A variant of the F4 algorithm

Antoine Joux[1,2] and Vanessa Vitse[2]

[1] Direction Générale de l'Armement (DGA)
[2] Université de Versailles Saint-Quentin, Laboratoire PRISM, 45 av. des États-Unis, 78035 Versailles cedex, France
antoine.joux@m4x.org vanessa.vitse@prism.uvsq.fr

Abstract. Algebraic cryptanalysis usually requires to find solutions of several similar polynomial systems. A standard tool to solve this problem consists of computing the Gröbner bases of the corresponding ideals, and Faugère's F4 and F5 are two well-known algorithms for this task. In this paper, we adapt the "Gröbner trace" method of Traverso to the context of F4. The resulting variant is a heuristic algorithm, well suited to algebraic attacks of cryptosystems since it is designed to compute with high probability Gröbner bases of a set of polynomial systems having the same shape. It is faster than F4 as it avoids all reductions to zero, but preserves its simplicity and its efficiency, thus competing with F5.

Key words: Gröbner basis, Gröbner trace, F4, F5, multivariate cryptography, algebraic cryptanalysis

1 Introduction

The goal of algebraic cryptanalysis is to break cryptosystems by using mathematical tools coming from symbolic computation and modern algebra. More precisely, an algebraic attack can be decomposed in two steps: first the cryptosystem and its specifics have to be converted into a set of multivariate polynomial equations, then the solutions of the obtained polynomial system have to be computed. The security of a cryptographic primitive thus strongly relies on the difficulty of solving the associated polynomial system. These attacks have been proven to be very efficient for both public key or symmetric cryptosystems and stream ciphers (see [2] for a thorough introduction to the subject).

In this article, we focus on the polynomial system solving part. It is well known that this problem is very difficult (NP-hard in general). However, for many instances coming from algebraic attacks, the resolution is easier than in the worst-case scenario. Gröbner bases, first introduced in [6], are a fundamental tool for tackling this problem. Historically, one can distinguish two families of Gröbner basis computation algorithms: the first one consists of developments of Buchberger's original algorithm [8, 14, 15, 19], while the second can be traced back to the theory of elimination and resultants and relies on Gaussian elimination of Macaulay matrices [10, 24–26]. Which algorithm to use depends on the shape and properties of the cryptosystem and its underlying polynomial system (base field, degrees of the polynomials, number of variables, symmetries...).

Faugère's F4 algorithm [14] combines ideas from both families. It is probably the most efficient installation of Buchberger's original algorithm, and uses Gaussian elimination to speed up the time-consuming step of "critical pair" reductions. It set new records in Gröbner basis computation when it was published a decade ago, and its implementation in Magma [5] is still considered as a major reference today. However, F4 shares the main drawback of Buchberger's algorithm: it spends a lot of time computing useless reductions. This issue was addressed by Faugère's next algorithm, F5 [15], which first rose to fame with the cryptanalysis of the HFE Challenge [16]. Since then, it has been successfully used to break several other cryptosystems (e.g. [4, 17]), increasing considerably the popularity of algebraic attacks. It is often considered as the most efficient algorithm for computing Gröbner bases over finite fields and its performances are for the main part attributable to the use of an elaborate criterion. Indeed, the F5 criterion allows to skip much more unnecessary critical pairs than the classical Buchberger's criteria [7]; actually it eliminates a priori all reductions to zero under the mild assumption that the system forms a semi-regular sequence [3]. Nevertheless, this comes at the price of degraded performances in the reduction step: during the course of the F5 algorithm, many reductions are forbidden for "signature" compatibility conditions, giving rise to polynomials that are either redundant (not "top-reduced" [13]), or whose tails are left almost unreduced.

In many instances of algebraic attacks, one has to compute Gröbner bases for numerous polynomial systems that have the same shape, and whose coefficients are either random or depend on a relatively small number of parameters. In this context, one should use specifically-devised algorithms that take this information into account. A first idea would be to compute a parametric or *comprehensive Gröbner basis* [30]; its specializations yield the Gröbner bases of all the ideals in a parametric polynomial system. However, for the instances arising in cryptanalysis, the computational cost of a comprehensive Gröbner basis is prohibitive. Another method was proposed by Traverso in the context of modular computations of rational Gröbner bases [29]: by storing the *trace* of an initial execution of the Buchberger algorithm, one can greatly increase the speed of almost all subsequent computations. Surprisingly, it seems that this approach was never applied to cryptanalysis until now.

We present in this paper how a similar method allows to avoid all reductions to zero in the F4 algorithm after an initial precomputation. A list of relevant critical pairs is extracted from a first F4 execution, and is used for all following computations; the precomputation overhead is largely compensated by the efficiency of the F4 reduction step, yielding theoretically better performances than F5. This algorithm is by nature probabilistic: the precomputed list is in general not valid for all the subsequent systems. One of the main contribution of this article is to give a complete analysis of this F4 variant and to estimate its probability of failure, which is usually very small.

The paper is organized as follows. After recalling the basic structure of Buchberger-type algorithms, we explain in section 2 how to adapt it to the context of several systems of the same shape. The detailed pseudo-code of our

variant of F4, which consists of the two routines `F4Precomp` and `F4Remake` for the first precomputation and the subsequent iterations respectively, is given in the appendix. In section 3, we recall the mathematical frame for the otherwise imprecise notion of "similar looking systems" and derive probability estimates for the correctness of our algorithm, depending on the type of the system and the size of the base field. We also compare the complexities of our variant and of F5, and explain when it is better to use our algorithm. The last section is devoted to applications: the first example comes from the index calculus method of [20] and is a typical case where our algorithm outperforms F4 and F5. We then show how it fits into the hybrid approach of [4] and consider the example of the cryptanalysis of the UOV signature scheme [22]. The next example is provided by the Kipnis-Shamir attack on the MinRank problem: we compare our results to those of [17]. Finally, we evaluate the performances of our F4 variant on the classical `Katsura` benchmarks. We would like to mention that the Gröbner trace method has already been applied to Faugère's algorithms for the decoding of binary cyclic codes [1]; however, the analysis therein was limited to this very specific case, and no implementation details nor probability estimates were given. This idea was then independently rediscovered in [20], where it was applied to the discrete log problem on elliptic curves.

2 The F4 variant

2.1 Description of the algorithm

We begin with the standard characterization of Gröbner bases. The notations LM and LT stand for the leading monomial and the leading term of a polynomial and $\vee$ denotes the least common multiple of two monomials.

