# Small Linearization: Memory Friendly Solving of Non-Linear Equations over Finite Fields

Christopher Wolf & Enrico Thomae
Horst Görtz Institute for IT-security
Faculty of Mathematics
Ruhr-University of Bochum, 44780 Bochum, Germany
{christopher.wolf,enrico.thomae}@ruhr-uni-bochum.de,
chris@Christopher-Wolf.de

First Version: 2011-12-14

**Abstract**

Solving non-linear and in particular $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic equations over finite fields is an important cryptanalytic problem. Apart from needing exponential time in general, we also need very large amounts of memory, namely $\approx Nn^2$ for $n$ variables, solving degree $D$, and $N \approx n^D$. Exploiting systematic structures in the linearization matrix, we show how we can reduce this amount of memory by $n^2$ to $\approx N$. For practical problems, this is a significant improvement and allows to fit the overall algorithm in the RAM of *one* machine, even for larger values of $n$. Hence we call our technique Small Linearization ($s\ell$).

We achieve this by introducing a probabilistic version of the $F_5$ criterion. It allows us to replace (sparse) Gaussian Elimination by black box methods for solving the underlying linear algebra problem. Therefore, we achive a drastic reduction in the algorithm's memory requirements. In addition, Small Linearization allows for far easier parallelization than algorithms using structured Gauss.

**Keywords:** $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic systems of equations, Finite Fields, $F_5$, XL, Hybrid$F_5$, Gröbner Basis, XL, eXtended Linearization, Algebraic Cryptanalysis

## 1 Introduction

Solving systems of non-linear equations is a subject of intensive study for more than hundred years. In cryptanalysis, these systems proved particularly useful not only in the case of stream ciphers [Cou02, FM07], but also for block ciphers [MR02b, BPW06] and hash functions [SKPI07]. Similarly, it was possible using algebraic techniques to break asymmetric schemes such as variants of McEliece [FOPT10], $\mathcal{MQ}$-schemes such as Hidden Field Equations [FJ03], or the Isomorphism of Polynomials identification scheme [Per05]. They are all subsumed under the term *algebraic cryptanalysis*.

A main drawback of algebraic cryptanalysis is the huge memory consumption, which is even the case for advanced methods such as Hybrid$F_5$ [BFP09]. In this article we concentrate on the class of a Gröbner Bases algorithms like $F_4$, $F_5$ and XL (eXtended Linearization) to solve equations of degree 2. We call these equations $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic ($\mathcal{MQ}$). Note that it is sufficient to concentrate on them as $\mathcal{MQ}$ is already $\mathcal{NP}$-complete [GJ79]. In addition, any systems of degree higher than 2 can be reduced efficiently to $\mathcal{MQ}$. In particular we show how to fuse $F_5$ and XL into a variant that uses substantially less memory than any other known solver of $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic equations. Even for theoretical running times of above $2^{80}$, the amount of memory needed still fits into the RAM of a normal computer ($\approx$ 4 GB, cf. Sect. 4.4).

### 1.1 Related Work

To solve $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic equations, different algorithms are used in cryptanalysis. They can be traced back to two different origins, namely algorithms based on Buchberger's thesis [Buc65], and algorithms based on a technique called *Relinearization*, introduced by Shamir *et al.* [KS99]. We call the first *Gröbner basis*, the second *XL* algorithms. While Gröbner basis algorithms take a more symbolic view, XL is strongly connected to linear algebra. In either case, we can use these algorithms to solve a system $\mathcal{P}(x)$ of $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic equations over a finite field $\mathbb{F}$.

For Gröbner basis algorithms, we compute the Gröbner basis of the ideal associated to the system $\mathcal{P}(x)$ by the mean of so-called *S-Polynomials*. This yields at least one univariate equation which we can then solve. Main step forward in this area was the $F_4$ algorithm [Fau99] as it computed S-Polynomials blockwise rather than pairwise and thereby omitted many unnecessary steps in Buchberger's initial algorithm. It was later improved to the so-called $F_5$ algorithm [Fau02b] which removed trivial syzygies by keeping track of its computation through so-called *signatures*. For example, this algorithm was used to break HFE challenge 1 [Fau02a] in under 96 hours. To the knowledge of the authors, the fastest general purpose realization of $F_5$ is the so-called *hybrid strategy* which was used to break many symmetric and asymmetric challenges [BFP09]. However, there are also special instances of $F_5$ for cryptographic purposes [JV11]. We summarize the last three algorithms under the heading *F-class*.

In the case of XL algorithms, finding solutions for $\mathcal{P}(x)$ is reduced to linear algebra from the start and a univariate equation over the ground field $\mathbb{F}$ is obtained by producing a matrix in row echelon form. As for the previous class, we can then solve the overall system, cf. Sect. 2 for more details. A refined version of this idea was given in [CKPS00]. In particular, linear dependencies were removed after each intermediate step. While there have been rumours that AES can be broken this way, they have later been discarded [MR02a]. However, this has seriously damaged the reputation of XL in the cryptographic community. Still, there was progress made in three areas. ***First***, it was discovered that XL becomes more efficient by guessing $r$ variables for some small value $r \in \mathbb{N}$. This can be seen as a time-time tradeoff between $\mathcal{O}(q^r)$ and the running time of XL for $(n-r)$ variables. ***Second***, Faugère and Joux showed that a careful implementation of the linear algebra step can drastically reduce memory [FJ03, Sect. 5.2&5.3]. In fact, the idea sketched there is elaborated in Sect. 4.1. In addition, they provide a randomized version of the algorithm presented in Sect. 4 neglecting systematic linear dependencies between matrix rows. ***Third***, XL was used in connection to sparse equation solvers such as Wiedemann. This way, the inherent problem of memory consumption was mitigated [YCBC07, ADK+10]. However, the fact that the linear algebra to be solved was significantly larger than for the F-class remained. ***Forth***, the so-called "Mutant strategy" was invented to make use of values, which were previously discarded. As a result, XL was sped up by one order of magnitude [SAD+08] as the size of the matrix was significantly reduced. It is the most efficient representative of the XL class known to the authors.

During the last few years, both XL and $F_5$ were treated as two rather different algorithms. In addition, there was a heated discussion, which algorithm is better. In particular, Ars *et al.* [AFI+04] showed that XL is a redundant version of $F_4$, *i.e.* it produces more equations than necessary. As $F_5$ is always more efficient than $F_4$, this was a devastating blow for the XL family. Moreover, in 2011 Albrecht *et al.* [ACFP11] showed that even MutantXL is a redundant version of $F_4$. A "unifying attempt" is due to Albrecht who introduced the Matrix-$F_5$ algorithm in [Alb10, Ch. 5]. This is an $F_5$ version of the XL algorithm. However, it is not competitive with $F_5$ in terms of speed or memory. In particular, the whole matrix needs to be stored to compute the Gröbner basis. All in all, these results suggest that $F_4/F_5$ is always faster than the XL-family. Still, their main problem is memory consumption. We overcome this problem with our algorithm *Small Linearization* ($s\ell$).

## 1.2 Organization and Achievement

Our main aim is to ***minimize memory*** for $F_5$ while keeping a comparable speed. To this aim, we provide a careful analysis of the linear dependencies occuring in the linear algebra step of XL (Sect. 3). Thus we address the open problem if we can remove trivial syzygies *and* fully preserve the sparse structure of the underlying matrix at the same time. We achieve this by introducing a ***probabilistic $F_5$ criterion*** in Sect. 3. In a nutshell, it works without checking the condition of weak semi-regular sequences but leaving this to a random shuffle of the rows in the linear matrix step.

On the up-side, we do not need to store the full matrix but can evaluate it in a ***black box fashion***. As black box algorithms are among the fastest to solve matrix vector equations, we do not pay a penalty in terms of speed for this strategy. However, we save on storing the matrix. This is explained in more detail in Sect. 4. In addition, we do not need to keep track of the signature of each row but leave this to ***randomness***. As we see in Sect. 3, the odds are pretty good that this will actually destroy all trivial syzygies this way. Consequently, we do not need to compute an iterative Gröbner basis like $F_5$, as we do not perform reductions.

Apart from reducing the memory requirements, it is also very easy to ***parallelize*** our approach. This is mainly due to the existence of efficient, black box solvers for linear systems of equations, but also to

the way we can organize computations and data flow.

In order to make optimal use of black box evaluation, we introduce a ***new transfer algorithm*** to move the solution of the matrix step back to the original problem space (cf. 4.2). As a consequence, this algorithm allows to reduce the solving degree $D$ for many instances and hence can work with a smaller matrix then other black box XL algorithms.

So in contrast to $F_5$, we need far less memory but loose on efficiency. In contrast to black box XL implementations like [YCBC07, ADK+10], we also save a factor of $n^2$ on memory. In addition, we avoid linear dependencies and hence gain on efficiency. In a sense, we have fused ideas from both areas to obtain a memory efficient $\mathcal{MQ}$-solver.

Taking another point of view, we have started with Matrix-$F_5$ and removed both the incremental computation of the Gröbner basis as the need for Gaussian elimination after such each step. So on the up-side, we can use black box methods instead of Gaussian (less memory). On the down-side, Matrix-$F_5$ starts at XL and has hence a bigger matrix. In terms of algorithmics, the first noticeable difference between our algorithm and Matrix-$F_5$ is that the latter proceeds step-by-step and thus produce a Gröbner Basis. We only use the last step, what is fine as long as we only want to find *one* solution of the system $\mathcal{P}$. Nevertheless we could easily adapt our algorithm to proceed step-by-step, by using all the techniques of Matrix-$F_5$ such as *signatures*.

In this context, probabilistic means that we produce all possible equations without *any* reduction to zero. Due to random dependencies between the coefficients of $\mathcal{P}$, this can be done with sufficiently high probability. Note that in practice our algorithm is still useful, even if a few reductions to zero occur. As we see in App. B, the probability that there is at least one such reduction is very small for values $q$ occuring in cryptography.

In the other parts of this paper, some basic definitions on $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic systems and some basic facts about simultaneous equation solving are given in Sect. 2. Moreover, App. A deals with working degree $D = 3$, and App. B explores the running time for Small Linearization.

## 2 Some Basics about Solving $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic Equations

In this section we fix some notation and repeat the basic facts about solving non-linear systems of equations over a finite field $\mathbb{F}_q$ with $q$ elements. To ease the analysis we restrict to homogeneous systems, *i.e.* no linear and constant terms. Note that we can reformulate any inhomogeneous system in $(n-1)$ variables $x_1, \ldots, x_{n-1}$ as a homogeneous system by adding the variable $x_n$ and the implicit equation $x_n = 1$. Hence, solving the homogeneous case for $n$ variables and solving the inhomogeneous case for $(n-1)$ variables are equivalent. Now a $\mathcal{MQ}$-system of $m$ equations and $n$ variables is defined by interpreting $m$ polynomials $p^{(1)}, \ldots, p^{(m)}$ as equations $p^{(i)} = 0$ with coefficients $\gamma_{ij}^{(k)} \mathbb{F}$ and

$$p^{(k)}(x_1, \ldots, x_n) := \sum_{1 \le i \le j \le n} \gamma_{ij}^{(k)} x_i x_j. \tag{1}$$

Further, we need the set of all monomials of a certain degree $D$.

