

# Improved Strategies for Solving Multivariate Polynomial Equation Systems over Finite Fields

Vom Fachbereich Informatik der  
Technischen Universität Darmstadt genehmigte

## Dissertation

zur Erlangung des Grades  
Doktor rerum naturalium (Dr. rer. nat.)

von

**M.Sc. Mohamed Saied Emam Mohamed**

geboren in Qaliubiya, Ägypten.



Referenten:	Prof. Dr. Johannes Buchmann Prof. Dr. Jintai Ding
Tag der Einreichung:	6. April 2011
Tag der mündlichen Prüfung:	6. June 2011
Hochschulkenziffer:	D 17

Darmstadt 2011



*To my God and my parents,  
To my wife and my daughters,  
To all my family and my in-laws,  
To my country Egypt*



# Wissenschaftlicher Werdegang

## **März 2007 - heute**

Promotionsstudent am Lehrstuhl von Professor Johannes Buchmann, Fachgebiet Theoretische Informatik, Fachbereich Informatik, Technische Universität Darmstadt

## **Oktober 2004 - heute**

Lehrbeauftragter an der Fakultät für Informatik und Informationssysteme, Fachbereich Informatik, Zagazig Universität, (Zagazig, Ägypten)

## **Dezember 1998 - Oktober 2004**

Ausbilder an der Fakultät für Informatik und Informationssysteme, Fachbereich Informatik, Zagazig Universität (Zagazig, Ägypten)

## **September 2004**

Master of Science in Informatik an der Fakultät für Informatik und Informationssysteme, Fachbereich Informatik, Helwan Universität (Helwan, Ägypten)

## **Juli 2000 - September 2004**

Masterstudium der Informatik an der Fakultät für Informatik und Informationssysteme, Fachbereich Informatik, Helwan Universität (Helwan, Ägypten)

## **September 1998 - Juni 1999**

Besuch vorbereitender Kurse für das Masterstudium an der Fakultät für Informatik und Informationssysteme, Fachbereich Informatik, Helwan Universität (Helwan, Ägypten)

## **Juni 1997**

Bachelor of Science in Reinen Mathematik und Informatik an der Fakultät für Naturwissenschaften, Ain Shams Universität (Kairo , Ägypten)

## **September 1993 - Juni 1997**

Studium der Reinen Mathematik und Informatik an der Fakultät für Naturwissenschaften, Ain Shams Universität (Kairo, Ägypten)

# Publikationsliste

- [1] Ding, J., Buchmann, J., Mohamed, M. S. E., Mohamed, W. S. A., Weinmann, R. P.: MutantXL. The First International Conference on Symbolic Computation and Cryptography - SCC 2008, pages 16 – 22, 2008.
- [2] Mohamed, M. S. E., Mohamed, W. S. A., Ding, J., Buchmann, J.: MXL2: Solving Polynomial Equations over  $GF(2)$  Using an Improved Mutant Strategy. The Second International Workshop on Post-Quantum Cryptography - PQCrypto 2008, Volume 5299 of Lecture Notes in Computer Science, pages 203–215, Springer-Verlag, 2008.
- [3] Mohamed, M. S. E., Ding, J., Buchmann, J., Werner, F.: Algebraic Attack on the MQQ Public Key Cryptosystem. 8th International Conference on Cryptology And Network Security - CANS 2009, Volume 5888 of Lecture Notes in Computer Science, pages 392–401, Springer-Verlag, 2009.
- [4] Mohamed, M. S. E., Cabarcas, D., Ding, J., Buchmann, J., Bulygin, S.: MXL3: An Efficient Algorithm for Computing Gröbner Bases of Zero-Dimensional Ideals. 12th International Conference on Information Security and Cryptology - ICISC 2009, Volume 5984 of Lecture Notes in Computer Science, pages 87–100, Springer-Verlag, 2010.
- [5] Buchmann, J., Cabarcas, D., Ding, J., Mohamed, M. S. E.: Flexible Partial Enlargement to Accelerate Gröbner Basis Computation over  $\mathbb{F}_2$ . 3rd Africacrypt Conference - Africacrypt 2010, Volume 6055 of Lecture Notes in Computer Science, pages 69–81, Springer-Verlag, 2010.
- [6] Mohamed, M. S. E., Ding, J., Buchmann, J.: Towards Algebraic Cryptanalysis of HFE Challenge 2 Using MXL3. 5th International Conference on Information Security and Assurance - ISA 2011, To appear, Lecture Notes in Computer Science, Springer-Verlag, 2011.
- [7] Mohamed, M. S. E., Ding, J., Buchmann, J.: A Tight Upper Bound for the Complexity of Computing Gröbner Bases over  $\mathbb{F}_2$ . The 18th International Workshop on Selected Areas in Cryptography - SAC 2011, Submitted, Springer-Verlag, 2011.
- [8] Mohamed, M. S. E., Bulygin, S., Buchmann, J.: Improved Differential Fault Analysis of Trivium. 2nd International Workshop on Constructive Side-Channel Analysis and Secure Design - COSADE 2011, pages 147–158, 2011.

