_1, \ldots, g_j$ are polynomials that do not depend on any of the variables $x_{i_1}, \ldots, x_{i_k}$ and $t_1, \ldots, t_j$ are all the distinct terms in the variables $x_{i_1}, \ldots, x_{i_k}$ that appear in $f_{S(t)}$.

Then there are constants $c_1, \ldots, c_j \in \mathrm{GF}(p)$ such that

$$f_t(\mathbf{u}) = c_1 g_1 + \cdots + c_j g_j$$

(the latter can also be viewed as $c_1 t_1 g_1 + \cdots + c_j t_j g_j$ evaluated at $\mathbf{v}$). The exact values for the constants c_i can be determined as follows: if $t_i = x_{i_1}^{b_1} \ldots x_{i_k}^{b_k}$, then $c_i = \prod_{\ell=1}^{k} m_\ell! S(m_\ell + b_\ell, m_\ell)$ where $S()$ are Stirling numbers of the second kind.

In particular, if $f_{S(t)}$ does not depend on any of the variables $x_{i_1}, \ldots, x_{i_k}$ then

$$f_t(\mathbf{x}) = m_1! \ldots m_k! f_{S(t)}.$$

Proof We first assume $f = x_{i_1}^{d_1} \cdots x_{i_k}^{d_k} g$ where $d_i \geq m_i$ and g does not depend on the variables $x_{i_1}, \ldots, x_{i_k}$. Then $f_{S(t)} = x_{i_1}^{d_1 - m_1} \cdots x_{i_k}^{d_k - m_k} g$, so $f_{S(t)}(\mathbf{v}) = g(\mathbf{v})$. Using Theorem 5(ii) we have $f_t(\mathbf{u}) = g(\mathbf{u}) \prod_{\ell=1}^{k} m_\ell! S(d_\ell, m_\ell)$. But $g(\mathbf{u}) = g(\mathbf{v}) = g(\mathbf{x})$ since g does not depend on the variables $x_{i_1}, \ldots, x_{i_k}$. Hence $f_t(\mathbf{u}) = g \prod_{\ell=1}^{k} m_\ell! S(d_\ell, m_\ell)$. We can then extend this result to an arbitrary f by the linearity of the Δ operator. □

4.3 Connection to the Moebius transform

Throughout this section we have $f : \mathrm{GF}(p)^n \to \mathrm{GF}(p)$ and $I = \{i_1, \ldots, i_k\}$. We denote by g the polynomial f viewed as a polynomial in $x_{i_1}, \ldots, x_{i_k}$ with coefficients being polynomials in the remaining variables. The Moebius transform is defined over $\mathrm{GF}(p)$ as described towards the end of Sect. 2. Finally, $\mathbf{u}$ is the vector having zeroes in positions $i_1, \ldots, i_k$ and indeterminates elsewhere, and $\mathbf{v}$ is the vector having ones in positions $i_1, \ldots, i_k$ and indeterminates elsewhere.

In Sect. 3.1 we saw that in the binary case the following three notions are equivalent:

A. $\sum_{\mathbf{w} \in C_I} f(\mathbf{w})$
B. $\Delta^{(k)}_{\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_k}} f$
C. $g^M(1, 1, \ldots, 1)$.

What happens to the relation between these three items when we move to $\mathrm{GF}(p)$? We saw that when generalising the derivative we can differentiate several times w.r.t. each variable, say m_1 times w.r.t. x_{i_1}, m_2 times w.r.t. x_{i_2}, and so on, i.e.

B. $\Delta^{(m_1)}_{\mathbf{e}_{i_1},\ldots,\mathbf{e}_{i_1}} \ldots \Delta^{(m_k)}_{\mathbf{e}_{i_k},\ldots,\mathbf{e}_{i_k}} f$

For the Moebius transform we use again g:

C. $g^M(y_1, \ldots, y_k)$

In Sect. 3.2 we saw that if we use alternating signs in **A**, and we only allow at most one differentiation w.r.t each variable in **B** then we obtain **A** = **B**. Their relationship with **C** is given in Theorem 4, which can be reformulated as

$$\left(\Delta^{(k)}_{\mathbf{e}_{i_1},\ldots,\mathbf{e}_{i_k}} f\right)(\mathbf{u}) = \sum_{(1,\ldots,1)\preceq \mathbf{y}} g^M(\mathbf{y}) \tag{6}$$

where the partial order relation $\preceq$ is defined as for Theorem 2. To see that the formula above is indeed equivalent to Theorem 4, note that

$$f_{S(I)} = \sum_{(1,\ldots,1)\preceq \mathbf{y}} g^M(\mathbf{y}) x_{i_1}^{y_1-1} \cdots x_{i_k}^{y_k-1}$$

so when evaluating $f_{S(I)}(\mathbf{v})$ as required in Theorem 4, we obtain the right hand side of Eq. (6) above.

