A Simple Deterministic Algorithm for Systems of Quadratic Polynomials over $\mathbb{F}_2$

Charles Bouillaguet
LIP6 laboratory, Sorbonne Université, Paris, France
charles.bouillaguet@lip6.fr

Claire Delaplace
MIS Laboratory, Université de Picardie Jules Verne, Amiens, France
claire.delaplace@u-picardie.fr

Monika Trimoska
MIS Laboratory, Université de Picardie Jules Verne, Amiens, France
monika.trimoska@u-picardie.fr

Abstract

This article discusses a simple deterministic algorithm for solving quadratic Boolean systems which is essentially a special case of more sophisticated methods. The main idea fits in a single sentence: guess enough variables so that the remaining quadratic equations can be solved by linearization (*i.e.* by considering each remaining monomial as an independent variable and solving the resulting linear system). Under strong heuristic assumptions, this finds all the solutions of m quadratic polynomials in n variables with $\tilde{\mathcal{O}}\left(2^{n-\sqrt{2m}}\right)$ operations. Although the best known algorithms require exponentially less time, the present technique has the advantage of being simpler to describe and easy to implement. In strong contrast with the state-of-the-art, it is also quite efficient in practice.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography; Theory of computation → Design and analysis of algorithms

Keywords and phrases Boolean quadratic polynomials, exhaustive search, linear algebra

Supplement Material Source code available at : `https://gitlab.lip6.fr/almasty/mq/`

Funding We acknowledge financial support from the French Agence Nationale de la Recherche under projects "PostCryptum" (ANR20-ASTR-0011) and "GORILLA" (ANR-20-CE39-0002).

1 Introduction

We consider the problem of solving systems of multivariate Boolean quadratic equations. Given a set of quadratic Boolean polynomials $\{f_1, \ldots, f_m\}$, the problem consists in finding a satisfying assignment $\hat{x} \in \{0,1\}^n$ such that $f_i(\hat{x}) = 0$ for all $1 \leq i \leq m$, or determining that no such $\hat{x}$ exists. Each polynomial is represented as the sum of quadratic terms $x_i x_j$, linear terms x_i and a constant term. Because $x^2 = x$ modulo 2, we assume that the exponent of each variable is either 0 or 1.

It is well-known that the problem is NP-complete, with a simple reduction from SAT. It is relevant to the cryptology community because its hardness can be used to build secure "post-quantum" encryption or signature schemes, as there are no known quantum algorithms capable of solving an NP-complete problem in polynomial time. Many such "multivariate" cryptographic schemes have been proposed with concrete sets of parameters, such as HFE [23], UOV [19], MQDSS [8] and variants thereof. Several have been submitted to the ongoing competition launched in 2017 by the (American) National Institute of Standards and Technology in order to select a portfolio of "post-quantum" public-key cryptographic algorithms. For instance, among these, the GeMSS signature scheme [7] exposes a public key consisting of $m = 162$ Boolean quadratic polynomials in $n = 192$ variables. It follows that the relevant range of parameters for cryptographic instances is essentially $m \approx n$ and $n \approx 200$ at this point: solving quadratic Boolean systems of this size is assumed to be intractable (much smaller systems seem completely out of reach as well). Note that in the context of cryptology, the average case complexity is more relevant than the worst case, because cryptographic instances of the problem are (presumably indistinguishable from) random. In addition, there is empirical evidence that random systems are hard to solve. The algorithmic problem is thus both of theoretical and of practical interest, and many algorithms have been proposed to solve it.

In this paper, we present a *decremental* improvement over the state-of-the-art: a simple deterministic algorithm that 1) is a special case of known techniques, and 2) has exponentially worse complexity. It only achieves a sub-exponential advantage over exhaustive search in the average case, but it is extremely simple and quite easy to implement efficiently. The algorithm works as follows:

> Guess sufficiently many variables so that the remaining polynomial system can be solved by *linearization* (*i.e.* by considering each remaining monomial as an independent variable, solving the resulting linear system and checking each solution against the original polynomial system).

More precisely, guess the values of all variables except the $\lfloor \sqrt{2m} - 2 \rfloor$ last ones. There remain strictly less than m (non-constant) monomials of degree less than two in the remaining variables, which enables the use of the linearization technique. This results in a complexity of $\tilde{\mathcal{O}}\left(2^{n-\sqrt{2m}}\right)$.

This algorithm is a particularly simple special case of several other more complex algorithms from the cryptology community, including but not limited to [10, 3, 2, 18]. However, to the best of our knowledge, this simple form was not discussed *per se*.

Related Work

Exhaustive search is the baseline method to solve systems of Boolean quadratic polynomial equations, with a running time $\tilde{\mathcal{O}}(2^n)$ and negligible space complexity. Using several algorithmic tricks and low-level optimizations, it can be implemented extremely efficiently [6]. In particular, the implementation in the libfes-lite library[1], which is state-of-the-art to the best of our knowledge, checks several candidate solutions per CPU cycle, which means that the factors hidden in the big Oh notation are extremely small. It follows that "beating brute force" *in practice*, namely assembling an implementation that runs faster on existing hardware than exhaustive search, for problem sizes that are feasible, is a significant achievement.

Systems that are very underdetermined ($n \geq m^2$) or very overdetermined ($m \geq 0.5n^2$) can be solved in polynomial time by simple techniques [9]. This suggest that $m \approx n$ is the hardest case, and it is common in cryptology (we usually have $m = 2n$ for encryption and $n = 2m$ or $n = 3m$ for signatures). Very overdetermined systems can (heuristically) be solved by linearization: there are $n(n+1)/2$ non-constant Boolean monomials in n variables; consider each one as an independent fresh variable; provided there are as many (linearly independent) quadratic equations, this yields a linear system with a (presumably) small number of solutions. This system can be solved in polynomial time, and each solution reveals a possible value of the variables. On random quadratic systems, we expect to have a single solution.

