A correlation duet: Correlation attacks on correlation generators

Antoine Joux

CISPA Helmholtz Center for Information Security, Germany
joux@cispa.de

Abstract. Pseudo-random correlation generators based on the Quasi-Abelian syndrome decoding problem were first attacked in an article published at Asiacrypt 2025, using compressed sensing. In this paper, we revisit the security of the problem using a more traditional cryptanalytic tool, namely correlation attacks.
As a result, we get a new cryptanalysis which outperforms the attack from Asiacrypt 2025 in several directions. It allows recovery of secret error polynomials with larger Hamming weights, runs approximately 1 000 times faster and uses 1 000 times less memory over $\mathbb{F}_3$. Over $\mathbb{F}_4$, the speed-up and memory gain are even higher.
Due to this new attack, it becomes necessary to entirely revisit the parameters of several pseudo-random correlation generator proposals, including FOLEAGE.

1 Introduction

The use of the Quasi-Abelian syndrome decoding (QA-SD) problem for advanced cryptographic application originates in [3], where it is used a foundation to build pseudo-random correlation generators.

This problem has similarities with the LWE or LPN problems, but all computations involve multivariate polynomials in a polynomial ring of the form:

$$\mathcal{R}^{(n)} = \frac{\mathbb{F}[X_1, \cdots, X_n]}{\left(X_1^d - 1, \cdots, X_n^d - 1\right)},$$

where $\mathbb{F}$ is a finite field and d a fixed exponent, usually $d = |\mathbb{F}| - 1$. However, there can be exceptions to this choice of d, e.g. in [9], $\mathbb{F}$ is a large prime field and d is 2.

In $\mathcal{R}^{(n)}$, we define a subset $\mathcal{R}_0^{(n)}$ of *short* elements which contains all polynomials formed as a sum of a limited number of terms. Like $\mathbb{F}$ and d, this limit is a parameter of the QA-SD problem and we denote it by the letter t (for terms).

We now define the problem as follows:

Definition 1. *The search QA-SD problem consists, given a c-tuple $(a_0, a_1, \cdots, a_{c-1})$ of elements from $\mathcal{R}^{(n)}$ and a target element b in $\mathcal{R}^{(n)}$, in finding a c-tuple of short*

elements $(e_0, e_1, \cdots, e_{c-1})$ *in* $\mathcal{R}_0^{(n)}$*, such that:*

$$\sum_{i=0}^{c-1} e_i a_i = b.$$

While it is standard to restrict the definition to the case where $a_0 = 1$*, we include a general* a_0 *in our definition for the sake of symmetry.*

Remark 1. Here, we focus on the search version of QA-SD, but there also is a distinguishing version of QA-SD, where given a $(c+1)$-tuple $(a_0, a_1, \cdots, a_{c-1}, T)$, the taks is to distinguish between the case where T is uniformly random in $\mathcal{R}^{(n)}$ and the properly formed case (as above). It is shown in [3], that there is a search-to-decision reduction for QA-SD.

Our goal in this paper is to analyze the security of QA-SD using techniques derived from the analysis of classical stream ciphers and Boolean functions. For this reason, we start by analyzing the simple case $\mathbb{F} = \mathbb{F}_3$ and $d = 2$, where the link to Boolean functions is almost immediate. In later sections, we show how this can be extended to $\mathbb{F}_4$. Other small fields are accessible with the set of techniques used for $\mathbb{F}_4$, however, we have not do a complete analysis or an implementation for these bigger fields.

2 Preliminaries and state-of-the-art

The QA-SD problem can be viewed in different ways. A classical coding theory approach, taken in [2, Section 6.1] is to view it as the standard syndrome decoding problem for a code over $\mathbb{F}$ given by an equation:

$$(\mathbf{A_0}\ \mathbf{A_1}\ \cdots\ \mathbf{A_{c-1}}) \begin{pmatrix} \mathbf{e_0} \\ \vdots \\ \mathbf{e_{c-1}} \end{pmatrix} = \mathbf{b}$$

This code has special structure since the individual matrices $\mathbf{A_i}$ represent multiplication in $\mathcal{R}^{(n)}$ by the polynomial a_i. In addition, the error vector formed by concatenating the coefficients of the error polynomials has a split structure which somewhat improves combinatorial algorithms. Yet, as shown in [6, Theorem 1] the hardness of decoding with a split structure can be related to the hardness of usual decoding. Another noteworthy technique involving decoding of multiple syndromes was proposed in [12].

In addition, folding attacks, introduced in [3] and refined in [2] offer a way to reduce the dimensions of such structured codes. We do not describe them here and we introduce a simplified subclass of the folding attack that is useful for us in Section 4.4.

2.1 A polynomial evaluation view of QA-SD

We adopt here a view of QA-SD similar to [4]. Namely, we aim at reconstructing the noise polynomial e_0 from evaluations of the given instance.

Indeed, since the elements of $\mathcal{R}^{(n)}$ are polynomials, it is convenient to work with their evaluation at points. In fact, the cryptographic functionnalities based on QA-SD already rely on evaluation at points. More precisely, in order to be coherent with the modular reduction with $X_i^d - 1$ of these polynomials, the evaluations are only taken at points whose coordinates are d-th roots of unity in $\mathbb{F}$. We denote this set of roots of unity by $\mathbb{F}^{(d)}$ and note that when $d = |\mathbb{F}| - 1$ this reduces to $\mathbb{F}^{(d)} = \mathbb{F}^*$.

Definition 2. *Given a search QA-SD instance $(a_0, a_1, \cdots, a_{c-1}, b)$ and a point $P \in (\mathbb{F}^{(d)})^n$, we say that a candidate c-tuple $(s_0, s_1, \cdots, s_{c-1})$ is compatible with P when:*

$$\sum_{i=0}^{c-1} s_i(P) a_i(P) = b(P).$$

By extension, $(s_0, s_1, \cdots, s_{c-1})$ is said to be compatible with a set of points when it is compatible with all points in that set.

Let S_a be the set of all points in $(\mathbb{F}^{(d)})^n$ which are common roots of all the a_is except a_0. Equivalently, we have:

$$S_a = \{P \in (\mathbb{F}^{(d)})^n | a_0(P) \neq 0 \text{ and } \forall i \in [1 \cdots c-1] : a_i(P) = 0\}.$$

It is easy to see that for a solution $(e_0, e_1, \cdots, e_{c-1})$ of the QA-SD problem, every c-tuple of the form $(e_0, s_1, \cdots, s_{c-1})$ with arbitrary s_i is compatible with the set S_a. Thus, focusing on the set S_a reduces the QA-SD problem to recovering (a sparse) e_0 from its evaluations on S_a. Indeed, for each $P \in S_a$, we have $e_0(P) = T(P)/a_0(P)$.

This approach of recovering e_0 from evaluations has already been used in [4]. Let's remark that after finding e_0, we are left with another (easier) QA-SD instance with the parameter c decreased by 1. Thus, finding e_0 from its evaluation on S_a is a sound approach to completely solve QA-SD.

Several reconstruction techiques for e_0 are proposed in [4]. They start by remarking that the cases where e_0 has zero, one or two terms can be easily recognized. Then, they remark that the sparse e_0 can be decomposed into several polynomials with fewer variables using interpolations from groups of evaluation points. If we single out one variable, say X_n, we can write:

$$e_0 = \sum_{i=0}^{d-1} e_0^{(i)} X_n^i,$$

where each $e_0^{(i)}$ is a polynomial in $n-1$ variables. This clearly reduces the number of terms in the partial polynomials $e_0^{(i)}$, compared to e_0. However, it also prunes down the number of evaluation points. Indeed, their proposed technique

to obtain evaluations of the $e_0^{(i)}$ is to perform of single variable interpolation on groups of evaluation points $(x_1, \cdots, x_{n-1}, \star)$ where the $n-1$ first variables take fixed values and the last variable runs through all possibilities in $\mathbb{F}^{(d)}$. Applying this technique to sequentially remove a small number of variables, they greatly reduce the average number of terms in the partial polynomials compared to e_0. Furthermore, since the terms are split in a random way, fluctuations have the ability to create partial polynomials with a very small number of terms.

Finally, they introduce a new technique based on compressed sensing that can directly recover polynomials with up to 4 terms. Combined with the interpolation technique this allows them to break many of the proposed QA-SD instantiations.

Despite these impressive results, the compressed sensing approach has some drawbacks. First, the limit of four terms remains quite low and it would be useful to improve it. Second, each call to the compressed sensing algorithm is costly in terms of runtime and memory which limits the attack in practice. Finally, it is difficult to precisely predict the behavior of the compressed sensing algorithm when changing the parameters of the QA-SD problem.

In this paper, we propose an alternative technique that replaces the compressed sensing approach and precisely addresses these issues. It allows direct recovery of many more terms, has a faster implementation, uses less memory and its behavior is easier to predict.

3 Applying Boolean correlations to functions on $\mathbb{F}_3$

3.1 Defining related Boolean functions

Let f be any arbitrary function defined on a subset of $\mathbb{F}_3^n$, that only contains n-tuples without any zero coordinate. Since each element $\vec{X}$ of $\{-1,1\}^n$ can be written as $\vec{X} = (-1)^{\vec{x}}$, where $\vec{x}$ is a vector of $\mathbb{F}_2^n$, it seems natural to convert f into a Boolean function.

It turns out that several Boolean functions can be derived from f. A first approach is to simply identify $\{-1,1\}^n$ with $\mathbb{F}_2^n$ and view f as a function from $\mathbb{F}_2^n$ to $\mathbb{F}_3$. To get a Boolean function out of this, we can additionally restrict f to the set of vectors such that $f(x) \neq 0$, and convert the output to its binary form where $\{0,1\}$ in $\mathbb{F}_2$ respectively encode $\{1,-1\}$ in $\mathbb{F}_3$.