Section 4.1 shows that if we are to differentiate several times in each variable in **B** then we need to generalise **A** as in (4) in order to have **A** = **B**. Theorem 6 can be viewed as giving the relation between **B** and **C**, i.e. between higher order differentiation and the Moebius transform. More precisely, it states that :

$$f_t(\mathbf{u}) = \sum_{\mathbf{m}\preceq \mathbf{y}} \mathbf{m}! S(\mathbf{y}, \mathbf{m}) g^M(\mathbf{y}) \tag{7}$$

where we used again the notation $t = x_{i_1}^{m_1} \cdots x_{i_k}^{m_k}$ and $f_t = \Delta^{(m_1)}_{\mathbf{e_{i_1}},\ldots,\mathbf{e_{i_1}}} \ldots \Delta^{(m_k)}_{\mathbf{e_{i_k}},\ldots,\mathbf{e_{i_k}}} f$ and by abuse of notation we wrote $\mathbf{m}! = \prod_{\ell=1}^k m_\ell!$ and $S(\mathbf{y}, \mathbf{m}) = \prod_{\ell=1}^k S(y_\ell, m_\ell)$ with $S(\,)$ Stirling numbers of the second kind. Again, the equivalence of (7) above to Theorem 6 can be seen using:

$$f_{S(t)} = \sum_{\mathbf{m}\preceq \mathbf{y}} g^M(\mathbf{y}) x_{i_1}^{y_1-m_1} \cdots x_{i_k}^{y_k-m_k}.$$

Conversely, we can also express the Moebius transform in terms of the higher order derivatives. We could write Eq. (7) above for each of the p^k possible values of $(m_1, \ldots, m_k)$ and view them as a system of linear equations in the p^k unknowns $g^M(\mathbf{y})$. Note that the system can be easily solved as it is already in triangular form (the triangular form becomes clear if we write the Eq. (7) in decreasing order of $\sum_{i=1}^k m_i$, and any order within the same value of $\sum_{i=1}^k m_i$; then each equation contains exactly one new unknown, namely $g^M(m_1, \ldots, m_k)$ which has not appeared in the previous equations). This could yield an alternative to the algorithm for computing the Moebius transform over GF(p) in the book by Joux [12, Algorithm 9.9]. Namely, we first compute all the higher order derivatives $f_t(\mathbf{0})$ for all terms t in $x_1, \ldots, x_n$ using Algorithm 2 only once, with $m_1 = \cdots = m_n = p - 1$. Then we compute the Moebius Transform by solving the triangular system of linear equations as described above. However, Algorithm 2 already has the same complexity as [12, Algorithm 9.9], so the overall complexity of this proposed algorithm cannot be better than [12, Algorithm 9.9].

We do not have to generalise **C** as a Moebius transform tough. Instead of using the ANF, we can express any function $f : \mathrm{GF}(p) \to \mathrm{GF}(p)$ as a polynomial in the basis $x^{\underline{i}}$, $i = 0, 1, \ldots,$

where we used the notation $x^{\underline{i}} = x(x-1)\cdots(x-i+1)$ for the falling factorial. We can then consider for any function $f : \mathrm{GF}(p)^n \to \mathrm{GF}(p)$ the transform that associates to it the function $f^F : \mathrm{GF}(p)^n \to \mathrm{GF}(p)$ so that $f^F(y_1, \ldots, y_n)$ is the coefficient of $x_1^{\underline{y_1}} \ldots x_n^{\underline{y_n}}$ in f. It is well known that this representation is very convenient for working with discrete derivatives, as we have $\Delta_{\mathbf{e_1}} x^{\underline{d}} = d x^{\underline{d-1}}$. So in our case:

$$f_t(\mathbf{x}) = \sum_{\mathbf{m} \preceq \mathbf{y}} \frac{\mathbf{y}!}{(\mathbf{y}-\mathbf{m})!} g^F(\mathbf{y}) x_{i_1}^{\underline{y_1 - m_1}} \cdots x_{i_k}^{\underline{y_k - m_k}}$$

and when evaluating at $\mathbf{u}$ only the term for $\mathbf{y} = \mathbf{m}$ remains:

$$f_t(\mathbf{u}) = \mathbf{m}! g^F(\mathbf{m}). \tag{8}$$

This means that if we generalise $\mathbf{C}$ to be the transform g^F rather than the Moebius transform, then $\mathbf{B}$ does become equal to $\mathbf{C}$ multiplied by a suitable constant (Note that in the binary case the two transforms coincide, $g^M = g^F$). We can also obtain an alternative proof of Theorem 6, by using the well known formula for transforming one representation into the other $x^d = \sum_{j=0}^{d} S(d, j) x^{\underline{j}}$. Namely:

$$\begin{aligned}
f(\mathbf{x}) &= \sum_{\mathbf{y} \succeq \mathbf{0}} g^M(\mathbf{y}) x_{i_1}^{y_1} \cdots x_{i_k}^{y_k} \\
&= \sum_{\mathbf{y} \succeq \mathbf{0}} g^M(\mathbf{y}) \sum_{\mathbf{z} \preceq \mathbf{y}} S(\mathbf{y}, \mathbf{z}) x_{i_1}^{\underline{z_1}} \cdots x_{i_k}^{\underline{z_k}} \\
&= \sum_{\mathbf{z} \succeq \mathbf{0}} \left(\sum_{\mathbf{y} \succeq \mathbf{z}} S(\mathbf{y}, \mathbf{z}) g^M(\mathbf{y}) \right) x_{i_1}^{\underline{z_1}} \cdots x_{i_k}^{\underline{z_k}} \\
f_t(\mathbf{x}) &= \sum_{\mathbf{z} \succeq \mathbf{m}} \frac{\mathbf{z}!}{(\mathbf{z}-\mathbf{m})!} \left(\sum_{\mathbf{y} \succeq \mathbf{z}} S(\mathbf{y}, \mathbf{z}) g^M(\mathbf{y}) \right) x_{i_1}^{\underline{z_1 - m_1}} \cdots x_{i_k}^{\underline{z_k - m_k}} \\
f_t(\mathbf{u}) &= \sum_{\mathbf{y} \succeq \mathbf{m}} \mathbf{m}! S(\mathbf{y}, \mathbf{m}) g^M(\mathbf{y}).
\end{aligned}$$