# Acknowledgement

First of all I would like to thank Prof. Dr. Johannes Buchmann for accepting my application to develop my PhD in his research group and for supporting me through the last four years. Also, I would like to thank Prof. Dr. Jintai Ding for supporting me with his suggestions and ideas. I would have never finished this work without their help. They always provide me continuous advices and guide me to further clear up my ideas.

Further, I would like to thank Wael Saied Tawah and Ralf-Philipp Weinmann for sharing me the discussions and helping me during the first part of my thesis. Also, I would like to thank Daniel Cabarcas and Stanislav Bulygin for their efforts in the second part of my thesis. Additionally, I would like to thank my colleagues at the CDC-team for their support, especially to Albrecht Petzoldt for translating me the abstract of this thesis into German. Also, I would like to thank CASED and CASED members for supporting me during the last year.

Mohamed Yosif, Ahmed abdel kader, and Slim Kallel are not only my best friends during my stay in Darmstadt, but also they helped me a lot for successfully finishing this thesis. Also, I would like to thank all my Egyptian friends in Darmstadt.

Special thanks go to my Moroccan friends Sidi Mohamed El Yousfi Alaoui and Mohammed Meziani for their help and their support during the last year.

Moreover, I expressly gratitude to my country Egypt for supporting my PhD studies financially at Darmsttat University of Technology.

Most of all, I would like to thank my parents for their continuous support throughout my whole life. It is fair to say that without them, I would not become what I am now.

Last but not least, I would like to thank my wife for her support and for giving me the environment that enabled me to finish this work. I will never forget how she took care of me, overcame all hard moments and insisted on completing with me which I already started. Further, I thank my tow daughters (Jana and Judi) for giving me the smile which renew my energy for making a good research.

# Zusammenfassung

Eine der wichtigen Fragestellungen in der Kryptographie ist das Problem, multivariate polynomiale Gleichungen über endlichen Körpern zu lösen. Die Komplexität dieses Problems ist das Maß für die Sicherheit vieler Public-Key-Kryptosysteme sowie vieler symmetrischer Kryptosysteme wie Block und Stromchiffren. In den letzten Jahren wurde die algebraische Kryptanalyse als eine Methode des Angriffs auf Kryptosysteme vorgestellt. Diese Methode beruht auf der Lösung multivariater polynomialer Systeme. Daher ist die Entwicklung von Algorithmen zur Lösung solcher Systeme ein heies Forschungsthema.

In den letzten Jahren wurden verschiedene Algorithmen vorgeschlagen, multivariate polynomiale Systeme über endlichen Körpern zu lösen. Eine vielversprechende Methode hierbei basiert darauf, eine Lösung durch die Erweiterung des Systems durch Generierung zusätzlicher Gleichungen mit Hilfe von Techniken der linearen Algebra zu finden. Theoretische Abschätzungen der Komplexität haben ergeben, dass auf viele realistische Anwendungen algebraische Angriffe mit den derzeitigen Algorithmen nicht möglich sind. Der Grund hierfür ist die Tatsache, dass die Berechnungen mit diesen Algorithmen viel Zeit und Speicherressourcen erfordern. Eine große Herausforderung ist es daher, diese Algorithmen zu verbessern, um in der Lage zu sein, große multivariate polynomiale Systeme mit den in der Praxis begrenzt verfügbaren Zeit und Speicherressourcen zu lösen.