The next family of algorithms are algebraic manipulation techniques that derive, in a way or another, from the Buchberger algorithm for computing Gröbner bases. Given a Gröbner basis of the original polynomial equations, it is easy to read a potential solution. These algorithms are neither limited to quadratic polynomials nor to the Boolean field. Their average case complexity is notoriously difficult to study, and it requires algebraic assumptions (regularity or semi-regularity) on the input polynomials. The state of the art, at this point, seems to be the F4 and F5 algorithms by Faugère [15, 16]. F4 is essentially a reformulation of the Buchberger algorithm that does batch processing using efficient sparse linear algebra instead of polynomial manipulations. F5 strives to eliminate some useless computations. Bardet, Faugère and Salvy [1] show that a simplified version of F5 computes a Gröbner basis of a regular sequence of quadratic polynomials in $\tilde{\mathcal{O}}\left(2^{4.295n}\right)$ field operations, over any finite field (therefore it "beats brute force" on fields with more than 20 elements). Efficient implementations of F4 are available in off-the-shelf computer algebra systems, notably MAGMA [5]. Faugère's algorithms have been successful in breaking some cryptosystems, most notably an instance of HFE with $n = 80$ variables, which turned to be spectacularly weak against Gröbner basis computations [17]. Variants of these algorithms have been discovered or rediscovered by the crypto community, notably under the form of the XL algorithm by Courtois, Klimov, Patarin and Shamir [10].

All these algorithms have exponential space complexity and existing implementation run into memory limitations even for a moderate number of variables. Implementing them is non-trivial, because they require either sophisticated data-structure for large-degree multivariate polynomials and/or sparse linear algebra over large matrices. Existing implementations are usually available inside full-blown computer algebra systems, which are large and complex software projects.

[1] `https://gitlab.lip6.fr/almasty/libfes-lite`

It is well-known that solving systems by Gröbner basis computation is easier on overdetermined systems. In the extreme, on sufficiently overdetermined polynomial systems, Gröbner basis computations degenerate into Gaussian elimination and work in polynomial time. This is the basis for the "hybrid method" which combines exhaustive search and algebraic techniques: guess the values of some variables, then compute a Gröbner basis of the remaining system which has become overdetermined.

Yang and Chen [24] as well as Bettale, Faugère and Perret [3] discuss the optimal number of variables to fix. The BooleanSolve algorithm of Bardet, Faugère, Salvy and Spaenlehauer [2] is the best embodiment of this idea at this point, with running time $\tilde{\mathcal{O}}\left(2^{0.792n}\right)$ on average, under algebraic assumptions. It guesses some variables, then checks if a polynomial combination of the remaining polynomials is equal to 1. If it is the case, then the guessed values are incorrect (by Hilbert's Nullstellensatz). Checking this is accomplished by deciding whether large sparse linear systems have a solution. The inventors of BooleanSolve claim that it is slower than exhaustive search when $n \leq 200$, which seems to make it practically useless. While conceptually simple, the algorithm is likely hard to implement because it requires a sparse linear system solver for exponentially large matrices. To the best of our knowledge, no implementation has ever been written.

The Crossbred algorithm of Joux and Vitse [18] also belongs to the "guess variables then solve a linear system" family of algorithms. Its asymptotic complexity is not precisely known, but its practical efficiency is spectacular: it has been used to solve a random system with $n = 74$ variables and $m = 148$ equations, which is the current record (this would require about 150 million CPU hours using exhaustive search). There is a public implementation that uses GPUs by Niederhagen, Ning and Yang [22]. It is the first algorithm that has beaten brute-force in practice on random non-overdetermined systems. This algorithm is discussed more in-depth in section 5.1.

The simple algorithm presented in this paper is, to a large extent, a special case of *all* the works surveyed up to this point.

A completely different family of algorithms emerged in 2017 when Lokshtanov, Paturi, Tamaki, Williams and Yu [21] presented a randomized algorithm of complexity $\tilde{\mathcal{O}}\left(2^{0.8765n}\right)$ based on the "polynomial method". In strong contrast with almost all the previous ones, it does not require any assumption on the input polynomials, which is a theoretical breakthrough. The algorithm works by assembling a high-degree polynomial that evaluates to 1 on partial solutions, then approximates it by lower-degree polynomials. The technique was later improved by Björklund, Kaski and Williams [4], reaching $\tilde{\mathcal{O}}\left(2^{0.804n}\right)$, then again by Dinur [13], reaching $\tilde{\mathcal{O}}\left(2^{0.6943n}\right)$.

Noting that the self-reduction that results in this low asymptotic complexity only kicks in for very large values of n, Dinur proposed a simpler, lightweight and "concretely efficient" version of his algorithm for the crypto community with complexity $\mathcal{O}\left(n^2 2^{0.815n}\right)$ using $n^2 2^{0.63n}$ bits of memory [12]. A closer look reveals that this is, in fact, concretely impractical: the algorithm requires more than 2^n operations as long as $n \leq 65$, and for $n \geq 66$ it requires at least 1.5 petabyte of memory (the most powerful computer in the world at the time of writing, `fugaku`, has about 5 petabyte of memory spread over more than 150 000 computing nodes). All other incarnations of the "polynomial method" [21, 4, 13] are even worse from a practical standpoint. They are therefore mostly of theoretical interest.