We keep the notation f for the original function over $\mathbb{F}_3$ and denote by f_2 its binary conversion restricted to non-zero outputs. The set of values with a well-defined f_2 is denoted by $S_2 \subset \mathbb{F}_2^n$. For every vector $\vec{x} \in S_2$, we have:

$$f((-1)^{\vec{x}}) = (-1)^{f_2(\vec{x})}.$$

In addition, as we indicated earlier, there are other Boolean functions that can be obtained from f. Indeed, for any $z \in \{0,1,-1\}$, we can define a function F_z as the indicator function of the set of values $x \in \{-1,1\}^n$ such that $f(x) = z$. Thus, $F_z(x) = 1$ when $f(x) = z$ and $F_z(x) = 0$ otherwise.

In terms of the multiplicative Boolean encoding, we restrict to the set $S_z \subset \mathbb{F}_2^n$ of points where $f(\vec{x}) = z$. On this set, F_z if the constant 1, i.e., the multiplicative encoding of the constant function $f_z = 0$.

The restricted function f_2 defined above can be seen as a combination of the information from F_1 and F_{-1}. Indeed, we have $S_2 = S_1 \cup S_{-1}$ and for each $\vec{x} \in S_2$:

$$(-1)^{f_2(\vec{x})} = F_1(\vec{x}) - F_{-1}(\vec{x}).$$

Because f_2 can be derived from $\{f_0, f_1, f_{-1}\}$, we mostly ignore it in the sequel and focus on the other three Boolean functions.

Remark 2. Initially, we constructed a correlation attack based on f_2 only. However, this attack is less satisfied that the version we are presenting. The main reason is that f_2 discards the information corresponding to the points where the original function vanishes. Of course, each individual f_z for $z \in \{0, 1, -1\}$ also discard information. However, our attack analyses the three functions together in order to keep track of all known evaluation points.

3.2 Correlations on Boolean functions

A standard approach in correlation attacks is to look for good approximations of a Boolean function by an affine function. A large body of theory has been developped to study and design Boolean functions for cryptographic applications with correlation attacks in mind, a good survey is Carlet's book [5].

We now want to use correlation techniques on our functions f_0, f_1, and f_{-1}. More precisely, given a function F_i taking values in $\{-1, 1\}$ and a Boolean function f_i defined over S_i that satisfies $(-1)^{f_i(\vec{x})} = F_i(\vec{x})$ for all $\vec{x} \in S_i$); we can measure the quality of the approximation of f_i by an affine function A using the correlation bias defined as:

$$c_A(f_i) = \frac{|\{\vec{x} \in S_i | f_i(\vec{x}) = A(\vec{x})\}| - |\{\vec{x} \in S_i | f_i(\vec{x})] \neq A(\vec{x})\}|}{|S_i|}.$$

We can remark that $c_A(f_i) = -c_{A+1}(f_i)$, which implies (the well-known fact) that it suffices to consider linear functions when effectively computing correlations.

Let L be the linear function given by $L(\vec{x}) = \vec{\ell} \cdot \vec{x}$. In multiplicative notation, it can be interpreted as the evaluation of a monomial $M_{\vec{\ell}}$ on the vector $\vec{X} = (-1)^{\vec{x}}$, where $M_{\vec{\ell}}(\vec{X}) = \prod_{i=1}^{n} X_i^{\vec{\ell}_i}$. We see that:

$$\begin{aligned} c_L(f_i) &= \frac{1}{|S_i|} \sum_{\vec{x} \in S_i} (-1)^{\vec{\ell} \cdot \vec{x} + f_i(\vec{x})} = \frac{1}{|S_i|} \sum_{\vec{x} \in S_i} M_{\vec{\ell}}((-1)^{\vec{x}}) \, (-1)^{f_i(\vec{x})} \\ &= \frac{1}{|S_i|} \sum_{\vec{X} \in \{-1,1\}^n} M_{\vec{\ell}}(\vec{X}) F_i(\vec{X}). \end{aligned}$$

Changing the range of the summation in the last equality does not change the total sum since we are only adding points where $F_i(\vec{X}) = 0$. Note that this final

sum is not computed in a finite field but over the integers. In other words, using Carlet's terminology, we are reinterpreting the function F_i with values in $\{-1, 1\}$ as the pseudo-Boolean function associated to f_i.

When a function f_i has one (or several) good affine approximations, the bias of the corresponding linear function (without the constant term) is large in absolute value. It is well-known that such as bias can be estimated from the values of f_i on a (large enough) random subset. The number of sample points required for a good estimation can be derived from the analysis of (multivariate) hypergeometric distributions. In our case, the use of Hoeffding's inequality is sufficient, see Section 4.3.

Furthermore, for a function with n inputs evaluated on a sample of N points, we can clearly compute (the numerator) of each bias as a sum of N values. This basic approach costs $O(N)$ arithmetic operations for each individual bias. In fact, there exists an amortized way to compute all the correlation coefficients simultaneously using the Walsh transform algorithm. This simultaneous evaluation requires 2^n integer counters and runs in time $O(n\,2^n)$. Note that the theory of Walsh transforms has lots of applications to the study of Boolean functions, it is in particular covered in [5, Chapter 2]. In terms of implementation of our attack, we use the Walsh transform algorithm given in [7, Chapter 9].

In our use case, there is a minor difference with the standard application of the Walsh transform. Indeed, we apply the Walsh transform on a table which is initialized with values in $\{-1, 0, 1\}$ (or even $\{0, 1\}$ for indicator functions) instead of $\{-1, 1\}$ for full pseudo-Boolean functions. This comes from the fact that our Boolean functions are only defined on a subset of $\mathbb{F}_2^n$. Points out of the set of definition are coded by a 0 and do not contribute to any of the computed sums. This affects neither the efficiency nor the functionality of the algorithm. After running it, the table entry whose index has binary expansion $\vec{\ell}$ contains the numerator of the coefficient $c_{\vec{\ell}}$.

3.3 Theoretical correlation bias for sparse polynomials

In order to apply correlation techniques to recover sparse polynomials over $\mathbb{F}_3$, we need to know the good linear approximation of the associated Boolean functions. In this section, we give a theoretical analysis of the bias associated with an arbitrary linear approximation. More precisely, we consider an arbitrary sparse polynomial with t (non constant) monomials on n variables given by:

$$F_t(\vec{X}) = m_0 + \sum_{i=1}^{t} m_i M_{\vec{\ell_i}}(\vec{X}),$$

with coefficient vectors $(\vec{\ell}_1, \cdots, \vec{\ell}_t)$ and coefficients m_i in $\{-1, 1\}$. The extra coefficient m_0 is the contant term of the polynomial in $\{0, 1, -1\}$.

To simplify the computation, we assume that the coefficients vectors are linearly independent over $\mathbb{F}_2$. In that situation, we can complete $(\vec{\ell}_1, \cdots, \vec{\ell}_t)$ into a basis $(\vec{\ell}_1, \cdots, \vec{\ell}_n)$ of $\mathbb{F}_2^n$. This provides us with a linear change of variables in the

exponents defined by $\vec{y_i} = \vec{\ell_i} \cdot \vec{x}$. For convenience, we also want a multiplicative change of variables. To do that, we first extend the scalar coefficients by the convention $m_i = 1$ when $i > t$. This allows us to define the multiplicative change of variables:

$$\vec{Y_i} = m_i \prod_{j=1}^{n} \vec{X_i}^{\ell_{i,j}},$$

which bijectively maps $\{-1, 1\}^n$ to itself.

As a consequence, to understand the bias in the linearly independent case, we just need to compute correlations for pure sums:

$$m_0 + \vec{Y_1} + \cdots + \vec{Y_t}.$$

Remark 3. For the linearly dependent case, the situation appears to be much more difficult to analyze fully. However, in our attack, we can approximately model the linearly dependent case as a linearly independent case where the sample are selected in a linear subspace. Together with the heuristic assumption that this dependent sampling does not utterly destroy the ordering of the biases, recovery remains possible. Thus, and thanks to the fact that a candidate solution can be easily tested, knowing the biases in the linearly independent case is enough to firmly ground the principles of our cryptanalysis. Our experiments have also confirmed that the attack still works in the linearly dependent case.

In the case of pure sums without a constant term, i.e. $m_0 = 0$, we have the following theorem.

Theorem 1. *In the ring $\mathbb{F}_3[Y_1, \cdots, Y_n]$, let F_i be the indicator function of the event where the sum of the first t variables is equal to i. We denote by $c_L^{(i)}(Y_1 + \cdots + Y_t)$ the correlation coefficient of F_i with the monomial $\vec{Y}^{\vec{\ell}}$, where $\vec{\ell}$ is the coefficient vector associated to the non-zero linear form L, i.e. $L(\vec{y}) = \vec{\ell} \cdot \vec{y}$. The values of these coefficients are:*

- $c_L^{(i)}(Y_1 + \cdots + Y_t) = 0$, *if $\vec{\ell}$ contains a 1 outside of the positions $1 \cdots t$.*
- *When $\vec{\ell}$ contains s coordinates among the positions $1 \cdots t$ (and none outside), we have two cases. First, when s is odd:*

$$\begin{aligned} c_L^{(0)}(Y_1 + \cdots + Y_t) &= 0, \\ c_L^{(1)}(Y_1 + \cdots + Y_t) &= \frac{(-1)^{t-(s+1)/2} \cdot 3^{(s+1)/2}}{2^t - (-1)^t} \text{ and} \\ c_L^{(-1)}(Y_1 + \cdots + Y_t) &= -c_s^{(1)}(Y_1 + \cdots + Y_t). \end{aligned}$$

- *Second for even $s > 0$, we have:*

$$\begin{aligned} c_L^{(0)}(Y_1 + \cdots + Y_t) &= \frac{2 \cdot (-1)^{t-s/2} \cdot 3^{s/2}}{2^t + 2\,(-1)^t} \text{ and} \\ c_L^{(\pm 1)}(Y_1 + \cdots + Y_t) &= \frac{(-1)^{t+1-s/2} \cdot 3^{s/2}}{2^t - (-1)^t}. \end{aligned}$$

The proof is given in Section 6. This theorem implies:

Corollary 1. *If F_t is a sparse polynomial*

$$F_t(\vec{X}) = m_0 + \sum_{i=1}^{t} m_i M_{\vec{\ell_i}}((\vec{X})),$$

containing t non-constant monomials whose exponent vectors are linearly independent modulo 2, then for any subset $S \subset [1..t]$ of cardinality $s \neq 0$, the correlation coefficients $c^{(i)}_{\oplus_{i\in S}\vec{\ell_i}}$ are, up to signs and rotation, identical to the coefficients given in Theorem 1 for the product of the s first variables.