DEFINITION 2.1 *Let $\mathcal{P} := \{p^{(k)} | \ 1 \le k \le m\}$ be the set of homogeneous quadratic polynomials defined in (1). We define the set of all monomials of degree $D$ by*

$$\mathtt{Mon}_0 := \{1\}, \ \mathtt{Mon}_D := \{ab : a \in \mathtt{Mon}_{D-1}, b \in \{x_1, \ldots, x_n\}\}.$$

*Multiplying $\mathcal{P}$ by all monomials of degree $D$ is described by the set*

$$\mathtt{Blow}_D := \{ab \mid a \in \mathtt{Mon}_D \text{ and } b \in \mathcal{P}\}.$$

*Note that we also reduce $\mathtt{Blow}_D$ by the field equations $x_i^q - x_i = 0$ as we are only interested in solutions over the ground field and not the algebraic closure.*

Let $\tilde{D} := D + 2$. To solve $\mathcal{P}$ by linearizing $\mathtt{Blow}_D$, we need the following definition.

DEFINITION 2.2 *Let $M := |\mathtt{Mon}_{\tilde{D}}|$. We call $\pi \in \mathtt{Mon}_{\tilde{D}}^M$ a* semantic *of $\mathtt{Mon}_{\tilde{D}}$ if $\forall i \neq j : \pi_i \neq \pi_j$. Let $N := |\mathtt{Blow}_D|$ and $A \in \mathbb{F}^{N \times M}$ be the* coefficient matrix *of $\mathtt{Blow}_D$. Then each polynomial $b \in \mathtt{Blow}_D$ with $b = \sum_{i=1}^M b_i \pi_i$ corresponds to some row $A_k$ of $A$,* i.e. *we have $\forall b \in \mathtt{Blow}_D \ \exists k : A_k = (b_1, \ldots, b_M)$.*

Throughout the paper, the semantic $\pi$ will be in lexicographical ordering. We use it for simplicity, although other orderings are possible, too, and do in particular not affect the proofs. The only requirement is that the last $D+2$ entries of $\pi$ depend only on the variables $x_{n-1}, x_n$. If $|Mon\tilde{D}| - \text{rank}A \leq \tilde{D}$, we call $\tilde{D}$ the *saturation degree* for $A$. Working with a matrix $A$ instead of the get $\texttt{Blow}_D$ is the *linearization step*. In addition, we call the matrix $A$ the *XL matrix* if it was derived in the XL algorithm and *sℓmatrix* in the case of *sℓ*. Last but not least, we use "number of linear independent equations" for the number of linear independent *rows* in the matrix $A$.

**Algorithm 1** High-Level view of the linearization algorithm

**Require:** Let $\mathcal{P}$ be a homogeneous, degree 2 system of polynomials in $n$ variables and $m$ equations. Let $D \in \mathbb{N}$ be a solving degree, and $\pi$ be a semantic for $\texttt{Mon}_{\tilde{D}}$

**Ensure:** A solution $x \in \mathbb{F}^n$ for $\mathcal{P}(x) = 0$

1: $\texttt{Blow} := \{p_i\mu : p_i \in \mathcal{P}, \mu \in \texttt{Mon}_D\}$, $N := |\texttt{Blow}|$, $M := |\texttt{Mon}_{\tilde{D}}|$,
2: $A :=$ ZeroMatrix$(\mathbb{F}^{N \times M})$, $i \leftarrow 1$
3: **for all** $p \in \texttt{Blow}_D$ **do**
4: Assign coefficients of $p$ to $i^{\text{th}}$ row of $A$, according to semantic $\pi$
5: $i \leftarrow i + 1$
6: **end for**
7: Solve $A\tilde{x} = 0$ for $\tilde{x} \in \mathbb{F}^M$
8: Derive univariate equation from $\tilde{x}$, resubstitute into $A\tilde{x}$, compute full $x$
9: **return** $x$

The idea of eXtended Linearization (XL) [Laz79, CKPS00] is quite simple. Basically we multiply every polynomials from $\mathcal{P}$ by every monomial of a certain degree $D$, *i.e.* we produce $\texttt{Blow}_D$. Obviously this preserves the original solution. For some degree $D$ the corresponding vector space spanned by $\{A_i \,|\, 1 \leq i \leq N\}$ is large enough, *i.e.* we obtain roughly as many linearly independent equations as monomials and thus we can solve the system by linearization, cf. also Alg. 1. Note that for inhomogeneous equations XL would proceed step-wise and multiply by all monomials of degree $\leq D$. However this is equivalent to just use $\texttt{Blow}_D$ for the homogenised system.

As previously mentioned, there is another unifying view on the question XL vs. $F_5$ called Matrix-$F_5$ [Alb10]. It is an $F_5$ version of the XL algorithm. For $F_5$, the role of the saturation degree $\tilde{D}$ is taken by the degree of regularity ($d_{\text{reg}}$)—another way of phrasing this is to say that $F_5$ solves as soon as degree $d_{\text{reg}}$ is reached. In a nutshell, Matrix-$F_5$ is not competitive with $F_5$ as its saturation degree is almost always larger than the degree of regularity $d_{\text{reg}}$ in $F_5$ if we take the same number of variables. To keep track for each row in the coefficient matrix $A$ by which polynomial and monomial it was produced, Matrix-$F_5$ uses *signatures*. The main difference of both algorithms is the application of the $F_5$ criterion. This criterion enables $F_5$ and Matrix-$F_5$ to remove all redundant calculations caused by so called *trivial syzygies*. Essientially, the $F_5$ criterion consists of applying the definition of weak semi-regular sequences to its input.

Definition 2.3 (weak semi-regular sequence) *The sequence $(p^{(1)}, \ldots, p^{(m)})$ of $m$ homogeneous polynomials is called* weak semi-regular *if for all $i = 1, \ldots, m$ and $g$ such that $gp^{(i)} \in \langle p^{(1)}, \ldots, p^{(i-1)} \rangle$ and $deg(gp^{(i)}) < d_{reg}$ then $g \in \langle p^{(1)}, \ldots, p^{(i-1)} \rangle$ also holds.*

To apply this criterion, we have to represent the ideal $\langle p^{(1)}, \ldots, p^{(i-1)} \rangle$ in a way that allows an easy membership test of $gp_i$. So to use Def. 2.3, Matrix-$F_5$ has to produce the set $\texttt{Blow}_D$ equation by equation alternated by Gaussian Eliminations. Consequently, the sparsity of the equations decreases, as well as does the efficiency of Matrix-$F_5$.

The crucial question to analyse the complexity of XL is, how many of $A$'s equations are linearly independent. Obviously this can become very hard, as soon as $\mathcal{P}$ contains some structure. Therefore, we restrict our analysis to generic $\mathcal{MQ}$-systems. A first attempt to describe such systems was due to Macaulay [Mac16], who defined regular sequences for $m = n$. Bardet *et al.* extended this definition to weak semi-regular sequences for $m \geq n$ in their complexity analysis of $F_5$ [BFS04]. Moh [Moh01] and later Yang and Chen [YC04] went on the same path for XL and mentioned the following lemma, although the proof in [YC04] is disputed.

**Lemma 2.4** *If a homogeneous $\mathcal{MQ}$-system $\mathcal{P}$ is a weak semi-regular sequence over $\mathbb{F}_q$ and $\tilde{D} < q$, then the number of linearly independent equations produced for $D = 2k + b$ and $b \in \{0, 1\}$ by $\texttt{Blow}_D$ is*

$$I_D := \sum_{i=0}^{k} (-1)^i \binom{m}{i+1} \binom{n + 2(k-i) + b - 1}{2(k-i) + b} \tag{2}$$

Unfortunately it is not proven yet that generic sequences are weak semi-regular. However, we have experimentially verified for values up to $D = 8$ that the conjecture holds, cf. Sect. 4.3.

# 3 Systematic Dependencies

By definition a weak semi-regular sequences can be viewed as a sequence without any special internal structure. So the only relations are trivial ones. We therefore call them *trivial syzygies*; the case of $\mathrm{F}_5$, they are called *principal syzygies*. Our overall goal is to relax the $\mathrm{F}_5$ criterion in a way that we still can identify trivial syzygies but without checking for ideal membership according to Def. 2.3. As we will see in the remainder of this section, this is possible by carefully mixing random choices with educated guesses on the monomials used in the generation of the set $\texttt{Blow}_D$ and finally leads to the formulation of Alg. 2.

Let us denote two quadratic polynomials by $f := \sum_{i=1}^{\sigma} \alpha_i a_i$ and $g := \sum_{i=1}^{\tau} \beta_i b_i$ for $\alpha_i, \beta_j \in \mathbb{F}_q$ and monomials $a_i, b_i \in \texttt{Mon}_2$ then a trivial syzygy is given by

$$gf = \sum_{i=1}^{\tau} \beta_i b_i f = \sum_{i=1}^{\tau} \beta_i b_i \sum_{i=1}^{\sigma} \alpha_i a_i = \sum_{i=1}^{\sigma} \alpha_i a_i \sum_{i=1}^{\tau} \beta_i b_i = \sum_{i=1}^{\sigma} \alpha_i a_i g = fg. \tag{3}$$

According to Lem. 2.4 the number of linearly independent equations produced by $\texttt{Blow}_2$ is $m\binom{n+1}{2} - \binom{m}{2}$, *i.e.* all dependencies are due to the $\binom{m}{2}$ possible trivial syzygies. The question we want to answer is, which of the equations we have to remove from $\texttt{Blow}_2$ to derive a set $\texttt{Blow}'_2$ of exactly $m\binom{n+1}{2} - \binom{m}{2}$ linearly independent equations. Let

$$\pi_M(f, g) := \sum_{b_i f \in M} \beta_i b_i f - \sum_{a_i g \in M} \alpha_i a_i g$$

with $f, g$ as above and $M$ a set of monomials. (3) can now be written as $\pi_M(f, g) = 0$.

## 3.1 Solving Degree $D = 2$

Example 3.1 Let $m = n = 3$ and $\mathcal{P}$ be defined by homogeneous quadratic equations $p^{(1)}, p^{(2)}$ and $p^{(3)}$. Obviously $M_i = \{ap^{(i)} \mid a \in \texttt{Mon}_2\}$ contains no linear dependencies for $i = 1, 2, 3$. Due to (3) there is each a syzygy over the sets $M_1 \cup M_2$ and $M_1 \cup M_3$. Thus we have to choose $c, d \in \texttt{Mon}_2$ and remove $cp^{(2)}$ from $M_2$ and $dp^{(3)}$ from $M_3$. Note $c = d$ is possible but not necessary. Denote $\alpha_c^{p^{(1)}}$ the coefficient of the monomial $c$ in $p^{(1)}$. Then the following hold.

$$\begin{aligned}
\pi_{M_1 \cup M_2}(p^{(1)}, p^{(2)}) &= \alpha_c^{p^{(1)}} cp^{(2)} \\
\pi_{M_1 \cup M_3}(p^{(1)}, p^{(3)}) &= \alpha_d^{p^{(1)}} dp^{(3)} && (4) \\
\pi_{M_2 \cup M_3}(p^{(2)}, p^{(3)}) &= \alpha_d^{p^{(2)}} dp^{(3)} - \alpha_c^{p^{(3)}} cp^{(2)}. && (5)
\end{aligned}$$

And thus $M_1 \cup M_2 \cup M_3$ still contains one linear dependency. We have to choose $e \in \texttt{Mon}_2$ with $e \neq d$ and remove $ep^{(3)}$ from $M_3$ to destroy this last dependency by changing equation (4) and (5). This leads to the following equations:

$$\begin{aligned}
\pi_{M_1 \cup M_2}(p^{(1)}, p^{(2)}) &= \alpha_c^{p^{(1)}} cp^{(2)} && (6) \\
\pi_{M_1 \cup M_3}(p^{(1)}, p^{(3)}) &= \alpha_d^{p^{(1)}} dp^{(3)} + \alpha_e^{p^{(1)}} ep^{(3)} && (7) \\
\pi_{M_2 \cup M_3}(p^{(2)}, p^{(3)}) &= \alpha_d^{p^{(2)}} dp^{(3)} + \alpha_e^{p^{(2)}} ep^{(3)} - \alpha_c^{p^{(3)}} cp^{(2)}. && (8)
\end{aligned}$$

Changing our point of view, investigate when

$$\begin{pmatrix} \alpha_c^{p^{(1)}} & 0 & 0 \\ 0 & \alpha_d^{p^{(1)}} & \alpha_e^{p^{(1)}} \\ 0 & \alpha_d^{p^{(2)}} & \alpha_e^{p^{(2)}} \end{pmatrix} \tag{9}$$

is regular over $\mathbb{F}_q$. Consequently (6–8) are now linearly independent with probability

$$\left(1 - \frac{1}{q}\right) \frac{(q^2-1)(q^2-q)}{q^4} \overset{q\to\infty}{\longrightarrow} 1. \tag{10}$$

So with high probability we produced the maximal number of linearly independent equations. See App. B for a more detailed analysis.