2 A Toy Example

Consider the following system in 5 variables:

$$
\begin{aligned}
f_1 &= ae + bc + be + cd + a + d + e + 1,\\
f_2 &= ac + ad + ae + bc + bd + ce + de + a + b + d,\\
f_3 &= ad + be + cd + a + b + d + 1,\\
f_4 &= ab + ad + bd + be + b + d + e,\\
f_5 &= ab + ae + bc + bd + cd + ce + de + a + e + 1
\end{aligned}
$$

These polynomials can be seen as vectors in the vector space spanned by all quadratic monomials. The system can thus be written as a matrix:

$$
M = \begin{array}{c}
\begin{array}{ccccccccccccccc} ab & ac & ad & ae & bc & bd & be & cd & ce & de & a & b & c & d & e & 1 \end{array}\\
\left[\begin{array}{ccccccccccccccc}
 & & & 1 & 1 & & 1 & 1 & & & 1 & & & 1 & 1 & 1\\
 & 1 & 1 & 1 & 1 & 1 & & & 1 & 1 & 1 & 1 & & 1 & & \\
 & & 1 & & & & 1 & 1 & & & 1 & 1 & & 1 & & 1\\
1 & & 1 & & & 1 & 1 & & & & & 1 & & 1 & 1 & \\
1 & & & 1 & 1 & 1 & & 1 & 1 & 1 & 1 & & & & 1 & 1
\end{array}\right]
\end{array}
\begin{array}{c} f_1\\ f_2\\ f_3\\ f_4\\ f_5 \end{array}
$$

Next, separate the first $u = 3$ variables, and write the polynomials in $\mathbb{F}_2[a, b, c][d, e]$, *i.e.* as polynomials in d, e whose coefficients are themselves polynomials in a, b, c. This yields a matrix with coefficients in $\mathbb{F}_2[a, b, c]$:

$$
M(a,b,c) = \begin{array}{c}
\begin{array}{cccc} de & d & e & 1 \end{array}\\
\left[\begin{array}{c|c|c|c}
0 & c+1 & a+b+1 & bc+a+1\\
1 & a+b+1 & a+c & ac+bc+a+b\\
0 & a+c+1 & b & a+b+1\\
0 & a+b+1 & b+1 & ab+b\\
1 & b+c & a+c+1 & ab+bc+a+1
\end{array}\right]
\end{array}
\begin{array}{c} f_1\\ f_2\\ f_3\\ f_4\\ f_5 \end{array}
\tag{1}
$$

The columns corresponding to quadratic (resp. linear, constant) monomials in d, e contain constant (resp. linear, quadratic) terms in a, b, c. Perform linear combinations of the rows to put the columns corresponding to quadratic terms in reduced row echelon form :

$$
\widetilde{M}(a,b) = \begin{array}{c}
\begin{array}{cccc} de & d & e & 1 \end{array}\\
\left[\begin{array}{c|c|c|c}
1 & a+b+1 & a+c & ac+bc+a+b\\
\hline
0 & c+1 & a+b+1 & bc+a+1\\
0 & a+c+1 & b & a+b+1\\
0 & a+b+1 & b+1 & ab+b\\
0 & a+c+1 & 1 & ab+ac+b+1
\end{array}\right]
\end{array}
\begin{array}{c} f_2\\ \\ f_1\\ f_3\\ f_4\\ f_2+f_5 \end{array}
$$

Any solution to the initial polynomial system is also a solution of the following equations,

Algorithm 1

Let A denote a $\ell \times v$ matrix of bits and b a size-ℓ vector of bits
Compute a basis $g_1, \ldots, g_\ell$ of $\mathcal{L}$
Write $g_i(y, z) = q_i(y) + yB_iz^t + C_iz^t$
for $\hat{y} \in \{0,1\}^u$ **do**
    for $1 \leq i \leq \ell$ **do**
        $b[i] \leftarrow q_i(\hat{y})$
        $A[i, \cdot] \leftarrow \hat{y}B_i + C_i$
    Solve the linear system $Az^t = b$
    for each solution $\hat{z}$ **do**
        if $0 = f_1(\hat{y}, \hat{z}) = \cdots = f_m(\hat{y}, \hat{z})$ **then**
            return $(\hat{y}, \hat{z})$
return $\perp$

taken by extracting non-pivotal rows:

$$\underbrace{\begin{pmatrix} c+1 & a+b+1 \\ a+c+1 & b \\ a+b+1 & b+1 \\ a+c+1 & 1 \end{pmatrix}}_{L(a,b,c)} \begin{pmatrix} d \\ e \end{pmatrix} = \underbrace{\begin{pmatrix} bc+a+1 \\ a+b+1 \\ ab+b \\ ab+ac+b+1 \end{pmatrix}}_{Q(a,b,c)}$$

Enumerate all the possible values of the first three variables (a, b, c) ; for each combination, solve the linear system $L(a, b, c) \cdot (d, e)^t = Q(a, b, c)$ for (d, e). Any solution of the linear system is automatically a satisfying assignment for $\{f_1, f_3, f_4, f_2 + f_5\}$. Check candidate solutions against f_2 ; they are then guaranteed to satisfy the original system.

The linear system, which is overdetermined, is inconsistent except for $(a, b, c) = (1, 0, 1)$, where it admits a single solution $(e, d) = (0, 0)$. This solution is indeed a valid satisfying assignment.