Equation (8) can also be used to determine the probability distribution of the values of the higher order derivative, which in turn can be used for generalising cube testers (see [2]) from the binary case to $\mathrm{GF}(p)$:

Theorem 7 *Let $d > 0$ be an integer and let $t = x_1^{m_1} \cdots x_n^{m_n}$ with $0 \le m_j < p$ and $m_1 + \cdots m_n \le d$. If we choose a function f uniformly at random from the set $\{f : \mathrm{GF}(p)^n \to \mathrm{GF}(p) | \deg(f) \le d\}$ then for each $i = 0, 1, \ldots, p-1$ we have*

$$P(f_t(\mathbf{0}) = i) = \frac{1}{p}.$$

Moreover, for different terms $t_1, \ldots, t_r$ of degree at most d, if we choose f as above, the values $f_{t_1}(\mathbf{0}), \ldots, f_{t_r}(\mathbf{0})$ are independent.

Proof Using Eq. (8) with $k = n$ and $i_j = j$, we see that $f_t(\mathbf{0}) = \mathbf{m}! f^F(\mathbf{m})$. On the other hand $f^F(\mathbf{m})$ is uniformly distributed when f is chosen uniformly at random as in the statement of the theorem, as there is a bijection between the functions in that set and the "truth tables" of the corresponding f^F, i.e. tables where the entries $f^F(y_1, \ldots, y_n)$ are zero for all $y_1 + \cdots + y_n > d$, and the remaining table entries take all possible values. For the independence, note that the values of $f^F(\mathbf{m})$ for different values of $\mathbf{m}$ are independent. □

4.4 Proposed Algorithm for the cube attack in GF(p)

In this section we give more details of the algorithm, drawing on the results from previous sections. The main idea of our proposed attack is that when the degree in one variable is higher than one, we can differentiate w.r.t. that variable repeatedly, unlike the cube attacks described in the Sect. 3, which use differentiation at most once for each variable.

We are given a cryptographic "black box" function $f(v_1, \dots v_m, x_1, \dots, x_n)$ with v_i being public variables and x_i being secret variables.

Preprocessing phase

1. Choose a term in the public variables, $t = v_{i_1}^{m_1} \cdots v_{i_k}^{m_k}$, with $1 \leq m_i \leq p-1$.
2. Using formula (5) in Proposition 7 we evaluate $f_t(\mathbf{0}, \mathbf{x})$ for several choices of the secret variables $\mathbf{x}$, in order to decide whether, with reasonably high probability, the total degree of $f_t(\mathbf{0}, \mathbf{x})$ in $\mathbf{x}$ equals one. (For this, one can use the textbook definition of linearity; namely, for various values of $a, b \in \mathrm{GF}(p)$ and $\mathbf{y}, \mathbf{z} \in \mathrm{GF}(p)^n$ test whether $a(f_t(\mathbf{0}, \mathbf{y}) - f_t(\mathbf{0}, \mathbf{0})) + b(f_t(\mathbf{0}, \mathbf{z}) - f_t(\mathbf{0}, \mathbf{0})) = f_t(\mathbf{0}, a\mathbf{y} + b\mathbf{z}) - f_t(\mathbf{0}, \mathbf{0})$; in GF($p$) with p large, we will need in general much fewer linearity tests than in the binary case, see [13]; one can at the same time check whether $f_t(\mathbf{0}, \mathbf{x})$ is non-constant).
3. If the decision above is "yes", we determine $f_t(\mathbf{0}, \mathbf{x})$ explicitly, as $f_t(\mathbf{0}, \mathbf{x}) = c_0 + \sum_{i=1}^{n} c_i x_i$ where $c_0 = f_t(\mathbf{0}, \mathbf{0})$ and $c_i = f_t(\mathbf{0}, \mathbf{e}_i) - c_0$; we store $(t, c_0, c_1, \dots, c_n)$.
4. Repeat the steps above for different values of t until one obtains n linearly independent stored tuples $(c_1, \dots, c_n)$, or until one runs out of time/memory.

For the heuristic of choosing t one could take into account the computational cost for a term t, see Proposition 7 and the comment following it. However a full heuristic is beyond the scope of this paper. Using Algorithm 2, we can actually compute $f_{t'}$ for all terms $t'|t$ at negligible additional cost, similar to the use of Moebius transform for efficiency in the binary cube attack. Other optimisations and variations have been proposed for the binary cube attack; many of them could potentially be transferred to the modulo p case, but again, this is beyond the scope of this paper.

Online phase

1. For each $(t, c_0, c_1, \dots, c_n)$ stored in the preprocessing phase, compute $f_t(\mathbf{0}, \mathbf{x})$ (with $\mathbf{x}$ being now unknown) using formula (5) in Proposition 7. Form the linear equation: $c_1 x_1 + \cdots, +c_n x_n + c_0 = f_t(\mathbf{0}, \mathbf{x})$.
2. Solve the system of linear equations thus obtained, determining the secret variables $x_1, \dots, x_n$. If the preprocessing phase only produced $r < n$ equations, we would need to do an exhaustive search in the $(n-r)$-dimensional space of solutions.