The strategy of example 3.1 can easily be extended to more than 3 equations, which leads to Alg. 5, cf. App. B. Note that we can choose $c = d$ in example 3.1. We use this to simplify the algorithm accordingly. In App. B we will see that this choice also increases the probability of (9) to be regular to $\left(\frac{q-1}{q}\right)^{m-1}$.

The extension to degree $D = 3$ is given in App. A. To deduce a general algorithm it is more important to understand the case $D = 4$ shown in the next section.

### 3.2 Solving Degree $D = 4$

The number of linear independent equations produced by $\mathtt{Blow}_4$ is

$$m\binom{n+3}{4} - \binom{m}{2}\binom{n+1}{2} + \binom{m}{3}. \tag{11}$$

Let us consider the following three quadratic polynomials $f := \sum_{i=1}^{\sigma} \alpha_i a_i$, $g := \sum_{j=1}^{\tau} \beta_j b_j$ and $h := \sum_{k=1}^{\upsilon} \gamma_k c_k$ for $\alpha_i, \beta_j, \gamma_k \in \mathbb{F}_q$ and monomials $a_i, b_j, c_k \in \mathtt{Mon}_2$. Besides the dependency $fg = gf$ we have new dependencies through $ghf = fhg = fgh$.

$$\sum_{j=1}^{\tau} \beta_j b_j \sum_{k=1}^{\upsilon} \gamma_k c_k f = \sum_{i=1}^{\sigma} \alpha_i a_i \sum_{k=1}^{\upsilon} \gamma_k c_k g = \sum_{i=1}^{\sigma} \alpha_i a_i \sum_{j=1}^{\tau} \beta_j b_j h\,. \tag{12}$$

Define

$$\pi_M^k(f,g) := \sum_{b_i c_k f \in M} \beta_i b_i c_k f - \sum_{a_j c_k g \in M} \alpha_j a_j c_k g$$

with $f, g$ as above and $c_k \in \mathtt{Mon}_2$ fixed. Let us explain the algorithm at the following example.

EXAMPLE 3.2 *Let $m = n = 3$ and $\mathcal{P}$ be given by homogeneous quadratic equations $p^{(1)}, p^{(2)}$ and $p^{(3)}$. Obviously $M_i = \{ap^{(i)} \mid a \in \mathtt{Mon}_4\}$ contains no linear dependency for $i = 1, 2, 3$. Due to $c_k(p^{(i)}p^{(j)} - p^{(j)}p^{(i)}) = 0$ the sets $M_1 \cup M_2$ and $M_1 \cup M_3$ are linear dependent for every $c_k \in \mathtt{Mon}_2$. To destroy these dependencies we have to choose $c \in \mathtt{Mon}_2$ and remove all $dp^{(2)}$ from $M_2$ for $d \in \mathtt{Mon}_4$ and $c|d$. Analogous we choose $e \in \mathtt{Mon}_2$ and remove all $\ell p^{(3)}$ from $M_3$ with $\ell \in \mathtt{Mon}_4$ and $e|\ell$ . Then the following holds.*

$$\begin{aligned} \pi^k_{M_1\cup M_2}(p^{(1)}, p^{(2)}) &= \alpha_c^{p^{(1)}} c_k c p^{(2)} \\ \pi^k_{M_1\cup M_3}(p^{(1)}, p^{(3)}) &= \alpha_e^{p^{(1)}} c_k e p^{(3)} \\ \pi^k_{M_2\cup M_3}(p^{(2)}, p^{(3)}) &= \alpha_e^{p^{(2)}} c_k e p^{(3)} - \alpha_c^{p^{(3)}} c_k c p^{(2)} \end{aligned} \tag{13}$$

*Thus $M_1 \cup M_2 \cup M_3$ is still linearly dependent. We have to choose additionally $u \in \mathtt{Mon}_2$ with $u \neq e$ and remove all $vp^{(3)}$ from $M_3$ with $v \in \mathtt{Mon}_4$ and $u|v$. This leads to the same situation as for degree 3. If $v = \ell$ we have to choose a new monomial $e' \in \mathtt{Mon}_2$ and destroy all linear dependencies obtained by producing monomials twice by removing $\ell' p^{(3)}$ with $\ell' \in \mathtt{Mon}_4$ and $\ell'$ contains $e'$ from $M_3$.*

*Counting the number of produced equations at this stage give us a value greater than* $m\binom{n+3}{4} - \binom{m}{2}\binom{n+1}{2} + \binom{m}{3}$. *The reason is that we did not use* $fh\ g = gh\ f = gf\ h$ *until now. This additional systematic dependency tells us that destroying the systematic dependency between* $f$ *and* $g$ *also destroys the systematic dependency between* $g$ *and* $h$. *This holds for every* $c_k g$ *we remove from* $M_2$. *Thus we have to remove less equations from* $M_3$ *as some dependencies are already destroyed.*

Now we are able to handle arbitrary solving degrees $D$ with Alg. 2. The step from even to odd degree is analogous to the step from $D = 2$ to $D = 3$ and the step from odd to even degree is analogous to the step from $D = 3$ to $D = 4$.

**Algorithm 2** Generating linear independent equations ($D = 2k + b$ with $b \in \{0, 1\}$)

```
1: eqn ← {};
2: list ← Mon_{D−2};
3: for μ ∈ Mon_D do
4:    eqn ← eqn ∪ {μp^(1)};
5: end for
6: μ ∈_R Mon_2; miniList ← Mon_2\{μ}; bigList ← Mon_D;
7: for i := 2 to m do
8:    count ← 0;
9:    bound ← C(n+D−3, D−2);
10:   for p := 1 to min{i − 2, k − 1} do
11:      bound ← bound + (−1)^p C(n+D−3−2p, D−2−2p) C(i−2, p);
12:   end for
13:   repeat
14:      j ← 0;
15:      repeat
16:         j ← j + 1;
17:         if list[j] · μ ∈ bigList then
18:            bigList ← bigList\{list[j] · μ};
19:            count ← count + 1;
20:         end if
21:      until j = C(n+D−3, D−2) or count = bound
22:      if j = C(n+D−3, D−2) then
23:         μ ∈_R miniList; miniList ← miniList\{μ};
24:      end if
25:   until count = bound
26:   for η ∈ bigList do
27:      eqn ← eqn ∪ {ηp^(i)};
28:      if |eqn| = T − D − 2 then
29:         Stop;
30:      end if
31:   end for
32: end for
```

## 3.3 General Case

**Lemma 3.3** *The number of equations produced by Alg. 2 equals the number of linearly independent equations given by Lem. 2.4.*

Proof. Let us restrict to $D = 2k$, as the case $D = 2k+1$ is analogous. We have to show that algorithm 2 cut out the right number of equations. According to (2) the number of columns of $\mathbf{M}$, *i.e.* the sum

$\sum_{i=1}^{m-1} s_i$ of all block sizes have to be

$$\sum_{i=1}^{k} \underbrace{(-1)^{i+1}\binom{m}{i+1}\binom{n+2(k-i)-1}{2(k-i)}}_{:=\sigma_i}.$$

To show this we make us of the following equality

$$\sum_{i=1}^{m-1} s_i = \sum_{i=1}^{m-1} s_1 i + \sum_{j=1}^{k-1}\sum_{i=j+1}^{m-1} (m-i)\tau_{ij}.$$

First $\sum_{i=1}^{m-1} s_1 i = s_1 \sum_{i=1}^{m-1} i = s_1\binom{m}{2} = \sigma_1$ holds. We finish the proof by showing that

$$\sigma_j = \sum_{i=j}^{m-1} (m-i)\tau_{i(j-1)}$$

holds for $j \in \{2,\ldots,k\}$.

$$\begin{aligned}
\sum_{i=j}^{m-1}(m-i)\tau_{i(j-1)} &= (-1)^{j-1}\binom{n+D-3-2(j-1)}{D-2-2(j-1)} \sum_{i=j}^{m-1}(m-i)\binom{i-1}{j-1} \\
&\overset{*}{=} (-1)^{j+1}\binom{n+D-2j-1}{D-2j}\binom{m}{j+1} \\
&= \sigma_j
\end{aligned}$$

Equality $*$ used that we can rearrange the order of the terms of the sum and get

$$\sum_{i=j}^{m-1}(m-i)\binom{i-1}{j-1} = \sum_{i=1}^{m-j} i\binom{m-i-1}{j-1} = \sum_{i=1}^{m-1}\binom{i}{1}\binom{m-i-1}{j-1} = \binom{m}{j+1}.$$

The last equality is due to the well known equality $\sum_{m=0}^{n}\binom{m}{j}\binom{n-m}{k-j} = \binom{n+1}{k+1}$. □

# 4 Implementation Issues

Until now we have explored the structure of the matrix $A$ from a mathematical point of view. In particular, we do not generate linearly dependent rows in our approach. In this section, we deal with implementation issues, in particular connected to memory consumption and sparse equation solving.

As the authors of [YCBC07], we have chosen to use Wiedemann's algorithm here [Wie86], in particular to its easier structure and fewer requirements on the matrix given. In a nutshell, Wiedemann allows us to solve matrix equations of the form

$$Ax = 0$$

for a matrix $A \in \mathbb{F}^{N\times M}$, and an unknown vector $x \in \mathbb{F}^M$. The key point for our memory friendly approach is that we do not need access to the matrix $A$ itself to *solve* this equation. Instead, a function $f_A(x)$ which *computes* the matrix-vector product $Ax$ *(black box access)* is sufficient. This means that we have $f_A(x) = Ax\ \forall x \in \mathbb{F}^M$, without the need to write down the matrix $A$ explicitly.

In general, all solutions to $Ax = 0$ form a *kernel*. We denote this space by $\ker(A)$. For $N \geq M$ and $R := \mathrm{Rank}(A)$, we have $r = M - R$ as corresponding nullity.

For $N \approx M$, Wiedemann's algorithm has a running time of $\mathcal{O}(N^2)$ and a memory requirement of $5N$ as it needs to store 5 vectors elements for intermediate computations.

### 4.1 Implicit Evaluation

We have $N := \#\text{rows}$ and $M := \#\text{columns}$ of the $s\ell$ matrix $A$. Moreover, for a given semantic $\pi$ we define an access function $\Pi(\mu) : \mathtt{Mon}_{\tilde{D}} \to \mathbb{N} : \mu \mapsto i$ such that $\pi_i = \mu$. So $\forall 1 \leq i \leq M : \Pi(\pi_i) = i$. Analogous, we define $\Pi_2(a) : \mathtt{Mon}_2 \to \mathbb{N} : a \mapsto k$.