3 Formal Description and Heuristic Analysis

The ring of Boolean polynomials in n variables $x_1, \ldots, x_n$, hereafter denoted by $\mathcal{B}[x_1, \ldots, x_n]$, is the quotient of the polynomial ring $\mathbb{F}_2[x_1, \ldots, x_n]$ by the ideal spanned by the "field equations" $\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$. As before, we consider a system of m quadratic polynomial equations $\{f_1, \ldots, f_m\}$ in $\mathcal{B}[x_1, \ldots, x_n]$.

Suppose that the n variables $x = (x_1, \ldots, x_n)$ are arbitrarily partitioned in two sets with $x = (y, z)$, $y = (y_1, \ldots, y_u)$, $z = (z_1, \ldots, z_v)$ and $u + v = n$. In the sequel, we choose $v = \lfloor \sqrt{2m} - 2 \rfloor$. This choice guarantees that there are less than m non-constant quadratic monomials in the variables $z_1, \ldots, z_v$ and will make it possible to solve quadratic systems of m equations in v variables by linearization. Indeed, there are $v(v + 1)/2$ such monomials, and this evaluates to $m - 3v/2 - 2$ without rounding v towards zero.

We stated in the introduction that the algorithm consists in guessing the first u variables, and solve the remaining system by linearization. In fact, we present below a slight algorithmic refinement which, in our opinion, leads to a simpler formal exposition.

The ring of Boolean polynomials is in particular a vector space. Denote by $\mathcal{Z}$ the subspace formed by the monomials that contain at most one variable from z. Now, consider the linear span of the input polynomials $\langle f_1, \ldots, f_m \rangle$ and let $\mathcal{L}$ denote its intersection with $\mathcal{Z}$. In other terms, $\mathcal{L}$ contains the linear combinations of the input polynomials in which all monomials contain at most one z_i. The point is that once the values of the $y_1, \ldots, y_u$ variables are fixed, then polynomials in $\mathcal{L}$ depend only on $z_1, \ldots, z_v$ and they have degree at most one.

Computing a basis $g_1, \ldots, g_\ell$ of $\mathcal{L}$ is easy: this can be done by putting the original polynomials in reduced row echelon form with a suitable choice of pivots to eliminate the "bad" monomials $z_i z_j$. The algorithm works by enumerating all possible $\hat{y} \in \{0,1\}^u$, solving $g_1(\hat{y}, z) = \cdots = g_\ell(\hat{y}, z) = 0$, which is a linear system in z, then checking each candidate solution $(\hat{y}, \hat{z})$ against the original quadratic polynomials. This is shown in Algorithm 1.

We write each g_i as $g_i(y, z) = q_i(y) + yB_i z^t + C_i z^t$, where q_i is a quadratic polynomial in $\mathcal{B}[y]$, B_i is a $u \times v$ matrix with coefficients in $\mathbb{F}_2$ and C_i is a length-v vector with coefficients in $\mathbb{F}_2$. This amounts to distinguish monomials that depend only on y, are bilinear in (y, z) or linear in z, respectively.

Assuming that the input polynomials are linearly independent (which seems a mild assumption), then the intersection $\mathcal{L}$ has dimension greater than or equal to $\ell = m - v(v-1)/2$. Our choice of v guarantees that $\ell \geq 5v/2$.

Assembling the linear system (steps 5–7) requires evaluating the ℓ quadratic polynomials q_i's in u variables and performing ℓ matrix-vector products with the B_i's which are of size $u \times v$. This requires $\mathcal{O}(\ell u(u+v))$ operations, while solving the linear system using Gaussian elimination requires $\mathcal{O}(\ell v^2)$ operations.

Let us assume that $m = \Theta(n)$, so that $v = \mathcal{O}(\sqrt{n})$ and $\ell = \mathcal{O}(\sqrt{n})$. Assembling the linear system costs $\mathcal{O}(n^{2.5})$ while solving it costs $\mathcal{O}(n^{1.5})$. Assembling the linear system is dominated by the evaluation of the quadratic polynomials. We show in section 4.1 that it is possible to decrease this cost to $\mathcal{O}(n)$.

The main problem of this algorithm is that it is difficult to bound the total number of iterations of the solution-checking loop (step 9–11). Morally speaking, because the linear system $Az^t = b$ is quite overdetermined, then most of the time there should be no solution at all.

Let us make the heuristic assumption that the b vectors are uniformly random (in fact, they are the result of the evaluation of quadratic polynomials). The image of A is a subspace of dimension less than or equal to v in $\{0,1\}^\ell$, therefore it contains the random vector b with probability less than $2^{v-\ell}$. When the linear system is consistent, it has at most 2^v solutions. This yields a crude upper-bound on the expected number of solutions N of each random linear system:

$$E(N) \leq 2^v \Pr(\text{the linear system is consistent}) = 2^{2v-\ell} \leq 2^{-v/2}$$

It follows that the total time spent checking solutions (in steps 9–11) is asymptotically negligible compared to the rest of the algorithm. The total expected running time of the algorithm, under the heuristic assumption that the b vectors are random, is $\mathcal{O}\left(n^{2.5} 2^{n-\sqrt{2m}}\right)$.

It would be nice to drop the heuristic assumption that the b vectors are random, but this seems difficult. At the very least, the authors of the BooleanSolve algorithm face a comparable problem and resort to different, albeit comparatively strong assumptions: they assume that

the "specialized" polynomial systems $\{f_1(\hat{y}, z), \ldots, f_m(\hat{y}, z)\}$ only deviate from a generic behavior for a sufficiently small fraction of all the possible $\hat{y}$ (they call this the "strong semi-regularity" assumption). This implies in particular that the specialized systems only rarely have solutions, which is exactly what we need as well.