Theorem 1 ([8]) *A family $G = \{g_1, \ldots, g_s\}$ in $\mathbb{K}[X_1, \ldots, X_n]$ is a Gröbner basis if and only if for all $1 \leq i < j \leq s$, the remainder of $S(g_i, g_j)$ on division by G is zero, where $S(g_i, g_j)$ is the S-polynomial of g_i and g_j:*

$$S(g_i, g_j) = \frac{LM(g_i) \vee LM(g_j)}{LT(g_i)} g_i - \frac{LM(g_i) \vee LM(g_j)}{LT(g_j)} g_j$$

It is straightforward to adapt this result into the Buchberger's algorithm [8], which outputs a Gröbner basis of an ideal $I = \langle f_1, \ldots, f_r \rangle$: one computes iteratively the remainder by G of every possible S-polynomial and appends this remainder to G whenever it is different from zero. In the following, we will rather work with critical pairs instead of S-polynomials: the critical pair of two polynomials f_1 and f_2 is defined as the tuple $(lcm, u_1, f_1, u_2, f_2)$ where $lcm = LM(f_1) \vee LM(f_2)$ and $u_i = \frac{lcm}{LM(f_i)}$.

The reduction of critical pairs is by far the biggest time-consuming part of the Buchberger's algorithm. The main idea of Faugère's F4 algorithm is to use linear algebra to simultaneously reduce a large number of pairs. At each iteration step, a Macaulay-style matrix is constructed, whose columns correspond to monomials

and rows to polynomials. This matrix contains the products $(u_i f_i)$ coming from the selected critical pairs (classically, all pairs with the lowest total degree lcm, but other selection strategies are possible) and also all polynomials involved in their reductions, which are determined during the preprocessing phase. By computing the reduced row echelon form of this matrix, we obtain the reduced S-polynomials of all pairs considered. This algorithm, combined with an efficient implementation of linear algebra, yields very good results.

As mentioned in the introduction, F4 has the drawback of computing many useless reductions to zero, even when the classical criteria of Buchberger [7] are taken into account. But when one has to compute several Gröbner bases of similar polynomial systems, it is possible to avoid, in most cases, all reductions to zero by means of a precomputation on the first system.

Here is the outline of our F4 variant:

1. For precomputation purposes, run a standard F4 algorithm on the first system, with the following modifications:
   - At each iteration, store the list of all polynomial multiples (u_i, f_i) coming from the critical pairs.
   - During the row echelon computing phase, reductions to zero correspond to linear dependency relations between the rows of the matrix; for each such relation, remove a multiple (u_i, f_i) from the stored list (some care must be taken in the choice of the multiple, see the appendix for details).
2. For each subsequent system, run a F4 computation with these modifications:
   - Do not maintain nor update a queue of untreated pairs.
   - At each iteration, instead of selecting pairs from the queue, pick directly from the previously stored list all the relevant multiples (u_i, f_i).

We give in the appendix the detailed pseudo-code of the `F4Precomp` algorithm which performs the precomputation, and of the `F4Remake` algorithm which is used for the subsequent systems.

2.2 Additional features

For the sake of concision, the pseudo-code of `F4Remake` given in the appendix does not contain a test of the correctness of the computation, except for the basic verification of line 9. More checks could be easily included: for instance, it is possible to store during the precomputation the leading monomials of the generators created at each step, and check in `F4Remake` if the new generators have the correct LM. In case of a failed computation, proper error handling would be recommended, e.g. by resuming the computation with the standard F4. At the end of the execution, a last check would be to verify whether the result (which is always a basis of the ideal) is indeed a Gröbner basis. This can be quite expensive, but is usually unnecessary: indeed, the output is always correct if the sets of leading monomials of the bases returned by `F4Remake` and `F4Precomp` coincide, assuming that the precomputation behaved generically (see section 3). Anyway, when the ideal is zero-dimensional with a small degree (as

is often the case in the context of algebraic attacks), a verification is almost immediate.

It is also possible to store during precomputation all the relevant polynomial multiples appearing in the matrices M, instead of only those arising from the critical pairs. This increases greatly the size of `F4Precomp`'s output, but allows to skip the preprocessing phase in `F4Remake`. However, the gain provided by this optimization is relatively minor, since the cost of the preprocessing is usually small compared to the computation of the reduced row echelon form. A different approach is outlined in [1]: instead of recording the information about relevant polynomials in a file, the precomputation directly outputs a program (in the C language) containing the instructions for the subsequent computations. Clearly, this code generating technique is much more complicated, but should be faster even when the compilation time of the output program is taken into account.

3 Analysis of the algorithm and complexity

3.1 Similar systems

Our algorithm is designed to be applied on many systems of the "same shape". If $\{f_1, \ldots, f_r\}$ and $\{f'_1, \ldots, f'_r\}$ are two similarly-looking polynomial systems, we want to estimate the probability that our algorithm computes the Gröbner basis of the second system, the precomputation having been done with the first system. This requires some more precise definitions.

Definition 2 *A generic polynomial F of degree d in n variables $X_1, \ldots, X_n$ is a polynomial with coefficients in $\mathbb{K}[\{Y_{i_1,\ldots,i_n}\}_{i_1+\ldots+i_n \leq d}]$ of the form*

$$F = \sum Y_{i_1,\ldots,i_n} X_1^{i_1} \ldots X_n^{i_n}.$$

A generic polynomial is thus a polynomial in which each coefficient is a distinct variable. Such polynomials are interesting to study because a system of random polynomials $f_1, \ldots, f_r$ (i.e. such that each coefficient is random) of total degree $d_1, \ldots, d_r$ respectively, is expected to behave like the corresponding system of generic polynomials.

Let $F_1, \ldots, F_r$ be a system of generic polynomials. If we consider F_i as an element of $\mathbb{K}(\underline{Y})[\underline{X}]$, we can compute the Gröbner basis of this system with the F4 algorithm, at least theoretically (in practice, the rational fraction coefficients will likely become extremely large). Now let $f_1, \ldots, f_r$ be a random system with $\deg(f_i) = \deg(F_i)$. We say that $f_1, \ldots, f_r$ behaves generically if we encounter the same number of iterations as with $F_1, \ldots, F_r$ during the computation of its Gröbner basis using F4, and if the same number of new polynomials with the same leading monomials are generated at each step of the algorithm.