In dieser Arbeit schlagen wir Strategien vor, um den Erweiterungsschritt dieser Algorithmen zu verbessern. Wir wenden diese Strategien aufgrund seiner einfachen Struktur auf den gut untersuchten XL-Algorithmus an und zeigen, dass die Kombination dieser Strategien mit XL diesen zu einem im Vergleich mit State-of-the-art Algorithmen in hohem Maße wettbewerbsfähigem Algorithmus macht.

Im Jahre 2006 präsentierte Jintai Ding das Konzept der mutants. Mutants sind Polynome von kleinerem als dem erwarteten Grad, die man während des Lineare-Algebra-Schrittes von XL erhält. Der in dieser Arbeit vorgestellte MutantXL Algorithmus nutzt das Konzept der mutants um den Lösungsprozess des XL-Algorithmus zu verbessern. Der  $MXL_2$ -Algorithmus wird als eine verbesserte Version des MutantXL-Algorithmus eingeführt, indem eine partielle Erweiterungsstrategie entwickelt wird. Konkret ändern wir MutantXL in einer Weise, dass im Erweiterungsschritt die Menge der generierten Polynome bis zu einem bestimmten Grad  $D$  betrachtet wird, eine Teilmenge nach der anderen, ohne dass dazu die Menge der Polynome auf einmal gespeichert werden muss. Dies führt zur Lösung von Systemen mit weniger erweiterten Polynomen als bei MutantXL.

Der Hauptnachteil von  $MXL_2$  sowie XL und MutantXL ist es, dass nur Systeme mit einer einzigen Lösung gelöst werden können. Um Systeme mit einer endlichen

## Publikationsliste

Zahl von Lösungen zu lösen, präsentieren wir eine neue hinreichende Bedingung dafür, dass eine Menge von Polynomen eine Gröbnerbasis ist. Wir haben diese neue Bedingung als Abbruchkriterium des  $\text{MXL}_2$ -Algorithmus benutzt. Diese Änderung wird gemeinsam mit weiteren Verbesserungen des Erweiterungsschrittes von  $\text{MXL}_2$  in den  $\text{MXL}_3$ -Algorithmus zur Berechnung von Gröbnerbasen eingeführt. Diese Arbeit stellt auch den  $\text{MGB}$ -Algorithmus vor, der eine flexible partielle Erweiterungsstrategie verwendet, um eine wesentliche Verbesserung gegenüber  $\text{MXL}_3$  zu bieten. Die am Ende der Arbeit vorgestellte vorläufige Studie schlägt eine neue Obergrenze für die Komplexität der Berechnung von Gröbnerbasen vor, die zu neuen Denkmödeln über die Komplexität von Gröbnerbasisberechnungen anregt.

Die Ergebnisse dieser Arbeit zeigen, dass die vorgeschlagenen Strategien die Leistung des  $\text{XL}$ -Algorithmus dramatisch verbessern. Darüber hinaus führen sie zu Algorithmen, die die  $\text{MAGMA}$ -Implementierung von  $F_4$ , einen der derzeit effizientesten Algorithmen, bezüglich Zeit und Speicherverbrauch in vielen Fällen übertreffen. Darüber hinaus wird eine angepasste Version des  $\text{MutantXL}$ -Algorithmus dazu verwendet, das  $\text{MQQ}$  Kryptosystem schneller und mit weniger Speicherverbrauch anzugreifen, als das mit  $F_4$  möglich ist.