Furthermore, the correlation coefficients for non-constant monomials which are not of this restricted form are all zero.

Our strategy to solve QA-SD is based on recognizing the correlation patterns of the highly correlated monomials and then deriving back the monomials of F_t from them. It hinges on the remark that the product of the t non contant monomials of F_t has the highest bias and that the products of $t-1$ such monomials also have large (but slightly smaller) biases.

4 Solving QA-SD for polynomials over $\mathbb{F}_3$

With the tools from the previous Section, we can now describe our general strategy as follows:

- First, obtain evaluation samples for the polynomial e_0 of the QA-SD instance. Split them into three classes depending on the output value 0, 1 or -1. These classes respectively contain N_0, N_1 and N_{-1} samples.
- Second, run the Walsh transform algorithm to estimate the correlation biases of the three indicator functions of these samples for all possible monomials.
- Third, identify the most correlated monomial together with a list of highly correlated monomials. This is used to derive a list of candidate monomials for e_0.
- Finally, knowing candidate monomials for e_0 use linear algebra to reconstruct the corresponding coefficients. Any extraneous candidates in the list are removed at this point, since their coefficients become zero. Note that this only works if the number of candidate monomials we obtain is smaller than the number of evaluation points.

This basic approach to reconstructing e_0 requires that the product of t is correctly identified and that all products of $t-1$ are included in the lift of highly correlated monomials. Even though our general strategy can be adapted to larger field, the third step we use here is specific to $\mathbb{F}_3$. We describe it in more details in the next Section.

4.1 Elementary reconstruction of the monomials of e_0

For this simple case, we assume as in Theorem 1 that the exponents of the non-constant monomials of e_0 are linearly independent and that their number t is sufficiently small. During the sampling step, we learn the value of e_0 on N points (out of 2^n). These evaluation points split into three classes, depending on the output being 0, 1 or -1. These classes respectively contain N_0, N_1 and N_{-1} points (out of approximately $2^n/3$), with $N = N_0 + N_1 + N_{-1}$.

For each monomial, we thus observe three sums (via a Walsh transform computation) and want to use them to find highly correlated monomials.

We first focus on recovering the product of the t monomials of e_0. Thanks to Theorem 1, we know that when t is even the corresponding biaises are approximately $(\pm 2\varepsilon, \mp\varepsilon, \mp\varepsilon)$, up to rotation, with $\varepsilon = (3/4)^{t/2}$. These values are approximate and come from replacing the denominators by by 2^t instead of $2^t \pm 1$ or $2^t \pm 2$. When t is odd, we instead have biases of the form $(0, \pm 3\varepsilon, \mp 3\varepsilon)$, up to rotation, with $\varepsilon = 3^{(t-1)/2}/2^t$. Furthermore, all other monomials have comparatively smaller biases.

To find this highly correlated monomial, we look for the maximum of:

$$\left|\sum_{\vec{x}\in S_0} M_{\vec{\ell}}((-1)^{\vec{x}})\,(-1)^{f_0(\vec{x})}\right| + \left|\sum_{\vec{x}\in S_1} M_{\vec{\ell}}((-1)^{\vec{x}})\,(-1)^{f_1(\vec{x})}\right| + \left|\sum_{\vec{x}\in S_{-1}} M_{\vec{\ell}}((-1)^{\vec{x}})\,(-1)^{f_{-1}(\vec{x})}\right|$$

among all possible monomials $M_{\vec{\ell}}$.

It is natural to expect this to give us the product of the t (non-constant) monomials of e_0. We back this up with the experiments of Section 4.2 and the theoretical justification of Section 4.3.

Once this maximum is identified, we analyse the triple bias pattern to identify the proper rotation and the parity of t. We know know that the product of $t-1$ monomials have biases (up to the known rotation) of the form $(0, \pm\varepsilon, \mp\varepsilon)$, when t is even and $(\pm 2\varepsilon, \mp\varepsilon, \mp\varepsilon)$, when t is odd.

To identify these highly correlated monomials, in the even case, we compute the difference of the two Walsh sums with biases $\pm\varepsilon$. In the odd case, we take the Walsh sum corresponding to bias $\pm 2\varepsilon$ and substract the other two Walsh sums from it. After that, we collect a list of positions which maximize the absolute values of the composed Walsh sums we just created.

The attack fully succeeds when all the products of $t-1$ monomials for e_0 appear in the list.

4.2 Experiments on $\mathbb{F}_3$

In this section, we report experiments of our correlation attack for QA-SD over $\mathbb{F}_3$. These experiments were performed on an a single core of an AMD EPYC 9374F server running at 3.85GHz. The parameters affecting the efficiency of the recovery technique are the number of (non-constant) terms t, the number of variables n and the number of available sample points N. For each set of parameters, we performed 100 runs and we report success according to five different

measures. By increasing of quality, the first level is recovering the product of all monomials as the maximally biased entry. The second level is to correctly estimate the rotation of the bias pattern. The third level is to recover at least one product of $t-1$ monomials in our list of highly correlated monomials for this pattern. The fourth level is recovery of half of the products of $t-1$ monomials. Finally, the fifth level is total success, recovering everything we need to interpolate e_0. We estimate the success rates for a list of 100 maximum values. This choice and the definition of the levels of success are somewhat arbitrary but reflect the accessible instances when more optimization are applied (see Section 4.4). In addition, we also report the required memory and average runtime for our tests. Note that the runtime excludes the cost of generating the instances and there samples, it counts on the recovery algorithm alone (without the final linear algebra reconstruction).

The concrete instances of QA-SD for $\mathbb{F}_3$ all come from [3, Table 1]. These instances have a number of variables $n = 25, 30$ or 35. There are two families, one with $c = 2$ and total error weight 152, the other with $c = 4$ and total error weight 64. This corresponds to a number of monomials in e_0 respectively equal to 76 or 16. The overall number of evaluation points available for e_0 are approximately $2^n/3$ in the first case and $2^n/27$ in the second. These instances are reported as, at least partially or theoretically broken in [4, Section 4.1]. For the first parameter set, they do that by interpolation on three variables, this reduces the average weight of e_0 to 9.5 and the number of available points to $\approx 2^{n-3}/3^8$, i.e. above 600 for $n = 25$. They rely on random fluctuations to get a reduced e_0 with no more than 4 monomials. For the second parameter set, they interpolate on two variables, this lowers the average weight to 4 and the number of evaluation points to $2^{n-2}/3^{12}$, while not enough with $n = 25$ this provides more than 500 points when $n = 30$.

For our tests, we explore regimes which are meaningfull for the parameters of [3, Table 1], with the smallest possible value of $n = 25$. Indeed, with larger n values, the number of evaluations is larger and recovery becomes more likely. We rediscuss larger values of n in Section 4.4. For $c = 2$, we consider the options of interpolation on three, two or a single variable. With three interpolated variables, we are, as in [4], left with 22 variables and about 600 evaluation points. In this setting, we easily recover partial polynomials with up to 8 terms with probability above 75% using out basic recovery technique. This is to be compared to the 4 terms achieved by [4]. Furthermore, our time and memory requirements are about 1 000 times smaller.

With two interpolated variables, we have 23 leftover variables and about 100 000 evaluation points. We easily recover partial polynomials with up to 24 (non-constant) terms. This is seems to be the overall best choice for the parameter set with $c = 2$. It is worth noticing that the attack works despite the fact that there are more terms than the number of independent variables.

For $c = 4$, interpolation on two variables for $n = 25$ only provides 15 evaluation point, which is definitely not enough, thus we consider interpolation on a single variable. This leads to 24 variables and about 23 000 points. In this

regime, variables, we would already easily attack up to 20 terms in the partial polynomial, which is way above the requirements for the parameters of [3, Table 1].

For all these configuration, we report the success rates on 100 experiments for a variety of value of t and points number in these regimes. The detailed tables containing the results are given in Section 8.

Remark 4. Note that our current implementation is limited to $n \leq 31$ due to the use of 32-bit words for the encoding of monomials. With $n = 31$, each run takes about 5 minutes and uses 48Gb of memory. If needed, it would be easy to modify the implementation to deal with slightly larger values of n.

4.3 Theoretical analysis based of Hoeffding's inequality

For our analysis, we need Hoeffding's inequality for sampling without replacement from [1, Proposition 1.2].

Theorem 2 (Hoeffding's inequality). *Let $\mathcal{X} = (x_1, \ldots, x_N)$ be a finite population of N (real) points and $X_1, \ldots, X_n$ be a random sample drawn without replacement from $\mathcal{X}$. Let $a = \min_{1 \leq i \leq N} x_i$ and $b = \max_{1 \leq i \leq N} x_i$, then, for all $\varepsilon > 0$:*

$$Pr\left(\frac{1}{n}\sum_{i=1}^{n} X_i - \mu \geq \varepsilon\right) \leq \exp\left(-\frac{2n\varepsilon^2}{(b-a)^2}\right),$$

where $\mu = \sum_{i=1}^{N} x_i/N$ is the mean of $\mathcal{X}$.