We will now translate this condition algebraically. Assume that the system $f_1, \ldots, f_r$ behaves generically until the $(i-1)$-th step; this implies in particular that the critical pairs involved at step i for both systems are similar, in the following sense: $(lcm, u_1, p_1, u_2, p_2)$ is similar to $(lcm', u'_1, p'_1, u'_2, p'_2)$ if $LM(p_1) =$

$LM(p'_1)$ and $LM(p_2) = LM(p'_2)$ (so that $u_i = u'_i$ and $lcm = lcm'$). Let M_g be the matrix of polynomial multiples constructed by F4 at step i for the generic system, and M be the one for $f_1, \ldots, f_r$. It is possible that after the preprocessing M is smaller than M_g, but for the purpose of our discussion, we may assume that the missing polynomial multiples are added to M; the corresponding rows will have no effect whatsoever later in the algorithm. Thus the k-th rows of M and M_g, seen as polynomials, have identical leading monomial; we note s the number of distinct leading monomials in M (or M_g). Remark that the matrices constructed by F4 are usually almost upper triangular, so that s is close to the number of rows. If we compute the reduced row echelon form of M_g, up to a well-chosen permutation of columns we obtain the following matrix $\tilde{M}_g$ where $r = \ell + s$ is the rank of M_g. Using the same transformations on M with adapted coefficients, we obtain a matrix $\tilde{M}$ where B is a matrix with ℓ columns.

$$\tilde{M}_g = \left(\begin{array}{c|c|c} I_s & 0 & A_{g,1} \\ \hline 0 & I_\ell & A_{g,2} \\ \hline 0 & 0 & 0 \end{array}\right) \qquad \tilde{M} = \left(\begin{array}{c|cc} I_s & \multicolumn{2}{c}{B_1} \\ \hline 0 & B & B_2 \end{array}\right)$$

Then the system $f_1, \ldots, f_r$ behaves generically at step i if and only if this matrix B has full column rank. Finally, the condition for generic behavior is that at each step, the corresponding matrix B has full column rank. Heuristically, since the system is random, we will assume that these matrices B are random. This hypothesis will allow us to give estimates for the probability that a system behaves generically, using the following easy lemma:

Lemma 3 *Let $M = (m_{ij}) \in \mathcal{M}_{n,\ell}(\mathbb{F}_q)$, $n \geq \ell$, be a random matrix, i.e. such that the coefficients m_{ij} are chosen randomly, independently and uniformly in $\mathbb{F}_q$. Then M has full rank with probability $\prod_{i=n-\ell+1}^{n}(1 - q^{-i})$. This probability is greater than the limit*

$$c(q) = \prod_{i=1}^{\infty}(1 - q^{-i}) = 1 - 1/q + \underset{q\to\infty}{O}(1/q^2).$$

Since a system behaves generically if and only if all the matrices B have full rank, we obtain the probability that our F4 variant works successfully:

Theorem 4 *The algorithm* `F4Remake` *outputs a Gröbner basis of a random system $f_1, \ldots, f_r \in \mathbb{F}_q[\underline{X}]$ with a probability that is heuristically greater than $c(q)^{n_{step}}$, assuming that the precomputation has been done with* `F4Precomp` *in n_{step} steps, for a system $f_1^0, \ldots, f_r^0 \in \mathbb{F}_q[\underline{X}]$ that behaves generically.*

This estimate is relevant as soon as the distribution of the matrices B is sufficiently close to the uniform one and the correlation between the steps is small enough.

For a system of generic polynomials, it is known that the number of steps n_{step} during the execution of F4 (for a degree-graded monomial order) is at most

equal to the degree of regularity d_{reg} of the homogenized system, which is smaller than the Macaulay bound $\sum_{i=1}^{r}(\deg F_i - 1) + 1$ [24]; this bound is sharp when the system is underdetermined. Since $c(q)$ converges to 1 when q goes to infinity, for a fixed degree of regularity the probability of success of our algorithm will be very close to 1 when the base field $\mathbb{F}_q$ is sufficiently large.

In practice, it is rather uncommon to deal with completely random polynomials. For many applications, the involved polynomial systems actually depend on some random parameters, hence a more general framework is the following:

Definition 5 *Let V be an algebraic variety in $\mathbb{K}^\ell$ and $F_1, \ldots, F_r$ be polynomials in $\mathbb{K}(V)[\underline{X}]$, where $\mathbb{K}(V)$ is the function field of V. We call the image of the map $V \to \mathbb{K}[\underline{X}]^r, \quad y \mapsto (F_1(y), \ldots, F_r(y))$ a parametric family (or family for short) of systems. We call the system $(F_1, \ldots, F_r)$ the generic parametric system of the family.*

A system of generic polynomials is of course a special case of a generic parametric system. As above, the `F4Remake` algorithm will give correct results for systems $f_1, \ldots, f_r$ in a family that behave like its associated generic parametric system. The probability for this is difficult to estimate since it obviously depends on the family considered, but is usually better than for systems of generic polynomials. An important class of examples is when the highest degree homogeneous part of the F_i has coefficients in $\mathbb{K}$ (instead of $\mathbb{K}(V)$). Then all systems of this parametric family behave generically until the first fall of degree occurs. As a consequence, the probability of success of our algorithm can be quite good even when the base field is relatively small, see section 4.2 for an example.

3.2 Change of characteristic

Another application of our algorithm is the computation of Gröbner bases of "random" polynomial systems over a large field, using a precomputation done over a small finite field. Even for a single system $f_1, \ldots, f_r$ in $\mathbb{F}_p[\underline{X}]$, it is sometimes more advantageous to precompute the Gröbner basis of a system $f'_1, \ldots, f'_r$ with $\deg f_i = \deg f'_i$ in $\mathbb{F}_{p'}[\underline{X}]$ for a small prime p', and then use `F4Remake` on the initial system, than to directly compute the Gröbner basis with F4. The estimated probabilities derived in section 3.1 do not directly apply to this situation, but a similar analysis can be done.

We recall that for every prime number p, there exists a well-defined reduction map $\mathbb{Q}[\underline{X}] \to \mathbb{F}_p[\underline{X}]$, which sends a polynomial P to $\bar{P} = cP \mod p$, where $c \in \mathbb{Q}$ is such that cP belongs to $\mathbb{Z}[\underline{X}]$ and is primitive (i.e. the gcd of its coefficients is one). Let $I = \langle f_1, \ldots, f_r \rangle$ be an ideal of $\mathbb{Q}[\underline{X}]$, and let $\bar{I} = \langle \bar{f}_1, \ldots, \bar{f}_r \rangle$ be the corresponding ideal in $\mathbb{F}_p[\underline{X}]$; we note $\{g_1, \ldots, g_s\}$ the minimal reduced Gröbner basis of I. According to [12], we say that p is a "lucky" prime if $\{\bar{g}_1, \ldots, \bar{g}_s\}$ is the minimal reduced Gröbner basis of $\bar{I}$, and "unlucky" otherwise. There is a weaker, more useful notion (adapted from [27]) of "F4 (or weak) unlucky prime": a prime number p is called so if the computation of the Gröbner bases of I and $\bar{I}$ with F4 differs. By doing the same analysis as in section 3.1, we can show that p is weakly

unlucky if and only if one of the above-defined matrices B is not of full rank. As before, these matrices can heuristically be considered as random and thus we obtain that the probability that a prime p is not weakly unlucky, is bounded from below by $c(p)^{n_{step}}$. So, if we want to compute the Gröbner basis of a system $f_1, \ldots, f_r \in \mathbb{F}_p[\underline{X}]$ where p is a large prime, we can lift this system to $\mathbb{Q}[\underline{X}]$ and then reduce it to $f'_1, \ldots, f'_r \in \mathbb{F}_{p'}[\underline{X}]$ where p' is a small prime number. Then we execute `F4Precomp` on the latter system and use the precomputation on the initial system with `F4Remake`. This will produce the correct result if p and p' are not weakly unlucky, thus p', while small enough so that the precomputation takes the least time possible, must be large enough so that the probability $c(p')^{n_{step}}$ is sufficiently close to 1. In practice, this last approach should be used whenever possible. If one has to compute several Gröbner bases over a large field $\mathbb{F}_q$ of systems of the same parametric family, the precomputation should not be done over $\mathbb{F}_q$, but rather over a smaller field. We will adopt this strategy in almost all the applications presented in section 4.

