Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations

Nicolas Courtois[1,3], Alexander Klimov[2], Jacques Patarin[3], and Adi Shamir[4]

[1] SIS, Toulon University, BP 132, F-83957 La Garde Cedex, France
courtois@minrank.org
[2] Dept. of Appl. Math. & Cybernetics, Moscow State University, Moscow, Russia
ask@ispras.ru
[3] Bull CP8, 68, route de Versailles, BP45, 78431 Louveciennes Cedex, France
J.Patarin@frlv.bull.fr
[4] Dept. of Applied Math. The Weizmann Institute of Science, Rehovot 76100, Israel
shamir@wisdom.weizmann.ac.il

Abstract. The security of many recently proposed cryptosystems is based on the difficulty of solving large systems of quadratic multivariate polynomial equations. This problem is NP-hard over any field. When the number of equations m is the same as the number of unknowns n the best known algorithms are exhaustive search for small fields, and a Gröbner base algorithm for large fields. Gröbner base algorithms have large exponential complexity and cannot solve in practice systems with $n \geq 15$. Kipnis and Shamir [9] have recently introduced a new algorithm called "relinearization". The exact complexity of this algorithm is not known, but for sufficiently overdefined systems it was expected to run in polynomial time.
In this paper we analyze the theoretical and practical aspects of relinearization. We ran a large number of experiments for various values of n and m, and analysed which systems of equations were actually solvable. We show that many of the equations generated by relinearization are linearly dependent, and thus relinearization is less efficient that one could expect. We then develop an improved algorithm called XL which is both simpler and more powerful than relinearization. For all $0 < \epsilon \leq 1/2$, and $m \geq \epsilon n^2$, XL and relinearization are expected to run in polynomial time of approximately $n^{\mathcal{O}(1/\sqrt{\epsilon})}$. Moreover, we provide strong evidence that relinearization and XL can solve randomly generated systems of polynomial equations in subexponential time when m exceeds n by a number that increases slowly with n.

Note: An extended version of this paper is available from the authors.

Key words: NP-completeness, cryptography, multivariate cryptography, polynomial equations over finite fields, relinearization, Gröbner bases.

1 Introduction

In this paper we consider the problem of solving systems of multivariate polynomial equations. This problem is NP-complete even if all the equations are quadratic and the field is $GF(2)$. It has many applications in cryptography, since a large number of multivariate schemes had been proposed (and cryptanalysed) over the last few years. In addition, the problem arises naturally in other subareas of Mathematics and Computer Science, such as optimization, combinatorics, coding theory, and computer algebra.

The classical algorithm for solving such a system is Buchberger's algorithm for constructing Gröbner bases, and its many variants (see, e.g., [1]). The algorithm orders the monomials (typically in lexicographic order), and eliminates the top monomial by combining two equations with appropriate polynomial coefficients. This process is repeated until all but one of the variables are eliminated, and then solves the remaining univariate polynomial equation (e.g., by using Berlekamp's algorithm over the original or an extension field). Unfortunately, the degrees of the remaining monomials increase rapidly during the elimination process, and thus the time complexity of the algorithm makes it often impractical even for a modest number of variables. In the worst case Buchberger's algorithm is known to run in double exponential time, and on average its running time seems to be single exponential. The most efficient variant of this algorithm which we are aware of is due to Jean-Charles Faugere (private communication [5, 6]) whose complexity in the case of $m = n$ quadratic equations is:

- If K is big, the complexity is proved to be $\mathcal{O}(2^{3n})$ and is $\mathcal{O}(2^{2.7n})$ in practice.
- When K =GF(2), the complexity is about $\mathcal{O}(2^{2n})$ (which is worse than the $O(2^n)$ complexity of exhaustive search).

In practice, even this efficient variant cannot handle systems of quadratic equations with more than about $n = 15$ variables.

In this paper we are interested in the problem of solving overdefined systems of multivariate polynomial equations in which the number of equations m exceeds the number of variables n. Random systems of equations of this type are not expected to have any solutions, and if we choose them in such a way that one solution is known to exist, we do not expect other interference solutions to occur. We are interested in this type of systems since they often occur in multivariate cryptographic schemes: if the variables represent the cleartext then we want the decryption process to lead to a unique cleartext, and if the variables represent the secret key we can typically write a large number of polynomial equations which relate it to the known public key, to the cleartexts, and to the ciphertexts.

Gröbner base techniques do not usually benefit from the fact that the number of equations exceeds the number of variables, since they proceed by sequentially eliminating a single monomial from a particular pair of equations. Unfortunately, this cryptographically important case received very little attention in the vast literature on Gröbner base algorithms. To see that much better algorithms exist in this case, consider a system of $n(n+1)/2$ random homogeneous quadratic equations in n variables $x_1, ...x_n$. The well known linearization technique replaces each product $x_i x_j$ by a new independent variable y_{ij}. The quadratic equations give a system of $n(n+1)/2$ linear equations in $n(n+1)/2$ variables which can be solved efficiently by Gauss elimination. Once we find all the y_{ij} values, we can find two possible values for each x_i by extracting the square root of y_{ii} in the field, and use the values of y_{ij} to combine correctly the roots of y_{ii} and y_{jj}.

At Crypto 99, Kipnis and Shamir [9] introduced a new method for solving overdefined systems of polynomial equations, called *relinearization*. It was designed to handle systems of ϵn^2 quadratic equations in n variables where ϵ is smaller than $1/2$. The basic idea of relinearization is to add to the given system of linear equations in the y_{ij} additional nonlinear equations which express the fact that these variables are related rather than independent. In its simplest form, relinearization is based on the commutativity of multiplication of 4-tuples of variables: For any a, b, c, d, $(x_a x_b)(x_c x_d) = (x_a x_c)(x_b x_d) = (x_a x_d)(x_b x_c)$ and thus $y_{ab}y_{cd} = y_{ac}y_{bd} = y_{ad}y_{bc}$. There are several generalizations of relinearization, including higher degree variants and a recursive variant. The relinearization technique can solve many systems of equations which could not be solved by linearization, but its exact complexity and success rate are not well understood.

In the first part of this paper, we analyse the theoretical and practical aspects of the relinearization technique. We concentrate in particular on the issue of the linear independence of the generated equations, and show that many of the generated equations are provably dependent on other equations, and can thus be eliminated. This reduces the size of the linearized systems, but also limits the types of polynomial equations which can be successfully solved by the technique.

In the second part of the paper, we introduce the XL (eXtended Linearization) technique which can be viewed as a combination of bounded degree Gröbner bases and linearization. The basic idea of this technique is to generate from each polynomial equation a large number of higher degree variants by multiplying it with all the possible monomials of some bounded degree, and then to linearize the expanded system. This is a very simple technique, but we prove that it is at least as powerful as relinearization. We analyse the time complexity of the XL technique, and provide strong theoretical and practical evidence that the expected running time of this technique is:

- Polynomial when the number m of (random) equations is at least ϵn^2, and this for all $\epsilon > 0$.
- Subexponential if m exceeds n even by a small number.

If the size of the underlying field is not too large, we can sometimes apply this subexponential technique even to an underdefined (or exactly defined) systems of equations by guessing the values of some of the variables and simplifying the resulting equations.