# Abstract

One of the important research problems in cryptography is the problem of solving multivariate polynomial equations over finite fields. The hardness of solving this problem is the measure of the security of many public key cryptosystems as well as of many symmetric cryptosystems, like block and stream ciphers. In recent years, algebraic cryptanalysis has been presented as a method of attacking cryptosystems. This method consists in solving multivariate polynomial systems. Therefore, developing algorithms for solving such systems is a hot research topic.

Over the recent years, several algorithms have been proposed to solve multivariate polynomial systems over finite fields. A very promising type of these algorithms is based on enlarging a system by generating additional equations and using linear algebra techniques to obtain a solution. Theoretical complexity estimates have shown that algebraic attacks made using these algorithms are infeasible for many realistic applications. This is due to the fact that, in many practical cases, the computations made by these algorithms require a lot of time and memory resources. A big challenge is to improve this algorithm in order to be able to use the limited available memory and time resources to solve large multivariate polynomial systems which exist in practice.

In this thesis we propose strategies to improve the enlargement step of these algorithms. We apply these strategies to the well studied XL algorithm, due to its simple structure, and show that combining these strategies with XL makes it highly competitive to the state-of-the-art algorithms.

In 2006, Jintai Ding presented the concept of mutant polynomials [19]. Mutants are polynomials of a lower degree than expected that appear during the linear algebra step of XL. The MutantXL algorithm presented in this thesis uses the concept of mutants to improve the solving process of the XL algorithm. The  $MXL_2$  algorithm is introduced as an improved version of the MutantXL algorithm by developing a partial enlargement strategy [40]. Specifically, we modify MutantXL in a way such that when it enlarges the system, it partitions the set of polynomials of the maximal degree  $D$  into some subsets using a special criteria. After that it explores this set of polynomials, one subset at a time, without being forced to store the whole set at once. This results in solving systems with fewer number of enlarged polynomials than MutantXL.

The main drawback of  $MXL_2$ , as well as XL and MutantXL algorithms, is that it can solve only systems having a unique solution. In order to solve systems with a finite number of solutions, we present a new sufficient condition for a set of polynomials to be a Gröbner basis [37]. We used this new condition as a termination criteria for the  $MXL_2$  algorithm. This modification together with further improvements to

the enlargement step of  $\text{MXL}_2$  are introduced in the  $\text{MXL}_3$  algorithm for computing Gröbner bases. This thesis also introduces the MGB algorithm which uses a flexible partial enlargement strategy [11] to provide an important improvement to  $\text{MXL}_3$ . The preliminary study presented at the end of the thesis suggests a new upper bound for the complexity of computing Gröbner bases which motivates thinking of new paradigms for estimating the complexity of Gröbner bases computation.

The results in this thesis show that the proposed strategies dramatically improve the performance of the XL algorithm and, moreover, introduce algorithms that outperform Magma's implementation of  $F_4$ , one of the currently most efficient algorithms, in terms of time and memory consumption in many cases. Moreover, an adapted version of MutantXL is used to attack the MQQ cryptosystem faster and uses less memory than attacks using  $F_4$  [39].

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Preliminaries</b>	<b>5</b>
2.1	Basic notations . . . . .	5
2.1.1	Polynomial ring . . . . .	5
2.1.2	Ideals . . . . .	5
2.1.3	Term orderings . . . . .	6
2.2	The MP problem . . . . .	7
2.2.1	Multivariate cryptosystems . . . . .	7
2.2.2	HFE cryptosystems . . . . .	8
2.3	Techniques for solving multivariate systems . . . . .	9
2.3.1	Linearization . . . . .	9
2.3.2	EXtended Linearization (XL) . . . . .	10
2.3.3	Gröbner bases . . . . .	12
<b>3</b>	<b>Mutant Strategy</b>	<b>15</b>
3.1	The basic idea . . . . .	15
3.2	The MutantXL algorithm . . . . .	18
3.3	Improved mutant strategy . . . . .	22
3.4	Complexity bounds . . . . .	23
3.5	Implementation and experimental results . . . . .	30
<b>4</b>	<b>Partial Enlargement Strategy</b>	<b>34</b>
4.1	The basic idea . . . . .	34
4.2	The MXL <sub>2</sub> algorithm . . . . .	39
4.3	Implementation and experimental results . . . . .	42
<b>5</b>	<b>Algebraic cryptanalysis of MQQ</b>	<b>49</b>
5.1	MQQ cryptosystems . . . . .	49
5.2	MQQ cryptanalysis . . . . .	51
<b>6</b>	<b>Computing Gröbner Basis</b>	<b>57</b>
6.1	The MXL <sub>3</sub> criterion . . . . .	57
6.2	The MXL <sub>3</sub> algorithm . . . . .	59
6.3	Implementation and experimental results . . . . .	64
6.4	Towards cryptanalysis of HFE challenge 2 . . . . .	69