4 Practicality

Algorithm 1 can actually be implemented and executed. Toy implementations in a computer algebra systems such as SageMath are easy to write. A user-friendly and competitive implementation in pure C using low-level optimizations is about 650 lines long (it is accessible in the supplementary material). This is possible because the algorithm itself is simple, and because it does not rely on sophisticated data structures or complex sub-algorithms such as fast multivariate polynomial multiplication, fast multipoint evaluation/interpolation, Gröbner basis computations or large sparse linear system solvers. In addition, its space complexity is negligible and it is trivially parallelizable.

This section discusses what is needed to obtain a serious implementation.

4.1 Faster Polynomial Enumeration Using a Gray Code

As discussed in section 3, the running time of the algorithm is dominated by the need to evaluate quadratic polynomials (step 6 of algorithm 1). Evaluating a quadratic polynomial on $\hat{y} \in \{0,1\}^u$ naively requires $\mathcal{O}\left(u^2\right)$ operations. This can be reduced by enumerating all values of $\hat{y}$ using a *Gray code*, so that only a single bit of $\hat{y}$ changes at each iteration. This technique seems to belong to the folklore. The point is that once the value of $q_i(\hat{y})$ is known, only the monomials that depend on the flipped variable must be reevaluated, and there are only u.

Consider an arbitrary quadratic polynomial $f(x) = c + \sum_{i=1}^{u}\sum_{j=1}^{u} M[i,j]y_i y_j$.

Observe that $f(x+y) = f(x) + f(y) + x\left(M + M^t\right)y^t + f(0)$. Let $\hat{e}_k \in \{0,1\}^u$ denote the vector which is zero everywhere except on the k-th coordinate. Define the partial "derivative" of f with respect to its k-th variable as:

$$\frac{\partial f}{\partial k}(y) = f(y) + f(y + \hat{e}_k).$$

Using the above observation, one quickly find that:

$$\frac{\partial f}{\partial k}(y) = M[k,k] + \sum_{i=1}^{u}(M[i,k] + M[k,i])y_i.$$

These modifications transform Algorithm 1 into Algorithm 2. Maintaining A and b consistent with the single-bit updates to $\hat{y}$ now requires $\mathcal{O}\left(u\right)$ operations on step 10 and $\mathcal{O}\left(v\right)$ operations on step 11, respectively. All in-all, the total cost of assembling the linear system has dropped from $\mathcal{O}\left(n^{2.5}\right)$ to $\mathcal{O}\left(n^{1.5}\right)$, and it now matches that of solving it.

This can again be improved a little by observing that each individual "partial derivative" is evaluated on related inputs (only two bits differ from one evaluation to the next). Taking advantage of this observation leads to the fast exhaustive search algorithm of Bouillaguet, Chen, Cheng, Chou, Niederhagen, Shamir and Yang [6]. Using this technique allows to

Algorithm 2 Rearrangement of Algorithm 1 (incremental updates to A and b)

$\hat{y} \leftarrow 0$ ▷ Setup $\hat{y}$, A and b
for $1 \leq i \leq \ell$ **do**
    $b[i] \leftarrow q_i(0)$
    $A[i, \cdot] \leftarrow C_i$
for $1 \leq i \leq 2^u$ **do** ▷ Main loop
    Solve the linear system $Az^t = b$ and process its solution as before
    $k \leftarrow$ index of the rightmost bit of i ▷ Update $\hat{y}$
    $\hat{y} \leftarrow \hat{y} + \hat{e}_k$
    for $1 \leq i \leq \ell$ **do** ▷ Update A and b
        $b[i] \leftarrow b[i] + \frac{\partial q_i}{\partial k}(\hat{y})$
        $A[i, \cdot] \leftarrow A[i, \cdot] + B_i[k, \cdot]$

update each $b[i]$ in constant time, and brings down the total time needed to update A and b to $\mathcal{O}(n)$.

4.2 Simplifying the Linear Algebra

Implementing these algorithmic optimizations result in a program that spends all its time examining a large number of very small overdetermined linear systems modulo 2 (say, of size 20×10). Optimizing the linear algebra is therefore the next step. The overwhelming majority of the linear systems will have full rank and be inconsistent. Therefore, it makes sense to process them in two phases:

1. Check whether the linear system is both full-rank and inconsistent. If this is the case, we can move on. This is performance-critical.
2. Otherwise, we actually need to compute a particular solution, or even possibly a basis of the solution space. This is a rare occurrence, therefore it is not necessary to optimize this part.

We now argue that the second phase is (heuristically) rarely invoked. As argued in section 3, each linear system $Az^t = b$ is consistent with probability less than $2^{-3v/2}$ (under the heuristic assumption that b is random). We now crudely lower-bound the probability that a random $\ell \times v$ matrix is full rank. This happens when each column is chosen out of the linear span of the previous columns, and the probability of this event is:

$$p := \prod_{j=\ell-v+1}^{\ell} \left(1 - 2^{-j}\right)$$

This is always greater than $\left(1 - 2^{v-\ell}\right)^v$. Let e denote the *excess ratio* $e := \ell/v - 1$. Because we always have $\ell \geq 5v/2$, then $e \geq 3/2$. This implies that $p \geq (1 - 2^{-ev})^v$. Taking an asymptotic expansion for $v \to +\infty$ shows that this is $1 - v2^{-ev} + \mathcal{O}\left(v^2 2^{-2ev}\right)$. Therefore, we expect the proportion of rank-defective linear systems to be smaller than $v2^{v-\ell}$. In practice, this holds quite well.