2 Experimental Analysis of the Relinearization technique

In this part we concentrate on systems of randomly generated homogeneous quadratic equations of the form:

$$\sum_{1\le i\le j\le n} a_{ijk}x_ix_j = b_k, \qquad k = 1\dots m \tag{1}$$

The general idea of the *relinearization* method is to first use linearization in order to solve the system of m linear equations in the $n(n+1)/2$ variables $y_{ij} = x_ix_j$. The system is typically underdefined, and thus we express each y_{ij} as a linear combination of $l < n(n+1)/2$ new parameters $t_1, \dots, t_l$. We then create additional equations which express the commutativity of the multiplication of x_i which can be paired in different orders. Let $(a, b, c, d, \dots, e, f) \sim (a', b', c', d', \dots, e', f')$ denote that the two tuples are permuted versions of each other. Then:

$$(x_ax_b)(x_cx_d)...(x_ex_f) = (x_{a'}x_{b'})(x_{c'}x_{d'})...(x_{e'}x_{f'}) \tag{2}$$

This can be viewed as an equation in the y_{ij} variables, and thus also as an equation in the (smaller number of) parameters t_s expressing them. The new system of equations derived from all the possible choices of tuples of indices and their permutations can be solved either by another linearization or by recursive relinearization.

2.1 Degree 4 relinearization

We have applied the degree 4 relinearization technique to a large number of systems of randomly generated homogeneous quadratic equations of various sizes. We always got linearly independent equations (except when the field was very small). For several small values of n, the critical number of equations which make the system (barely) solvable is summarized in the following table:

n	m	l	n'	m'
6	8	13	104	105
8	12	24	324	336
10	16	39	819	825
15	30	90	4185	4200

Table 1. Fourth degree relinearization

n Number of variables in original quadratic system
m Number of equations in original quadratic system
l Number of parameters in the representation of the y_{ij}
n' Number of variables in the final linear system
m' Number of equations in the final linear system

Assuming the linear independence of the derived equations (which was experimentally verified), we can easily derive the asymptotic performance of degree 4 relinearization for large n: The method is expected to find the solution (in polynomial time) whenever the number of equations exceeds ϵn^2 for $\epsilon > 1/2 - 1/\sqrt{6} \approx 0.1$. This case is thus well understood.

2.2 Higher degree relinearization

The problem becomes much more complicated when we consider degree 6 relinearizations, which are based on all the equations of the form:

$$y_{ab}y_{cd}y_{ef} = y_{gh}y_{ij}y_{kl}, \quad \text{where } (a, b, c, d, e, f) \sim (g, h, i, j, k, l) \tag{3}$$

Note that these equations are cubic in the free parameters t_s (even if the original equations are quadratic), so we need many more equations to relinearize it successfully.

Unlike the case of degree 4 relinearizations, many of these equations were experimentally found to be linearly dependent. We have identified several distinct causes of linear dependence, but its complete characterization is still an open research problem.

We first have to eliminate trivial sources of linear dependence. We only have to consider 6-tuples of indices (a, b, c, d, e, f) which are sorted into non-decreasing order within each successive pair (a, b), (c, d), (e, f), and then into non-decreasing lexicographic order on these pairs. For 6-tuples which contain 6 distinct indices such as (0,1,2,3,4,5), we get 15 (rather than $6! = 720$) legal permutations:

$$\begin{array}{lll} (0,1,\ 2,3,\ 4,5) & (0,1,\ 2,4,\ 3,5) & (0,1,\ 2,5,\ 3,4) \\ (0,2,\ 1,3,\ 4,5) & (0,2,\ 1,4,\ 3,5) & (0,2,\ 1,5,\ 3,4) \\ (0,3,\ 1,2,\ 4,5) & (0,3,\ 1,4,\ 2,5) & (0,3,\ 1,5,\ 2,4) \\ (0,4,\ 1,2,\ 3,5) & (0,4,\ 1,3,\ 2,5) & (0,4,\ 1,5,\ 2,3) \\ (0,5,\ 1,2,\ 3,4) & (0,5,\ 1,3,\ 2,4) & (0,5,\ 1,4,\ 2,3) \end{array}$$

so we can create 14 possible equations. But for the 6-tuple $(0, 1, 1, 1, 1, 2)$, there are only 2 legal permutations $(0, 1,\ 1, 1,\ 1, 2)$ and $(0, 2,\ 1, 1,\ 1, 1)$ and thus we get only one equation. In general, there are 32 types of repetition of values in the given 6-tuple, and each one of them gives rise to a different number of equations. Table (2) summarizes the number of non-trivial equations which can actually be formed using 6-tuples for small values of n.

n	equations
4	136
5	470
6	1309
7	3136
8	6720
9	13212
10	24255
11	42108
12	69784
20	1388520

Table 2. Number of non trivial equations defined by 6-tuples

2.3 Eliminating redundant linear equations

In this section we show that most of the non-trivial equations defined so far are redundant, since they can be linearly derived from other equations. Consider a typical non-trivial equation generated by degree r relinearization:

$$y_{i_1 i_2} y_{i_3 i_4} \dots y_{i_{r-1} i_r} = y_{j_1 j_2} y_{j_3 j_4} \dots y_{j_{r-1} j_r} \text{ with } (i_1, \dots, i_r) \sim (j_1, \dots, j_r) \tag{4}$$

We call such an equation *special* if the lists of y's are the same on both sides of the equation, except for exactly two y's whose indices are permuted. For example, the non-trivial equation

$$y_{01} y_{23} y_{45} y_{67} y_{89} = y_{01} y_{27} y_{36} y_{45} y_{89} \tag{5}$$

is special since 3 out of the 5 terms are common in the two expressions. For large n only a small fraction of the equations are special, but we can prove:

Lemma: The set of special equations linearly span the set of all the non-trivial equations for the same relinearization degree.

Proof (sketch): Consider two particular permutations A and B of the same r-tuple of indices, which define one of the possible equations. A basic property of permutation groups is that any permutation can be derived by a sequence of transpositions which affect only adjacent elements. Consider the pairing of consecutive indices which defines the sequence of y's. Applying a single transposition of adjacent indices

can permute the indices of at most two y's, and thus we can derive the equality of the product of y's for any two permuted versions of some subset of indices from the transitivity of the equality in special equations.

To further reduce the number of equations, recall that each y_{ij} variable is a linear combination of a smaller number of parameters t_s. Instead of having all the possible common products of y_{ij} variables on both sides of the equation, it suffices to consider only common products of t_s parameters, since each product of the first type is expressible as a linear combination of products of the second type. We can thus consider only the smaller number of equations of the form:

$$y_{ab}y_{cd}t_e t_f \cdots t_g = y_{ac}y_{bd}t_e t_f \cdots t_g = y_{ad}y_{bc}t_e t_f \cdots t_g \tag{6}$$

The common t's on both sides of the equation seem to be cancellable, and thus we are led to believe that degree r relinearization is just a wasteful representation of degree 4 relinearization, which can solve exactly the same instances. However, division by a variable is an algebraic rather than linear operation, and thus we cannot prove this claim. The surprising fact is that these seemingly unnecessary common variables are very powerful, and in fact, they form the basis for the XL technique described in the second part of this paper. As a concrete example, consider a slightly overdefined system of 10 quadratic equations in 8 variables. Experiments have shown that it can be solved by degree 6 relinearization, whereas degree 4 relinearizations need at least 12 quadratic equations in 8 variables. Other combinations of solvable cases are summarized in table 3.

n	m	l	n'	m"
4	8	2	9	9
4	7	3	19	19
4	6	4	34	40
4	5	5	55	86
5	9	6	83	83
5	8	7	119	129
5	7	8	164	215
5	6	9	219	443
6	10	11	363	394
6	9	12	454	548
6	8	13	559	806
6	7	14	679	1541
7	11	17	1139	1363
7	10	18	1329	1744
7	9	19	1539	2318
8	12	24	2924	3794
8	11	25	3275	4584
8	10	26	3653	5721
9	13	32	6544	9080
9	12	33	7139	10567
9	11	34	7769	12716

n Number of variables in the original quadratic system
m Number of equations in the original quadratic system
l Number of parameters in the representation of the y_{ij}
n' Number of variables in the final linear system
m" number of equations which were required to solve the final linear system

Table 3. Experimental data for degree 6 relinearization

As indicated in this table, even the equations derived from special equations are still somewhat dependent, since we need more equations than variables in the final linear system. We have found several other sources of linear dependence, but due to space limitations we cannot describe them in this extended abstract.