*Contents*

<b>7 Flexible Partial Enlargement</b>	<b>76</b>
7.1 The basic idea . . . . .	76
7.2 The MGB algorithm . . . . .	78
7.3 MGB in action . . . . .	81
7.4 Experimental results . . . . .	82
7.5 Complexity bounds . . . . .	85
<b>8 Conclusion and Future Work</b>	<b>91</b>
<b>Bibliography</b>	<b>93</b>

# List of Algorithms

2.1	XL . . . . .	11
3.1	StepwiseXL . . . . .	18
3.2	$XL_{Mutant}$ . . . . .	19
3.3	MutantXL . . . . .	21
3.4	ImprovedMutantXL . . . . .	24
4.1	$MXL_2$ . . . . .	41
6.1	$MXL_3$ . . . . .	63
7.1	MGB . . . . .	79
7.2	$Echelonize(P, M, ED)$ . . . . .	79
7.3	$Gr\ddot{o}bner(P, M, S, D, ED)$ . . . . .	79
7.4	$Enlarge(P, M, S, x, D, ED, newExtend)$ . . . . .	80

# List of Figures

3.1	Compare the investigated parts of the set $H_d$ for XL, $XL_{Mutant}$ , and MutantXL. . . . .	27
3.2	Compare upper bounds of $I$ using (3.8) and (3.9) to the exact value of $I$ for HFE-96 systems solved by MutantXL at degree 4. . . . .	29
3.3	Comparison between XL and MutantXL maximal degrees for random systems of size $n$ . . . . .	30
3.4	Comparison between the maximum number of terms generated by MutantXL and $F_4$ for random systems of size $n$ . . . . .	33
4.1	Variable partitions of polynomials generated by XL for a random system of size $n = 26$ . . . . .	36
4.2	The partitions of $H_{d-1}^+$ . . . . .	40
4.3	Compare the performance of mxl2 . . . . .	48
5.1	Comparison between MutantXL and $F_4$ for MQQ . . . . .	54
5.2	Relation between $n$ and the number of mutants obtained . . . . .	55
6.1	Generating degree 4 polynomials of a random system of size $n = 14$ using $MXL_2$ . . . . .	60
6.2	Time complexity in seconds . . . . .	74
6.3	Relation between O-Notation of maximum matrix and system size . . . . .	74
6.4	Relation between the memory usage of solving HFE challenge 2 systems ( $y$ -axis) and the number of variables $n'$ ( $x$ -axis) after guessing $g$ variables, while the number of equation is 128. . . . .	75
7.1	Behavior of the algorithm for a sequence of 24 degree 2 equations in 24 variables. Horizontal stripes represent variable-partitions, darker ones are full. Vertical stripes represent terms that do not appear in the given polynomials. . . . .	81
7.2	Comparison between $MXL_3$ and $F_4$ for HFE(d,25) . . . . .	84
7.3	MGB different degree level number of variables. . . . .	88
7.4	Compares $n$ to $n_d$ for a random system of size $n = 77$ . . . . .	89
7.5	Compares $n$ to $n_d$ for a random system of sizes $n = 3 \dots 77$ at the degree of regularity. . . . .	89
7.6	Experimental results compared with the number of terms at the degree of regularity. . . . .	90