3.3 Precomputation correctness

The output of `F4Precomp` is correct if the first system behaves generically; we have seen that this occurs with a good probability $c(q)^{n_{step}}$. We will now consider what can happen when the precomputation is not correct, and how to detect it. We can, at least theoretically, run `F4Remake` on the generic system; following Traverso's analysis [29] two cases are then possible:

1. This would produce an error. Then `F4Remake` will fail for most subsequent systems, so this situation can be easily detected after very few executions (the probability of no detection is very low: rough estimates have been given in [29] for the different characteristic case). More precisely, as soon as an error occurs with `F4Remake`, one has to determine whether the precomputation was incorrect or the current system does not behave generically. This can be done by looking at the course of the algorithm: if at some step `F4Remake` computes more new generators than `F4Precomp`, or generators with higher leading monomials, then clearly it is the precomputation which is incorrect.
2. The computation would succeed but the resulting output is not a Gröbner basis. This situation, while unlikely, is more difficult to detect: one has to check that the outputs of `F4Remake` on the first executions are indeed Gröbner bases. If there is a system for which this is not true, then the precomputation is incorrect.

Alternatively, one can run `F4Precomp` on several systems and check that the outputs coincide. If it is not the case, one should obviously select the most common output; the probability that a majority of precomputations is similarly incorrect is extremely low. Of course, if $c(q)^{n_{step}}$ is sufficiently close to 1, then the probability of an incorrect precomputation is low enough not to have to worry about these considerations.

3.4 Complexity

Generally, it is difficult to obtain good estimates for the complexity of Gröbner basis computation algorithms, especially of those based on Buchberger's approach. However, we can give a broad upper bound of the complexity of `F4Remake`, by observing that it can be reduced to the computation of the row echelon form of a D-Macaulay matrix of the homogenized system, whose useless rows would have been removed. In the case of generic systems, D is equal to the degree of regularity d_{reg} of the homogenized system. Thus we have an upper-bound for the complexity of our algorithm:

Proposition 6 *The number of field operations performed by* `F4Remake` *on a system of random polynomials over* $\mathbb{K}[X_1, \ldots, X_n]$ *is bounded by*

$$O\left(\binom{d_{reg}+n}{n}^{\omega}\right)$$

where d_{reg} *is the degree of regularity of the homogenized system and* ω *is the constant of matrix multiplication.*

Since there is no reduction to zero as well with F5 (under the assumption that the system is semi-regular), the same reasoning applies and gives the same upper-bound, cf [3]. However, we emphasize that these estimates are not really sharp and do not reflect the difference in performances between the two algorithms. Indeed, `F4Remake` has two main advantages over F5: the polynomials it generates are fully reduced, and it avoids the incremental structure of F5. More precisely, the F5 criterion relies on the use of a signature or label for each polynomial, and we have already mentioned in the introduction that signature compatibility conditions prohibit some reductions; therefore, the polynomials generated by F5 are not completely reduced, or are even redundant [13]. This incurs either more costly reductions later in the algorithm or a larger number of critical pairs. Secondly, the incremental nature of F5 implies that the information provided by the last system polynomials cannot be used to speed up the first stages of the computation.

Thus, our F4 variant should be used preferentially as soon as several Gröbner bases have to be computed and the base field is large enough for this family of systems. Nevertheless, the F5 algorithm remains irreplaceable when the Gröbner basis of only one system has to be computed, when the base field is too small (in particular over $\mathbb{F}_2$) or when the systems are so large that a precomputation would not be realisable.

4 Applications

In all applications, the variant `F4Remake` is compared with an implementation of F4 which uses the same primitives and structures (in language C), and also with the proprietary software Magma (V2.15-15) whose implementation is probably the best publicly available for the considered finite fields. Unless otherwise specified, all tests are performed on a 2.6 GHz Intel Core 2 Duo processor and times are given in seconds.

4.1 Index calculus

An index calculus method has been recently proposed in [11, 18] for the resolution of discrete logarithm on $E(\mathbb{F}_{q^n})$ where E is an elliptic curve defined over a small degree extension field. In order to find "relations", they make use of Semaev's idea [28] which allows to convert the relation search into the resolution of a multivariate polynomial system. A variation of this approach is given in [20], where relations with a slightly different form are considered: it has the advantage of leading to overdetermined systems and is thus faster in practical cases. We focus on the resolution of the polynomial systems arising from this last attack in the special case of $E(\mathbb{F}_{p^5})$ where p is a prime number. The polynomial systems in this example fit into the framework of parametric families: the coefficients polynomially depend on the x-coordinate of a random point $R \in E(\mathbb{F}_{p^5})$ (and also of the equation of the curve E). Our algorithm is particularly relevant for this example because of the large number of relations to collect, leading to an average of $4!p^2$ systems to solve. Moreover, p is large in all applications so the probability of success of our F4 variant is extremely good.

The systems to solve are composed of 5 equations defined over $\mathbb{F}_p$ of total degree 8 in 4 variables. Degrevlex Gröbner bases of the corresponding ideals over several prime fields of size 8, 16, 25 and 32 bits are computed. The probabilities of failure are estimated under the assumption that the systems are random, and knowing that the computation takes 29 steps.

size of p	est. failure probability	F4Precomp	F4Remake	F4	F4/F4Remake	F4 Magma
8 bits	0.11	8.963	2.844	5.903	2.1	9.660
16 bits	4.4×10^{-4}	(19.07)	3.990	9.758	2.4	9.870
25 bits	2.4×10^{-6}	(32.98)	4.942	16.77	3.4	118.8
32 bits	5.8×10^{-9}	(44.33)	8.444	24.56	2.9	1046

Step	degree	F4Remake matrix size	F4 matrix size	size ratio
14	17	1062×3072	1597×3207	1.6
15	16	1048×2798	1853×2999	1.9
16	15	992×2462	2001×2711	2.2
17	14	903×2093	2019×2369	2.5
18	13	794×1720	1930×2000	2.8

Fig. 1. Experimental results on $E(\mathbb{F}_{p^5})$

As explained in section 3.2, it is sufficient to execute the precomputation on the smaller field to get a list of polynomial multiples that works for the other cases; the timings of `F4Precomp` over the fields of size $16, 25$ and 32 bits are

thus just indicative. The above figures show that the precomputation overhead is largely compensated as soon as there are more than two subsequent computations. Note that it would have been hazardous to execute `F4Precomp` on a smaller field as the probability of failure increases rapidly. It is mentioned in [20] that the systems have also been solved with a personal implementation of F5, and that the size of the Gröbner basis it computes at the last step before minimization is surprisingly large (17249 labeled polynomials against no more than 2789 polynomials for both versions of F4). As a consequence, the timings of F5 obtained for these systems are much worse than those of F4 or its variants. This shows clearly that on this example, it is much more efficient to apply our algorithm rather than F5.