3 The XL Algorithm

We present another algorithm for solving systems of multivariate polynomial equations called XL (which stands for eXtended Linearizations, or for multiplication and linearization). As we will see, each independent equation obtained by relinearization exists (in a different form) in XL, and thus XL can be seen as a simplified and improved version of relinearization.

Let K be a field, and let $\mathcal{A}$ be a system of multivariate quadratic equations $l_k = 0 \qquad (1 \leq k \leq m)$ where each l_k is the multivariate polynomial $f_k(x_1, \ldots, x_n) - b_k$.

The problem is to find at least one solution $x = (x_1, \ldots, x_n) \in K^n$, for a given $b = (b_1, \ldots, b_n) \in K^m$.

In all the following notations we suppose the powers of variables taken over K, i.e. reduced modulo q to the range $1, \ldots, q-1$, because of the equation $a^q = a$ of the finite field K.

We say that the equations of the form $\prod_{j=1}^{k} x_{i_j} * l_i = 0$ are of type $x^k l$, and we call $x^k l$ the set of all these equations. For example the initial equations $\mathcal{A}$ are of type l.

We also denote by x^k the set of all terms of degree exactly k, $\prod_{j=1}^{k} x_{i_j}$. It is a slightly modified extension of the usual convention $x = (x_1, \ldots, x_n)$.

Let $D \in \mathbb{N}$. We consider all the polynomials $\prod_j x_{i_j} * l_i$ of total degree $\leq D$.

Let $\mathcal{I}_D$ be the set of equations they span. $\mathcal{I}_D$ is the linear space generated by all the $x^k l,\ 0 \leq k \leq D-2$.

$\mathcal{I}_D \subset \mathcal{I}$, $\mathcal{I}$ being the ideal spanned by the l_i (could be called $\mathcal{I}_\infty$).

The idea of the XL algorithm is to find in some $\mathcal{I}_D$ a set of equations which is easier to solve than the initial set of equations $\mathcal{I}_0 = \mathcal{A}$. As we show later, the XL algorithm with maximal degree D completely contains the relinearization technique of degree D.

Definition 3.01 (The XL algorithm) *Execute the following steps:*

1. **Multiply:** *Generate all the products $\prod_{j=1}^{k} x_{i_j} * l_i \in \mathcal{I}_D$ with $k \leq D-2$.*
2. **Linearize:** *Consider each monomial in the x_i of degree $\leq D$ as a new variable and perform Gaussian elimination on the equations obtained in 1.*
   The ordering on the monomials must be such that all the terms containing one variable (say x_1) are eliminated last.
3. **Solve:** *Assume that step 2 yields at least one univariate equation in the powers of x_1. Solve this equation over the finite fields (e.g., with Berlekamp's algorithm).*
4. **Repeat:** *Simplify the equations and repeat the process to find the values of the other variables.*

The XL algorithm is very simple, but it is not clear for which values of n and m it ends successfully, what is its asymptotic complexity, and what is its relationship to relinearization and Gröbner base techniques. As we will see, despite it's simplicity XL may be one of the best algorithms for randomly generated overdefined systems of multivariate equations.

Note 1: The equations generated in XL are in $x^k l$ and belong to $\mathcal{I}$, the ideal generated by the l_i. There is no need to consider more general equations such as l_1^2 since they are in $\mathcal{I}_4$ and are thus in the linear space generated by the equations of type $x^2 l \cup xl \cup l$.

Note 2: Sometimes it is more efficient to work only with a subset of all the possible monomials. For example, when all the equations are homogeneous quadratic equations, it suffices to use only monomials of odd (or even) degrees.

Note 3: A related technique was used by Don Coppersmith to find small roots of univariate modular equations [2]. However, in that application he used LLL rather than Gauss elimination to handle the generated relations, and relied heavily on the fact that the solution is small (which plays no role in our application).

4 A toy example of XL

Let $\mu \neq 0$. Consider the problem of solving:

$$\begin{cases} x_1^2 + \mu x_1 x_2 = \alpha \ \ (4.1) \\ x_2^2 + \nu x_1 x_2 = \beta \ \ (4.2) \end{cases}$$

For $D = 4$ and even degree monomials, the equations we generate in step 1 of the XL algorithm are $l \cup x^2 l$. Those are the 2 initial equations and $6 = 2 * 3$ additional equations generated by multiplying the initial 2 equations l_i by the 3 possible terms of degree 2: $x_1^2, x_1 x_2, x_2^2 \in x^2$.

$$\begin{cases} x_1^4 + \mu x_1^3 x_2 = \alpha x_1^2 & (4.3) \\ x_1^2 x_2^2 + \nu x_1^3 x_2 = \beta x_1^2 & (4.4) \\ x_1^2 x_2^2 + \mu x_1 x_2^3 = \alpha x_2^2 & (4.5) \\ x_2^4 + \nu x_1 x_2^3 = \beta x_2^2 & (4.6) \\ x_1^3 x_2 + \mu x_1^2 x_2^2 = \alpha x_1 x_2 & (4.7) \\ x_1 x_2^3 + \nu x_1^2 x_2^2 = \beta x_1 x_2 & (4.8) \end{cases}$$

In step 2 we eliminate and compute:

From (4.1): $x_1 x_2 = \frac{\alpha}{\mu} - \frac{x_1^2}{\mu}$;

From (4.2): $x_2^2 = (\beta - \frac{\alpha\nu}{\mu}) + \frac{\nu}{\mu} x_1^2$;

From (4.3): $x_1^3 x_2 = \frac{\alpha}{\mu} x_1^2 - \frac{x_1^4}{\mu}$;

From (4.4): $x_1^2 x_2^2 = (\beta - \frac{\alpha\nu}{\mu}) x_1^2 + \frac{\nu}{\mu} x_1^4$;

From (4.8): $x_1 x_2^3 = \frac{\alpha\beta}{\mu} + (\frac{\alpha\nu^2}{\mu} - \beta\nu - \frac{\beta}{\mu}) x_1^2 - \frac{\nu^2}{\mu} x_1^4$;

From (4.6): $x_2^4 = (\beta^2 - \frac{2\alpha\beta\nu}{\mu}) + (\frac{2\nu\beta}{\mu} + \beta\nu^2 - \frac{\alpha\nu^2}{\mu}) x_1^2 + \frac{\nu^3}{\mu} x_1^4$;

Finally from (4.5) we get one equation with only one variable x_1:

$$\alpha^2 + x_1^2(\alpha\mu\nu - \beta\mu^2 - 2\alpha) + x_1^4(1 - \mu\nu) = 0.$$

5 Experimental results on XL

5.1 Experimental results with $m = n$ over GF(127)

When $m = n$ our simulation has shown that we need $D = 2^n$ in order to be able to solve the equations (so the algorithm works only for very small n).

An explanation of this is given in the paragraph 6.2.