4.3 Vectorization

Vectorization is a key implementation technique to obtain competitive performance on current hardware, leading to a constant speedup of about 40×.

Consider a Boolean circuit $\mathcal{C}$ with $(\ell+1)v$ input wires for A and b as well as v output wires for z and two extra output wires c and d. The c output wire indicates whether the linear system $Az^t = b$ is *consistent* while the d output wire indicates whether the matrix A is *rank-defective*.

The point is that on a CPU equipped with w-bit registers, w copies of the circuit can be evaluated in parallel on w distinct inputs by performing normal Boolean operations between registers (the i-th copy operates on the i-th bit of all w-bit values). Most current x86-64 CPUs have AVX2 instructions, which allows to perform Boolean operations on 256-bit registers.

This allows to group the iterations of the main loop of the algorithm in batches of size w. The most common situation is that all linear systems in a batch are full-rank and inconsistent (which results in $d \vee c = 0000\ldots000$). In this case, there is nothing to do except moving on to the next batch. Otherwise, the individual elements of the batch must be examined, but this is a rare occurrence.

Such a Boolean circuit is not difficult to obtain, as this amounts to perform an LU factorization of A with partial pivoting. Of course, conditional instructions have to be rewritten: `if b then swap x and y` becomes `delta = b & (x ^ y); x ^= delta; y ^= delta`. This kind of conversion can be done automatically [14].

We actually use the following relaxed specification: if $d = 1$ (the system does not have full-rank), then the other outputs are unspecified. If $d = 0$ and $c = 0$, then the system is inconsistent ; lastly, if $d = 0$ and $c = 1$ then the only solution of the system can be read on the z wires. Giving up on producing a meaningful result when the system is rank-defective allows several simplifications that reduce the size of the circuit. Actually obtaining the solution z when the system is consistent has negligible cost. The circuit we use has approximately $3\ell v^2$ gates.

4.4 Comparison with exhaustive search

The wall-clock running time of fast software implementations of exhaustive search is $T = \alpha 2^n$, for some constant α which depends on the implementation and on the machine. The running time of Algorithm 1 is $T' = P(m)2^{n-2\sqrt{m}}$, for some polynomial P that also depends on the machine. An implementation of Algorithm 1 "beats brute force" when $T/T' > 1$, in other terms when $\alpha 2^{\sqrt{2m}} > P(m)$. This always happens for sufficiently large m, *i.e.* if there are sufficiently many equations, regardless of the number of variables.

We use the exhaustive search implementation in libfes-lite for comparison. Determining the threshold m such that both programs take the same time is a simple matter ; using a single core on the recent laptop of one of the authors, we found that it is $m = 48$, which we consider to be rather low. In that case, both program take about two hours to run.

5 Making it complicated with Higher-Degree Multiples

Algorithm 1 exploits the well-known idea that a polynomial system can be solved in polynomial time by linearization when the number of (linearly independent) equations exceeds the number of non-constant monomials that appear in them. The number of non-constant monomials of degree $\leq D$ in n variables is $N_D(n) = \sum_{i=1}^{D} \binom{n}{i}$. A degree-$D$ Boolean system with m linearly independent polynomials can be linearized when $m \geq N_D(n)$. By guessing variables, we decrease the value of n until this "linearization condition" is satisfied.

A contrasting idea consists in increasing the number of linearly independent polynomial equations. Any polynomial contained in the ideal $I = \langle f_1, \ldots, f_m \rangle$ can indeed by appended to the original equation system without altering its set of solutions. Seen as a vector space, I is spanned by the tf_i, where t ranges across all possible monomials and $1 \leq i \leq m$. Note that when we add these "multiples" of the original polynomials, the degree of the system grows and as a result, the total number of monomials in the system grows as well.

A problem is that the tf_i are not linearly independent, if only because of the trivial relations $f_i f_j + f_j f_i = 0$ and $f_i(f_i + 1) = 0$, not to mention the fact that there are $m2^n$ such multiples while $\mathcal{B}[x]$ has dimension only 2^n.

Let I_D denote the vector space spanned by the tf_i for all monomials t of degree less than or equal to $D-2$ and all $1 \leq i \leq m$. Generally speaking, we expect to have $\dim I_3 = (n+1)m$ and $\dim I_4 = (n^2 - m + n + 1)m/2$, with the assumption that there are no surprising algebraic dependencies between the f_i's except the trivial ones mentioned above. Estimating the dimension of I_D for larger D is more complicated. In general, it depends not only on n and m but on the actual polynomials $f_1, \ldots, f_m$ (it is related to the Hilbert function of the ideal, see [11]). The usual regularity and semi-regularity assumptions state that, up to a certain degree, the trivial linear dependencies are the only ones, which means that $\dim I_D$ is a fixed function of n and m (independently of the actual polynomials), at least when D is not too large.

Several algorithms related to Gröbner basis computation use these multiples [15, 16, 10, 2, 18]. This follows from the observation by Lazard [20] that the tf_i are linearly independent in $\mathbb{K}[x]$ when the f_i form a Gröbner basis; *a contrario*, echelonizing the tf_i of sufficiently high degree provides a way to compute a Gröbner basis of $\{f_1, \ldots, f_m\}$.

In this section, we explore how the common method of using higher-degree multiples can be used to extend Algorithm 1. Two different directions naturally suggest themselves. One of them leads to the Crossbred algorithm of Joux-Vitse [18], the other one gives the FXL [10] and the BooleanSolve [2] algorithms.