Corollary 2. *A standard and immediate corollary is that in addition:*

$$Pr\left(\frac{1}{n}\sum_{i=1}^{n} X_i - \mu \leq -\varepsilon\right) \leq \exp\left(-\frac{2n\varepsilon^2}{(b-a)^2}\right),$$

Remark 5. Note that since we are in a setting of sampling with replacement, Serfling's inequality would be an improvement on Hoeffding's. However, since we work in a regime where the number of samples is small compared to the total population, this improvement is minor and would not provide significant improvement. Also, the use of variable names in the theorem, i.e. n, N and ε, is different than in the rest of the paper.

Equiped with this Lemma, let us first lower bound the probability of recovering the maximally correlated monomial (the product of all t) from our observations. Since we are computing a sum of absolute values, Hoeffding's inequality does not really apply. However, we can simulate the sum of absolute values:

$$\left|\sum_{\vec{x}\in S_0} M_{\vec{\ell}}((-1)^{\vec{x}})\,(-1)^{f_0(\vec{x})}\right| + \left|\sum_{\vec{x}\in S_1} M_{\vec{\ell}}((-1)^{\vec{x}})\,(-1)^{f_1(\vec{x})}\right| + \left|\sum_{\vec{x}\in S_{-1}} M_{\vec{\ell}}((-1)^{\vec{x}})\,(-1)^{f_{-1}(\vec{x})}\right|$$

by guessing the right formula among the signed options:

$$\pm \sum_{\vec{x}\in S_0} M_{\vec{\ell}}((-1)^{\vec{x}})\,(-1)^{f_0(\vec{x})} \pm \sum_{\vec{x}\in S_1} M_{\vec{\ell}}((-1)^{\vec{x}})\,(-1)^{f_1(\vec{x})} \pm \sum_{\vec{x}\in S_{-1}} M_{\vec{\ell}}((-1)^{\vec{x}})\,(-1)^{f_{-1}(\vec{x})}.$$

This views our problem as a single bias reconstruction on the full poputation of 2^n points, using the sample set $S_0 \cup S_1 \cup S_{-1}$. Furthermore, adding together the predicted sums (and denominators) given in the proof of Theorem 1, we can check that the maximal composed bias we want to observe becomes exactly $(3/4)^{t/2-1}$ in the even-t case and $(3/4)^{(t-1)/2}$ in the odd case. All other biases are bounded away by a factor $1/2$ in the even case and $2/3$ in the odd case. Assuming knowledge of the proper signed sum, this becomes a classical problem of finding a planted biased value amond 2^n decoys. Using a standard analysis based on Hoeffding's inequality and the union bound it is easy to build a simple threshold distinguisher and to compute the number of samples needed to have a success rate above $1/2$. Namely, let ε_h be the high bias value, ε_l be our upper bound on the low bias values, and $N = N_0 + N_1 + N_{-1}$ be the total number of evaluation points. We fix a threshold N_{th} between the expected values $N_h = \varepsilon_h\, N$ and $N_l = \varepsilon_l\, N$ of the properly signed sum (out of the eight options). We also let ε_{th}i stand for N_{th}/N. For the good monomial, the probability that the observed sum is $\leq N_{\text{th}}$ is bounded by Hoeffding's inequality below $\exp(-2N(\varepsilon_h - \varepsilon_{\text{th}})^2)$. Similarly, for each bad monomial, the probability of being $\geq N_{\text{th}}$ is below $\exp(-2N(\varepsilon_{\text{th}} - \varepsilon_l)^2)$. By a union bound, the probability that there exists a bad monomial above the threshold can be upper bounded by multiplying by the number of potential bad monomials. Since we do not know the proper sign assignement for our sum, the number of decoys becomes $8 \cdot 2^n$ instead of the more standard 2^n. We thus get an upper bound of:

$$2^{n+3} \cdot \exp(-2N(\varepsilon_{\text{th}} - \varepsilon_l)^2).$$

To equate this with the probability of failure for the good monomial, we take:

$$N_{\text{th}} = \frac{(n+3)\log 2}{4(\varepsilon_h - \varepsilon_l)} + \frac{N_h + N_l}{2}.$$

To make sure that we succeed with probability at least $1/2$, we set:

$$\exp(-2N(\varepsilon_h - \varepsilon_{\text{th}})^2) = 1/4 \quad \text{i.e.} \quad N = \frac{\log 2}{(\varepsilon_h - \varepsilon_{\text{th}})^2}.$$

Replacing ε_{th} by its value give a quadratic equation in N:

$$\delta_\varepsilon^2\, N^2 - (n+4)\,\log 2\, N + \frac{(n\log 2)^2}{4\,\delta_\varepsilon},$$

where δ_ε stands for $\varepsilon_h - \varepsilon_l$. We thus get a theoretical lower bound on the required number of points of:

$$N \geq \frac{\left(n + 7 + 2\sqrt{2}\sqrt{n+5}\right)\,\log 2}{2\delta_\varepsilon^2}.$$

This theoretical bound could be optimized by remarking that the vast majority of the decoys are further away from the maximally biased value. However, this simplified analysis already backs the experimental findings. For example, for $t = 10$ and $n = 22$, we find a bound of 873, which is a slight overestimate compatible with our observations.

The main contribution in the number of points is δ_ε^{-2}. Thus, over $\mathbb{F}_3$, we expect the required number of points for a successful attack to grow proportionally to $(4/3)^t$ as t increases.

4.4 Optimizations of the reconstruction

As presented, our reconstruction technique as a time complexity of $\tilde{O}(2^n)$ and memory complexity 2^n independently of the value of t. This dependency is a clear disadvantage and would prevent the attack from scaling to many variables even with a very small t. Our first optimization removes this disadvantage. It relies on a special case of the folding technique proposed in [3]. This special case consists in evaluating at points whose coordinates are correlated, thus reducing the number of variables.

At its core, our folding approach works as follows: to reduce the number of variables by one, we pick a Boolean vector $\vec{\rho}$ and a coefficient $\alpha \in \{-1, 1\}$ to produce a reduced e'_0 on $n-1$ variables defined as:

$$e'_0(\vec{X}_1, \cdots, \vec{X}_{n-1}) = e_0(\vec{X}_1, \cdots, \vec{X}_{n-1}, \alpha \prod_{i=1}^{n-1} \vec{X}_i^{\vec{\rho_i}}).$$

To reduce by more than one variable, we simply iterate this process.

We easily see that any term of $e_0(\vec{X})$ is transformed in a term of the polynomial e'_0. As always with folding, this has the potential to reduce the number of terms since the multiplicative substitution is not bijective. However, the number of terms after one or more steps of variable reduction remains bounded by the number of terms in e_0. Another effect of the transformation is to reduce the number of evaluation points by an approximate factor of 2 for each variable we are removing. However, the number of evaluation points is comparable to what would have been available from the start on a comparable instance with the reduced value of n and the same parameters.

As a consequence, we can recover e'_0 using our basic attack. Furthermore, for any monomial of e'_0, there are only two monomials that can map to it under the substitution defined by $\vec{\rho}$. Thus, it is easy to interpolate e_0 using the evaluation point by using a linear algebra reconstruction on the $2t$ candidate monomials. This removes the bad preimages. We repeat this for each of the consecutive single variable reductions we have performed.

A second improvement is to remark that the partial successes we report in our tables can be leveraged upon to recover e_0 with a smaller number of evaluations then required for full success. For example, if we have only recovered some of products of $t-1$ monomials among many false positives, i.e. success levels

L3 or L4, we can test each of these candidate monomials simply. By dividing them from the product of all, we obtain a candidate monomial of e_0, by either adding or subtracting it, we can make it disappear from the modified e_0. The correct guess reduces the number of terms, while bad guesses increase it. The two cases can be distinguished by observing whether the modified product of all monomials is more or less correlated with the modified e_0. Similarly, this can also be attempted at success levels L1 and L2. However, we now need to test all of the 2^n monomials as candidates, thus increasing the runtime by a large factor. Since the basic attack already works for all concrete parameters proposed for $\mathbb{F}_3$, we have not implemented these refined techiques.

5 Extension to QA-SD over $\mathbb{F}_4$

The attack we have presented over $\mathbb{F}_3$ can be adapted to larger fields, in particular to $\mathbb{F}_4$. This case is particularly interesting since many parameters have been proposed, we sum them up in Table 1. In the same table, we also indicate the number of variables n_0 to fix in order to lower the expected number of terms t' below the number of leftover variables and the number of remaining evaluation points. Note that this expected number of terms does not account for folding and does not take advantage of the fact that the weight of the sub-polynomials varies. This variation is of huge importance in [4], since they rely of the fact that by trying many possibilities for variables that are fixed it is possible to get sub-polynomials with extremely low weight.

In Table 1, we mark in boldface the number of points which are sufficient for a direct application of our attack, without relying on probabilistic weight variations. This includes the aggressive FOLEAGE parameters with $c = 3$ and at least 17 variables. With fewer variables, we need to rely on some probabilistic sparsification. With $c = 3$ and $n = 16$, we have 3 500 evaluation points and can attack sub-polynomials when their weight falls to 8. With $c = 3$ and $n = 15$, the weight needs to fall to 7, and for $n = 14$, we need weight at most 5. For the conservative $c = 4$, with $n = 18$ the weight needs to fall to 5 and for $n = 17$ we need 4. As a consequence, all these parameters can be practically broken much faster than with the method of [4].

Table 1: $\mathbb{F}_4$-QA-SD params from [2, Table 5] and [10, Table 6/7].

t	c	n	n_0	t'	N
9	9	20	0	9	**53 000**
27	3	14/15/16/17/18/19	1	9	389/1167/3500/**10 500**/**31 000**/**94 000**
27	4	14/15/16/17/18	1	9	6/18/54/164/492
27	5	20/21	1	9	69/207
33	4	18/19/20	1	11	492/1400/4400
43	2	18/19/20	1	14-15	**2M**/**6M**/**18M**
66	2	17/18/19/20	2	7-8	54/164/492/**4433**

t	c	n	n_0	t'	N
81	2	19	2	9	492

To adapt our technique to $\mathbb{F}_4$, the main steps are the following:

1. Extend the Boolean correlations and Walsh transform techniques to deal with a larger set of values.
2. Compute the theoretical correlation with monomials and understand the link with the terms of the sparse error polynomials.
3. Design a method to reconstruct the error polynomial from the correlated information.