3 variables and 3 homogenous quadratic equations, $GF(127)$

XL equations		Δ	B	XL unknowns (B degrees)	
type	Free/All	(Free+B-T-1)		T	type
l	3/3	-3	1	6	x^2
$xl \cup l$	12/12	-5	3	19	$x^3 \cup x^2 \cup x$
$x^3 l \cup xl$	30/39	-2	3	34	$x^5 \cup x^3 \cup x$
$x^5 l \cup x^3 l \cup xl$	66/102	-1	4	70	$x^7 \cup x^5 \cup x^3 \cup x$
$x^6 l \cup x^4 l \cup x^2 l \cup l$	91/150	0	4	94	$x^8 \cup x^6 \cup x^4 \cup x^2$
$x^7 l \cup x^5 l \cup x^3 l \cup xl$	121/210	0	5	5	125 $x^9 \cup x^7 \cup x^5 \cup x^3 \cup x$
$x^{17} l \cup x^{15} l \cup x^{13} l \cup \ldots$	821/1845	4	9	825	$x^{19} \cup x^{17} \cup x^{15} \cup \ldots$

4 variables and 4 homogenous quadratic equations, $GF(127)$

XL equations		Δ	B	XL unknowns (B degrees)	
type	Free/All	(Free+B-T-1)		T	type
l	4/4	-6	1	10	x^2
$x^4 l \cup x^2 l \cup l$	122/184	-5	3	129	$x^6 \cup x^4 \cup x^2$
$x^8 l \cup x^6 l \cup x^4 l \cup x^2 l \cup l$	573/1180	-3	5	580	$x^{10} \cup x^8 \cup x^6 \cup x^4 \cup x^2$
$x^{12} l \cup x^{10} l \cup \ldots$	1708/4144	-1	7	1715	$x^{14} \cup x^{12} \cup \ldots$
$x^{14} l \cup x^{12} l \cup x^{10} l \cup \ldots$	2677/6864	0	8	2684	$x^{16} \cup x^{14} \cup x^{12} \cup \ldots$

T: number of monomials **Δ** ≥ 0 when XL solves the equations, (Δ = Free+B-T-1)

B: nb. of monomials in one variable e.g. x_1 **Free/All**: numbers of free/all equations of given **type**

5.2 Experimental results with $m = n + 1$ over GF(127)

When $m = n + 1$ our simulations show that we have to take $D = n$ in order to obtain $\Delta \geq 0$ and be able to solve the equations.

4 variables and 5 homogenous quadratic equations, $GF(127)$

XL equations		Δ	B	XL unknowns (B degrees)	
type	Free/All	(Free+B-T-1)		T	type
l	5/5	-4	1	10	x^2
$xl \cup l$	25/25	-8	3	34	$x^3 \cup x^2 \cup x$
$x^2l \cup l$	45/55	1	2	45	$x^4 \cup x^2$

8 variables and 9 homogenous quadratic equations, $GF(127)$

XL equations		Δ	B	XL unknowns (B degrees)	
type	Free/All	(Free+B-T-1)		T	type
l	9/9	-27	1	36	x^2
$x^2l \cup l$	297/333	-68	2	366	$x^4 \cup x^2$
$x^4l \cup x^2l \cup l$	2055/3303	-25	3	2082	$x^6 \cup x^4 \cup x^2$
$x^5l \cup x^3l \cup xl$	4344/8280	-5	4	4352	$x^7 \cup x^5 \cup x^3 \cup x$
$x^6l \cup x^4l \cup x^2l \cup l$	8517/18747	3	4	8517	$x^8 \cup x^6 \cup x^4 \cup x^2$

T: number of monomials **Δ** ≥ 0 when XL solves the equations, (Δ = Free+B-T-1)

B: nb. of monomials in one variable e.g. x_1 **Free/All**: numbers of free/all equations of given **type**

5.3 Experimental results with $m = n + 2$ over GF(127)

In case $m = n+2$ it may be possible to take $D = \sqrt{n}+C$ but the data is still inconclusive. We are currently working on larger simulations, which will be reported in the final version of this paper.

8 variables and 10 homogenous quadratic equations, $GF(127)$

XL equations		Δ	B	XL unknowns (B degrees)	
type	Free/All	(Free+B-T-1)		T	type
l	10/10	-26	1	36	x^2
$x^2l \cup l$	325/370	-40	2	366	$x^4 \cup x^2$
$x^3l \cup xl$	919/1280	1	3	920	$x^5 \cup x^3 \cup x$

9 variables and 11 homogenous quadratic equations, $GF(127)$

XL equations		Δ	B	XL unknowns (B degrees)	
type	Free/All	(Free+B-T-1)		T	type
l	11/11	-34	1	45	x^2
$x^3l \cup xl$	1419/1914	-40	3	1461	$x^5 \cup x^3 \cup x$
$x^4l \cup x^2l \cup l$	3543/5951	2	3	3543	$x^6 \cup x^4 \cup x^2$

T: number of monomials **Δ** ≥ 0 when XL solves the equations, (Δ = Free+B-T-1)

B: nb. of monomials in one variable e.g. x_1 **Free/All**: numbers of free/all equations of given **type**

6 Complexity evaluation of XL

Given m quadratic equations with n variables, we multiply each equation by all the possible $x_{i_1} \cdot \ldots \cdot x_{i_{D-2}}$. The number of generated equations (of type $x^{D-2}l$) is about $\alpha = \frac{n^{D-2}}{(D-2)!} \cdot m$ while we have about $\beta = \frac{n^D}{D!}$ linear variables of type $x^D \cup x^{D-2}$.

If most of the equations are linearly independent in XL (we will comment on this critical hypothesis below), we expect to succeed when $\alpha \geq \beta$, i.e. when

$$m \geq \frac{n^2}{D(D-1)} \tag{7}$$

We get the following evaluation

$$D \geq \quad \text{about} \quad \frac{n}{\sqrt{m}}. \tag{8}$$

6.1 Case $m \approx n$

If $m \approx n$, and if we expect most of the equations to be independent, we expect the attack to succeed when $D \approx \sqrt{n}$. The complexity of the algorithm is thus lower bounded by the complexity of a Gaussian reduction on about $\frac{n^D}{D!}$ variables, $D \approx \sqrt{n}$. Its working factor is thus at least

$$WF \geq \left(\frac{n^{\sqrt{n}}}{\sqrt{n}!} \right)^{\omega}$$

where $\omega = 3$ in the usual Gaussian reduction algorithm, and $\omega = 2.3766$ in improved algorithms. By simplifying this expression, we get the subexponential complexity bound of approximately:

$$WF \geq e^{\omega\sqrt{n}(\frac{ln n}{2}+1)} \tag{9}$$

Notes:

- **When n is fixed** the XL algorithm is expected to run in polynomial time (in the size of K).
- **When K is fixed and** $n \to \infty$, the formula indicates that XL may run in sub-exponential time. We will see however that this is likely to be true only when $m - n$ is "sufficiently" big while still $m \approx n$. This point is the object of the study below.

6.2 Case $m = n$

When $m = n$ our simulation showed that $D = 2^n$ (instead of $D \approx \sqrt{n}$).

It is possible to give a theoretical explanation of this fact: If we look at the algebraic closure $\overline{K}^n$ of K we have generally 2^n solutions for a system of n equations with n variables. So the final univariate equation we can derive should be generally of degree 2^n.

6.3 Case $m = n + 1$

For $m = n+1$ our simulations show that $D = n$ (instead of $\sqrt{n}$). The reason for this is not clear at present.