Remark 6 Let ℓ be the length of the binary representation of p. We can view each bit in the binary representation of an element in GF(p) as one binary variable. If f is a function of n variables over GF(p), we can also view it as ℓ binary functions $f^{(0)}, \dots, f^{(\ell-1)}$ in ℓn binary variables. We could therefore apply the classical (binary) cube attack on each of these functions $f^{(j)}$. A rough estimate suggests that the running time for corresponding cubes will be approximately the same. (Differentiating $p-1$ times with respect to one variable x_i in GF(p) takes p evaluations of f; differentiating once w.r.t. each of the binary variables that are components of x_i will take 2^ℓ evaluations of f; we have $p \approx 2^\ell$.) The chances of success on a particular cube bear no easy relationship between the two approaches, because the degree of f and the degrees of the ℓ binary functions are not related in a simple way.

Hence we would argue that in general one cannot tell which of the attacks will work better, so one should try both. If the cipher has a structure that would suggest that the degree as

polynomial over GF(p) is relatively low, then a cube attack over GF(p) should certainly be an approach to consider.

An example of a situation when the cube attack modulo p is much more efficient than a binary cube attack is presented here:

Example 4 Consider $f(v,x) = v^3 + v^2x + vx^2 + x^3 + v^2 + vx + x^2 + v + x + 1$, a function in one public variable v and one secret variable x over the field GF(p).

Computing the first and second derivative w.r.t. the public variable v we obtain:

$$f_v(v,x) = 3v^2 + 2vx + x^2 + 5v + 2x + 3$$
$$f_{v^2}(v,x) = 6v + 2x + 8.$$

Evaluating at $v = 0$ we have:

$$f_v(0,x) = x^2 + 2x + 3$$
$$f_{v^2}(0,x) = 2x + 8$$

so the second derivative w.r.t. v is linear in the secret variable. Considering f as a black box function, our proposed attack over GF(p) is successful for second order differentiation w.r.t. v (i.e. choosing $t = v^2$ at point 1. in the preprocessing phase described above), regardless of the value of p. Also, each evaluation of the derivative only requires 4 calls to f, so the algorithm is very efficient. (Note that the previously existing attack over GF(p), as described in Sect. 3.2, would not be successful as it would only compute the first derivative, which is not linear in the secret variable).

We have also applied the binary cube attack to this function for $p = 2^\ell - 1$ with $\ell = 19$ and $\ell = 13$. To apply the binary attack, we view f as ℓ Boolean functions $f^{(0)}, \ldots, f^{(\ell-1)}$ (corresponding to the ℓ-bit output of f), each being a function in ℓ binary public variables and ℓ binary secret variables (each of these variables being one bit in the binary representation of v and of x respectively).

For $p = 2^{19} - 1$ we used the binary Moebius transform as described in [11] to examine simultaneously all the subcubes of the cube of size 2^ℓ corresponding to choosing all the public variables. That means we performed the cube attack for all possible cubes. We used the technique proposed in [23] to test linearity and also obtain an estimate of the degree of each f_I. We used 6 values for the secret variable x (including the all-zero one) and their linear combinations. None of the cubes produced an f_I of degree one (and none of degree two either), so the binary cube attack fails.

We also considered the same function for $p = 2^{13} - 1$. This case is sufficiently small to be able to compute the full ANF of each of the 13 functions $f^{(0)}, \ldots, f^{(12)}$ in 26 binary variables (13 public and 13 secret variables) using the binary Moebius transform algorithm. Having the explicit representation of the function means we can then write down for each set I of cube indices the explicit expression for $f^{(j)}_{S(I)}(\mathbf{0}, \mathbf{x})$ and examine its degree and density. There were no results of degree one, which confirms the findings for $p = 2^{19} - 1$. There is a very small number of constant results and all the remaining ones are of degree 12 or 13 (13 being the maximum possible degree in 13 variables). The density is in most cases close to 0.5 i.e. about half of the possible terms appear in the polynomial. So from this point of view each $f^{(j)}$ behaves like a random polynomial.

We also considered a few other functions f of the same degree and number of variables as the one above, and also a few other primes. All results were similar to the situation above, i.e. the binary cube attack failed.

These results are not too surprising, as it is well known that multiplication modulo p of two numbers a and b, when expressing each output bit as a binary polynomial in the bits of a and b, yields very large degrees. Indeed, this is one of the reasons why many encryption algorithms use operations over several different fields/rings to prevent algebraic attacks.

One can expand this example by constructing a full cipher along the lines of the cipher constructed in [5, Section 6], but this time over GF(p) i.e. an LFSR over GF(p) followed by SBoxes with inputs in GF(p) and the output represented as a polynomial of moderate degree over GF(p). We considered such an example, and, as expected, the attack modulo p works quite efficiently, whereas for the binary attack we could not find any linear results.

We also attempted running the proposed attack modulo $2^{31} - 1$ on simplified versions of the stream cipher ZUC, but this was unsuccessful. We suspect the main reason is that in ZUC operations modulo $2^{31} - 1$ in the LFSR are followed by a non-linear filtering function (containing addition modulo 2^{32}, Sboxes, etc), which successfully increases beyond reach the degree of the output when viewed as polynomial over GF($2^{31} - 1$).

5 Generalisations to GF(p^s)

In this section we take our generalisation further, to arbitrary finite fields GF(p^s). An important particular case would be GF(2^s), as many cryptographic algorithms (e.g. SNOW, AES) include operations over a field of this type.