The first of these steps requires p-ary Walsh transforms, which are less well-known than their Boolean counterpart. With this tool in hand, the second step is easily adapted from the $\mathbb{F}_3$ result. Somewhat surprisingly, the third step is the one that differs the most from the ternary case.

5.1 Redefining correlations for the case of $\mathbb{F}_4$

The analog of the Walsh transform modulo p is not as well-known as the Boolean case. For a theoretical definition, the reader can refer to [11, Chapter 13]. An earlier reference which refers to the Walsh transform simply as Fourier transform is [8]. For completeness, this section redefines what we need for our attack in the $\mathbb{F}_4$ case, namely the analog of the Walsh transform modulo 3.

As before, we start with a polynomial

$$F_t(\vec{X}) = m_0 + \sum_{i=1}^{t} m_i M_{\vec{\ell_i}}(\vec{(X)}),$$

containing t terms. The differences from the previous case are that the polynomial ring is now defined with an exponent $d = 3$, i.e. each variable X_i is computed modulo $X_i^3 - 1$ and that it takes values in $\mathbb{F}_4$. Thus, the vector of exponents $\vec{\ell}$ now belongs to $\mathbb{F}_3^n$. With the convention that $\mathbb{F}_4 = \{0, 1, \alpha, \alpha + 1\}$, we have four indicator functions F_0, F_1, F_α and $F_{1+\alpha}$ taking values 0 or 1. Again, this functions are constant (and equal to 1) on their respective sets S_0, S_1, S_α, and $S_{1+\alpha}$. As we did before, we would like to view it as a multiplicative encoding of functions f_0, f_1, f_α and $f_{1+\alpha}$ taking the constant 0 value in $\mathbb{F}_3$ on the respective set of definition (as before the encoding is the constant 1). Recalling that in the Boolean case, the multiplicative encoding of x is $(-1)^x$ with -1 a unit of order two, it is natural to encode x modulo 3 has J^x where $J = \exp(2i\Pi/3)$ is a unit of order 3.

With this, we can now define our replacement for the correlation biases as:

$$c_L(f_i) = \frac{1}{|S_i|} \sum_{\vec{x} \in S_i} J^{\vec{\ell} \cdot \vec{x} + f_i(\vec{x})} = \frac{1}{|S_i|} \sum_{\vec{x} \in S_i} M_{\vec{\ell}}(J^{\vec{x}}) \, (J)^{f_i(\vec{x})}.$$

To express $c_L(f_i)$ in terms of $\vec{X} \in \{1, \alpha, 1+\alpha\}^n$, we use the encoding E that maps $\{0, 1, \alpha, 1+\alpha\}$ to $\{0, 1, J, J^2\}$ to switch from elements in $\mathbb{F}_4$ to complex numbers. This encoding, which is also used in [4] allows us to rewrite:

$$c_L(f_i) = \frac{1}{|S_i|} \sum_{\vec{X} \in \{1,\alpha,1+\alpha\}^n} E\left(M_{\vec{\ell}}(\vec{X})\, F_i(\vec{X})\right).$$

As in the Boolean case, we changed the summation range without affecting the sum, since all extra terms are equal to zero.

The numerators of the $c_L(f_i)$ are the ternary equivalents of the Walsh sums. We easily compute all these sums simultaneously in quasi-linear time by a straightforward adaptation of the fast Walsh transform algorithm. We do not have an explicit reference for this adaptation, but it is folklore and is based on the following recursion step. Let G by a function of n variables and assume that we have already computed the generalized Walsh sums for the three partial functions on $n-1$ variables obtained by fixing the last variable. Let:

$$S_{L'}(G_i) = \sum_{\vec{X} \in \{1,\alpha,1+\alpha\}^{n-1}} E\left(M_{\vec{\ell'}}(\vec{X})\, G(\vec{X}, i)\right).$$

Then, we have:

$$\begin{aligned}
S_{L'\|0}(G) &= S_{L'}(G_1) + S_{L'}(G_\alpha) + S_{L'}(G_{1+\alpha}),\\
S_{L'\|1}(G) &= S_{L'}(G_1) + J\, S_{L'}(G_\alpha) + J^2\, S_{L'}(G_{1+\alpha})\\
S_{L'\|2}(G) &= S_{L'}(G_1) + J^2\, S_{L'}(G_\alpha) + J\, S_{L'}(G_{1+\alpha}).
\end{aligned}$$

The fast algorithm simply derives from repeated use of these relations. As in the Boolean case, the resulting algorithm also give the denominator $|S_i|$ in position 0 of the generalized Walsh transform.

Remark 6. When implementing the algorithm, we need to choose a representation of the complex numbers. One could choose the standard representation of a complex number by two floating point numbers representing the real and imaginary part. Instead, we remark that every complex number that arises during the computation belongs to the ring of integers of the number field $\mathbb{Q}[J]$. Since $1 + J + J^2 = 0$, we can represent each of them by a sum $aJ + bJ^2$, where a and b are integers.

5.2 Analysis of the correlated monomials over $\mathbb{F}_4$

As in the case of $\mathbb{F}_3$, when the coefficient vectors of the non constant monomials are linearly independent, we can transform the polynomial into a pure sum:

$$m_0 + \vec{Y}_1 + \cdots + \vec{Y}_t.$$

When $m_0 = 0$, we have the following adapted theorem.

Theorem 3. *In the ring $\mathbb{F}_4[Y_1, \cdots, Y_n]$, let F_i be the indicator function of the event where the sum of the first t variables is equal to i (in $\{0, 1, \alpha, 1+\alpha\}$). We denote by $c_L^{(i)}(Y_1 + \cdots + Y_t)$ the correlation coefficient of F_i with the monomial $\vec{Y}^{\vec{\ell}}$, where $\vec{\ell}$ is the coefficient vector associated to the non-zero linear form L, i.e. $L(\vec{y}) = \vec{\ell} \cdot \vec{y}$. The values of these coefficients are:*

- $c_L^{(i)}(Y_1 + \cdots + Y_t) = 0$, *if $\vec{\ell}$ contains a non-zero value outside of the positions $1 \cdots t$.*
- *When $\vec{\ell}$ contains s_1 coordinates equal to 1 and s_2 equal to 2 among the positions $1 \cdots t$ (and none outside), we have two cases. First, when $s_1 \not\equiv s_2 \pmod 3$:*

$$c_L^{(0)}(Y_1 + \cdots + Y_t) = 0 \text{ and}$$
$$c_L^{(i)}(Y_1 + \cdots + Y_t) = \frac{(-1)^{t+1-s_1-s_2} \cdot J^{(s_1-s_2)\log_\alpha i} \cdot 2^{s_1+s_2+1}}{3^t - (-1)^t} \text{ when } i \neq 0.$$

- *Second for $s_1 \equiv s_2 \pmod 3$, we have:*

$$c_L^{(0)}(Y_1 + \cdots + Y_t) = \frac{(-1)^{t-s_1-s_2} \cdot 2^{s_1+s_2}}{3^{t-1} + (-1)^t} \text{ and}$$
$$c_L^{(i)}(Y_1 + \cdots + Y_t) = \frac{(-1)^{t+1-s_1-s_2} \cdot 2^{s_1+s_2}}{3^t - (-1)^t} \text{ when } i \neq 0.$$

The proof is given in Section 7.

5.3 Recovering the terms from e_0

The main difference between the $\mathbb{F}_4$ situation and the $\mathbb{F}_3$ case is that there no longer is a single maximally correlated monomial. Instead, every monomial whose coefficient vector is a linear combination involving every term of e_0 is highly correlated. There are 2^t such monomials. Because of this high number, there is little hope to recover e_0 when the number of terms is higher then the number of variables. Another consequence is that we would like to recover the monomials of e_0 from these 2^t highly correlated monomials, without collecting monomials with lower correlations. We thus need a new method completely different from the one used in the $\mathbb{F}_3$ case.

Our basic strategy is to look for the 2^t maximal values of the sum of square modules of the complex Walsh sums:

$$\sum_{i \in \{0,1,\alpha,1+\alpha\}} \left| \sum_{\vec{x} \in S_i} M_{\vec{\ell}}(J^{\vec{x}}) \, (J)^{f_i(\vec{x})} \right|^2 .$$

With enough evaluation points, these 2^t values correspond to linear combination in coefficients $\{1, 2\}$ of the exponent vectors of the (non constant) monomials

in e_0. Let $\mathcal{S}$ be this set of 2^t vectors. In order to recover the monomials of e_0, we form the multiset $\mathcal{S}-\mathcal{S}$. This difference set contains 3^t distinct elements (all vectors in the subspace spanned by the exponents of the monomials of e_0) and counting multiplicities, it has 4^t elements. The multiplicity of each elements is the lever that allows us to recover e_0. Indeed, let:

$$\vec{L} = \sum_{i=1}^{t} \alpha_i \vec{\ell_i},$$

be a linear combination of the monomials of e_0. Let w be the Hamming weight of (α_i), i.e the number of non-zero coefficients in the expression of $\vec{L}$. Then, we see that $\vec{L}$ appears in $\mathcal{S}-\mathcal{S}$ with multiplicity 2^{t-w}. Since there are $2^w\binom{t}{w}$ linear possibilities with Hamming weight w, the overall size with multiplicity is as expected:

$$\sum_{w=0}^{t} 2^w \cdot \binom{t}{w} \cdot 2^{t-w} = 2^t \cdot \sum_{w=0}^{t} \binom{t}{w} = 4^t.$$

In total, $2t$ vectors of $\mathcal{S}-\mathcal{S}$ have multiplicity 2^{t-1}, these are $(\vec{\ell_i})$ and $(2\vec{\ell_i})$. After recovering them, a final linear algebra computation allows to recover the correct monomials together with the corresponding coefficients.