We now discuss evaluating function $f_A$ for a given $s\ell$ matrix. To this aim, we exploit its regular block structure. Denote with $A^{(2)} \in \mathbb{F}^{m \times |Mon_2|}$ the coefficient matrix of the initial system $\mathcal{P}$. From a high level view, we choose some monomial $\mu \in \mathtt{Mon}_D$ and interpret the initial system $\mathcal{P}$ as a block $\mu\mathcal{P}$. For this block we compute the monomials $\mathtt{Mon}_{\mathcal{P},\mu} := \{a\mu : a \in \mathtt{Mon}_2\}$, assign each variable $\nu \in \mathbb{F}^{|Mon2|}$ its corresponding value $\nu_{\Pi_2(a)} := \tilde{x}_{\Pi(a\mu)}$, and then evaluate this block $\mathcal{P}\mu$ by $A^{(2)}\nu$.

This works as each column in $A$ is associated to *one* particular monomial. Consequently, we can identify each column in $A^{(2)}$ with exactly one column in $A$ and hence one monomial $a\mu$. Hence, each coefficient in this column will be multiplied by the same field element $\tilde{x}_{\Pi(\mu)}$ for some fixed monomial from $\mathtt{Mon}_{\tilde{D}}$. Note that the definition of $\mathtt{Mon}_{\mathcal{P},\mu}$ is *independent* of the coefficients of the initial system $\mathcal{P}$. So each block $\mu\mathcal{P}$ is associated with $|\mu\mathtt{Mon}_2| = |\mathtt{Mon}_2| = n(n+1)/2$ fixed monomials, and always the same coefficients, namely these of the system $\mathcal{P}$. So we do not need to store the coefficients of $\mathtt{Blow}_D$ for each individual block $\mu\mathcal{P}$, but only once for *all* blocks.

The algorithm sketched above is actually a memory efficient implementation of the original linearization algorithm (Alg. 1). To take care of the linear dependencies from sect. 3, we also need to delete some rows in the corresponding computation. To transfer Alg. 2 into a memory friendly version, we need to make a few changes. See alg. 3–4 for the result. In particular, we only compute one block at a time and hence store only $n^2/2$ intermediate field elements. Note that we can have $M > N$. As the system is overdetermined in this case, we cut off the remaining rows here (lines 10–12 in Alg. 4) for efficiency reasons.

**Algorithm 3** Precomputation for a given $s\ell$ matrix A

**Require:** $D \in \mathbb{N}$ be a solving degree, $n, m$ be the number of variables and equations of $\mathcal{P}$
**Ensure:** Random permutation $\Gamma$ from $\mathtt{Mon}_D$, start-values in start

1: $\Gamma \leftarrow ()$, $\text{perm}_2 = \text{Permute}(\mathtt{Mon}_2)$, $k \leftarrow \lfloor D/2 \rfloor$, binStop $\leftarrow \binom{n+D-3}{D-2}$, bound $\leftarrow 0$
2: **for all** $\alpha \in_R \text{perm}_2$ **do**
3:  **for all** $\beta \in \mathtt{Mon}_{D-2}$ **do**
4:  **if** $\alpha\beta \notin \rho$ **then**
5:  $\Gamma$.append($\alpha\beta$)
6:  **end if**
7:  **end for**
8: **end for**
9: starts $\leftarrow (\infty, \ldots, \infty)$, spares $\leftarrow (0, \ldots, 0)$ $\{m+1$ values each$\}$
10: **for** $i \leftarrow 0$ **to** m-1 **do**
11:  **for** $p \leftarrow 1$ **to** $k-1$ **do**
12:  spares$[p] \leftarrow \binom{i-1}{p}$ **if** $i \leq 1+p$ **else** 0
13:  **end for**
14:  **for** $p \leftarrow 1$ **to** $k$ **do**
15:  bound $\leftarrow$ bound $+ (-1)^p * \binom{n+D-3-2*p}{D-2-2*p} *$spares$[p]$
16:  **end for**
17:  starts$[i]$ = bound **if** bound $\geq 0$ **else** 0
18:  bound $\leftarrow$ bound $+ \binom{n+D-3}{D-2}$
19: **end for**
20: **return** $\Gamma$, starts

**Discussion.** There are two subtilities in Alg. 4 worth mentioning: First, the monomials $\alpha \in_R \text{perm}_2$ are chosen in *random* order while the monomials from $\beta \in \mathtt{Mon}_{D-2}$ are chosen in any order, as explained in Sect. 3 and in more detail in Ex. A.1: Here, we need to destroy dependencies of degree 2. On an empirical level, we have tried all other possible choices of $\alpha\beta$ here, namely: Choosing $\alpha$ in a systematic fashion, $\beta$ at random, choosing the coefficient $\omega := \alpha\beta$ at random, and choosing $\omega$ systematically. Not

**Algorithm 4** Fast Evaluation of $f_A$ for $A \in \mathbb{F}^{N \times M}$ an $s\ell$ matrix

**Require:** $\tilde{x} \in \mathbb{F}^N$ a vector, $\Gamma$ a random permutation of $\mathtt{Mon}_D$, $\Pi$ an access function to the semantic $\pi$ of $\mathtt{Mon}_{\tilde{D}}$, and $m+1$ starts values from Alg. 3.
**Ensure:** $\tilde{y} \leftarrow A\tilde{x}$

```
ỹ = (), e = 1
for j = 0 to |Mon_D| − 1 do
    for all η ∈ Mon_2 do
        μ = ηΓ[j], ν[i] = x̃_{Π(μ)}
    end for
    while starts[e] ≤ j do
        e ← e + 1
    end while
    for i ← 0 to e − 1 do
        if |ỹ| ≥ M then
            return ỹ
        end if
        ỹ.append(p^(i)(ν))
    end for
end for
return ỹ
```

surprisingly, for all three algorithms linear dependencies became more likely than in the setting of this algorithm.

Second, in Alg. 4 the end marker $e$ is used to determine the last polynomial $p_e$ which is used in the construction of the $s\ell$ matrix $A$. By the choice of the *start* values, this destroys all systematic linear dependencies (*cf.* Sect. 3).

**Speed.** To evaluate the cost for one matrix-vector-multiplication or function evaluation $f_A(x)$ is a bit problematic. When only counting operations in $\mathbb{F}$, we have the same speed as any other sparse evaluation function for the $s\ell$ matrix $A$, namely $n^2/2$ operations per row. In terms of memory access to the long vector $\tilde{x}$, we are *better*, as each block only needs to access $n^2/2$ elements in comparison to $n^2/2$ elements in the case of unstructured sparse matrices. However, we need to compute all monomials from $\mathtt{Mon}_{[}\mathcal{P}, \mu]$, which results in $n^2/2$ monomial multiplications. To derive a realistic comparison, we need to know if costs for a given computer are dominated by computations or (more likely) memory access.

**Memory Considerations.** We see that our procedure needs to store $2N \approx n^{\tilde{D}}$ field elements for $\tilde{y}, \tilde{x}$. All other values such as the $m$ start values or the $D|\mathtt{Mon}_D| \approx n^D$ field elements we need to store for the permutation $\Gamma$ are negligible in comparison. Hence, the overall memory consumption is *not dominated* by the storage requirements of the implicit matrix $A$ but by the memory requirement of Wiedemann's algorithm. Taking into account the $3N$ elements of the Berlekamp-Massey sub-routine, this number becomes $5N \approx 5n^{\tilde{D}} = \mathcal{O}(n^{\tilde{D}})$.

In any case, we can use the whole tool-kit of different Wiedemann implementations such as parallelization, block-Wiedemann algorithm to speed up our solution finding algorithm, cf. [Tho02, ADK+10]. Moreover, Alg. 4 is easily paralleliszable: We distribute the small matrix $A^{(2)}$ to k computers, as the permutation $\Gamma$, and can then evaluate the function $f_A(x)$ in parallel.

In addition, for $n \leq 25$ $n^2/2$ is bounded from above by $\approx 300$. This fits easily into the cache of a modern micro-processor, so we gain an additional speed-up to linear solvers which do not exploit a similar structure.

## 4.2 Computing Solutions of Kernels

In the orginal XL algorithm, the matrix $A$ was reduced to upper triangular form and all rows containing only univariate equations were then solved. As we work with homogenised equations, and know $x_n$, this

corresponds to rows which only contain monomials of the form $x_i x_n$ for some $i : 1 \leq i < n$. Substituting $x_n = 1$, leads to

$$\beta_{\tilde{D}} x_i^{\tilde{D}} + \ldots + \beta_1 x_i + \alpha = 0$$

for some coefficients $\alpha, \beta_k \in \mathbb{F}$ and $1 \leq k \leq \tilde{D}$, which can now be efficiently solved over the finite field $\mathbb{F}$.

In case of black box solving, the situation is different. For simplicity, we assume a 1-dimensional kernel space, *i.e.* $a \in \mathbb{F}^M, a \neq 0$ with $Aa = 0$ is the *only* kernel vector (up to a scalar factor $s \in \mathbb{F}^*$). Then we can compute the values of $x \in \mathbb{F}^n$ using

$$x_i^{\tilde{D}} - sa_{\Pi(x_i^{\tilde{D}})} = 0, \ldots, x_i - sa_{\Pi(x_i x_n^{\tilde{D}-1})} = 0, 1 - sa_{\Pi(x_n^{\tilde{D}})} = 0 \tag{14}$$

for $1 \leq i < n$, $a = (a_1, \ldots, a_M)$. By the construction of the $s\ell$ matrix $A$, (14) yields a consistent solution for $x = (x_1, \ldots, x_{n-1}, 1)$.

However, in general we have $\tilde{D} \geq N - M \geq 1$ and the nullity of $A$ becomes $r := N - M$. Here, the situation is a little more cumbersome. In particular, it seems we are dealing with $(q^r - 1)$ parasiteric solutions here. However, we know that these solutions need to be *consistent* with a solution $x \in \mathbb{F}^n$ of the original system. Let $K \in \mathbb{F}^{r \times M}$ be the base vectors of $\ker(A)$, written as a matrix. We know that the correct solution must be expressible as linear combination of *rows* of $K$. Denote the corresponding vector $\eta \in \mathbb{F}^r$. We generalize (14) to

$$\begin{aligned} x_i^{\tilde{D}} &- \eta_1 k_{1,\Pi(x_i^{\tilde{D}})} - \ldots - \eta_r k_{r,\Pi(x_i^{\tilde{D}})} = 0, \\ &\vdots \\ x_i &- \eta_1 k_{1,\Pi(x_i x_n^{\tilde{D}-1})} - \ldots - \eta_r k_{r,\Pi(x_i x_n^{\tilde{D}-1})} = 0, \\ 1 &- \eta_1 k_{1,\Pi(x_n^{\tilde{D}})} - \ldots - \eta_r k_{r,\Pi(x_n^{\tilde{D}})} = 0 \end{aligned} \tag{15}$$

for $1 \leq i < n$ and $k_{i,j} \in \mathbb{F}$ the coefficients of the kernel matrix $K$. Each block in one $x_i$ can be solved individually by eliminating all $r$ variables $\eta_1, \ldots, \eta_r$. As the degee of these variables is one, we obtain one *univariate equation* in $x_i$ of degree at most $\tilde{D}$. Note that we have $r \leq \tilde{D}$, *i.e.* we can eliminate *all* variables $\eta_1, \ldots, \eta_r$. As $A\tilde{x}$ has a solution, at least one block $x_i$ will yield a vector $\eta \in \mathbb{F}^r$, so we can now solve all other blocks and recover the full solution $x \in \mathbb{F}^n$. Empirically, we have found a solution to (15) already for the first choice of $i$ in most cases.