5.1 The Crossbred algorithm

The Crossbred algorithm can be described as follows. Proceed exactly as in Algorithm 1, but replace the original polynomials $f_1, \ldots, f_m$ by the degree-D multiples tf_j where t ranges across all monomials of degree less than or equal to $D-2$ and $1 \leq j \leq m$. Algorithm 1 is in fact exactly the Crossbred algorithm with $D = 2$, which is the simplest possible case. Computing the intersection with the subspace $\mathcal{Z}$ yields degree-D polynomials where the $z_1, \ldots, z_v$ variables only occur linearly.

Working with higher-degree multiples enables the use of larger values of v, and therefore reduces the number of variables that must be guessed. For instance, let us see what happens when $D = 3$. Starting from the m initial polynomials, we obtain $(n+1)m$ (hopefully) linearly independent degree-3 multiples. To obtain a basis of $\mathcal{L}$, which is the intersection of the linear span of the multiples with $\mathcal{Z}$, we need to eliminate the "bad" monomials that contain more than one variable of z. These are monomials of the form $z_j z_k, y_i z_j z_k$ and $z_i z_j z_k$. Their number is easy to determine:

$$\#\{z_i z_j\} + \#\{y_i z_i z_j\} + \#\{z_i z_j z_k\} = \binom{v}{2} + (n-v)\binom{v}{2} + \binom{v}{3} = v(v-1)(3n-2v+1)/6.$$

The dimension of $\mathcal{L}$ is thus at least $\ell = m(n+1) - v(v-1)(3n-2v+1)/6$ and as in section 3 we want to have $\ell \geq 5v/2$. The right value of v is therefore the largest positive root of :

$$3nv^2 - 2v^3 - 3nv + 3v^2 + 14v = 6m(n+1)$$

The solutions to this cubic equation are hairy expressions of n and m, but one can check that values slightly larger than $\sqrt{2m}$ are permitted. For instance, with $n = m = 64$, $v = 12$ is admissible, while the original presentation of Algorithm 1 needs $v \leq 8$. However, when $n \approx m$ and $n \to +\infty$, the cubic equation essentially becomes $3nv^2 = 6mn$ and the value of v converges towards $\sqrt{2m}$.

Extensions to higher degrees are possible, however both the analysis and the implementation become more complicated. Joux and Vitse used $D = 4$ to solve a random system with $n = 74$ and $m = 148$, a computational record. Larger values of D require sparse linear algebra on matrices of size $\mathcal{O}\left(n^D\right)$, which quickly become problematic.

5.2 The FXL and BooleanSolve Algorithms

Lastly, we consider a different way to extend Algorithm 1 with higher-degree multiples: guess some variables, then solve the higher-degree system by plain linearization. More precisely,

1. Generate the degree-D multiples of the form tf_i where t ranges across all degree-$(D-2)$ monomials in $z_1, \ldots, z_v$ (only).
2. Guess the values of $y_1, \ldots, y_u$.
3. Solve the remaining degree-D polynomial system in $z_1, \ldots, z_v$ by linearization.

The FXL and BooleanSolve algorithms essentially do this, with steps 1 and 2 in reverse order (they commute). The parameter v is again chosen as the highest value where the linearization condition is satisfied, *i.e.* when there are more linearly independent multiples than $N_D(v)$. The linearized system has size $N_D(v) = \mathcal{O}\left(v^D\right)$, and solving it by Gaussian elimination therefore takes time $\mathcal{O}\left(v^{3D}\right)$. This can be reduced a little by observing that degree-D monomials in $z_1, \ldots, z_v$ are unaffected by guessing $y_1, \ldots, y_u$. They can therefore be eliminated in advance, before guessing any variables, by performing suitable linear combinations of the multiples and focusing on the remaining equations, which only contain monomials of degree $D-1$ in the z_i. Solving this subsystem costs $\mathcal{O}\left(v^{3(D-1)}\right)$.

The problematic part is that the dimension of the vector space spanned by the multiples is difficult to compute, for reasons stated above. But the case where $D = 3$ is easy: we can create $(v+1)m$ multiples of degree three, which we assume to be linearly independent, and there are $(v^2-v+6)(v+1)/6$ monomials of degree less than 3 in $z_1, \ldots, z_v$. The linearization condition is satisfied by $v = \lfloor\sqrt{6m}\rfloor$ in particular, a much higher value than what is possible with the Crossbred algorithm discussed above. The drawback is that the linear systems that have to be solved for each choice of $\hat{y}$ have size $\mathcal{O}(m)$, which is larger than the $\mathcal{O}(\sqrt{m})$ used in the Crossbred algorithm. This yields an algorithm of complexity $\mathcal{O}\left(m^3 2^{n-\sqrt{6m}}\right)$.

While this complexity is asymptotically better than that of Algorithm 1, it seems that the concrete number of operations needed by this degree-3 extension is actually larger when $m \leq 200$. This suggests that using higher degree multiples in this fashion gives a better asymptotic complexity, but is not more efficient for practical parameters. This is consistent with the fact that the Crossbred algorithm is very practical, while the BooleanSolve algorithm has not even been implemented.

References

1 Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. On the complexity of the F5 gröbner basis algorithm. *J. Symb. Comput.*, 70:49–70, 2015. doi:10.1016/j.jsc.2014.09.025.

2 Magali Bardet, Jean-Charles Faugère, Bruno Salvy, and Pierre-Jean Spaenlehauer. On the complexity of solving quadratic boolean systems. *J. Complexity*, 29(1):53–75, 2013. doi:10.1016/j.jco.2012.07.001.