It turns out that studying the multiplicity in the multiset difference is a pretty robust strategy which still works if we do not fully recover $\mathcal{S}$ but instead a noisy version where some of the monomials are replaced by incorrect ones. In our implementation, we check for each monomial of e_0 whether it appears in the difference multiset with high multiplicity.

Remark 7. From an algorithm point of view, the multiplicities of each element in the multiset can be computed directly by forming all pairs, which requires times $O(4^t)$. However, for large t, this becomes the limiting factor. For this reason, it is useful to note that the multiplicities can be directly obtained by fast polynomial arithmetic. For this, we first remark that the valid set $\mathcal{S}$ is symmetric, i.e. that for every vector in $\mathcal{S}$ its opposite also lies in $\mathcal{S}$. The reconstructed $\mathcal{S}$ we have might be incorrect and not symmetric. Thus, we first expand our candidate $\mathcal{S}$ to make it symmetric. Then, we form the polynomial:

$$F_{\mathcal{S}} = \sum_{\vec{s}\in\mathcal{S}} \vec{X}^{\vec{s}}.$$

Then, we compute $F_{\mathcal{S}}^2$ in the complex polynomial ring where each variable is reduced modulo X^3-1 and remark that the coefficient of each monomial in this squared polynomial are the cardinalities we require.

We have implemented this attack and we report experiments in tables in Section 9. By constrast with the $\mathbb{F}_3$ we only report three levels of success. The first level is achieved when at least one monomial of e_0 is among the 100 highest multiplicities in the difference multiset that we create from highly correlated

monomials. The second level means that at least half of the monomials of e_0 are found. The highest is probability of full success, meaning that all monomials of e_0 appear with the 100 highest multiplicities.

Finally, it is important to note that the $\mathbb{F}_4$ recovery technique is also compatible with a reduction of a number of variables. The only difference is that when we add back variables one by one, each monomial from the projected polynomial can be lift in three different ways (instead of two). However, the important fact is that when there are a lot of evaluation points, we can lower the number of variables to speed-up the attack.

Compared to the code of [4] which could only run with up to 13 variables over $\mathbb{F}_4$, our code runs with up to 21 variables on our server[1]. However, as the number of variable grows, the running time increases. As a consequence, we only built tables of experiments for 13 and 14 variables, since it is enough for the FOLEAGE parameters. According to the analysis of Section 4.3 and the bias given in Theorem 3, we expect the number to required evaluation points to grow proportionally to $(9/4)^t$. Somewhat surprisingly, the growth of the number of evaluation points in the results of Section 9 is a bit slower. Furthermore, the attack works at least partially when t is marginally bigger than n.

6 Proof of Theorem 1

We recall the statement of the theorem.

Theorem (Theorem 1). *In the ring $\mathbb{F}_3[Y_1, \cdots, Y_n]$, let F_i be the indicator function of the event where the sum of the first t variables is equal to i. We denote by $c_L^{(i)}(Y_1 + \cdots + Y_t)$ the correlation coefficient of F_i with the monomial $\vec{Y}^{\vec{\ell}}$, where $\vec{\ell}$ is the coefficient vector associated to the non-zero linear form L, i.e. $L(\vec{y}) = \vec{\ell} \cdot \vec{y}$. The values of these coefficients are:*

- *$c_L^{(i)}(Y_1 + \cdots + Y_t) = 0$, if $\vec{\ell}$ contains a 1 outside of the positions $1 \cdots t$.*
- *When $\vec{\ell}$ contains s coordinates among the positions $1 \cdots t$ (and none outside), we have two cases. First, when s is odd:*

$$c_L^{(0)}(Y_1 + \cdots + Y_t) = 0,$$
$$c_L^{(1)}(Y_1 + \cdots + Y_t) = \frac{(-1)^{t-(s+1)/2} \cdot 3^{(s+1)/2}}{2^t - (-1)^t} \text{ and}$$
$$c_L^{(-1)}(Y_1 + \cdots + Y_t) = -c_s^{(1)}(Y_1 + \cdots + Y_t).$$

- *Second for even $s > 0$, we have:*

$$c_L^{(0)}(Y_1 + \cdots + Y_t) = \frac{2 \cdot (-1)^{t-s/2} \cdot 3^{s/2}}{2^t + 2\,(-1)^t} \text{ and}$$
$$c_L^{(\pm 1)}(Y_1 + \cdots + Y_t) = \frac{(-1)^{t+1-s/2} \cdot 3^{s/2}}{2^t - (-1)^t}.$$

[1] With 21 variables, we use 780Gb of memory and each run takes under 35 minutes. Potentially, it would be possible to add one more variable by taking care to only store a single Walsh table in memory at a time.

Proof. To compute $c_L^{(i)}(Y_1 + \cdots + Y_t)$, we need both their numerators and denominators. Let us start with the numerators:

$$\sum_{\substack{\vec{Y} \in \{-1,1\}^s \\ Y_1 + \cdots + Y_t \equiv i \pmod 3}} \vec{Y}^{\vec{\ell}}.$$

Remark that if $\vec{\ell}_j = 1$ for $j > t$, we can factor out the sum on $\vec{Y}_j$, thus deriving the first case with bias 0.

Thus, it suffices to study the case where $\vec{Y}^{\vec{\ell}}$ is a product of a subset of s of the variables $Y_1, \cdots, Y_t$, with s being the number of 1s in $\vec{\ell}$. Reordering the variables, we can assume that $\vec{\ell}$ starts with s values equal to 1, i.e. we restrict ourselves to the case $\vec{Y}^{\vec{\ell}} = \prod_{i=1}^{s} Y_i$.

To compute the sums in general, we note that the sum over the variables with indices in $[t+1 \cdots n]$ now simply contributes an overall factor of 2^{n-t}. We can safely ignore these indices larger than t for the moment.

We are reduced to computing the following sums:

$$P_i^{(s)} = \sum_{\substack{\vec{Y} \in \{-1,1\}^t \\ Y_1 + \cdots + Y_t \equiv i \pmod 3}} Y_1 \cdots Y_s.$$

It is convenient to regroup the three sums into the degree two polynomial $P^{(s)}(X) = P_0^{(s)} + P_1^{(s)} X + P_{-1}^{(s)} X^2$. To recover $P^{(s)}$, we just need its evaluations at three points. This is easily achieved if we remark that:

$$\Theta^3 = 1 \Rightarrow P^{(s)}(\Theta) = \sum_{\vec{Y} \in \{-1,1\}^t} \Theta^{Y_1 + \cdots + Y_t} \cdot Y_1 \cdots Y_s.$$

Using distributivity, we can write:

$$P^{(s)}(\Theta) = \prod_{j=1}^{s} \left(\sum_{Y_j \in \{-1,1\}} Y_j \Theta^{Y_j} \right) \cdot \prod_{j=s+1}^{t} \left(\sum_{Y_j \in \{-1,1\}} \Theta^{Y_j} \right) = (\Theta - 1/\Theta)^s \cdot (\Theta + 1/\Theta)^{t-s}.$$

It follows that $P^{(0)}(1) = 2^t$ and $P^{(s)}(1) = 0$, for $s > 0$. Furthermore, letting $J = e^{2I\Pi/3}$, we see that $P^{(s)}(J) = (-1)^{t-s} \left(J - J^2\right)^s$ and $P^{(s)}(J^2) = (-1)^{t-s} \left(J^2 - J\right)^s$. We can recover the individual values as follows:

$$\begin{aligned}
P_0^{(s)} &= \left(P^{(s)}(1) + P^{(s)}(J) + P^{(s)}(J^2) \right) /3, \\
P_1^{(s)} &= \left(P^{(s)}(1) + J^2\, P^{(s)}(J) + J\, P^{(s)}(J^2) \right) /3, \text{ and} \\
P_{-1}^{(s)} &= \left(P^{(s)}(1) + J\, P^{(s)}(J) + J^2\, P^{(s)}(J^2) \right) /3.
\end{aligned}$$

For $s = 0$, we see that $P_0^{(0)} = (2^t + 2\cdot(-1)^t)/3$ and $P_{\pm 1}^{(0)} = (2^t - (-1)^t)/3$. For odd s, we find $P_0^{(s)} = 0$, $P_1^{(s)} = (-1)^{t-(s+1)/2}\, 3^{(s-1)/2}$, and $P_{-1}^{(s)} = -P_1^{(s)}$. Finally, for even $s > 0$, we find $P_0^{(s)} = 2\cdot(-1)^{t-s/2}\, 3^{s/2-1}$ and $P_{\pm 1}^{(s)} = (-1)^{t+1-s/2}\, 3^{s/2-1}$.

We can also remark that the denominators of $c_L^{(i)}(Y_1 + \cdots + Y_t)$, are in fact the values $P_i^{(0)}$. Furthermore, the extra coefficient of 2^{n-t} that we mentionned earlier cancels out when considering the quotients $P_i^{(s)}/P_i^{(0)}$. Indeed, $P_i^{(0)}$ counts the number of vectors from $\{-1, 1\}^n$ for which i the evaluation of the sum of the first t coordinates is equal to i modulo 3. We have:

$$P_0^{(0)} = \frac{2^{n-t}\cdot\left(2^t + 2\,(-1)^t\right)}{3} \text{ and}$$
$$P_{\pm 1}^{(0)} = \frac{2^{n-t}\cdot\left(2^t - (-1)^t\right)}{3}$$

$$2^{n-t}\left(P_1^{(t)} + P_2^{(t)}\right) = \frac{2}{3}\cdot 2^{n-t}\cdot\left(2^t - (-1)^t\right),$$

which matches the intuition that each evaluation values should appears with proportion close to one third. Finally, denoting by $c_s^{(.)}$ the correlation coefficient for the Boolean linear form equal to the sum of the first s variables, we get the result that for odd s:

$$c_s^{(0)}(Y_1 + \cdots + Y_t) = 0,$$
$$c_s^{(1)}(Y_1 + \cdots + Y_t) = \frac{(-1)^{t-(s+1)/2}\cdot 3^{(s+1)/2}}{2^t - (-1)^t} \text{ and}$$
$$c_s^{(-1)}(Y_1 + \cdots + Y_t) = -c_s^{(1)}(Y_1 + \cdots + Y_t).$$

And for even $s > 0$, we have:

$$c_s^{(0)}(Y_1 + \cdots + Y_t) = \frac{2\cdot(-1)^{t-s/2}\cdot 3^{s/2}}{2^t + 2\,(-1)^t} \text{ and}$$
$$c_s^{(\pm 1)}(Y_1 + \cdots + Y_t) = \frac{(-1)^{t+1-s/2}\cdot 3^{s/2}}{2^t - (-1)^t}.$$

This concludes the proof.