To our knowledge, this solving algorithm has not been described so far in the open literature. In particular [YCBC07, ADK+10, Abd11] neglected this question and hence required $N - M = 1$, which also implies the existence of only *one* solution. For cryptographic systems, this is usually fine. Still, in practice this strategy leads to a significant loss of performance as we need a higher solving degree $D$ as we have replaced the condition $|\mathtt{Mon}_D| - I_D \leq \tilde{D}$ by $|\mathtt{Mon}_D| - I_D \leq 1$.

### 4.3 Implementation and Verification

We have implemented Small Linearization in SAGE [S+10], using a total of 2000 lines of code (including comments). However, most of the code is verifying the correctness of intermediate values, so the actual code contains only a few 100 lines. While SAGE is a good tool to investigate mathematical problems, it has some serious performance issues. Hence, we were only able to investigate systems up to $n = 7$ and $m = 8$. All in all, our implementation is a prototype of Small Linearization and was used to verify the agreement between theory and practice. In particular, we have worked with random systems as they are the hardest case for $\mathcal{MQ}$-systems. With random we mean to choose the coefficients $\gamma_{ij}^k \in_R \mathbb{F}$ uniformly at random for the polynomials in the $\mathcal{MQ}$-system $\mathcal{P}$.

To obtain a fair comparison between $\mathrm{F}_5$ and $s\ell$, we need a high speed implementation, for example in C++. Hence, we had to use the theory from Sect. 3 and the open literature to derive the values of tables 1–2.

In addition, we did extensive empirical verification of Lem. 2.4 by computer simulation written in [MAG], giving more than 10,000 data points. Parameters were running for various tuples $(n, m, D)$ in the range $3 \leq n \leq 15$, $3 \leq m \leq 50$, $1 \leq D \leq 8$. All data points were in line with theory and are hence an empirical verification of our analysis.

## 4.4 Comparison with Other Algorithms

**Random Systems** In this section, we compare the efficiency of our approach with previously known algorithms. Throughout this section, we use as field GF(256) and the most difficult case $n = m$ for benchmarking, cf. Table 1. In particular, our set of competitors includes the best known attack against $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic schemes, namly HybridF$_5$ [BFP09]. In addition to use F$_5$, they brute force $r$ variables and balance the workload of $q^r$ against the running time of F$_5$ for $(n-r)$ variables and $m = n$ equations. This has proved very efficient for XL and $s\ell$, too. To keep the comparison fair, we have hence included guessing $r$ variables for all algorithms, *i.e.* the fixing strategy in the notation of XL.

Table 1: HybridF$_5$ (H$_5$) from [BFP09], plain XL (Alg. 1), XL with Wiedemann solver (Wied), and Small Linearization ($s\ell$) regarding running time (T) and memory consumption (M). $d_{\text{reg}}$ the degree of regularity for F$_5$, and $\tilde{D}$ the solving degree for XL, WiedemannXL, and $s\ell$. In all cases, we have guess $r$ variables over $\mathbb{F} = \text{GF}(2^8)$ and use a system with $n = m$ variables, and equations. For each set parameter set in $m$, minimal values in T and M are marked in **bold**.

| $m$ | Parameters H$_5$ | T(H$_5$) [log$_2$] | T(XL/Wied) [log$_2$] | M(Wied) | T($s\ell$) [log$_2$] | M($s\ell$) | Parameters XL/Wied/$s\ell$ |
|---|---|---|---|---|---|---|---|
| 10 | $r = 1, d_{\text{reg}} = 6$ | **37.75** | 40.55 | 179 kB | 39.10 | 15 kB | $r = 2, \tilde{D} = 6$ |
| 10 | $r = 2, d_{\text{reg}} = 5$ | 41.90 | 44.46 | 35 kB | 43.26 | **4 kB** | $r = 3, \tilde{D} = 5$ |
| 12 | $r = 1, d_{\text{reg}} = 7$ | **43.66** | 46.27 | 2 MB | 44.49 | 95 kB | $r = 2, \tilde{D} = 7$ |
| 12 | $r = 2, d_{\text{reg}} = 6$ | 47.75 | 50.13 | 381 kB | 48.58 | **24 kB** | $r = 3, \tilde{D} = 6$ |
| 14 | $r = 1, d_{\text{reg}} = 8$ | **49.50** | 51.98 | 19 MB | 49.89 | 615 kB | $r = 2, \tilde{D} = 8$ |
| 14 | $r = 2, d_{\text{reg}} = 6$ | 50.81 | 52.44 | 1 MB | 51.19 | **60 kB** | $r = 3, \tilde{D} = 6$ |
| 16 | $r = 1, d_{\text{reg}} = 9$ | **55.28** | 57.65 | 183 MB | **55.28** | 4 MB | $r = 2, \tilde{D} = 9$ |
| 16 | $r = 2, d_{\text{reg}} = 7$ | 56.48 | 58.13 | 12 MB | 56.48 | **379 kB** | $r = 3, \tilde{D} = 7$ |
| 18 | $r = 1, d_{\text{reg}} = 10$ | 61.02 | 63.32 | 2 GB | **60.68** | 25 MB | $r = 2, \tilde{D} = 10$ |
| 18 | $r = 2, d_{\text{reg}} = 8$ | 62.15 | 63.80 | 109 MB | 61.81 | **2 MB** | $r = 3, \tilde{D} = 8$ |
| 20 | $r = 1, d_{\text{reg}} = 11$ | 66.73 | 68.96 | 15 GB | **66.09** | 165 MB | $r = 2, \tilde{D} = 11$ |
| 20 | $r = 2, d_{\text{reg}} = 9$ | 67.79 | 69.45 | 987 MB | 67.15 | **15 MB** | $r = 3, \tilde{D} = 9$ |
| 22 | $r = 1, d_{\text{reg}} = 12$ | 72.42 | 74.60 | 126 GB | **71.50** | 1 GB | $r = 2, \tilde{D} = 12$ |
| 22 | $r = 2, d_{\text{reg}} = 10$ | 73.43 | 75.08 | 8 GB | 72.51 | **96 MB** | $r = 3, \tilde{D} = 10$ |
| 24 | $r = 1, d_{\text{reg}} = 13$ | 78.09 | 80.23 | 1 TB | **76.92** | 7 GB | $r = 2, \tilde{D} = 13$ |
| 24 | $r = 2, d_{\text{reg}} = 11$ | 79.06 | 80.71 | 72 GB | 77.89 | **615 MB** | $r = 3, \tilde{D} = 11$ |
| 26 | $r = 1, d_{\text{reg}} = 14$ | 83.74 | 85.84 | 9 TB | **82.34** | 45 GB | $r = 2, \tilde{D} = 14$ |
| 26 | $r = 2, d_{\text{reg}} = 12$ | 84.67 | 86.33 | 604 GB | 83.27 | **4 GB** | $r = 3, \tilde{D} = 12$ |
| 28 | $r = 1, d_{\text{reg}} = 15$ | 89.38 | 91.45 | 71 TB | **87.77** | 295 GB | $r = 2, \tilde{D} = 15$ |
| 28 | $r = 2, d_{\text{reg}} = 13$ | 90.28 | 91.94 | 5 TB | 88.67 | **25 GB** | $r = 3, \tilde{D} = 13$ |
| 30 | $r = 1, d_{\text{reg}} = 16$ | 95.01 | 97.06 | 573 TB | **93.20** | 2 TB | $r = 2, \tilde{D} = 16$ |
| 30 | $r = 2, d_{\text{reg}} = 14$ | 95.89 | 97.54 | 39 TB | 94.07 | **164 GB** | $r = 3, \tilde{D} = 14$ |

To compute this table, we have determined the minimal workload (first line for each parameter $m$), individually for HybridF$_5$ (H$_5$) and XL with fixing. Note that H$_5$ on the one hand and the XL family on the other hand differ in the number of variables they need to guess for an optimal workload. The workload for Wied and $s\ell$ was computed by $(\#\text{columns})^\omega$ for $\omega = 2$, and $\#$columns implied by the corresponding saturation degree $\tilde{D}$. The corresponding degree of regularity $d_{\text{reg}}$ (F$_5$) is given in the left-most column, the saturation degree $\tilde{D}$ (for XL/Wied/sl) in the right-most column. We see that HybridF$_5$ consistently needs to guess one variable less than the XL family. The reason is that the saturation degree $\tilde{D}$ is larger than the degree of regularity $d_{\text{reg}}$ for the same number of variables guessed in Table 1. To account for this disadvantage, XL needs to invest more time in the precomputation step.

First we see that the memory consumption of $s\ell$ is far less than that of Wied. Even for an overall workload close to $2^{80}$ ($m = 24$) we can fit the whole algorithm in memory (7 GB and 615 MB, respectively). For Wied, this is more tricky (1 TB or 72 GB, respectively). The reason is that Wied still uses

Table 2: Comparison between Small Linearization ($s\ell$) and HybridF$_5$ (H$_5$)from [BFP09]. In all cases, we have guess $r$ variables over $\mathbb{F} = \mathrm{GF}(2^8)$. Fields with $\star$ were not given in [BFP09]. Particularly relevant values are **bold**.

| | $m$ | $r$ | HybridF$_5$ Time [log$_2$] | HybridF$_5$ Memory | Small Linearization Time [log$_2$] | Small Linearization Memory |
|---|---|---|---|---|---|---|
| UOV ($n = 30$) | 10 | 1 | **37.75** | **2 MB** | 40.99 | 451 kB |
| | | 2 | 41.90 | $\star$ | **39.10** | **15 kB** |
| UOV ($n = 60$) enTTS ($n = 28$) | 20 | 1 | 66.73 | 139 GB | 80.01 | 321 GB |
| | | 2 | **67.79** | **12 GB** | 66.09 | 165 MB |
| | | 3 | 71.62 | $\star$ | **67.15** | **15 MB** |
| Rainbow ($n = 48$) amTTS ($n = 34$) | 24 | 1 | 78.09 | 10 TB | 95.75 | 73 TB |
| | | 2 | **79.06** | **816 GB** | 76.92 | 7 GB |
| | | 3 | $\star$ | $\star$ | **77.89** | **615 MB** |

the randomly dependent rows of the XL matrix. In any case, the running times for $r$ and $(r+1)$ (first and second line for each value $m$) gives a running time not more than a factor of 4 apart. On the other hand, the memory reduction is quite drastic, so it may be sensible in practice to choose the higher of the two values. For HybridF$_5$, we are unfortunately not aware of a closed formula to compute its memory consumption. We hence refer to Table 2 for a pointwise comparison and see that $s\ell$ needs far less memory than H$_5$.

Second, $s\ell$ seems to outperform HybridF$_5$ for all values $m \geq 18$ in Table 1. This is in clear violation of the theorems from [AFI+04]. As they show that XL is a redundant version of F$_4$, this basically implies that our upper bound for the number of computations in $s\ell$ is tighter than the corresponding bound for H$_5$. However, we want to stress that for the *same* number of guessed variables ($r$), we had $d_{\mathrm{reg}} < \tilde{D}$ in Table 1, so F$_5$ is faster than $s\ell$ according to the corresponding bounds.