# List of Tables

3.1	Performance of $XL_{Mutant}$ versus XL . . . . .	20
3.2	Performance of MutantXL versus XL . . . . .	20
3.3	Reductions to zero of the MutantXL algorithm . . . . .	22
3.4	Reductions to zero of the improved MutantXL algorithm . . . . .	23
3.5	New elements generated from $H_d$ by XL, $XL_{Mutant}$ , and MutantXL	27
3.6	Performance of MutantXL versus Magma's $F_4$ on random system . .	31
3.7	Performance of MutantXL versus Magma's $F_4$ on HFE-288 system .	32
4.1	Experiments of MutantXL and XL on random systems. . . . .	35
4.2	Experiments of XLP on some random systems. . . . .	37
4.3	Compare the effectiveness of mutant and partial enlargement strategies to improve XL and the impacts of combining both strategies in $MXL_2$ . . . . .	44
4.4	Compare the steps of the $MXL_2$ improvements, the mutant strategy over the partial enlargement strategy. . . . .	45
4.5	Compare the steps of the $MXL_2$ improvements, the partial enlargement strategy over the mutant strategy. . . . .	46
4.6	Compare the efficiency of our $MXL_2$ implementation to the Magma's implementation of $F_4$ on random systems. . . . .	46
4.7	Compare the efficiency of our $MXL_2$ implementation to the Magma's implementation of $F_4$ on HFE-288 systems. . . . .	47
5.1	Definition of the the nonlinear mapping $P'$ . . . . .	51
5.2	Performance of MutantXL versus $F_4$ . . . . .	53
5.3	Performance of MutantXL versus $F_4$ . . . . .	53
5.4	Steps of solving MQQ-200 by MutantXL . . . . .	53
5.5	Steps of solving MQQ-200 by Magma- $F_4$ . . . . .	54
6.1	Performance of $MXL_3$ versus $F_4$ for dense random system . . . . .	65
6.2	Performance of $MXL_3$ versus $F_4$ for HFE(288,n) systems . . . . .	65
6.3	Results for the system Random-30 by $MXL_3$ . . . . .	66
6.4	Results for the system Random-30 by Magma . . . . .	66
6.5	Performance of $MXL_3$ versus $MXL_2$ for dense random system . . . .	67
6.6	Performance of $MXL_3$ versus $F_4$ for HFE(96,n) systems . . . . .	68
6.7	Performance of MutantXL versus $F_4$ for HFE(d,25) systems . . . . .	68
6.8	results of $MXL_3$ for HFE Challenge 2 system ( $n = 144$ , $m = 128$ and $h = 16$ ) . . . . .	70

*List of Tables*

6.9	results of $MXL_3$ for HFE Challenge 2 system ( $n = 108, m = 96$ and $h = 12$ ) . . . . .	71
6.10	results of $MXL_3$ for HFE Challenge 2 system ( $n = 72, m = 64$ and $h = 8$ ) . . . . .	71
6.11	results of $MXL_3$ for HFE Challenge 2 system ( $n = 36, m = 32$ and $h = 4$ ) . . . . .	72
6.12	results of $F_4$ for HFE Challenge 2 system ( $n = 144, m = 128$ and $h = 16$ ) . . . . .	72
6.13	Results for HFE2 system ( $n = 72, m = 64, h = 8, g = 8$ ) by $MXL_3$ .	73
7.1	Experiments for dense random systems . . . . .	83
7.2	Experiments for HFE(288,n) systems . . . . .	83
7.3	Results for the system Random-32 with the MGB algorithm . . . . .	84
7.4	Partitions generated by XL at degree 5 for Random systems . . . . .	86
7.5	Compare Partitions generated by XL and MGB at degree 5 for Random systems . . . . .	87

# 1 Introduction

Solving systems of multivariate polynomial equations over finite fields is one of the important research problems in cryptography and many other areas. For example, in cryptography, the intractability of solving this problem assesses the security of a type of public-key cryptosystems. The public key in this case is defined by a set of multivariate polynomials. These cryptosystems constitute the so-called Multivariate Cryptography [22, 48], which is one of the candidates for post-quantum cryptography [7]; cryptographic primitives that could resist potentially quantum computer attacks.