5.1 Preliminaries

We need some known results regarding the values of binomial coefficients and multinomial coefficients in fields of finite characteristic.

Theorem 8 (Kummer's Theorem, [15, p. 115]) *Let $n \geq k \geq 0$ be integers and p a prime. Let j be the highest exponent for which $\binom{n}{k}$ is divisible by p^j. Then j equals the sum of carries when adding k and $n - k$ as numbers written in base p.*

Kummer's theorem has been generalised to multinomial coefficients by various authors (see for example [6] and citations therein).

Theorem 9 *Let $d, k_1, k_2, \ldots, k_m$ be integers such that $\sum_{i=1}^{m} k_i = d$ and $k_i \geq 0$ and let p be a prime. Let j be the highest exponent for which $\binom{d}{k_1,k_2,\ldots,k_m}$ is divisible by p^j. Then j equals the sum of all the carries when adding all of $k_1, k_2, \ldots, k_m$ as numbers written in base p.*

We will be interested in the situations where the multinomial coefficients are not zero modulo p.

Corollary 1 *Let p be a prime. The following are equivalent:*

(i) *The multinomial coefficient $\binom{d}{k_1,k_2,\ldots,k_m}$ is not zero modulo p.*
(ii) *There are no carries when adding $k_1, k_2, \ldots, k_m$ as numbers written in base p.*
(iii) *In base p, each digit of d equals to the sum of the digits of $k_1, k_2, \ldots, k_m$ in the corresponding position.*

5.2 Differentiation in GF(p^s)

When moving from GF(p) to GF(p^s) several things work differently. For a start, there are monomials for which differentiating once w.r.t. a variable x decreases the degree in x by

more than one, regardless of the difference step. For example let us differentiate x^d once, obtaining $(x+h)^d - x^d$. The coefficient of x^{d-1} is dh, so when d is a multiple of p the degree is strictly less than $d-1$. We examine the general case:

Theorem 10 *Let $f : \mathrm{GF}(p^s) \to \mathrm{GF}(p^s)$, $f(x) = x^d$, $d < p^s$. Let $0 < m \leq s(p-1)$ and let $h_1, \ldots, h_m \in \mathrm{GF}(p^s) \setminus \{0\}$. The degree of $\Delta^{(m)}_{h_1\mathbf{e_1},\ldots,h_m\mathbf{e_1}} x^d$ is less than or equal to the integer d' computed as follows: write d in base p as $d = d_u d_{u-1} \ldots d_1 d_0$; let i be the highest integer for which $d_0 + d_1 + \cdots + d_i \leq m$; define $d'_{i+1} = d_{i+1} - (m - (d_0 + d_1 + \cdots + d_i))$; finally define d' as the number written in base p as $d' = d_u d_{u-1} \ldots d_{i+2} d'_{i+1} 0 \ldots 0$ (with $i+1$ zeroes at the end).*

In particular, for $p = 2$, the binary representation of the degree d' is obtained from the binary representation of d by replacing m of its ones by zeroes, starting from the least significant bit.

Proof By Theorem 1 and using the fact that $\binom{d}{j}\binom{j}{k_1,\ldots,k_m} = \binom{d}{k_1,\ldots,k_m,d-j}$, the degree d' will be less than or equal to $d-j$ where j is minimal such that the set

$$\{(k_1,\ldots,k_m) | 1 \leq k_\ell \leq d, k_1 + \cdots + k_m = j, \text{ and } \binom{d}{k_1,\ldots,k_m,d-j} \not\equiv 0 \bmod p\}$$

is not empty. Using Corollary 1(iii) we see that the minimum value for j for given d and m is achieved by choosing $k_1, \ldots, k_m \geq 1$ as small as possible while maintaining $\binom{d}{k_1,\ldots,k_m,d-j}$ not equal to zero modulo p. This is achieved by choosing $k_1, \ldots k_m$ as follows: d_0 of them will be equal to 1, d_1 will be equal to p (i.e. 10 in base p), d_2 will be equal to p^2 (i.e. 100 in base p), …, d_i of them will be equal to p^i and finally $k - (d_0 + d_1 + \cdots + d_i)$ will be equal to p^{i+1}. It can be verified that $d' = d - (i_1 + \cdots + i_k)$ will then have the form described in the theorem statement. □

The previous theorem implies:

Corollary 2 *Let $f = \sum_{j=0}^{d} c_j x^j$ be a polynomial function in one variable over $\mathrm{GF}(p^s)$. Let $m = \max\{\mathrm{S}_p(j) | c_j \neq 0\}$, where $\mathrm{S}_p(a)$ denotes the sum of the digits in the base p representation of a. Then differentiating f a total of m times w.r.t. x will always produce a polynomial function which does not depend on x (possibly the identically zero function).*

Corollary 3 *Any function $f : \mathrm{GF}(p^s) \to \mathrm{GF}(p^s)$ can be differentiated at most $s(p-1)$ times w.r.t. a given variable before the result becomes the identically zero function.*

Is the bound in Corollary 2 tight, in the sense that there are functions which are non-zero after m differentiations? In particular, for Corollary 3, are there functions which are still non-zero after $s(p-1)$ differentiations? We will show that this is indeed the case, but only if we choose the h_i carefully. First let us illustrate a choice of the steps h_i which we need to avoid. By Eq. (2), if we differentiate p times with all steps equal to 1 the result is identically zero regardless of the original function f:

$$\begin{aligned}\Delta^{(p)}_{\mathbf{e_1},\ldots,\mathbf{e_1}} f(x_1,\ldots,x_n) &= \sum_{i=0}^{p} (-1)^{p-i} \binom{p}{i} f(x_1 + i, x_2, \ldots, x_n) \\ &= f(x_1 + p, x_2, \ldots, x_n) - f(x_1, x_2, \ldots, x_n) \\ &= 0\end{aligned}$$

because all the coefficients $\binom{p}{i}$ for $0 < i < p$ are divisible by p.