3 Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Hybrid approach for solving multivariate systems over finite fields. *J. Math. Cryptol.*, 3(3):177–197, 2009. doi:10.1515/JMC.2009.009.

4 Andreas Björklund, Petteri Kaski, and Ryan Williams. Solving systems of polynomial equations over GF(2) by a parity-counting self-reduction. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 26:1–26:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.26.

5 Wieb Bosma, John J. Cannon, and Catherine Playoust. The Magma Algebra System I: The User Language. *J. Symb. Comput.*, 24(3/4):235–265, 1997.

6 Charles Bouillaguet, Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang. Fast exhaustive search for polynomial systems in $\mathbb{F}_2$. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2010. doi:10.1007/978-3-642-15031-9_14.

7 Antoine Casanova, Jean-Charles Faugère, Gilles Macario-Rat, Jacques Patarin, Ludovic Perret, and Jocelyn Ryckeghem. GeMSS: A Great Multivariate Short Signature. Research report, UPMC - Paris 6 Sorbonne Universités ; INRIA Paris Research Centre, MAMBA Team, F-75012, Paris, France ; LIP6 - Laboratoire d'Informatique de Paris 6, December 2017. URL: https://hal.inria.fr/hal-01662158.

8 Ming-Shing Chen, Andreas Hülsing, Joost Rijneveld, Simona Samardjiska, and Peter Schwabe. From 5-pass *MQ*-based identification to *MQ*-based signatures. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 135–165, 2016. doi:10.1007/978-3-662-53890-6_5.

9 Nicolas T. Courtois, Louis Goubin, Willi Meier, and Jean-Daniel Tacier. Solving underdefined systems of multivariate quadratic equations. In David Naccache and Pascal Paillier, editors, *Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings*, volume 2274 of *Lecture Notes in Computer Science*, pages 211–227. Springer, 2002. doi:10.1007/3-540-45664-3_15.

10 Nicolas T. Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer, 2000. doi:10.1007/3-540-45539-6_27.

11 David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra, 3/e (Undergraduate Texts in Mathematics)*. Springer-Verlag, Berlin, Heidelberg, 2007.

12 Itai Dinur. Cryptanalytic applications of the polynomial method for solving multivariate equation systems over GF(2). In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 374–403. Springer, 2021. doi:10.1007/978-3-030-77870-5_14.

13 Itai Dinur. Improved algorithms for solving polynomial systems over GF(2) by multiple parity-counting. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2550–2564. SIAM, 2021. doi:10.1137/1.9781611976465.151.

14 Amr Elmasry and Jyrki Katajainen. Lean programs, branch mispredictions, and sorting. In Evangelos Kranakis, Danny Krizanc, and Flaminia L. Luccio, editors, *Fun with Algorithms - 6th International Conference, FUN 2012, Venice, Italy, June 4-6, 2012. Proceedings*, volume 7288 of *Lecture Notes in Computer Science*, pages 119–130. Springer, 2012. doi:10.1007/978-3-642-30347-0_14.

15 Jean-Charles Faugère. A new efficient algorithm for computing grobner bases (f4). *Journal of Pure and Applied Algebra*, 139(1-3):61–68, 1999.

16 Jean Charles Faugère. A new efficient algorithm for computing gröbner bases without reduction to zero (f5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, page 75–83, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/780506.780516.

17 Jean-Charles Faugère and Antoine Joux. Algebraic cryptanalysis of hidden field equation (HFE) cryptosystems using gröbner bases. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 44–60. Springer, 2003. doi:10.1007/978-3-540-45146-4_3.

18 Antoine Joux and Vanessa Vitse. A Crossbred Algorithm for Solving Boolean Polynomial Systems. In *NuTMiC*, volume 10737 of *Lecture Notes in Computer Science*, pages 3–21. Springer, 2017. https://eprint.iacr.org/2017/372.pdf.

19 Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced oil and vinegar signature schemes. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 206–222. Springer, 1999. doi:10.1007/3-540-48910-X_15.

20 Daniel Lazard. Gröbner-bases, gaussian elimination and resolution of systems of algebraic equations. In J. A. van Hulzen, editor, *Computer Algebra, EUROCAL '83, European Computer Algebra Conference, London, England, March 28-30, 1983, Proceedings*, volume 162 of *Lecture Notes in Computer Science*, pages 146–156. Springer, 1983. doi:10.1007/3-540-12868-9_99.

21 Daniel Lokshtanov, Ramamohan Paturi, Suguru Tamaki, R. Ryan Williams, and Huacheng Yu. Beating brute force for systems of polynomial equations over finite fields. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2190–2202. SIAM, 2017. doi:10.1137/1.9781611974782.143.

22 Ruben Niederhagen, Kai-Chun Ning, and Bo-Yin Yang. Implementing joux-vitse's crossbred algorithm for solving MQ systems over GF(2) on gpus. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, volume 10786 of *Lecture Notes in Computer Science*, pages 121–141. Springer, 2018. doi:10.1007/978-3-319-79063-3_6.

23 Jacques Patarin. Hidden fields equations (HFE) and isomorphisms of polynomials (IP): two new families of asymmetric algorithms. In Ueli M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, volume 1070 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 1996. doi:10.1007/3-540-68339-9_4.

24 Bo-Yin Yang and Jiun-Ming Chen. Theoretical analysis of XL over small fields. In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *Information Security and Privacy: 9th Australasian Conference, ACISP 2004, Sydney, Australia, July 13-15, 2004. Proceedings*, volume 3108 of *Lecture Notes in Computer Science*, pages 277–288. Springer, 2004. doi:10.1007/978-3-540-27800-9_24.