6.4 Case $m = n + C$, $C \geq 2$

For $m = n+C$, $C \geq 2$, it seems from our simulations that even for small values of C we will have $D \approx \sqrt{n}$. This remark will lead to the FXL algorithm below.

In order to know for what value of C it is reasonable to assume that $D \approx \sqrt{n}$ we need more simulations. Many of them will be included in the extended version of this paper, however given the limited computing power available, the results do not give a precise estimation of C.

6.5 Case $m = \epsilon n^2$, $\epsilon > 0$

Let $0 < \epsilon \leq 1/2$ and $m = \epsilon n^2$. We expect XL to succeed when

$$D \approx \lceil 1/\sqrt{\epsilon} \rceil. \tag{10}$$

The working factor is in this case $WF \approx \frac{n^{\omega \lceil 1/\sqrt{\epsilon} \rceil}}{(\lceil 1/\sqrt{\epsilon} \rceil)!}$. So the algorithm is expected to be polynomial (in n) with a degree of about $\omega/\sqrt{\epsilon}$.

Remark: The fact that solving a system of $\epsilon \cdot n^2$ equations in n variables was likely to be polynomial was first suggested in [9]. Despite the fact that the relinearization is less efficient than what could have been expected, the complexity of solving ϵn^2 equations in n variables is still expected to be polynomial.

7 The FXL algorithm

In our simulations it is clear that when $m \approx n$, the smallest working degree D decreases dramatically when $m - n$ increases. For example, if $m = n$ then $D = 2^n$, if $m = n + 1$ then $D = n$, and if m is larger we expect to have $D \approx \sqrt{n}$.

We are thus led to the following extension of XL called FXL (which stands for Fixing and XL):

Definition 7.01 (The FXL algorithm)

1. *Fix μ variables (see below for the choice of μ).*
2. *Solve with XL the resultant system of m equations in $n - \mu$ variables.*

We choose the smallest possible μ such that in step 2 we have $D \approx \sqrt{n}$, in order to have minimal complexity in step 2.

The complexity of the FXL algorithm is $q^\mu e^{c\sqrt{n}lnn}$, as we have q^μ choices for μ variables in step 1, and XL is $e^{c\sqrt{n}lnn}$ for $D \approx \sqrt{n}$.

How μ increases when n increases is an open question. We can notice that if $\mu = \mathcal{O}(\sqrt{n})$, then the complexity of the FXL algorithm would be about $q^{\mathcal{O}(\sqrt{n})} e^{C\sqrt{n}lnn}$, which is approximately $e^{C\sqrt{n}(lnn+lnq)}$. Thus the FXL algorithm might be sub-exponential, even when $m = n$, but we have no rigorous proof of this conjecture.

8 XL and relinearization

In this part we explain how relinearization can be simplified and leads to XL.

We have formally proved that the set of equations defined by a successful relinearization of degree D is equivalent to a subset of equations derived from the XL algorithm with the same D.

Given a system of m quadratic equations in n variables $\mathcal{A}$.

The proof is based on a series of effective syntactic transformations on the system of equations $\mathcal{C}$ derived from the degree D relinearization of $\mathcal{A}$.

In order to simplify the proof we restricted to the case of one iteration of the relinearization method and to the homogenous equations case.

Note: It is not obvious to prove the claim, because it is not enough to show that the equations we obtain in a relinearization are contained in XL equations modulo some correspondence. Indeed in a polynomial elimination we have polynomials of very small degree (usually linear) that are always included in the ideal generated by the initial polynomials. However in order to find them we may need to generate a great many higher degree polynomials.

Therefore we need to find an effective correspondence between relinearization and XL, and this effectiveness must be of the sort to be able to transform a whole working relinearization computation process into an algorithm that is a subcase of XL.

8.1 The structure of the proof.

The proof is based on a series of effective syntactic transformations on the system of equations $\mathcal{C}$ derived from the degree D relinearization of $\mathcal{A}$.

On one hand, we transform the original relinearization.

1. We start from a working relinearization algorithm with n variables, m equations and with D tuples.
2. From the initial set of equations that we get in relinearization called $\mathcal{C}$, we generate another set of equations $\mathcal{D}$ that is an improved form of relinearization with less equations and less variables. This transformation is not bijective, but we show it preserves the feasibility of relinearization.
3. Then we transform $\mathcal{D}$ and $\mathcal{E}$ to equations on different variables, this transformation is bijective modulo an equivalence relation on the monomials, $\mathcal{D}$ and it is also shown to preserve the feasibility.

On the other hand, we do an effective, parallel construction.

1. We write the substitution that lead to $\mathcal{C}$, as a series of single substitutions, and we construct $\mathcal{C}'$ that contains the same equations written in a 'special form' that we introduce. We define a property called special degree SpecDeg and we show that $SpecDeg(\mathcal{C}') \leq D$.
2. We transform $\mathcal{C}'$ to $\mathcal{D}'$. We show that $\mathcal{D}'$ are the equations of $\mathcal{D}$ written in a special form, and with $SpecDeg(\mathcal{D}') \leq D$.
3. Then we transform $\mathcal{D}'$ to $\mathcal{E}'$. We show that $\mathcal{E}' \subset \mathcal{I}_D$, and that they are an effective expression of the equations of $\mathcal{E}$ as a subcase of XL algorithm with the same D.

The following theorem immediately follows from all the above steps:

Theorem 8.11 (Relinearization as a subcase of XL algorithm.) *For any working relinearization algorithm with n variables, m equations and with D tuples, and the equations $\mathcal{C}$. We have constructed a set of equations $\mathcal{E}$ that preserves the feasibility of the algorithm. Then we have constructed*

$$\mathcal{E}' \subset \mathcal{I}_D,$$

which is an effective expression of the equations of $\mathcal{E}$ as the subcase of XL algorithm.

Since the proof is constructive, one must be very careful about the details and we need to describe precisely each step. We are going to explain the whole process on an example, with the preservation proofs that come at the end.

Comments: One might say that the relinearization method is as a syntaxical transformation, and $\mathcal{E}$ gives a semantics (a meaning) to the equations obtained in the relinearization. It is not exactly all the equations that appear in the XL algorithm, and such is not the claim of this proof. We only prove that they are contained in XL. However we claim that it fills 90% of the XL equations space. The XL equation sets are smaller, for example, to solve 11 equations with 9 variables relinearization generated 12716 equations, 7769 of them were independent, while with XL the same computation is done with only about 3543 equations.

So we do not claim that relinearization gives exactly XL, but that XL does the same computation simpler and with less equations.

The interest of the proof is small, compared to the understanding that the whole idea of adding new variables and substituting is bad because it is bounded to generate a great many dependent equations. It has been discovered in the relinearization (in the beginning of the present paper), however there is a general reason it is so. Let us explain this reason first on a small example:

Why adding new variables is a bad idea: Let the equation to solve be $l_1 = 0$ with

$$l_1 = y_{13} + y_{12} - 1.$$

A parametrized solution to such equation(s) might contain:

$$y_{13} = 1 - y_{12}.$$

We observe that substituting a parametrized solution of the system to equations like, let's say $y_{12}y_{34} = y_{13}y_{24}$ is equivalent to adding terms like l_1y_{24} to these equations.

This is necessarily redundant because the l_i are linear in the y_{ij}, the number of all the l_iy_{jk} is usually about $\frac{n^3}{2}$ while they live in a space of dimension $\frac{n^3}{3!}$.

Such linear dependencies are commonly called syzygies in the vocabulary of Gröbner bases. Relinearization added new variables, and thus new syzygies. We claim (informal statement) that our transformation (described below) eliminates all the syzygies in the l_i and the y_{ij}. However it does not eliminate the syzygies between the l_i and the x_i. There is a method that removes them, according to the author of it [5]. It is called F_4, is much more complex compared to XL, see [5], and it is unclear if it really produces better results in practice.