Denote by $b_0, \ldots, b_{s-1}$ a basis of $\mathrm{GF}(p^s)$ viewed as a s-dimensional vector space over $\mathrm{GF}(p)$. We choose the sequence $h_1, \ldots, h_{(p-1)s}$ as follows: $p-1$ values of b_0, followed by $p-1$ values of b_1 etc. For these choices of h_i Proposition 1 becomes:

Proposition 8 *Let $f : \mathrm{GF}(p^s) \to \mathrm{GF}(p^s)$ be a polynomial function and $m_1 \leq s(p-1)$. Write $m_1 = q(p-1) + r$ with $0 < r \leq p-1$ (note the slight difference from the usual quotient and remainder, as here the remainder can be $p-1$ but cannot be 0). Then*

$$\Delta^{(m_1)}_{h_1\mathbf{e_1},\ldots,h_{m_1}\mathbf{e_1}} f(x) = \sum_{a_0,\ldots,a_q} (-1)^{m_1-\sum_{i=0}^{q} a_i} \binom{p-1}{a_0} \cdots \binom{p-1}{a_{q-1}} \binom{r}{a_q} f\left(x + a_0 b_0 + \cdots + a_q b_q\right)$$

where the sum is over all tuples $(a_0, \ldots, a_q) \in \{0, \ldots, p-1\}^q \times \{0, \ldots, r\}$.

All the coefficients in the sum above are non-zero. If f is a "black box" function, one evaluation of f_t needs $p^q(r+1)$ evaluations of f.

Proof Similar to the proof of Proposition 7. □

We show next that our choice of h_i is indeed a valid choice in the sense that there are functions which can be differentiated $m(p-1)$ times w.r.t. the same variable without becoming zero.

Proposition 9 *For each m_1 with $0 \leq m_1 \leq s(p-1)$ there is at least a univariate polynomial function $f : \mathrm{GF}(p^s) \to \mathrm{GF}(p^s)$ such that $\Delta^{(m_1)}_{h_1\mathbf{e_{i_1}},\ldots,h_{m_1}\mathbf{e_{i_1}}} f(x)$ is a non-zero constant.*

Proof We will construct f. Write $m_1 = q(p-1)+r$ with $0 < r \leq p-1$ as in Proposition 8. In the formula in Proposition 8 all the terms in the sum have a non-zero coefficient, and have distinct arguments for f. We will prescribe the values of f at these evaluation points and then interpolate f. Namely we will prescribe $f(a_0 b_0 + \cdots + a_q b_q) = 0$ for all $(a_0, \ldots, a_q) \in (\{0, \ldots, p-1\}^q \times \{0, \ldots, r\}) \setminus \{\mathbf{0}\}$, and $f(0) \neq 0$. The polynomial f is interpolated as

$$f(x) = \prod \left(x_1 - \left(a_0 b_0 + \cdots + a_q b_q\right)\right)$$

where the product is over all $(a_0, \ldots, a_q) \in (\{0, \ldots, p-1\}^q \times \{0, \ldots, r\}) \setminus \{\mathbf{0}\}$. It can be easily seen that $\deg_x(f) = (r+1)p^q - 1$, which, when written in base p consists of a digit of r followed by q digits of $p-1$. Hence in the notations of Corollary 2, $m = q(p-1)+r = m_1$.

On the other hand, denoting $g(x) = \Delta^{(m_1)}_{h_1\mathbf{e_{i_1}},\ldots,h_{m_1}\mathbf{e_{i_1}}} f(x)$ we know from Theorem 10 that g is a constant (possibly zero). However $g(0) = f(0) \neq 0$ by Proposition 8 and our choice of interpolation points. Hence g is a non-zero constant. □

Note that there are other possible valid choices of h_i, but we aimed to keep things simple computationally by using this particular choice.

Example 5 Consider $f(x_1) = x_1^5 \in \mathrm{GF}(9)[x_1]$. Hence $m = 3$ in Corollary 2. We define $h_1 = h_2 = 1$ and $h_3 = h_4 = \alpha$, where α is a primitive element of GF(9). We compute the third order derivative $\Delta^{(3)}_{\mathbf{e_1},\mathbf{e_1},\alpha\mathbf{e_1}} f(x_1)$ using either Proposition 1 or Theorem 1 and obtain:

$$\Delta^{(3)}_{\mathbf{e_1},\mathbf{e_1},\alpha\mathbf{e_1}} f(x_1) = 2\alpha^3 + \alpha = 2\alpha(\alpha+1)(\alpha+2) \neq 0.$$

Generalising the results of this section to differentiation w.r.t. several variables is not difficult but the notation becomes cumbersome. Moreover, we will see in the next subsection that such a generalisation is not very useful for a practical attack.