7 Proof of Theorem 3

Proof. As in the previous proof, we start with the numerators:

$$\sum_{\substack{\vec{Y}\in\{1,\alpha,1+\alpha\}^s \\ Y_1+\cdots+Y_t \equiv i \text{ (in } \mathbb{F}_4)}} E\left(\vec{Y}^{\vec{\ell}}\right).$$

Remark that if $\vec{\ell}_j \neq 0$ for $j > t$, we can again factor out the sum on $\vec{Y}_j$, thus deriving the first case of bias 0.

Thus, it suffices to study the case where $\vec{Y}^{\vec{\ell}}$ contains s_1 variables with exponent 1 and s_2 with exponent 2 all with indices between 1 and t. After reordering the variables, we restrict ourselves to the case $\vec{Y}^{\vec{\ell}} = \prod_{i=1}^{s_1} Y_i \cdot \prod_{i=1}^{s_2} Y_{s_1+i}^2$.

To compute the sums in general, we note that the sum over the variables with indices in $[t+1 \cdots n]$ now contributes a factor of 3^{n-t}.

We are reduced to computing the following sums:

$$P_i^{(s_1,s_2)} = \sum_{\substack{\vec{Y} \in \{1,\alpha,1+\alpha\}^t \\ Y_1+\cdots+Y_t \equiv i \text{ (in } \mathbb{F}_4)}} E\left(Y_1 \cdots Y_{s_1} \cdot Y_{s_1+1}^2 \cdots Y_{s_1+s_2}^2\right).$$

It is now convenient to regroup the four sums into a bilinear polynomial $P^{(s)}(X,Y) = P_0^{(s_1,s_2)} + P_1^{(s_1,s_2)} X + P_\alpha^{(s_1,s_2)} Z + P_\alpha^{(s_1,s_2)} XZ$. To recover $P^{(s_1,s_2)}$, we just need its evaluations at four points. This is easily achieved if we remark that:

$$\Theta^2 = 1 \text{ and } \Xi^2 = 1 \Rightarrow$$
$$P^{(s_1,s_2)}(\Theta,\Xi) = \sum_{\vec{Y}} \Theta^{\mathrm{Tr}((1+\alpha)(Y_1+\cdots+Y_t))} \cdot \Xi^{\mathrm{Tr}(Y_1+\cdots+Y_t)} \cdot E\left(Y_1 \cdots Y_{s_1} \cdot Y_{s_1+1}^2 \cdots Y_{s_1+s_2}^2\right).$$

Using linearity of the trace and distributivity, we can write:

$$P^{(s_1,s_2)}(\Theta,\Xi) = \left(\Theta + \Xi J + \Theta\Xi J^2\right)^{s_1} \cdot \left(\Theta + \Xi J^2 + \Theta\Xi J\right)^{s_2} \cdot (\Theta + \Xi + \Theta\Xi)^{t-s_1-s_2}.$$

It follows that $P^{(0,0)}(1,1) = 3^t$ and $P^{(s_1,s_2)}(1,1) = 0$, for $(s_1,s_2) \neq (0,0)$. Furthermore, we see that

$$\begin{aligned} P^{(s_1,s_2)}(1,-1) &= (-1)^{t-s_1-s_2} \cdot 2^{s_1+s_2}, \\ P^{(s_1,s_2)}(-1,1) &= (-1)^{t-s_1-s_2} \cdot 2^{s_1+s_2} J^{s_1+2s_2}, \\ P^{(s_1,s_2)}(-1,-1) &= (-1)^{t-s_1-s_2} \cdot 2^{s_1+s_2} J^{2s_1+s_2}. \end{aligned}$$

We can reconstruct the individual values as follows:

$$\begin{aligned} P_0^{(s_1,s_2)} &= \left(P^{(s_1,s_2)}(1,1) + P^{(s_1,s_2)}(1,-1) + P^{(s_1,s_2)}(-1,1) + P^{(s_1,s_2)}(-1,-1)\right)/4, \\ P_1^{(s_1,s_2)} &= \left(P^{(s_1,s_2)}(1,1) + P^{(s_1,s_2)}(1,-1) - P^{(s_1,s_2)}(-1,1) - P^{(s_1,s_2)}(-1,-1)\right)/4, \\ P_\alpha^{(s_1,s_2)} &= \left(P^{(s_1,s_2)}(1,1) - P^{(s_1,s_2)}(1,-1) + P^{(s_1,s_2)}(-1,1) - P^{(s_1,s_2)}(-1,-1)\right)/4, \\ P_{1+\alpha}^{(s_1,s_2)} &= \left(P^{(s_1,s_2)}(1,1) - P^{(s_1,s_2)}(1,-1) - P^{(s_1,s_2)}(-1,1) + P^{(s_1,s_2)}(-1,-1)\right)/4. \end{aligned}$$

For $(s_1,s_2) = (0,0)$, we find:

$$P_0^{(0,0)} = (3^t + 3 \cdot (-1)^t)/4 \quad \text{and} \quad P_1^{(0,0)} = P_\alpha^{(0,0)} = P_{1+\alpha}^{(0,0)} = (3^t - (-1)^t)/4.$$

For non-zero (s_1, s_2) pairs:

$$
\begin{aligned}
s_1 + 2s_2 \not\equiv 0 \pmod 3 \rightarrow P_0^{(s_1,s_2)} &= 0 \text{ and for non-zero } i\text{:} \\
P_i^{(s_1,s_2)} &= (-1)^{t+1-s_1-s_2} \cdot 2^{s_1+s_2-1} \cdot J^{(s_1+2s_2)\log_\alpha i}. \\
s_1 + 2s_2 \equiv 0 \pmod 3 \rightarrow P_0^{(s_1,s_2)} &= 3 \cdot (-1)^{t-s_1-s_2} \cdot 2^{s_1+s_2-2} \text{ and for non-zero } i\text{:} \\
P_i^{(s_1,s_2)} &= (-1)^{t+1-s_1-s_2} \cdot 2^{s_1+s_2-2}.
\end{aligned}
$$

Similarly to the $\mathbb{F}_3$ case, the denominators are the values $P_i^{(0,0)}$. Again, the extra coefficient of 3^{n-t} cancels out when considering the quotients $P_i^{(s_1,s_2)}/P_i^{(0,0)}$. This concludes the proof.

8 Tables of experiments over $\mathbb{F}_3$

In the following tables, we have selected the number of points to show the approximate range where full success becomes highly probable. More experiments are needed to get a more precise view of the required number of points, especially when t gets close or greater than n. Indeed, in that situation, the possibility of non-linear independence of monomials enters the picture and confuses the threshold.

Table 2: Experiments with 22 vars. Mem: 96Mb; Time: 0.15 sec

Parameters		Success rate per level					Parameters		Success rate per level				
t	#Pts	L1	L2	L3	L4	L5	t	#Pts	L1	L2	L3	L4	L5
3	100	97	96	96	96	50	3	150	100	99	99	99	96
4	150	100	91	91	91	78	4	200	100	98	98	98	98
5	200	93	93	93	93	46	5	300	97	97	97	97	95
6	300	100	94	94	94	81	6	400	100	97	97	97	95
7	500	98	98	98	98	84	7	700	100	100	100	100	100
8	700	100	92	92	92	88	8	1000	100	98	98	98	98
9	1000	100	99	99	99	91	9	1500	100	100	100	100	100
10	2000	100	97	97	97	97	10	3000	100	100	100	100	100
11	2000	100	100	100	100	99	11	3000	100	100	100	100	100
12	3000	100	99	99	99	99	12	4000	100	99	99	99	99
13	3000	99	99	99	99	74	13	5000	100	100	100	100	100
14	4000	100	97	97	97	93	14	6000	100	100	100	100	100
15	4000	97	97	97	97	20	15	6000	99	99	99	99	95
16	6000	100	95	95	95	68	16	10000	100	100	100	100	100
17	6000	87	86	86	86	3	17	10000	96	96	96	96	77
18	10000	99	92	92	92	46	18	20000	100	100	100	100	100
19	20000	99	99	99	99	90	19	30000	100	100	100	100	100
20	20000	100	92	92	92	71	20	30000	100	99	99	99	99
21	20000	92	92	92	92	5	21	40000	97	97	97	97	95
22	40000	100	96	96	96	83	22	60000	100	99	99	99	99

t	#Pts	L1	L2	L3	L4	L5	t	#Pts	L1	L2	L3	L4	L5
23	60000	95	95	95	95	86	23	80000	100	100	100	100	100
24	70000	100	95	95	95	79	24	90000	100	96	96	96	95