However, inspecting the formulæ for T(H$_5$) and T($s\ell$) closer, we see that both use $(\#rows)^\omega$ with $\omega = 2$. This choice is justified in both cases as they deal with very sparse equations. Now, the term for $\#rows$ is

$$\#rows\mathrm{F}_5(n, m, d_{\mathrm{reg}}) = m\binom{n + d_{\mathrm{reg}} - 1}{d_{\mathrm{reg}}} \text{ and } \#rows\mathrm{SL}(n, m, \tilde{D}) = \binom{n + \tilde{D} - 1}{\tilde{D}} \tag{16}$$

Apparently, for $\tilde{D} = d_{\mathrm{reg}}$ we have $\#rows\mathrm{SL}(n, m, \tilde{D}) \leq \#rows\mathrm{F}_5(n, m, d_{\mathrm{reg}})\ \forall (n, m) \in \mathbb{N}^2 : m \geq n$. So for large values $d_{\mathrm{reg}}$, and small field sizes $|\,\mathbb{F}\,|$, Hybrid-$s\ell$ can outperform H$_5$ according to this expression. However, as we know that XL is a redundant version of F$_5$, this basically implies that $\#rows\mathrm{F}_5(n, m, d_{\mathrm{reg}})$ is too pessimistic and needs to be corrected.

**Cryptographic Challenges.** We now compare HybridF$_5$ with Small Linearization in the case of three $\mathcal{MQ}$-schemes, cf. Table 2. As before, we see for $m = 20, 24$ a lower running time for $s\ell$ than for H$_5$, see previous section for an explanation. However, we want to stress that $s\ell$needs far less memory consumption than H$_5$. For example, we have 15 MB for $m = 20, r = 3$ for $s\ell$ and 12 GB for $m = 20, r = 2$ for H$_5$. Similarly, we see that 816 GB of memory ($m = 24, r = 2$) for H$_5$ is reduced to only 615 MB ($m = 24, r = 3$) for $s\ell$.

# 5 Conclusions

We have demonstrated that eXtended Linearization (XL) and F$_5$ can be fused into a combined algorithm that uses the best from both worlds: Taking a matrix view, we can use black box methods while taking a symbolic view, we get rid of trivial syzygies.

Using this formula, we have estimated the running time for $s\ell$ in *hybrid strategy*, that is by fixing some variables. This showed that $s\ell$ for practical parameters is of comparable speed to HybridF$_5$. More importantly, the memory requirements are drastically reduced. This is true for both algorithm classes,

Table 3: Asymptotic Running time for $\mathrm{F}_5$ and Small Linearization ($s\ell$) for given $n$ variables, $m$ equations, degree of regularity $d_{\mathrm{reg}}$ ($\mathrm{F}_5$), and solving degree $\tilde{D}$, respectively.

$$\mathrm{F}_5 \quad O\left(m\binom{n+d_{\mathrm{reg}}-1}{d_{\mathrm{reg}}}^{\omega}\right) \qquad\qquad s\ell \quad O\left(\binom{n+\tilde{D}-1}{\tilde{D}}^{\omega}\right)$$

namely the original XL algorithm, even when executed with the Wiedemann solver, cf. Table 1 and Hybrid$\mathrm{F}_5$, cf. tables 2. For example, the UOV-challenge with $m = 20$ equations requires 12 GB for $\mathrm{H}_5$, but only 15 MB (!) for $s\ell$.

An interesting open question is the integration of the Mutant strategy into Small Linearization. On the one hand, Mutants are alien to the central idea of $s\ell$: For the latter, we have exploited the very regular structure of the $s\ell$ matrix $A$ while the former are slightly random by definition. On the other hand, there are only very few mutants compared to the overall number of rows in the matrix $A$. Hence, it might be possible to reduce the saturation degree $\tilde{D}$ by one in some situations of practical relevance and hence speed up hybrid $s\ell$ without jeopardizing the small memory consumption. However, more research is needed to answer this questions.

# References


[Abd11] Wael Said Abdelmageed Mohamed. *Improvements for the XL Algorithm with Applications to Algebraic Cryptanalysis.* PhD thesis, TU Darmstadt, Germany, Juni 2011. `http://tuprints.ulb.tu-darmstadt.de/2621/`, 121+xii pages.

[ACFP11] Martin Albrecht, Carlos Cid, Jean-Charles Faugère, and Ludovic Perret. On the relation between the mutant strategy and the normal selection strategy in Gröbner Basis algorithms. Cryptology ePrint Archive, Report 2011/164, 2011. `http://eprint.iacr.org/2011/164/`.

[ACr04] Pil Joong Lee, editor. *Advances in Cryptology — ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*. Springer, 2004. ISBN 3-540-23975-8.

[ADK+10] Wael Said Abdelmageed Mohamed, Jintai Ding, Thorsten Kleinjung, Stanislav Bulygin, and Johannes Buchmann. PWXL: A parallel wiedemann-XL algorithm for solving polynomial equations over GF(2). In Carlos Cid and Jean-Charles Faugere, editors, *Proceedings of the 2nd International Conference on Symbolic Computation and Cryptography (SCC 2010)*, pages 89–100, Jun 2010.

[AFI+04] Gwénolé Ars, Jean-Charles Faugère, Hideki Imai, Mitsuru Kawazoe, and Makoto Sugita. Comparison between XL and Gröbner Basis algorithms. In ACr [ACr04], pages 338–353.

[Alb10] Martin Albrecht. Algorithmic algebraic techniques and their application to block cipher cryptanalysis. PhD thesis, Royal Holloway, University of London, 2010. `http://martinralbrecht.files.wordpress.com/2010/10/phd.pdf`, 176 pages.

[BFP09] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Hybrid approach for solving multivariate systems over finite fields. *In Journal of Mathematical Cryptology*, 3:177–197, 2009.

[BFS04] M. Bardet, J.-C. Faugère, and B. Salvy. On the complexity of Gröbner Basis computation of semi-regular overdetermined algebraic equations. In *Proceedings of the International Conference on Polynomial System Solving*, pages 71–74, 2004.

[BPW06] Johannes Buchmann, Andrei Pyshkin, and Ralf-Philipp Weinmann. Block ciphers sensitive to Gröbner Basis attacks. In *CT-RSA*, volume 3860 of *Lecture Notes in Computer Science*, pages 313–331. David Pointcheval, editor, Springer, 2006. ISBN 3-540-31033-9.

[Buc65] Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal.* PhD thesis, University of Innsbruck, Austria, 1965.

[CKPS00] Nicolas T. Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Bart Preneel, editor, Springer, 2000. Extended Version: `http://www.minrank.org/xlfull.pdf`.

[Cou02] Nicolas Courtois. Higher order correlation attacks, XL algorithm and cryptanalysis of Toyocrypt. In *ICISC*, volume 2587 of *Lecture Notes in Computer Science*, pages 182–199. Pil Joong Lee and Chae Hoon Lim, editors, Springer, 2002.

[Die04] Claus Diem. The XL-algorithm and a conjecture from commutative algebra. In ACr [ACr04]. ISBN 3-540-23975-8.

[Fau99] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases ($F_4$). *Journal of Pure anNational Institute of Standards and Technologyd Applied Algebra*, 139:61–88, June 1999.

[Fau02a] Jean-Charles Faugère. HFE challenge 1 broken in 96 hours. Announcement that appeared in `news://sci.crypt`, 19$^{\text{th}}$ of April 2002.

[Fau02b] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero ($F_5$). In *International Symposium on Symbolic and Algebraic Computation — ISSAC 2002*, pages 75–83. ACM Press, July 2002.

[FJ03] Jean-Charles Faugère and Antoine Joux. Algebraic cryptanalysis of Hidden Field Equations (HFE) using Gröbner Bases. In *Advances in Cryptology — CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 44–60. Dan Boneh, editor, Springer, 2003.

[FM07] Simon Fischer and Willi Meier. Algebraic immunity of S-boxes and augmented functions. In FSE [FSE07], pages 366–381. ISBN 978-3-540-74617-1.

[FOPT10] Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. Algebraic cryptanalysis of McEliece variants with compact keys. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 279–298. Henri Gilbert, editor, Springer, 2010. ISBN 978-3-642-13189-9.

[FSE07] Alex Biryukov, editor. *Fast Software Encryption — FSE 2007*, volume 4593 of *Lecture Notes in Computer Science*. Springer, 2007. ISBN 978-3-540-74617-1.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979. ISBN 0-7167-1044-7 or 0-7167-1045-5.

[JV11] Antoine Joux and Vanessa Vitse. A variant of the F4 algorithm. In *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 356–375. Aggelos Kiayias, editor, Springer, 2011. ISBN 978-3-642-19073-5.

[KS99] Aviad Kipnis and Adi Shamir. Cryptanalysis of the HFE public key cryptosystem. In *Advances in Cryptology — CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 19–30. Michael Wiener, editor, Springer, 1999. `http://www.minrank.org/hfesubreg.ps` or `http://citeseer.nj.nec.com/kipnis99cryptanalysis.html`.

[Laz79] Daniel Lazard. Systems of algebraic equations. In Edward W. Ng, editor, *EUROSAM*, volume 72 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 1979. ISBN 3-540-09519-5.

[Mac16] F. S. Macaulay. *Algebraic theory of modular systems*, volume 19 of *Cambridge Tracts*. Cambridge Univ., 1916.

[MAG] Computational Algebra Group, University of Sydney. *The MAGMA Computational Algebra System for Algebra, Number Theory and Geometry*. `http://magma.maths.usyd.edu.au/magma/`.

[Moh01] T. Moh. On the method of "XL" and its inefficiency to TTM. Cryptology ePrint Archive, Report 2001/047, 2001. http://eprint.iacr.org/2001/047.

[MR02a] S. Murphy and M.J.B. Robshaw. Comments on the security of the AES and the XSL technique. Report for [NES], Document NES/DOC/RHU/WP5/026/1, 2002. https://www.cosic.esat.kuleuven.be/nessie/reports/phase2/Xslbes8_Ness.pdf, 7 pages.

[MR02b] Sean Murphy and Matthew J.B. Robshaw. Essential algebraic structure within the AES. In *Advances in Cryptology — CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 1–16. Moti Yung, editor, Springer, 2002.

[NES] NESSIE: New European Schemes for Signatures, Integrity, and Encryption. Information Society Technologies programme of the European commission (IST-1999-12324). http://www.cryptonessie.org/.

[Per05] Ludovic Perret. A fast cryptanalysis of the isomorphism of polynomials with one secret problem. In *Advances in Cryptology — EUROCRYPT 2005*, Lecture Notes in Computer Science, pages 354–370. Ronald Cramer, editor, Springer, 2005.

[S+10] W. A. Stein et al. *Sage Mathematics Software (Version 4.6)*. The Sage Development Team, 2010. http://www.sagemath.org.

[SAD+08] Mohamed Saied, Wael Said Abdelmageed Mohamed, Jintai Ding, Johannes Buchmann, Stefan Tohaneanu, Ralf-Philipp Weinmann, Daniel Carbarcas, and Dieter Schmidt. Mutantxl and mutant Gröbner Basis algorithm. In *SCC '08: Proceedings of the 1st International Conference on Symbolic Computation and Cryptography*, pages 16–22, 2008.

[SKPI07] Makoto Sugita, Mitsuru Kawazoe, Ludovic Perret, and Hideki Imai. Algebraic cryptanalysis of 58-round SHA-1. In FSE [FSE07], pages 349–365. ISBN 978-3-540-74617-1.

[Tho02] Emmanuel Thomé. Subquadratic computation of vector generating polynomials and improvement of the block wiedemann algorithm. *J. Symb. Comput.*, 33(5):757–775, 2002.

[Wie86] Douglas H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, 32(1):54–62, January 1986.

[YC04] Bo-Yin Yang and Jiun-Ming Chen. All in the XL family: Theory and practice. In *ICISC 2004*, pages 67–86. Springer, 2004.

[YCBC07] Bo-Yin Yang, Chia-Hsin Owen Chen, Daniel J. Bernstein, and Jiun-Ming Chen. Analysis of quad. In FSE [FSE07], pages 290–308. ISBN 978-3-540-74617-1.