Comments, conclusion: Finally, the only (but a very important) contribution of the relinearization method is by drawing attention to the fact that algorithms similar to XL (known at least since the 80's) are likely to be subexponential when $m > n$ while a lot of people working with Gröbner bases have been considering mostly the case $m = n$.

Presentation of the original relinearization.

Linearization algorithm (folklore) consists of solving a system of equations, considering it as a system of linear equations. To achieve this simply add new variables that correspond to **all** the monomials present in the system.

Example: Given ϵn^2 quadratic equations with n variables with $\epsilon \geq 1/2$, we put $y_{ij} = x_ix_j$ and we have at least M linear equations with M variables, $M \approx n^2/2$.

Definition 8.12 (Trivial Equations) *We call trivial all the equations on the y_{ij} that reduce to $0 = 0$ after substitution of the $y_{ij} = x_ix_j$. Example: $y_{12}y_{34} = y_{24}y_{13}$. In fact they are generated by partitions of the set of indexes, in parts of size 1 and 2.*

When $m < n^2/2$ linearization doesn't work any more, and in this case we do what is called the relinearization technique [9]: We increase the number of equations by adding some new (but trivial) equations that express how the new variables y_{ij} are related.

Then we get a new set of bounded degree equations that is expected to be easier solve that the initial one.

Detailed description of the relinearization. We restrict to single-iteration relinearization applied to solving a set of homogenous quadratic equations.

We write the system to solve

$$f_i(x_1, \ldots, x_n) = c_i, i = 1..m$$

as

$$\mathcal{A} = \{l_i = 0\}$$

with $l_i = f_i - c_i$.

1. **Parametrize.** Let $y_{ij} = x_i x_j$ be new variables. We rewrite the initial m equations $\mathcal{A} = \{l_i = 0\}_{i=1..m}$ as linear expressions in about $n^2/2$ variables $\{y_{ij}\}$. It is called $\mathcal{A}' = \{l'_i = 0\}$.
   We are supposed to find a parametric solution of this linear system which expresses each of y_{ij} as a linear expression in some $n^2/2 - m = (1/2 - \epsilon)n^2$ new variables z_i. On a field, it is an unnecessary complication, because if we consider a maximum rank independent set among the y_{ij} expressions, we inverse this linear transformation, and express all the other y_{ij} as a function of the independent set.
   Therefore we may suppose without loss of generality that the z_i are a subset of the y_{ij} variables.
   Let $\mathcal{S}_{\mathcal{A}'}$ be such a parametric solution of $\mathcal{A}'$.
2. **Trivial equations.** We write trivial equations in all the variables y_{ij} of degree at most $D > 1$, e.g. $y_{12}y_{34} = y_{13}y_{24}$. These algebraically dependent equations $\mathcal{B}$ are linearly independent as multivariate polynomials of degree $\leq D$. Let $M_{\mathcal{B}}$ be the number of equations in $\mathcal{B}$.
3. **Substitution.** Let $\mathcal{C}$ be the set of equations that we obtain when substituted all the variables of $\mathcal{B}$ with their linear expressions of $\mathcal{S}_{\mathcal{A}'}$ (In a way $\mathcal{C} = \mathcal{B} \circ \mathcal{S}_{\mathcal{A}'}$).
   Let $M_{\mathcal{C}}$ be the number of linearly independent equations in $\mathcal{C}$. Let $T_{\mathcal{C}}$ be the number of all monomials in $\mathcal{C}$.
4. **Solving.** For some degree D we expect:
$$M_{\mathcal{C}} \geq T_{\mathcal{C}} \tag{11}$$
   and we solve by Gauss elimination on the $T_{\mathcal{C}}$ terms.

Starting from the description of relinearization above, we will subsequently transform it. The whole proof will be explained on a simple example.

A small example Let a system to solve be

$$\mathcal{A} = \begin{cases} 0 = l_1 \\ \vdots \\ 0 = l_n \end{cases}$$

Let one of the equations in $\mathcal{A}$ be

$$l_1 \;=\; f_1(x) - c_1 \;=\; x_1x_2 + x_1x_3 - 1.$$

In the transformed (linearized) system $\mathcal{A}'$ the same equation becomes:

$$\mathcal{A}' = \begin{cases} 0 = l'_1 \;-\; y_{12} + y_{13} - 1 \\ \vdots \end{cases}$$

The parametric solution could be $\mathcal{S}_{\mathcal{A}'} = \begin{cases} y_{13} = 1 - y_{12} \\ y_{12} = y_{12} \\ \vdots \end{cases}$

In $\mathcal{S}_{\mathcal{A}'}$, all the differences between both sides of the '=' sign are simple linear combinations of initial equations $l'_j \in \mathcal{A}'$.

We generate the set of all trivial equations with at most D indexes called $\mathcal{B}$, in our example $D = 4$ and the trivial equations contain for example this one:

$$\mathcal{B} = \begin{cases} 0 = y_{12}y_{34} - y_{13}y_{24} \\ \vdots \end{cases}.$$

For each trivial equations of $\mathcal{B}$ we substitute the parametric solution $\mathcal{S}_{\mathcal{A}'}$. Thus in the equation above we substitute $y_{13} = 1 - y_{12}$ and get:

$$\mathcal{C} = \begin{cases} 0 = y_{12}y_{34} - (1 - y_{12})y_{24} \\ \vdots \end{cases}.$$

The key argument of the whole construction is the following:

8.2 Decomposing substitutions and special equations.

Any substitution of the y_{ij} can be decomposed as a chain of single substitutions of one variable y_{ij}. Each single substitution is equivalent to adding a linear combination of l'_j multiplied by a multiplicative term in y_{ij} of degree ≥ 0.

Definition 8.21 *A 'special degree' of a product of the l'_j and the y_{ij} is:*

SpecDeg $\stackrel{def}{=} 2 \cdot$ (*number of l'_i with repetitions*) +
+ (*number of different indexes in the y_{ij} with repetitions*)

A 'special degree' (SpecDeg) of an expression being a sum of products of the l'_j and the y_{ij}, is by definition the Max of SpecDeg for all the terms in the expression.

Definition 8.22 *We call 'special expressions' the sums of products of the l'_j and the y_{ij}.*

Definition 8.23 *We call 'true special expressions' sums of the special expressions with all terms of the form $l'_i \cdot$(a special expression). The reason is we call them 'true', is that in practical equations solving $l'_i = 0$, and it means that the terms containing a factor l'_i are the equations that are true each time x satisfies the intial equations.*

The trivial equations of $\mathcal{B}$ are special equations with $\text{SpecDeg}(\mathcal{B}) \leq D$. We observe that

1. SpecDeg of an expression does not increase when we substitute an y_{ij} by it's linear expression in other y_{ij}.
2. SpecDeg of an expression does not increase when we rewrite this substitution of y_{ij} as adding an element of form $l'_i \cdot$ (a special expression).
3. SpecDeg of a special expression does not increase when we substitute an l'_j by it's linear expression in the y_{ij}. **An important detail** of the proof is that we never do this last thing, unless a term is quadratic in the l'_j (explained later).

We decompose a set of substitutions that lead from $\mathcal{B}$ to $\mathcal{C}$ into single substitutions, and each time we rewrite it conserving exactly the original equations, and adding 'true special expressions' (have to contain only terms being a multiple of one of the l'_i). We have described an effective way to express $\mathcal{C}$ as special expressions with $\text{SpecDeg}(\mathcal{C}') \leq D$.