4.2 Hybrid approach

The hybrid approach proposed in [4] relies on a trade-off between exhaustive search and Gröbner basis computation. The basic idea is that when one wants to find a solution of a given system $f_1, \ldots, f_r \in \mathbb{K}[X_1, \ldots, X_n]$, it is sometimes faster to try to guess a small number of variables $X_1, \ldots, X_k$. For each possible k-tuple $(x_1, \ldots, x_k)$, one computes the Gröbner basis of the corresponding specialized system $f_1(x_1, \ldots, x_k), \ldots, f_r(x_1, \ldots, x_k) \in \mathbb{K}[X_{k+1}, \ldots, X_n]$ until a solution is found; the advantage is that the specialized systems are much simpler to solve than the initial one.

The hybrid approach is thus a typical case when many systems of the same shape have to be solved and fits perfectly into the framework of parametric families we have described in section 3.1. However, this method is most useful when the search space is reasonably small, which implies in particular that the size of the base field cannot be too large, so one should be wary of the probability of success before applying our F4 variant to this context. Note that, when the right guess is made for the k-tuple $(x_1, \ldots, x_k)$, the corresponding specialized system does not have a generic behaviour. As soon as this is detected by `F4Remake` (see section 2.2), the computation can be continued with e.g. standard F4.

As an example, we consider the cryptanalysis of the Unbalanced Oil and Vinegar system (UOV, [22]), described in [4]. Briefly, the attack can be reduced to the resolution of a system of n quadratic equations in n variables over a finite field $\mathbb{K}$; for the recommended set of parameters, $n = 16$ and $\mathbb{K} = \mathbb{F}_{16}$. Although the base field is quite small, our F4 variant has rather good results in this cryptanalysis: this is due to the fact that the quadratic part of the evaluated polynomials $f_i(x_1, \ldots, x_k) \in \mathbb{K}[X_{k+1}, \ldots, X_n]$ does not depend on the values of the specialized variables $X_1, \ldots, X_k$, and hence all the systems behave generically until the first fall of degree.

For instance, for $k = 3$ the computation with F4 takes 6 steps, and no fall of degree occurs before the penultimate step, so a heuristic estimation of the probability of success is $c(16)^2 \simeq 0.87$. To check this estimate we have performed an exhaustive exploration of the search space $\mathbb{F}_{16}^3$ using `F4Remake`. The measured probability of success depends of the actual system and varies around 90%, which shows that our estimate is satisfying.

The timings obtained during this experiment confirm that our variant provides a non-negligible speed-up. In particular, our timings are better (after a precomputation of 32.3 sec) than the 9.41 sec of F5 given in [4]; of course, this comparison is not really meaningful since the implementation of finite field arithmetic and linear algebra cannot be compared and since different (but similar) computers have been used.

	F4Remake	F4	F4 Magma	F4/F4Remake
Timing (sec)	5.04	16.77	120.6	3.3
Largest matrix	5913×7005	10022×8329	10245×8552	2.0

Fig. 2. Experimental results on UOV with 3 specialized variables

4.3 MinRank

We briefly recall the MinRank problem: given $m+1$ matrices $M_0, M_1, \ldots, M_m \in \mathcal{M}_n(\mathbb{K})$ and a positive integer r, is there a m-tuple $(\alpha_1, \ldots, \alpha_m) \in \mathbb{K}^m$ such that $\text{Rank}\left(\sum_{i=1}^{m} \alpha_i M_i - M_0\right) \leq r$.

We focus on the challenge A proposed in [9]: $\mathbb{K} = \mathbb{F}_{65521}; m = 10; n = 6; r = 3$. The Kipnis-Shamir's attack converts instances of the MinRank problem into quadratic multivariate polynomial systems [23]. For the set of parameters from challenge A, we thus have to solve systems of 18 quadratic equations in 20 variables, and since they are underdetermined, we can specialize two variables without loss of generality. These systems can be solve either directly or with the hybrid approach [17]; in the first case, our F4 variant will be relevant only if one wants to break several different instances of the MinRank problem.

Experiments with F4 and our variant show that, either for the full systems or the systems with one specialized variable, the matrices involved at different steps are quite large (up to 39138×22968) and relatively sparse (less than 5% non-zero entries). With both types of systems, a lot of reductions to zero occurs; for example, we have observed that for the full system at the 8th step, 17442 critical pairs among 17739 reduce to zero. This makes it clear that the classic F4 algorithm is not well suited for these specific systems.

It is difficult to compare our timings with those given in [17] using F5: besides the fact that the experiments were executed on different computers, the linear algebra used in Faugère's FGb implementation of F5 (whose source code is not public) seems to be highly optimized, even more so than in Magma's implementation of F4. On this point, our own implementation is clearly not competitive: for example, at the 7th step for the full system, Magma's F4 reduces a 26723×20223 matrix in 28.95 sec, whereas at the same step our implementation

reduces a slightly smaller matrix of size 25918×19392 in 81.52 sec. Despite these limitations, we have obtained timings comparable with those of [17], listed in the table below. This means that with a more elaborate implementation of linear algebra, our F4 variant would probably be the most efficient for these systems.

	F5	F4Remake	F4	F4 Magma
full system	30.0	27.87	320.2	116.6
1 specialized variable	1.85	2.605	9.654	3.560

Fig. 3. Experimental results on MinRank

Computations were executed on a Xeon bi-processor 3.2 GHz for F5. The results of `F4Remake` have been obtained after a precomputation over $\mathbb{F}_{257}$ of 4682 sec for the full system and 113 sec for the system with one variable specialized.

4.4 Katsura benchmarks

To illustrate the approach presented in section 3.2, we have applied our algorithm to the computation of the Gröbner bases of the Katsura11 and Katsura12 systems [21], over two prime fields of size 16 and 32 bits. As already explained, the idea is to run a precomputation on a small prime field before executing `F4Remake` over a large field (actually, for Katsura12 the first prime $p = 251$ we chose was weakly unlucky). The timings show that for both systems, the speed gain on 32 bits compensates the precomputation overhead, contrarily to the 16 bits case.