# Appendix

## A SolvingDegree $D = 3$

Using degree 3 monomials do not introduce new linear dependencies described by equation (3). But all existing dependencies will be multiplied by $x_1$ to $x_n$. Intuitively it is clear that the number of linearly independent equations is $m\binom{n+2}{3} - \binom{m}{2}n$, *i.e.* we have to cut out $\binom{m}{2}n$ equations. But algorithmically it is not as clear at all. For example if we cut out $x_1x_1f$ and $x_1x_2f$ in the degree 2 step and we multiply the monomials $x_1x_1$ and $x_1x_2$ by all variables $x_i$ we produce the monomial $x_1x_1x_2$ twice. Let us explain the solution to this problem at the following example.

EXAMPLE A.1 *Let $m = n = 3$ and $\mathcal{P}$ be defined by homogeneous quadratic equations $p^{(1)}, p^{(2)}$ and $p^{(3)}$. Obviously $M_i = \{ap^{(i)} \mid a \in \mathtt{Mon}_3\}$ is linearly independent for $i = 1, 2, 3$. Due to following equations*

$$p^{(1)}x_kp^{(2)} = p^{(2)}x_kp^{(1)} \text{ and } p^{(1)}x_kp^{(3)} = p^{(3)}x_kp^{(1)} \tag{17}$$

*$M_1 \cup M_2$ and $M_1 \cup M_3$ are linearly dependent for every $x_k$. If we want to destroy this dependency we have to choose $c \in \mathtt{Mon}_2$ and remove all $dp^{(2)}$ from $M_2$ with $d \in \mathtt{Mon}_3$ and $d$ contains $c$. Analogous we choose $e \in \mathtt{Mon}_2$ and remove all $\ell p^{(3)}$ from $M_3$ with $\ell \in \mathtt{Mon}_3$ and $\ell$ contains $e$. Let define*

$$\pi^k_M(f, g) := \sum_{b_ix_kf \in M} \beta_ib_ix_kf - \sum_{a_jx_kg \in M} \alpha_ja_jx_kg$$

*with $f, g$ as in section 3. Equation (17) can now be formulated by $\pi^k_M(f, g) = 0$ for every $x_k$. Let denote $\alpha_c^{p^{(1)}}$ the coefficient of the monomial $c$ in $p^{(1)}$. Then the following hold.*

$$\begin{aligned}
\pi^k_{M_1\cup M_2}(p^{(1)}, p^{(2)}) &= \alpha_c^{p^{(1)}}x_kcp^{(2)} \\
\pi^k_{M_1\cup M_3}(p^{(1)}, p^{(3)}) &= \alpha_e^{p^{(1)}}x_kep^{(3)} \\
\pi^k_{M_2\cup M_3}(p^{(2)}, p^{(3)}) &= \alpha_e^{p^{(2)}}x_kep^{(3)} - \alpha_c^{p^{(3)}}x_kcp^{(2)}
\end{aligned} \tag{18}$$

*Thus $M_1 \cup M_2 \cup M_3$ is still linearly dependent. We have to choose $u \in \mathtt{Mon}_2$ with $u \neq e$ and remove all $vp^{(3)}$ with $v \in \mathtt{Mon}_3$ and $v$ contains $u$ from $M_3$. This leads to the same situation as in equation (6)-(8) for degree 2.*

$$\begin{aligned}
\pi^k_{M_1\cup M_2}(p^{(1)}, p^{(2)}) &= \alpha_c^{p^{(1)}}x_kcp^{(2)} \\
\pi^k_{M_1\cup M_3}(p^{(1)}, p^{(3)}) &= \alpha_e^{p^{(1)}}x_kep^{(3)} + \alpha_u^{p^{(1)}}x_kup^{(3)} \\
\pi^k_{M_2\cup M_3}(p^{(2)}, p^{(3)}) &= \alpha_e^{p^{(2)}}x_kep^{(3)} + \alpha_u^{p^{(2)}}x_kup^{(3)} - \alpha_c^{p^{(3)}}x_kcp^{(2)}
\end{aligned} \tag{19}$$

*But what happens, if $v = \ell$? For example if we choose $e = x_1x_1$ and $u = x_1x_2$ than $x_2\ x_1x_1 = x_1\ x_1x_2$ holds. In this case $\ell p^{(3)}$ is already removed. So if we choose $x_k$ such that $\ell = x_ke = v$ the equations system (18) still holds and thus $M_1 \cup M_2 \cup M_3$ is still linearly dependent. So we have to choose a new monomial $e' \in \mathtt{Mon}_2$ and destroy all linear dependencies obtained by producing monomials twice by removing $\ell' p^{(3)}$ with $\ell' \in \mathtt{Mon}_3$ and $\ell'$ contains $e'$.*

## B Success probability

With success probability we mean the probability that our algorithm produces the maximal number of equations without *any* reduction to zero. Equation (10) showed that with some probability some equations will still be linearly dependent. This dependencies are not systematic. They are a result of the special choice of the $\mathcal{MQ}$-system $\mathcal{P}$ and so we are not able to eliminate them beforehand. If even one equation is linearly dependent we call the algorithm not successful. Note that this event is no big problem for practical reasons. In most cases the equation system is still solvable if we produce few less linearly independent equations. But nevertheless we want to calculate the success probability of our algorithm to show that it works.

## B.1 Solving Degree $D = 2$

Let us first consider the degree 2 case. The formula $\frac{(q^2-1)(q^2-q)}{q^4}$ given by (10) treats the case $m = n = 3$. This could be generalized for arbitrary $m$. In table 4 we sketch the block structure of matrix (9) in the general case. The first line denote the unknowns for some $c, d, e, f, g, h \in \mathtt{Mon}_2$, empty entries denote zero and $*$ denotes some coefficients $\alpha_i^{p^{(j)}}$.

Table 4: Structure of the systematic dependencies

| | $cp^{(2)}$ | $dp^{(3)}$ | $ep^{(3)}$ | $fp^{(4)}$ | $gp^{(4)}$ | $hp^{(4)}$ | $\cdots$ |
|---|---|---|---|---|---|---|---|
| $\pi_{M_1 \cup M_2}$ | $*$ | | | | | | |
| $\pi_{M_1 \cup M_3}$ | | $*$ | $*$ | | | | |
| $\pi_{M_2 \cup M_3}$ | $*$ | $*$ | $*$ | | | | |
| $\pi_{M_1 \cup M_4}$ | | | | $*$ | $*$ | $*$ | |
| $\pi_{M_2 \cup M_4}$ | $*$ | | | $*$ | $*$ | $*$ | |
| $\pi_{M_3 \cup M_4}$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | |
| $\vdots$ | | | | $\vdots$ | | | |

Let $\mathbf{M}$ by the matrix given by the body of table 4. Obviously $\mathbf{M}$ is regular iff all block matrices on the diagonal are regular. As long as $c \neq d \neq e \neq f \neq g \neq h$ this matrices are uniformly random, as we assume the $\mathcal{MQ}$-system to be uniformly random. The probability of an arbitrary $(m \times m)$ matrix to be regular over $\mathbb{F}_q$ is given by

$$\frac{\prod_{i=0}^{m-1}(q^m - q^i)}{q^{m^2}}.$$

Thus the degree 2 algorithm succeed for some $m > 1$ with probability

$$\frac{\prod_{i=1}^{m-1}\prod_{j=0}^{i-1}(q^i - q^j)}{\prod_{i=1}^{m-1} q^{i^2}}. \tag{20}$$

The general analysis in equation (20) differs from Alg. 5 in one important point — we chose $c = d$ for simplicity and thus the coefficients in the matrix are no longer independent. More precisely every block-matrix on the diagonal is a submatrix of the left upper corner of the next bigger block (see figure 1). Thus the probability of our algorithm to succeed simplifies to

$$\left(\frac{q-1}{q}\right)^{m-1}. \tag{21}$$

This is always larger than equation (20). See table 5 for comparison with experimental data. Note $q = 5$ is the smallest field size for which the algorithm of degree 2 is valid as otherwise $D + 2 < q$ do not hold and thus equation (2) is not proper any longer.

Table 5: Probability of success for $m = n$. We run 10000 experiments each time to get an experimental result for the probability of success.

| | equation (20) | | | equation (21) | | | experimental | | |
|---|---|---|---|---|---|---|---|---|---|
| $q \backslash m$ | 5 | 6 | 7 | 5 | 6 | 7 | 5 | 6 | 7 |
| 11 | 0.665 | 0.599 | 0.540 | 0.683 | 0.621 | 0.564 | 0.680 | 0.620 | 0.574 |
| 29 | 0.866 | 0.835 | 0.805 | 0.869 | 0.839 | 0.810 | 0.868 | 0.839 | 0.808 |
| 127 | 0.969 | 0.961 | 0.953 | 0.969 | 0.961 | 0.954 | 0.972 | 0.963 | 0.957 |

## B.2 Solving Degree $D > 2$

Determining the exact probability of success becomes messy for $D > 2$ as it depends on the special selection of monomials of $\texttt{Mon}_2$ that we use to destroy the systematic dependencies. As our algorithm choose this monomials by random, it becomes hard to analyze the probability of success. See examples B.1 and B.2 for illustration. First let define again

$$\pi_M^k(f,g) := \sum_{b_i x_k f \in M} \beta_i b_i x_k f - \sum_{a_j x_k g \in M} \alpha_j a_j x_k g$$

with $f := \sum_{i=1}^{\sigma} \alpha_i a_i$ and $g := \sum_{i=1}^{\tau} \beta_i b_i$ for $\alpha_i, \beta_j \in \mathbb{F}_q$ and monomials $a_i, b_j \in \texttt{Mon}_2$. After removing some elements from $\texttt{Blow}_D$ we destroyed the systematic dependencies, *i.e.* $\pi_M^{x_k}(f,g) \neq 0$ for all $f, g \in \mathcal{P}$. With some probability over the random choice of the coefficients of $f$ and $g$ there are still dependencies among the destroyed systematic dependencies. Examples B.1 and B.2 will now show that the choice of $\texttt{Mon}_2$ monomials in algorithm 2 affects the exact probability. This monomials are chosen randomly in our algorithm and so it seems infeasible to give a general and exact formula for the probability of success. To ease notation we denote $\alpha_c^{p^{(i)}} x_k c p^{(j)}$ with $c = x_a x_b$ by $ab^i \cdot x_k c^{(j)}$. To keep the examples small we set $n = m = D = 3$.