5.3 Reducing differentiation over GF(p^s) to differentiation over GF(p)

Fix a basis $b_0, \ldots, b_{s-1} \in \mathrm{GF}(p^s)$ for $\mathrm{GF}(p^s)$ viewed as an s-dimensional vector space over $\mathrm{GF}(p)$. Any element $a \in \mathrm{GF}(p^s)$ can be uniquely written as $a = a_0 b_0 + \cdots + a_{s-1} b_{s-1}$ with $a_i \in \mathrm{GF}(p)$. Denote by $\varphi : \mathrm{GF}(p^s) \to \mathrm{GF}(p)^s$ the vector space isomorphism defined by $\varphi(a) = (a_0, \ldots, a_{s-1})$; this can be naturally extended to $\varphi : \mathrm{GF}(p^s)^n \to \mathrm{GF}(p)^{sn}$; denote by $\pi_j : \mathrm{GF}(p^s) \to \mathrm{GF}(p)$ the s projection homomorphisms defined as $\pi_j(a) = a_j$.

Let $f : \mathrm{GF}(p^s)^n \to \mathrm{GF}(p^s)$ be a polynomial function in n variables $x_1, \ldots, x_n$. By writing $x_i = x_{i0} b_0 + \cdots + x_{i,s-1} b_{s-1}$ the function f can be alternatively viewed as a function $\overline{f} : \mathrm{GF}(p)^{sn} \to \mathrm{GF}(p)^s$ defined by $\overline{f} = \varphi \circ f \circ \varphi^{-1}$:

$$\begin{array}{ccc} \mathrm{GF}(p^s)^n & \xrightarrow{f} & \mathrm{GF}(p^s) \\ \varphi \downarrow & & \downarrow \varphi \\ \mathrm{GF}(p)^{sn} & \xrightarrow{\overline{f}} & \mathrm{GF}(p)^s \end{array}$$

Alternatively we can view f as s polynomial (projection) functions $\overline{f}_0, \ldots, \overline{f}_{s-1} : \mathrm{GF}(p)^{sn} \to \mathrm{GF}(p)$, defined by $\overline{f}_i = \pi_i \circ f \circ \varphi^{-1}$ each in sn variables x_{ij} with $i = 1, \ldots, n$ and $j = 0, \ldots, s-1$.

Due to the linearity of the differentiation operator and the fact that φ is an isomorphism, the following diagram commutes:

$$\begin{array}{ccc} \mathcal{F}(\mathrm{GF}(p^s)^n, \mathrm{GF}(p^s)) & \xrightarrow{\Delta_{\mathbf{a}}} & \mathcal{F}(\mathrm{GF}(p^s)^n, \mathrm{GF}(p^s)) \\ \Phi \downarrow & & \downarrow \Phi \\ \mathcal{F}(\mathrm{GF}(p)^{sn}, \mathrm{GF}(p)^s) & \xrightarrow{\Delta_{\varphi(\mathbf{a})}} & \mathcal{F}(\mathrm{GF}(p)^{sn}, \mathrm{GF}(p)^s) \end{array}$$

where $\mathcal{F}(A, B)$ denotes the set of functions from A to B, for any sets A and B, and Φ is the operator defined as $\Phi(f) = \overline{f}$. For our purposes we will be interested in differentiations w.r.t. one variable. In the next theorem, since we are differentiating functions in a different number of variables, we will use for the canonical basis vectors the notation $\mathbf{e}_i^{(n)}$ instead of $\mathbf{e}_i$ to clarify the length n of the vector.

Theorem 11 *Let $f : \mathrm{GF}(p^s)^n \to \mathrm{GF}(p^s)$ be a polynomial function and let $r_0, \ldots, r_{s-1}$ with $0 \le r_i \le p-1$.*

$$\overline{\Delta^{(r_0)}_{b_0\mathbf{e_1^{(n)}}, \ldots, b_0\mathbf{e_1^{(n)}}} \cdots \Delta^{(r_{s-1})}_{b_{s-1}\mathbf{e_1^{(n)}}, \ldots, b_{s-1}\mathbf{e_1^{(n)}}} f} = \Delta^{(r_0)}_{\mathbf{e_1^{(sn)}}, \ldots, \mathbf{e_1^{(sn)}}} \cdots \Delta^{(r_{s-1})}_{\mathbf{e_{s-1}^{(sn)}}, \ldots, \mathbf{e_{s-1}^{(sn)}}} \overline{f}$$

the latter being a differentiation r_0 times w.r.t x_{10}, r_1 times w.r.t x_{11}, …, r_{s-1} times w.r.t. $x_{1,s-1}$.

Proof We use the fact that $\varphi(b_i \mathbf{e}_1^{(n)}) = \mathbf{e}_i^{(sn)}$ for all $i = 0, \ldots, s-1$ and we apply repeatedly the fact that the diagram above commutes. □

Remark 7 A cube attack for a polynomial function f over $\mathrm{GF}(p^s)$ can be developed based on the results in this section (generalised to several variables). However, by using Theorem 11, such an attack can be reduced to an attack in $\mathrm{GF}(p)$ on the polynomial functions $\overline{f}_0, \ldots, \overline{f}_{s-1}$ simultaneously. In the cube attack we are looking to differentiate f so that the result is linear in the secret variables. A polynomial function f is linear iff all the polynomial functions $\overline{f}_0, \ldots, \overline{f}_{s-1}$ are linear. However, if we mount an attack in $\mathrm{GF}(p)$ on $\overline{f}_0, \ldots, \overline{f}_{s-1}$ individually (rather than an attack translated from the attack in $\mathrm{GF}(p^s)$) there are chances that some of the $\overline{f}_0, \ldots, \overline{f}_{s-1}$ are linear, even if not all of them are linear. This suggests that

for functions f over GF(p^s) an attack in GF(p) on each component independently is more promising than an attack in GF(p^s) on the whole function f. Therefore, in Sect. 4.4 we only described the attack over GF(p).