	8 bits	16 bits			32 bits		
	Precomputation	F4Remake	F4	F4 Magma	F4Remake	F4	F4 Magma
Katsura11	27.83	9.050	31.83	19.00	15.50	60.93	84.1
Katsura12	202.5	52.66	215.4	143.3	111.4	578.8	$> 5\,h$

Fig. 4. Experimental results on Katsura11 and Katsura12

As a side note, we observed that surprisingly, the matrices created by F4 are quite smaller in our version than in Magma (e.g. 15393×19368 versus 20162×24137 at step 12 of Katsura12); of course, both version still find the same new polynomials at each step. This phenomenon was already present in the previous systems, but not in such a proportion. This seems to indicate that our implementation of the `Simplify` subroutine is much more efficient.

Step	degree	F4Remake matrix size	F4 matrix size	size ratio
9	10	14846×18928	18913×20124	1.4
10	11	15141×19235	17469×19923	1.2
11	12	8249×12344	16044×19556	3.1
12	13	2225×6320	15393×19368	21.2
13	14	–	15229×19313	–

(At the 13^{th} step, F4 finds no new generator so this step is skipped by `F4Remake`)

Fig. 5. Sizes of the matrices involved in the last steps of Katsura12

5 Conclusion

We have presented in this article a variant of the F4 algorithm that provides a very efficient probabilistic method for computing Gröbner bases; it is especially designed for the case where many similar polynomial systems have to be solved. We have given a precise analysis of this context, estimated the probability of success, and evaluated both theoretically and experimentally the performances of our algorithm, showing that it is well adapted for algebraic attacks on cryptosystems.

Since Faugère's F5 algorithm is considered as the most efficient tool for computing Gröbner bases, we have tried as much as possible to compare its performances with our F4 variant. Clearly, F5 remains irreplaceable when the Gröbner basis of only one system has to be computed or when the base field is too small, in particular over $\mathbb{F}_2$. However, our method should be used preferentially as soon as several Gröbner bases have to be computed and the base field is large enough for the considered family of systems. The obtained timings support in part this claim, indicating that with a more elaborate implementation of linear algebra our algorithm would outperform F5 in most cases.

References

1. D. Augot, M. Bardet, and J.-C. Faugère. On the decoding of binary cyclic codes with the Newton identities. *J. Symbolic Comput.*, 44(12):1608–1625, 2009.
2. G. Bard. *Algebraic Cryptanalysis*. Springer-Verlag, New York, first edition, 2009.
3. M. Bardet, J.-C. Faugère, B. Salvy, and B.-Y. Yang. Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems. Presented at MEGA'05, Eighth International Symposium on Effective Methods in Algebraic Geometry, 2005.
4. L. Bettale, J.-C. Faugère, and L. Perret. Hybrid approach for solving multivariate systems over finite fields. *Journal of Mathematical Cryptology*, pages 177–197, 2009.
5. W. Bosma, J. J. Cannon, and C. Playoust. The Magma algebra system I: The user language. *J. Symb. Comput.*, 24(3/4):235–265, 1997.

6. B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal.* PhD thesis, University of Innsbruck, Austria, 1965.
7. B. Buchberger. A criterion for detecting unnecessary reductions in the construction of Gröbner bases. In E. W. Ng, editor, *Proc. of the EUROSAM 79*, volume 72 of *Lecture Notes in Computer Science*, pages 3–21. Copyright: Springer, Berlin - Heidelberg - New York, 1979.
8. B. Buchberger. Gröbner bases: An algorithmic method in polynomial ideal theory. In N. Bose, editor, *Multidimensional systems theory, Progress, directions and open problems, Math. Appl. 16*, pages 184–232. D. Reidel Publ. Co., 1985.
9. N. Courtois. Efficient zero-knowledge authentication based on a linear algebra problem MinRank. In *Advances in Cryptology – ASIACRYPT 2001*, pages 402–421. Springer, 2001.
10. N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Advances in Cryptology – EUROCRYPT 2000*, pages 392–407. Springer, 2000.
11. C. Diem. On the discrete logarithm problem in elliptic curves. Preprint, available at: `http://www.math.uni-leipzig.de/~diem/preprints/dlp-ell-curves.pdf`, 2009.
12. G. L. Ebert. Some comments on the modular approach to Gröbner-bases. *SIGSAM Bull.*, 17(2):28–32, 1983.
13. C. Eder and J. Perry. F5C: a variant of Faugère's F5 algorithm with reduced Gröbner bases. arXiv/0906.2967, 2009.
14. J.-C. Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, June 1999.
15. J.-C. Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Proceedings of ISSAC 2002*, New York, 2002. ACM.
16. J.-C. Faugère and A. Joux. Algebraic cryptanalysis of hidden field equation (HFE) cryptosystems using Gröbner bases. In *CRYPTO*, pages 44–60, 2003.
17. J.-C. Faugère, F. Levy-Dit-Vehel, and L. Perret. Cryptanalysis of MinRank. In *Advances in Cryptology – CRYPTO 2008*, pages 280–296, Berlin, Heidelberg, 2008. Springer-Verlag.
18. P. Gaudry. Index calculus for abelian varieties of small dimension and the elliptic curve discrete logarithm problem. *J. Symbolic Computation*, 2008. doi:10.1016/j.jsc.2008.08.005.
19. R. Gebauer and H. M. Möller. On an installation of Buchberger's algorithm. *J. Symbolic Comput.*, 6(2-3):275–286, 1988.
20. A. Joux and V. Vitse. Elliptic curve discrete logarithm problem over small degree extension fields. Application to the static Diffie–Hellman problem on $E(\mathbb{F}_{q^5})$. Cryptology ePrint Archive, Report 2010/157, 2010.
21. S. Katsura, W. Fukuda, S. Inawashiro, N. M. Fujiki, and R. Gebauer. Distribution of effective field in the Ising spin glass of the $\pm J$ model at $T = 0$. *Cell Biochem. Biophys.*, 11(1):309–319, 1987.
22. A. Kipnis, J. Patarin, and L. Goubin. Unbalanced Oil and Vinegar signature schemes. In *Advances in Cryptology – EUROCRYPT'99*, pages 206–222. Springer, 1999.
23. A. Kipnis and A. Shamir. Cryptanalysis of the HFE public key cryptosystem by relinearization. In *Advances in Cryptology – CRYPTO' 99*, pages 19–30. Springer Berlin, Heidelberg, 1999.