Table 3: Experiments with 23 vars. Mem: 192Mb; Time: 0.35 sec

Parameters		Success rate per level					Parameters		Success rate per level				
t	#Pts	L1	L2	L3	L4	L5	t	#Pts	L1	L2	L3	L4	L5
3	100	95	95	95	95	41	3	150	100	100	100	100	98
4	150	100	94	94	94	76	4	200	100	96	96	96	96
5	200	97	96	96	96	35	5	300	100	100	100	100	94
6	300	100	93	93	93	76	6	400	100	94	94	94	94
7	500	96	96	96	96	83	7	700	99	99	99	99	98
8	700	100	93	93	93	91	8	1000	100	98	98	98	98
9	1000	97	95	95	95	86	9	1500	100	100	100	100	100
10	2000	100	97	97	97	97	10	3000	100	100	100	100	100
11	2000	99	99	99	99	97	11	3000	100	100	100	100	100
12	3000	100	98	98	98	98	12	4000	100	99	99	99	99
13	3000	95	95	95	95	75	13	5000	100	100	100	100	100
14	4000	100	98	98	98	88	14	6000	100	99	99	99	99
15	4000	92	92	92	92	25	15	6000	98	98	98	98	91
16	6000	99	94	94	94	58	16	10000	100	99	99	99	98
17	6000	85	84	84	84	0	17	10000	98	97	97	97	83
18	10000	100	93	93	93	38	18	20000	100	98	98	98	98
19	20000	96	96	96	96	90	19	30000	100	100	100	100	100
20	20000	100	94	94	94	59	20	30000	100	99	99	99	99
21	20000	87	86	86	86	1	21	40000	99	99	99	99	96
22	40000	100	95	95	95	79	22	60000	100	99	99	99	99
23	60000	96	96	96	96	76	23	80000	100	100	100	100	99
24	70000	100	98	98	98	74	24	90000	100	98	98	98	94

Table 4: Experiments with 24 vars. Mem: 384Mb; Time: 0.73 sec

Parameters		Success rate per level					Parameters		Success rate per level				
t	#Pts	L1	L2	L3	L4	L5	t	#Pts	L1	L2	L3	L4	L5
3	100	100	100	100	100	39	3	150	100	100	100	100	92
4	150	100	94	94	94	71	4	200	100	97	97	97	96
5	200	97	97	97	97	19	5	300	100	100	100	100	92
6	300	100	93	93	93	68	6	400	100	96	96	96	91
7	500	97	97	97	97	76	7	700	100	100	100	100	100
8	700	100	94	94	94	88	8	1000	100	98	98	98	98
9	1000	98	97	97	97	87	9	1500	100	100	100	100	100
10	2000	100	99	99	99	99	10	3000	100	100	100	100	100

t	#Pts	L1	L2	L3	L4	L5	t	#Pts	L1	L2	L3	L4	L5
11	2000	99	99	99	99	94	11	3000	100	100	100	100	99
12	3000	100	97	97	97	97	12	4000	100	98	98	98	98
13	3000	100	100	100	100	71	13	5000	100	100	100	100	100
14	4000	99	92	92	92	86	14	6000	100	96	96	96	96
15	4000	89	88	88	88	8	15	6000	98	98	98	98	86
16	6000	100	94	94	94	58	16	10000	100	98	98	98	98
17	6000	85	82	82	80	0	17	10000	97	96	96	96	75
18	10000	99	95	95	95	26	18	20000	100	98	98	98	98
19	20000	98	98	98	98	91	19	30000	100	100	100	100	100
20	20000	100	93	93	93	58	20	30000	100	99	99	99	99
21	20000	88	88	88	88	0	21	40000	100	100	100	100	99
22	40000	100	97	97	97	81	22	60000	100	98	98	98	98
23	60000	98	98	98	98	71	23	80000	98	98	98	98	97
24	70000	99	95	95	95	65	24	90000	100	96	96	96	94
25	70000	91	90	90	90	0	25	100000	97	97	97	97	53
32	1000000	100	99	99	99	99	35	2000000	94	94	94	94	58

9 Tables of experiments over $\mathbb{F}_4$

Table 5: Experiments with 13 vars. Mem: 121Mb; Time: 0.16 sec

Parameters		Success rate			Parameters		Success rate		
t	#Pts	L1	L2	L3	t	#Pts	L1	L2	L3
3	100	100	100	97	3	150	100	100	100
4	150	97	95	57	4	250	100	100	100
5	300	97	92	46	5	400	100	100	98
6	600	100	99	80	6	750	100	100	100
7	1000	100	99	23	7	1500	100	100	100
8	2000	100	100	37	8	3000	100	100	100
9	4000	100	100	46	9	6000	100	100	100
10	8000	100	100	77	10	9000	100	100	99
11	14000	100	100	72	11	16000	100	100	96
12	25000	100	100	59	12	30000	100	100	100
13	40000	100	99	8	13	60000	100	99	98
14	80000	100	93	39	14	100000	100	99	87
15	150000	100	95	50	15	200000	100	99	72

Table 6: Experiments with 14 vars. Mem: 364Mb; Time: 0.53 sec

Parameters		Success rate			Parameters		Success rate		
t	#Pts	L1	L2	L3	t	#Pts	L1	L2	L3
3	100	100	100	89	3	150	100	100	100
4	150	94	82	29	4	250	100	100	100
5	300	92	79	15	5	400	100	100	97
6	600	100	92	34	6	750	100	100	98
7	1000	97	69	4	7	1500	100	100	99
8	2000	99	76	3	8	3000	100	100	99
9	4000	100	98	10	9	6000	100	100	100
10	8000	100	99	28	10	9000	100	100	74
11	14000	100	97	4	11	16000	100	100	56
12	25000	100	90	2	12	30000	100	100	64
13	40000	97	14	0	13	60000	100	100	94
14	80000	100	83	0	14	100000	100	100	73
15	150000	100	100	1	15	200000	100	100	87
16	300000	100	98	44	16	400000	100	100	79

10 Conclusion

Based on classical correlation cryptanalysis techniques, we have obtained a new attack on QA-SD pseudo-random correlation generators that greatly outperforms the previous attack from [4]. Indeed, it recovers error polynomials with more terms, more quickly and using considerably less memory.

We have implemented and tested for QA-SD over $\mathbb{F}_3$ and $\mathbb{F}_4$, but the approach we use for $\mathbb{F}_4$ can be generalized to other small fields, like $\mathbb{F}_8$ or $\mathbb{F}_{16}$, which have been proposed in [10, Table 6/7]. However, for a field of size q, we expect the correlations biases to be proportional to $\left(q^{1/2}/(q-1)\right)^t$, becoming smaller as q grows. As a consequence, the attack becomes less efficient for larger fields. In particular, the large field QA-SD proposed in [9] is not vulnerable to our attack. It is an open problem to see whether it is possible to take advantage of the fact that the large field QA-SD proposal relies on evaluation points with coordinates in a very small multiplicative subgroup of $\mathbb{F}_p$.

All in all, this new attack should help designers of QA-SD based cryptographic systems to select more secure parameters. This selection requires a careful analysis, mixing in the probability that subpolynomials of low enough weight arise (as analyzed in [4]) and the updated bound on the meaning of low-weight in this context that comes from our improved reconstruction of the error polynomials.

References

1. Bardenet, R., Maillard, O.A.: Concentration inequalities for sampling without replacement. Bernoulli **21**(3), 1361–1385 (2015). https://doi.org/10.3150/14-

BEJ605, https://hal.science/hal-01216652
2. Bombar, M., Bui, D., Couteau, G., Couvreur, A., Ducros, C., Servan-Schreiber, S.: FOLEAGE: $\mathbb{F}_4$ OLE-Based Multi-party Computation for Boolean Circuits. In: Chung, K., Sasaki, Y. (eds.) Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part VI. pp. 69–101. Lecture Notes in Computer Science, Springer (2024). https://doi.org/10.1007/978-981-96-0938-3_3, https://doi.org/10.1007/978-981-96-0938-3_3
3. Bombar, M., Couteau, G., Couvreur, A., Ducros, C.: Correlated pseudorandomness from the hardness of quasi-abelian decoding. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part IV. pp. 567–601. Lecture Notes in Computer Science, Springer (2023). https://doi.org/10.1007/978-3-031-38551-3_18, https://doi.org/10.1007/978-3-031-38551-3_18
4. Bouillaguet, C., Delaplace, C., Hamdad, M., Vergnaud, D.: Practical cryptanalysis of pseudorandom correlation generators based on quasi-abelian syndrome decoding. In: Hanaoka, G., Yang, B. (eds.) Advances in Cryptology - ASIACRYPT 2025 - 31st International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, VIC, Australia, December 8-12, 2025, Proceedings, Part IV. pp. 436–465. Lecture Notes in Computer Science, Springer (2025). https://doi.org/10.1007/978-981-95-5113-2_14, https://doi.org/10.1007/978-981-95-5113-2_14
5. Carlet, C.: Boolean functions for cryptography and coding theory. Cambridge University Press (2021)
6. Feneuil, T., Joux, A., Rivain, M.: Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part II. pp. 541–572. Lecture Notes in Computer Science, Springer (2022). https://doi.org/10.1007/978-3-031-15979-4_19, https://doi.org/10.1007/978-3-031-15979-4_19
7. Joux, A.: Algorithmic cryptanalysis. Chapman and Hall/CRC (2009)
8. Kumar, P.V., Scholtz, R.A., Welch, L.R.: Generalized bent functions and their properties. Journal of Combinatorial Theory, Series A **40**(1), 90–107 (1985)
9. Li, Z., Liu, H., Xing, C., Yao, Y., Yuan, C.: Faster pseudorandom correlation generators via Walsh-Hadamard transform. IACR Cryptol. ePrint Arch. **2026**, 196 (2026), https://eprint.iacr.org/2026/196
10. Li, Z., Xing, C., Yao, Y., Yuan, C.: Efficient pseudorandom correlation generators for any finite field. In: Fehr, S., Fouque, P. (eds.) Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part V. pp. 145–175. Lecture Notes in Computer Science, Springer (2025). https://doi.org/10.1007/978-3-031-91092-0_6, https://doi.org/10.1007/978-3-031-91092-0_6
11. Mesnager, S.: Bent functions. Springer (2016)
12. Sendrier, N.: Decoding one out of many. In: International Workshop on Post-Quantum Cryptography. pp. 51–67. Springer (2011)