6 Conclusion

We examined higher order differentiation over integers modulo a prime p, as well as over general finite fields of p^s elements, proving a number of results applicable to cryptographic attacks, and in particular generalising the fundamental theorem on which the cube attack is based.

Using these results we proposed a generalisation of the cube attack to functions over the integers modulo p; the main difference from the binary case is that we can differentiate several times with respect to the same variable. Such an attack would be particularly suited to ciphers that use operations modulo p in their internal structure.

We also show that a further generalisation to general finite fields GF(p^s) is possible, but not as promising as the generalisation to GF(p), due to the fact that differentiation in GF(p^s) can be reduced to differentiation in GF(p).

Acknowledgments We would like to thank Dr Iain Phillips and Haochen Zhu for their help in the implementations discussed in Sect. 4.4.

References

1. Agnesse A., Pedicini M.: Cube attack in finite fields of higher order. In: Proceedings of the Ninth Australasian Information Security Conference—AISC '11, vol. 116, pp. 9–14 (2011).
2. Aumasson J.-P., Dinur I., Meier W., Shamir A.: Cube testers and key recovery attacks on reduced-round MD6 and Trivium. In: 16th International Workshop on Fast Software Encryption—FSE, pp. 1–22 (2009).
3. Biham E., Shamir A.: Differential cryptanalysis of DES-like cryptosystems. J. Cryptol. **4**(1), 3–72 (1991).
4. Daemen J., Govaerts R., Vandewalle J.: Block ciphers based on modular arithmetic. In: Wolfowicz W. (ed.) Proceedings of the 3rd Symposium on the State and Progress of Research in Cryptography, pp. 80–89. Fondazione Ugo Bordoni (1993).
5. Dinur I., Shamir A.: Cube attacks on tweakable black box polynomials. In: EUROCRYPT, pp. 278–299 (2009).
6. Dodd F., Peele R.: Some counting problems involving the multinomial expansion. Math. Mag. **64**(2), 115–122 (1991).
7. Duan M., Lai X.: Higher order differential cryptanalysis framework and its applications. In: International Conference on Information Science and Technology (ICIST), pp. 291–297 (2011).
8. Ekdahl P., Johansson T.: SNOW—a new stream cipher. In: Proceedings of the First NESSIE Workshop. Heverlee, Belgium (2000).
9. ETSI/SAGE: Specification of the 3GPP confidentiality and integrity algorithms 128-EEA3 & 128-EIA3. Document 2: ZUC specification. Technical Report 1.6, ETSI, (2011).
10. Filiol E.: A new statistical testing for symmetric ciphers and hash functions. In: Deng R., Bao F., Zhou J., Qing S. (eds.) Information and Communications Security. Lecture Notes in Computer Science, vol. 2513, pp. 342–353. Springer, Berlin (2002).
11. Fouque P.-A., Vannet T.: Improving key recovery to 784 and 799 rounds of Trivium using optimized cube attacks. In: 20th International Workshop on Fast Software Encryption—FSE, pp. 502–517 (2013).
12. Joux A.: Algorithmic Cryptanalysis, 1st edn. Chapman & Hall/CRC, Boca Raton (2009).

13. Kaufman T., Ron D.: Testing polynomials over general fields. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 413–422 (2004).
14. Knellwolf S., Meier W.: High order differential attacks on stream ciphers. Cryptogr. Commun. **4**(3–4), 203–215 (2012).
15. Kummer E.E.: Über die Ergänzungssätze zu den allgemeinen Reciprocitätsgesetzen J. Reine Angew. Math. **44**, 93–146 (1852).
16. Lai X.: Higher order derivatives and differential cryptanalysis. In: Blahut R.E., Costello D.J., Jr., Maurer U., Mittelholzer T. (eds.) Communications and Cryptography. The Springer International Series in Engineering and Computer Science, vol. 276, pp. 227–233. Springer, Berlin (1994).
17. Lai X., Massey J.L.: A proposal for a new block encryption standard. In: EUROCRYPT, pp. 389–404 (1990).
18. Lai X., Massey J.L., Murphy S.: Markov ciphers and differential cryptanalysis. In: EUROCRYPT, pp. 17–38 (1991).
19. MacWilliams F.J., Sloane N.J.A.: The Theory of Error-Correcting Codes. North Holland, Amsterdam (1977).
20. O'Neil S.: Algebraic structure defectoscopy. Cryptology ePrint Archive, Report 2007/378. http://eprint.iacr.org/ (2007).
21. Vielhaber M.: Breaking ONE.FIVIUM by AIDA an algebraic IV differential attack. Cryptology ePrint Archive, Report 2007/413. http://eprint.iacr.org/ (2007).
22. Vielhaber M.: AIDA breaks BIVIUM (A & B) in 1 minute dual core CPU time. Cryptology ePrint Archive, Report 2009/402. http://eprint.iacr.org/ (2009).
23. Winter R., Sălăgean A., Phan R.C.W.: Comparison of cube attacks over different vector spaces. In: Groth, J. (ed.) 15th IMA International Conference on Cryptography and Coding, IMACC. Lecture Notes in Computer Science, vol. 9496, pp. 225–238. Springer, Berlin (2015).