At the end we may eliminate all the $l'_i \cdot l'_j$ products by substituting one of the l'_i by it's expression in the y_{ij}. This operation also does not increase the SpecDeg.

We have just demonstrated that all the equations $\mathcal{C}$ are linear combinations of:

- trivial equations of $\mathcal{B}$ with SpecDeg $\leq D$.
- special expressions with SpecDeg $\leq D$ and terms with at least one l'_i.

Let $\mathcal{C}'$ be this (equivalent) way of writing $\mathcal{C}$.

On our example:

$$\mathcal{C} = \begin{cases} 0 = y_{12}y_{34} - (1 - y_{12})y_{24} \\ \vdots \end{cases}.$$

$$\mathcal{C}' = \begin{cases} 0 = (y_{12}y_{34} - y_{13}y_{24}) \quad + y_{24}l'_1 \\ \vdots \end{cases}.$$

$\mathcal{C}$ and $\mathcal{C}'$ are two different representations of the same equations.

8.3 Eliminating a part of the equations, $\mathcal{C} \mapsto \mathcal{D}$, $\mathcal{C}' \mapsto \mathcal{D}'$.

The next step of the transformation is to eliminate in $\mathcal{C}$ some terms and some equations, in order to have only one term with the same set of indexes present. We do it rearranging indexes in increasing order, it gives:

$$\mathcal{D} = \begin{cases} 0 = y_{12}y_{34} - y_{24} - y_{12}y_{24} \\ \vdots \end{cases}.$$

It will be shown later that this transformation preserves the feasibility of the original algorithm. (In practice also reduces the size of the system to solve.)

We do the same transformation on $\mathcal{C}'$ (replacing the terms in the y_{ij} **but** without changing the l'_i). It preserves SpecDeg and we call $\mathcal{D}'$ the output of this transformation on $\mathcal{C}'$. On the example

$$\mathcal{D}' = \begin{cases} 0 = (y_{12}y_{34} - y_{12}y_{34}) \quad + l'_1 y_{24} \\ \vdots \end{cases}.$$

$\mathcal{D}$ and $\mathcal{D}'$ are two different representations of the same equations.

8.4 Return to the original variables: $\mathcal{D} \mapsto \mathcal{E}$, $\mathcal{D}' \mapsto \mathcal{E}'$.

Let $\mathcal{E}$ be what we get when we substitute all the variables $y_{ij} \leftarrow x_i x_j$ in the equations $\mathcal{D}$:

$$\mathcal{E} = \begin{cases} 0 \;=\; x_1x_2x_3x_4 - x_2x_4 - x_1x_2^2x_4 \\ \vdots \end{cases}.$$

For $\mathcal{D}'$ we do the same thing, we substitute $y_{ij} \leftarrow x_i x_j$. We also substitute the variables absent in $\mathcal{D}$, namely we substitute $l'_j \leftarrow l_j$. We don't replace (yet) the l_i with their expressions in the x_i, Thus we get $\mathcal{E}'$:

$$\mathcal{E}' = \begin{cases} 0 \;=\; 0 \;+\; l_1x_2x_4 \\ \vdots \end{cases}.$$

Now we re-write the series of all the successive steps that led to $\mathcal{C}'$. Then for each step we replace $y_{ij} \leftarrow x_i x_j$ and $l'_i \leftarrow l_i$.

We note that:

1. The trivial equations will all disappear in this substitution.
2. Each term added was a 'true special expression' of SpecDeg$\leq D$ (it contained at least one l'_i). It will be replaced by a term of the form $(l_i) \cdot \prod_j x_j$ with the maximum degree $D-2$ in the x_i.

Since $\mathcal{E}$ and $\mathcal{E}'$ are two different representations of the same equations, given the form of $\mathcal{E}'$ we have proven that:

All the equations in $\mathcal{E}'$ are in what we call

$$\mathcal{I}_D = x^{D-2}l \cup x^{D-3}l \cup \ldots \cup l \subset \mathcal{I}$$

It is the subset of the ideal $\mathcal{I}$ generated by the equations l_i with the total degree upper bounded by D.

On our example, we have effectively obtained in $\mathcal{E}'$ the equation

$$l_1x_2x_4 \;\in\; x^2l \subset \mathcal{I}_4.$$

And in $\mathcal{E}$ there is the same equation but expanded.

8.5 Preservation proofs.

We have a precise construction how to obtain the same equations $\mathcal{E}$ under a different form $\mathcal{E}'$ that are explicitely proving their being in $x^{D-2}l \cup x^{D-3}l \cup \ldots \cup l$.

The only thing that remain to be proven is that the transformations

$$\mathcal{C} \to \mathcal{D} \to \mathcal{E}$$

preserve the solvability of equations.

First preservation proof. We start with an original relinearization that works with a given (n, m, D).

We usually have the number of variables y_{ij} (about $n^2/2$) much bigger than the number m of initial equations in $\mathcal{A}$. Thus most of the expressions of y_{ij} in the parametric solution $\mathcal{S}_{\mathcal{A}'}$ are synonymous $y_{ij} = y_{ij}$. They will not be changed in substitution of trivial equations $\mathcal{B}$ by $\mathcal{S}_{\mathcal{A}'}$ and many of the equations of $\mathcal{B}$ will remain trivial in $\mathcal{C}$. To avoid these equations we constructed $\mathcal{D}$:

Construction 8.51 *Detailed construction of $\mathcal{D}$. Repeat the substitutions that led to $\mathcal{C}$ and construct from the beginning $\mathcal{D}$ instead, writing all the indexes arranged in the increasing order, so the equations that remain trivial will be eliminated from the beginning without ever generating them.*

For a given term $Y = \prod y_{ij}$ of degree ν let $\overline{Y}$ be the the class of all terms that have the same set of indexes. The cardinal of $\overline{Y}$ is:

$$\#\overline{Y} = \Theta_\nu = (2\nu - 1)(2\nu - 3)(2\nu - 5) \ldots . = (2\nu)!/(2^\nu \nu!).$$

Since $\overline{Y}$ has Θ_ν elements, we have put in $\mathcal{B}$ exactly $\Theta_\nu - 1$ corresponding trivial equations. Trivial equations that are in $\mathcal{B}$ and remain the same in $\mathcal{C}$ will disappear in $\mathcal{D}$ because their terms have the same set of indexes. Therefore:

Proposition 8.52 *In each equivalence class $\overline{Y}$ the difference between $\mathcal{C}$ and $\mathcal{D}$ is:*

- *At most $\Theta_\nu - 1$ equations disappear (those that exist in $\mathcal{C}$).*
- *Exactly $\Theta_\nu - 1$ variables disappear.*

Example: Let us suppose that y_{12}, y_{34}, y_{13} and y_{24} are unchanged in $\mathcal{S}_{\mathcal{A}'}$. Each time we write $y_{13}y_{24}$ in $\mathcal{C}$, we write $y_{12}y_{34}$ instead in $\mathcal{D}$. There are $\Theta_2 = 2$ terms related to this one, we suppress $1 = \Theta_2 - 1$ equations and also we have removed $1 = \Theta_2 - 1$ variables.

Globally:

Corollary 8.53 *If there are more equations than terms in $\mathcal{C}$ it is also true for $\mathcal{D}$.*

$$T_{\mathcal{C}} - M_{\mathcal{C}} = T_{\mathcal{D}} - M_{\mathcal{D}} \tag{12}$$

Second preservation proof. We recall the construction: we substitute $y_{ij} = x_i x_j$ in all equations $\mathcal{D}$. We call $\mathcal{E}$ these equations, in fact we could have generated them directly without writing $\mathcal{C}$ and $\mathcal{D}$ equations.