24. D. Lazard. Gröbner bases, Gaussian elimination and resolution of systems of algebraic equations. In *Computer algebra (London, 1983)*, volume 162 of *Lecture Notes in Comput. Sci.*, pages 146–156. Springer, Berlin, 1983.
25. F. Macaulay. Some formulae in elimination. *Proceedings of London Mathematical Society*, pages 3–38, 1902.
26. M. S. E. Mohamed, W. S. A. E. Mohamed, J. Ding, and J. Buchmann. MXL2: Solving polynomial equations over $GF(2)$ using an improved mutant strategy. In *PQCrypto*, pages 203–215. Springer, 2008.
27. T. Sasaki and T. Takeshima. A modular method for Gröbner-basis construction over $\mathbb{Q}$ and solving system of algebraic equations. *J. Inf. Process.*, 12(4):371–379, 1989.
28. I. Semaev. Summation polynomials and the discrete logarithm problem on elliptic curves. Cryptology ePrint Archive, Report 2004/031, 2004.
29. C. Traverso. Gröbner trace algorithms. In *Symbolic and algebraic computation (Rome, 1988)*, volume 358 of *Lecture Notes in Comput. Sci.*, pages 125–138. Springer, Berlin, 1989.
30. V. Weispfenning. Comprehensive Gröbner bases. *J. Symbolic Comput.*, 14(1):1–29, 1992.

A Pseudo-code

A.1 The precomputation

Given a family of polynomials $\{f_1, \ldots, f_r\}$, the `F4Precomp` algorithm computes for each iteration step of the classical F4 algorithm, the list of polynomial multiples that will be used by `F4Remake` on subsequent computations. This algorithm follows very closely [14], with these additional features:

- A list L of lists of couples is introduced; at the end of the i-th main iteration, $L[i]$ contains the desired list of polynomial multiples for that step. Each polynomial multiple is represented by a couple (m, n), where m is a monomial and n is the index of the polynomial in a global list G (this list G will be progressively reconstructed by `F4Remake`). In the same way, a list L_{tmp} is used to temporary store these couples.
- Instead of just computing the reduced row echelon form M' of the matrix M, we also compute an auxiliary matrix A such that $AM = M'$. If reductions to zero occur, then the bottom part of M' is null and the corresponding bottom part of A gives the linear dependencies between the rows of M. This information is exploited in lines 21 to 26, in order to remove from the temporary list L_{tmp} the useless multiples before copy in $L[step]$. Actually, only the bottom-left part A' of A is of interest: it contains the linear dependencies between the rows of M coming from the critical pairs, modulo those coming from the preprocessing. It is clear that with each dependency relation, one polynomial multiple can be removed, but some care must be taken in this choice. To do so, the row echelon form $\tilde{A}$ of A' is then computed and the polynomial multiples corresponding to the pivots of $\tilde{A}$ are removed. Among the remaining polynomial multiples, those whose leading monomial is now unique can also be removed.

Apart from these modifications, the pseudo-code is basically the F4 algorithm with Gebauer and Möller installation of the Buchberger's criteria (`Update` subroutine) [19]. The only notable change concerns the implementation of the `Simplify` procedure: instead of searching through all the former matrices and their row echelon forms for the adequate simplification as in [14], we introduce an array $TabSimplify$ which contains for each polynomial f in the basis a list of couple of the form $(m, g) \in T \times \mathbb{K}[\underline{X}]$, meaning that the product mf can be simplified into the more reduced polynomial g. This array is updated after the reduced row echelon form is computed (lines 12 to 16 of `Postprocessing`).

Alg. 1 F4Precomp

INPUT: $f_1, \ldots, f_r \in \mathbb{K}[\underline{X}]$ OUTPUT: a list of lists of couples $(m, n) \in T \times \mathbb{N}$

1. $G \leftarrow [\,],\ G_{min} \leftarrow \emptyset,\ P \leftarrow \emptyset, TabSimplify \leftarrow [\,], L \leftarrow [\,]$
2. **for** $i = 1$ to r **do**
3. $\quad G[i] \leftarrow f_i, \quad TabSimplify[i] \leftarrow [(1, f_i)], \quad Update(f_i)$
4. $step = 1$
5. **while** $P \neq \emptyset$ **do**
6. $\quad P_{sel} \leftarrow Sel(P)$
7. $\quad F \leftarrow [\,],\ LM(F) \leftarrow \emptyset,\ T(F) \leftarrow \emptyset,\ L[step] \leftarrow [\,],\ L_{tmp} \leftarrow [\,]$
8. $\quad$ **for all** $pair = (lcm, t_1, g_1, t_2, g_2) \in P_{sel}$ **do**
9. $\quad\quad$ **for** $k = 1$ to 2 **do**
10. $\quad\quad\quad ind \leftarrow index(g_k, G)$
11. $\quad\quad\quad$ **if** $(t_k, ind) \notin L_{tmp}$ **then**
12. $\quad\quad\quad\quad Append(L_{tmp}, (t_k, ind))$
13. $\quad\quad\quad\quad f \leftarrow Simplify(t_k, ind)$
14. $\quad\quad\quad\quad Append(F, f)$
15. $\quad\quad\quad\quad LM(F) \leftarrow LM(F) \cup \{LM(f)\}$
16. $\quad\quad\quad\quad T(F) \leftarrow T(F) \cup \{m \in T : m \text{ monomial of } f\}$
17. $\quad Preprocessing(F, T(F), LM(F))$
18. $\quad M \leftarrow$ matrix whose rows are the polynomials in F
19. $\quad (M'|A) \leftarrow ReducedRowEchelonForm(M|I_{\#F})\ (\Rightarrow AM = M')$
20. $\quad rank \leftarrow Postprocessing(M', LM(F))$
21. $\quad$ **if** $rank < \#F$ **then**
22. $\quad\quad A' \leftarrow A[rank + 1..\#F][1..\#L_{tmp}]$
23. $\quad\quad \tilde{A} \leftarrow ReducedRowEchelonForm(A')$
24. $\quad\quad C \leftarrow \{c \in \{1, \ldots, \#L_{tmp}\} : c \text{ is not a column number of a pivot in } \tilde{A}\}$
25. $\quad\quad$ **for** $j \in C$ **do**
26. $\quad\quad\quad$ **if** $\exists k \in C,\ k \neq j$ and $LM(F[k]) = LM(F[j])$ **then** $Append(L[step], L_{tmp}[j])$
27. $\quad$ **else** $L[step] \leftarrow L_{tmp}$
28. $\quad step \leftarrow step + 1$
29. **return** L

In the pseudo-code, some variables are supposed to be global: G, a list of polynomials that forms a basis of $\langle f_1, \ldots, f_r \rangle$; G_{min}, a set of polynomials which is the minimized version of G; $TabSimplify$, an array of lists of couples used for the simplification of polynomials multiples; P, a queue of yet untreated critical pairs.