EXAMPLE B.1 *Let $c = x_1x_2$, $d = x_2x_3$ and $e = x_3x_3$ be the monomials we removed (see Sect. 3). Note, as $x_3\ x_1x_2 = x_1\ x_2x_3$ we need a third monomial $x_3x_3$ to remove the correct number of (linearly dependent) equations. Obviously $12^1 \neq 0$ is a necessary conditions for table 6 to be regular. Thus the first 3 rows are*

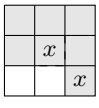
Table 6: Matrix of destroyed systematic dependencies.

| | $x_1x_1x_2^{(2)}$ | $x_1x_2x_2^{(2)}$ | $x_1x_2x_3^{(2)}$ | $x_1x_1x_2^{(3)}$ | $x_1x_2x_2^{(3)}$ | $x_1x_2x_3^{(3)}$ | $x_2x_2x_3^{(3)}$ | $x_2x_3x_3^{(3)}$ | $x_1x_3x_3^{(3)}$ |
|---|---|---|---|---|---|---|---|---|---|
| $\pi^1_{M^{(p^{(1)},p^{(2)})}}$ | $12^1$ | $22^1$ | $23^1$ | 0 | 0 | 0 | 0 | 0 | 0 |
| $\pi^2_{M^{(p^{(1)},p^{(2)})}}$ | $11^1$ | $12^1$ | $13^1$ | 0 | 0 | 0 | 0 | 0 | 0 |
| $\pi^3_{M^{(p^{(1)},p^{(2)})}}$ | 0 | 0 | $12^1$ | 0 | 0 | 0 | 0 | 0 | 0 |
| $\pi^1_{M^{(p^{(1)},p^{(3)})}}$ | 0 | 0 | 0 | $12^1$ | $22^1$ | $23^1$ | 0 | 0 | $33^1$ |
| $\pi^2_{M^{(p^{(1)},p^{(3)})}}$ | 0 | 0 | 0 | $11^1$ | $12^1$ | $23^1$ | $23^1$ | $33^1$ | 0 |
| $\pi^3_{M^{(p^{(1)},p^{(3)})}}$ | 0 | 0 | 0 | 0 | 0 | $12^1$ | $22^1$ | $23^1$ | $13^1$ |
| $\pi^1_{M^{(p^{(2)},p^{(3)})}}$ | $-12^3$ | $-22^3$ | $-23^3$ | $12^2$ | $22^2$ | $23^2$ | 0 | 0 | $33^2$ |
| $\pi^2_{M^{(p^{(2)},p^{(3)})}}$ | $-11^3$ | $-12^3$ | $-13^3$ | $11^2$ | $12^2$ | $23^2$ | $23^2$ | $33^2$ | 0 |
| $\pi^3_{M^{(p^{(2)},p^{(3)})}}$ | 0 | 0 | $-12^3$ | 0 | 0 | $12^2$ | $22^2$ | $23^2$ | $13^2$ |

*linearly independent with probability $\left(1 - \frac{1}{q}\right)^2$. After Gaussian Elimination the remaining lower right $(3 \times 3)$-submatrix do not contain systematic zeros or dependencies and thus the overall probability of 6 to be regular is*

$$\frac{\prod_{i=0}^{2}(q^3 - q^i)}{q^9}\left(1 - \frac{1}{q}\right)^2. \tag{22}$$

EXAMPLE B.2 *Let $c = x_1x_1$ and $d = x_2x_2$ be the monomials we remove (see Sect. 3). Obviously $11^1 \neq 0$ and $(22^1 \vee 22^2) \neq 0$ are necessary conditions for table 7 to be regular. This holds with probability $\left(1 - \frac{1}{q}\right)\left(1 - \frac{1}{q^2}\right)$. The first six lines are now linearly independent and after Gaussian Elimination the lower right $3 \times 3$ submatrix is of the following shape, where white stand for zero, gray for some entry and $x$ for the same entry:*




*The probability of this matrix to be regular is $\left(1 - \frac{1}{q}\right)^2$ and thus the overall probability is*

$$\left(1 - \frac{1}{q}\right)^3\left(1 - \frac{1}{q^2}\right). \tag{23}$$

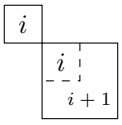
Table 7: Matrix of destroyed systematic dependencies.

| | $x_1x_1x_1^{(2)}$ | $x_1x_1x_2^{(2)}$ | $x_1x_1x_3^{(2)}$ | $x_1x_1x_1^{(3)}$ | $x_1x_1x_2^{(3)}$ | $x_1x_1x_3^{(3)}$ | $x_1x_2x_2^{(3)}$ | $x_2x_2x_2^{(3)}$ | $x_2x_2x_3^{(3)}$ |
|---|---|---|---|---|---|---|---|---|---|
| $\pi^1_{M^{(p^{(1)},p^{(2)})}}$ | $11^1$ | $12^1$ | $13^1$ | 0 | 0 | 0 | 0 | 0 | 0 |
| $\pi^2_{M^{(p^{(1)},p^{(2)})}}$ | 0 | $11^1$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $\pi^3_{M^{(p^{(1)},p^{(2)})}}$ | 0 | 0 | $11^1$ | 0 | 0 | 0 | 0 | 0 | 0 |
| $\pi^1_{M^{(p^{(1)},p^{(3)})}}$ | 0 | 0 | 0 | $11^1$ | $12^1$ | $13^1$ | $22^1$ | 0 | 0 |
| $\pi^2_{M^{(p^{(1)},p^{(3)})}}$ | 0 | 0 | 0 | 0 | $11^1$ | 0 | $12^1$ | $22^1$ | $23^1$ |
| $\pi^3_{M^{(p^{(1)},p^{(3)})}}$ | 0 | 0 | 0 | 0 | 0 | $11^1$ | 0 | 0 | $22^1$ |
| $\pi^1_{M^{(p^{(2)},p^{(3)})}}$ | $-11^3$ | $-12^3$ | $-13^3$ | $11^2$ | $12^2$ | $13^2$ | $22^2$ | 0 | 0 |
| $\pi^2_{M^{(p^{(2)},p^{(3)})}}$ | 0 | $-11^3$ | 0 | 0 | $11^2$ | 0 | $12^2$ | $22^2$ | $23^2$ |
| $\pi^3_{M^{(p^{(2)},p^{(3)})}}$ | 0 | 0 | $-11^3$ | 0 | 0 | $11^2$ | 0 | 0 | $22^2$ |

To study the behavior of our algorithm we first give a lower bound on the probability of success. This bound is bad in some cases (cf. table 8) and thus we secondly give a expected probability of success by using a heuristic.

As in equation (19) our algorithm destroy all systematic dependencies. For $D > 2$ we derive the same matrix as in (4), but with blocks of a larger size $s_i$. Due to our algorithm the $i$-th block still is a submatrix of the upper left part of the block $i+1$ (see figure 1). If block $i$ is regular we can remove

Figure 1: Blockwise dependence of regularity.




the first $s$ columns in the first $s$ rows of block $i+1$ by Gaussian elimination. Thus block $i+1$ is regular iff the obtained block of size $(s_{i+1} \times s_{i+1})$ is regular (see figure 1). Unfortunately we cannot assume the elements of a single block to be uniformly random. Depending on the choice of monomials of $\texttt{Mon}_2$ there could be strong dependencies among the elements, especially for blocks with small $i$ (see App. B.1, Tab. 6 for examples). We can derive the size $s_i$ directly from algorithm 2 and formulate the following corrolary.

**Corollary B.3** *Let $D = 2k + b$ with $b \in \{0, 1\}$ be the solving degree of XL and*

$$z_i := \binom{n+D-3}{D-2} + \sum_{j=1}^{\min\{i-1,k-1\}} \underbrace{(-1)^j \binom{n+D-3-2j}{D-2-2j}\binom{i-1}{j}}_{:=\tau_{ij}}. \tag{24}$$

*The size $s_i$ of the $i$-th block of matrix $\boldsymbol{M}$ with $1 \leq i \leq m-1$ is given by*

$$s_i = s_{i-1} + z_i \text{ with } s_1 := z_1 = \binom{n+D-3}{D-2}.$$

In order to determine the behavior of our algorithm we give a lower bound of the probability of success.

**Lemma B.4** *Let $m, n > 1$ be the number of equations respectively variables of an uniformly random $\mathcal{MQ}$-system, $D$ the solving degree of XL and $z_i$ as defined in corollary B.3. A lower bound for the probability of success of algorithm 2 is given by*

$$\left(1 - \frac{1}{q}\right)^{\sum_{i=1}^{m-1} z_i}. \tag{25}$$

**Algorithm 5** Generating linearly independent equations ($D = 2$)

```
eqn ← {}; miniList ← Mon_2;
for i := 1 to m do
   for μ ∈ miniList do
      eqn ← eqn ∪ {μp^(i)};
      if |eqn| = T − D − 2 then
         Stop;
      end if
   end for
   η ∈_R miniList;
   miniList ← miniList\{η};
end for
```

PROOF. Let $\mathbf{M}$ be the matrix given the rows $\pi_M^k(p^{(i)}, p^{(j)})$ with a fixed set $A$ of elements that are zero and a fixed set $B$ of the remaining elements, *i.e.* $|A| + |B| = z^2$ with $s_1$ (see corollary B.3) the size of the first block in $\mathbf{M}$. All the elements of $B$ are chosen uniformly random. We observe the probability space of all matrices $\mathbf{M}$ such that the probability of being regular is not zero. Note that matrix $\mathbf{M}$ has no systematic dependencies between the rows, *i.e.* for some choice of elements it has to be regular. The worst case structure of a matrix $\mathbf{M}$ regarding the probability of being regular, is an upper (or lower) triangular matrix. The probability of such a $(s_1 \times s_1)$ matrix to be regular is the probability of every diagonal element to be different from zero, *i.e.* $\left(1 - \frac{1}{q}\right)^{s_1}$. Obviously this probability gets better if we introduce dependencies between variables. If for example $x_1 = x_2$ then the probability raise to $\left(1 - \frac{1}{q}\right)^{s_1 - 1}$. It can be easily shown by induction that the probability of success also increase if we increase the number of elements in $B$, *i.e.* if we destroy the triangular structure. As we have $m - 1$ such matrices of size $z_i$ for $1 \leq i \leq m - 1$ to be regular, equation (25) is a lower bound for the probability of success. □

Table 8: Comparison experimental success probability, lower bound and heuristic.

| $m$ | $n$ | $D$ | q | Experimental | No. of Exp. | Lower Bound | Heuristic |
|---|---|---|---|---|---|---|---|
| 3 | 3 | 3 | 11 | 0.684 | $10^4$ | 0.564 | 0.812 |
| 3 | 3 | 3 | 127 | 0.972 | $10^4$ | 0.954 | 0.984 |
| 5 | 5 | 4 | 11 | 0.337 | $10^4$ | 0.006 | 0.659 |
| 5 | 5 | 4 | 127 | 0.738 | $10^4$ | 0.653 | 0.969 |
| 5 | 5 | 5 | 11 | 0.301 | $10^4$ | $3 \cdot 10^{-5}$ | 0.659 |
| 5 | 5 | 5 | 127 | 0.801 | $10^4$ | 0.419 | 0.969 |
| 6 | 6 | 3 | 11 | 0.452 | $10^4$ | 0.057 | 0.593 |
| 6 | 6 | 3 | 127 | 0.938 | $10^4$ | 0.789 | 0.961 |
| 9 | 8 | 3 | 11 | 0.258 | $10^3$ | 0.002 | 0.434 |
| 9 | 8 | 3 | 127 | 0.889 | $10^3$ | 0.603 | 0.938 |

Table 8 shows that (25) is a quite bad bound in some case. The proof of lemma B.4 suggests that for large $n$, $m$ and $D$ the $i$-th block of matrix $\mathbf{M}$, precisely the lower right $(z_i \times z_i)$ submatrix after Gaussian Elimination, is a random matrix and not, as for the lower bound assumed, a triangular matrix. This leads to the following heuristic.

**Heuristic.** For large $n$, $m$ and $D$ the probability of success of algorithm 2 is close to

$$\prod_{i=1}^{m-1} \prod_{j=0}^{z_i - 1} \left(1 - q^{j - z_i}\right). \tag{26}$$

For real world $\mathcal{MQ}$-systems, such as UOV with $m = 26, n = 24, q = 256$ and $D = 12$ the lower bound is almost 0. The heuristic tells us that our algorithm should succeed with probability 0.901. Clearly it will be subject of future research to get sharper bounds on the probability of success. Please mention that even if our algorithm 2 is not successful, *i.e.* there are at least one reduction to zero, this does not imply that algorithm 4 fails. Most times spending one or two extra equations gave us full rank again.