It's obvious that the numbers of terms and independent equations are the same in $\mathcal{D}$ and $\mathcal{E}$:

Proposition 8.54 *If there is more equations than terms in $\mathcal{C}$ it is also true for $\mathcal{D}$ and $\mathcal{E}$.*

$$T_{\mathcal{C}} - M_{\mathcal{C}} = T_{\mathcal{D}} - M_{\mathcal{D}} = T_{\mathcal{E}} - M_{\mathcal{E}} \tag{13}$$

The whole construction and proof is completed now.

Relinearization vs. XL in practice In practice, XL can be really better than relinearization. For example, to solve 11 equations with 9 variables, relinearization requires the solution of a linear system with 7769 variables (see Table 3), whereas XL requires the solution of a system with only 3543 variables (see 5.3).

Moreover, XL can use any D while relinearization can only use composite values of D. For example, to solve 10 quadratic equations with 8 variables we had to use the relinearization algorithm with $D = 6$, but the XL algorithm could use the smaller value of $D = 5$. Consequently, the system of linear equations derived from linearization had 3653 variables, while the system of linear equations derived from XL had only 919 variables (see 5.3).

9 Gröbner bases algorithms

One way of implementing the XL algorithm is to combine the equations in an organised way, rather than to multiply them by all the possible monomials. This would naturally lead to the classical Gröbner-bases algorithms.

We define $\mathcal{I}_{x_{i_1},\ldots,x_{i_j}}$ as a subspace of all the equations of $\mathcal{I}$ that can be written with just the variables $x_{i_1},\ldots,x_{i_j}$. The XL method checks if there are any (univariate) equations in some $(\mathcal{I}_D)_{x_1}$.

The Gröbner bases algorithms construct a basis of a space of (univariate) equations in $\mathcal{I}_{x_1} = \bigcup_k (\mathcal{I}_k)_{x_1}$. However in order to get there, they compute successively bases of the $\mathcal{I}_{x_1,\ldots,x_k}$ for $k = n \ldots 1$.

It is not clear what is the best way to use Gröbner bases to solve our problem of overdefined systems of equations. A large number of papers have been written on Gröbner base techniques, but most of them concentrate either on the case of fields of characteristic 0, or look for solution in an algebraic closure of $\overline{K}^n$, and the complexity analysis of these algorithms is in general very difficult.

10 Cryptanalysis of HFE with XL/relinearization attacks

The HFE (Hidden Field Equations) cryptosystem was proposed at Eurocrypt 1996 [11]. Two different attacks were recently developed against it [3, 9], but they do not compromise the practical security of HFE instances with well chosen parameters. Moreover it does not seem that these attacks can be extended against variations of the HFE scheme such as HFEv or HFEv$^-$ described in [8].

The first type of attack (such as the affine multiple attack in [11]) tries to compute the cleartext from a given ciphertext. It is expected to be polynomial when the degree d of the hidden polynomial is fixed, and not polynomial when $d = \mathcal{O}(n)$. In [3] Nicolas Courtois presented several improved attacks in this category, with an expected complexity of $n^{\mathcal{O}(ln(d))}$ (which is still not polynomial) instead of the original complexity of $n^{\mathcal{O}(d)}$.

A second line of attack tries to recover the secret key from the public key. The Kipnis-Shamir attack described in [9] was the first attack of this type. It is also expected to be polynomial when d is fixed but not polynomial when $d = \mathcal{O}(n)$.

To test the practicality of these attacks, consider the HFE "challenge 1" described in the extended version of [11] and in [4]. It is a trapdoor function over $GF(2)$ with $n = 80$ variables and $d = 96$. A direct application of the FXL to these 80 quadratic equations requires Gaussian reductions on about $80^9/9! \approx 2^{38}$ variables, and thus its time complexity exceeds the 2^{80} complexity of exhaustive search, in spite of its conjectured subexponential asymptotic complexity. The best attack on the cleartext (from [3]) is expected to run on "challenge 1" in time 2^{62}. The best attack on the secret key (from [9]) is expected to run in time 2^{152} when XL is used, and to take even longer when relinearization is used. A possible improvement of this attack (from [3], using sub-matrices) runs in time 2^{82}, which is still worse than the 2^{80} complexity of exhaustive search.

11 Conclusion

In this paper we studied the relinearization technique of Kipnis and Shamir, along with several improvements. We saw that in high degree relinearizations the derived equations are mostly linearly dependent, and thus the algorithm is much less efficient than originally expected.

We have related and compared relinearization to more general techniques, such as XL and Gröbner bases. We have proved that XL "contains" relinearization and demonstrated that it is more efficient in practice. We also concluded that the complexity of solving systems of multivariate equations drops rapidly when the number of equations exceeds the number of variables (even by one or two). Consequently, over a small field the FXL algorithm may be asymptotically subexponential even when $m = n$, since it guesses the values of a small number of variables in order to make the system of equations slightly overdefined. However in many practical cases with fixed parameters $m \approx n$, the best known algorithms are still close to exhaustive search.

Finally, when the number of equations m and the number of variables n are related by $m \geq \epsilon n^2$ for any constant $0 < \epsilon \leq 1/2$, the asymptotic complexity seems to be polynomial with an exponent of $\mathcal{O}(1/\sqrt{\epsilon})$.

References

1. Iyad A. Ajwa, Zhuojun Liu, and Paul S. Wang: "Grobner Bases Algorithm", ICM Technical Reports, February 1995, see http://symbolicnet.mcs.kent.edu/icm/reports/index1995.html.
2. Don Coppersmith: "Finding a small root of a univariate modular equation"; Proceedings of Eurocrypt'96, Springer-Verlag, pp.155-165.
3. Nicolas Courtois: *The security of Hidden Field Equations (HFE)*; Cryptographers' Track Rsa Conference 2001, San Francisco 8-12 Avril 2001, LNCS2020, Springer-Verlag.
4. Nicolas Courtois: The HFE cryptosystem home page. Describes all aspects of HFE and allows to download an example of HFE challenge. http://hfe.minrank.org
5. Jean-Charles Faugère: "A new efficient algorithm for computing Gröbner bases (F_4)." Journal of Pure and Applied Algebra 139 (1999) pp. 61-88. See www.elsevier.com/locate/jpaa
6. Jean-Charles Faugère: "Computing Gröbner basis without reduction to 0", technical report LIP6, in preparation, source: private communication.
7. Rudolf Lidl, Harald Niederreiter: "Finite Fields"; Encyclopedia of Mathematics and its applications, Volume 20, Cambridge University Press.
8. Aviad Kipnis, Jacques Patarin, Louis Goubin: "Unbalanced Oil and Vinegar Signature Schemes" ; Eurocrypt 1999, Springer-Verlag, pp. 216-222.
9. Aviad Kipnis, Adi Shamir: "Cryptanalysis of the HFE Public Key Cryptosystem"; Proceedings of Crypto'99, Springer-Verlag.
10. Neal Koblitz: "Algebraic aspects of cryptography"; Springer-Verlag, ACM3, 1998, Chapter 4 "Hidden Monomial Cryptosystems", pp. 80-102.
11. Jacques Patarin: "Hidden Field Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of Asymmetric Algorithms"; Eurocrypt'96, Springer Verlag, pp. 33-48. An extended up-to-date version can be found at http://www.univ-tln.fr/∼courtois/hfe.ps