The function Sel on line 6 is a selection function, whose expression depends on the chosen strategy; usually, selecting all pairs of lowest total degree lcm (normal strategy) yields the best performances. The notation $index(g, G)$ stands for the integer i such that $G[i] = g$, and the function $pair(f_1, f_2)$ outputs the critical pair $(lcm, u_1, f_1, u_2, f_2)$. Finally, $ReducedRowEchelonForm$ computes as expected the reduced row echelon form of its input matrix. We stress that great care should be taken in the implementation of this last function since almost all the execution time of the algorithm is spent in it. Note that the test on line 10 in `Update` is only necessary during the initialisation phase of `F4Precomp` (line 3).

Alg. 2 Update

INPUT: $f \in \mathbb{K}[\underline{X}]$

1. **for all** $pair = (lcm, t_1, g_1, t_2, g_2) \in P$ **do**
2. **if** $(LM(f) \vee LM(g_1)$ divides strictly $lcm)$ AND $(LM(f) \vee LM(g_2)$ divides strictly $lcm)$ **then** $P \leftarrow P \setminus \{pair\}$
3. $P_0 \leftarrow \emptyset, P_1 \leftarrow \emptyset, P_2 \leftarrow \emptyset$
4. **for all** $g \in G_{min}$ **do**
5. **if** $LM(f) \wedge LM(g) = 1$ **then** $P_0 \leftarrow P_0 \cup pair(f, g)$ **else** $P_1 \leftarrow P_1 \cup pair(f, g)$
6. **for all** $pair = (lcm, t_1, g_1, t_2, g_2) \in P_1$ **do**
7. $P_1 \leftarrow P_1 \setminus \{pair\}$
8. **if** $\nexists pair' = (lcm', t_1', g_1', t_2', g_2') \in P_0 \cup P_1 \cup P_2$ s.t.$lcm' | lcm$ **then** $P_2 \leftarrow P_2 \cup \{pair\}$
9. $P \leftarrow P \cup P_2$
10. **if** $\nexists g \in G_{min}$ such that $LM(g) | LM(f)$ **then**
11. **for all** $g \in G_{min}$ **do**
12. **if** $LM(f) | LM(g)$ **then** $G_{min} \leftarrow G_{min} \setminus \{g\}$
13. $G_{min} \leftarrow G_{min} \cup \{f\}$

Alg. 3 Preprocessing

INPUT: $F, T(F), LM(F)$

1. $Done \leftarrow LM(F)$
2. **while** $T(F) \neq Done$ **do**
3. $m \leftarrow \max(T(F) \setminus Done)$
4. $Done \leftarrow Done \cup \{m\}$
5. **for all** $g \in G_{min}$ **do**
6. **if** $LM(g) | m$ **then**
7. $g' \leftarrow Simplify\left(\frac{m}{LM(g)}, index(g, G)\right)$
8. $Append(F, g')$
9. $LM(F) \leftarrow LM(F) \cup \{m\}$
10. $T(F) \leftarrow T(F) \cup \{m' \in T : m'$ monomial of $g'\}$
11. **break**

Alg. 4 Simplify

INPUT: $t \in T, ind \in \mathbb{N}$ OUTPUT: $p \in \mathbb{K}[\underline{X}]$

1. **for** $(m, f) \in TabSimplify[ind]$ (from last to first) **do**
2. **if** $m = t$ **then return** f
3. **else if** $m|t$ **then**
4. $Append\left(TabSimplify[ind], \left(m, \frac{t}{m} f\right)\right)$
5. **return** $\frac{t}{m} f$

Alg. 5 Postprocessing

INPUT: a matrix M in reduced row echelon form with $\#F$ lines and an ordered set of monomials $LM(F)$

OUTPUT: the rank of the matrix M

1. **for** $i = 1$ to $\#F$ **do**
2. $f \leftarrow M[i]$
3. **if** $f = 0$ **then break**
4. **if** $LM(f) \notin LM(F)$ **then**
5. $Append(G, f)$
6. $Update(f)$
7. $TabSimplify[\#G] \leftarrow [(1, f)]$
8. **else**
9. **for** $g \in G_{min}$ **do**
10. $ind \leftarrow index(g, G)$
11. **if** $LM(g)|LM(f)$ **then**
12. **for** $j = 1$ to $\#TabSimplify[ind]$ **do**
13. **if** $TabSimplify[ind][j] = \left(\frac{LM(f)}{LM(g)}, .\right)$ **then**
14. $TabSimplify[ind][j] = \left(\frac{LM(f)}{LM(g)}, f\right)$
15. **break**
16. **if** $j > \#TabSimplify[ind]$ **then** $Append\left(TabSimplify[ind], \left(\frac{LM(f)}{LM(g)}, f\right)\right)$
17. **return** $i - 1$

A.2 F4Remake

The `F4Remake` algorithm uses the same routines `Simplify`, `Preprocessing` and `Postprocessing`. Since it no longer uses critical pairs, the subroutine `Update` can be greatly simplified and is replaced by `Update2`.

Alg. 6 F4Remake

INPUT: $f_1, \ldots, f_r \in \mathbb{K}[\underline{X}]$, a list L of lists of couples $(m, n) \in T \times \mathbb{N}$
OUTPUT: G_{min}, the reduced minimal Gröbner basis of $f_1, \ldots, f_r$

1. $G \leftarrow [\,]$, $G_{min} \leftarrow \emptyset, TabSimplify \leftarrow [\,]$
2. **for** $i = 1$ to r **do**
3. $\quad G[i] \leftarrow f_i$
4. $\quad TabSimplify[i] \leftarrow [(1, f_i)]$
5. $\quad Update2(f_i)$
6. **for** $step = 1$ to $\#L$ **do**
7. $\quad F \leftarrow [\,]$, $LM(F) \leftarrow \emptyset$, $T(F) \leftarrow \emptyset$
8. $\quad$ **for all** $(m, n) \in L[step]$ **do**
9. $\quad\quad$ **if** $n > \#G$ **then** computation fails ! **exit**
10. $\quad\quad f \leftarrow Simplify(m, n)$, $Append(F, f)$
11. $\quad\quad LM(F) \leftarrow LM(F) \cup \{LM(f)\}$
12. $\quad\quad T(F) \leftarrow T(F) \cup \{m \in T : m \text{ monomial of } f\}$
13. $\quad Preprocessing(F, T(F), LM(F))$
14. $\quad M \leftarrow$ matrix whose rows are the polynomials in F
15. $\quad M' \leftarrow ReducedRowEchelonForm(M)$
16. $\quad Postprocessing(M', LM(F))$
17. **return** $InterReduce(G_{min})$

Alg. 7 Update2

INPUT: $f \in \mathbb{K}[\underline{X}]$

1. **if** $\nexists g \in G_{min}$ such that $LM(g) | LM(f)$ **then**
2. $\quad$ **for all** $g \in G_{min}$ **do**
3. $\quad\quad$ **if** $LM(f) | LM(g)$ **then** $G_{min} \leftarrow G_{min} \setminus \{g\}$
4. $\quad G_{min} \leftarrow G_{min} \cup \{f\}$