In addition, the security of many symmetric cryptosystems, like block and stream ciphers, is connected to the problem of solving a large system of multivariate polynomial equations. This was firstly noticed by Claude Shannon in [46]. Based on this observation, an attacker can model a symmetric cryptosystem by multivariate polynomial equations. These equations are constructed such that, solving them reveals the secret information of the corresponding symmetric cryptosystem. This type of attacks is known as “algebraic cryptanalysis”. The designers of stream and block ciphers should take into account such attacks.

There are several algorithms which solve multivariate polynomial systems over finite fields. This work focuses mainly on algorithms which use the multivariate polynomials to “enlarge” the system by generating additional equations having the same set of solutions. The enlarged system could be thought of as a system of linear equations which has a larger set of variables. Using linear algebra techniques, such as Gaussian elimination, on the matrix representation of this linear system, a solution can be obtained. The prominent algorithms which use this strategy are XL,  $F_4$ , and  $F_5$  introduced in [13], [24] and [25] respectively. The XL algorithm follows a simple strategy; by fixing a degree bound  $D$ , it enlarges the polynomial system by generating as many equations of maximal degree  $D$  as possible. On the other hand,  $F_4$  and  $F_5$  algorithms use relatively complex structures to solve the system by computing a Gröbner basis for the ideal generated by the input polynomials. However, they are generally able to solve larger systems in terms of number of equations and number of variables compared to XL.

Although these algorithms have been used for several algebraic attacks [32, 14, 12, 26], theoretical complexity estimates have shown that this kind of attacks are infeasible for many realistic applications. This is due to the fact that, in many practical cases, the computations made by these algorithms lead to constructing a huge system of polynomial equations, and consequently a huge matrix, which requires a lot of time and memory resources.

A big challenge is to improve these algorithms in a way allowing them to use

## 1 Introduction

only the limited available memory and time resources for solving a multivariate polynomial system with as large number of equations and variables as possible.

One of the strategies to improve the efficiency of these algorithms is to find better linear algebra techniques. This mainly reduces the time consumption. On the other hand, strategies improving the enlargement step of the polynomial system, by reducing the matrix size, will affect both time and memory consumption.

This work proposes strategies to improve the enlargement step of the algorithms mentioned above. Afterwards, we apply these strategies to the XL algorithm and show that combining these strategies with XL makes it highly competitive to state-of-the-art algorithms which solve multivariate systems.

The contribution of this thesis can be divided into the following four main parts.

*Mutant Strategy.* In 2006, Jintai Ding pointed out in his notes [19] that during the linear algebra step, certain polynomials of degrees lower than expected appear. These polynomials are called mutants. The mutant strategy aims to distinguish mutants from the rest of polynomials and give them a priority in the process of solving the system. We modify the XL algorithm such that, instead of enlarging the system blindly and increasing the degree, we first use the mutants, if any, at the lowest possible degree to enlarge the system. We call this new algorithm MutantXL [20]. Our experimental results show that MutantXL can indeed outperform XL and can solve multivariate systems at a lower degree than the usual XL algorithm. Also, we present further improvements to MutantXL and show that our implementation of this improved version is efficient in terms of time and memory consumption compared to the widely used Magma's implementation of  $F_4$ .

*Partial Enlargement Strategy.* In many experiments with XL and MutantXL algorithms, we observed that MutantXL can not produce any mutants and therefore, solves as XL. We introduce a new enlargement method that deals with this problem. We modify XL in such a way that it explores the set of polynomials of maximal degree  $D$ , one subset at a time, without being forced to store the whole set at once. This results in solving systems with fewer number of enlarged polynomials than XL. We also introduce a new algorithm, called  $MXL_2$  [40], which combines both mutant and partial enlargement strategies to improve over XL. Our experiments show that  $MXL_2$  not only outperforms MutantXL, but also in most of the studied cases, it outperforms  $F_4$  in terms of memory consumption.

A new strategy for *Computing Gröbner basis*. The main drawback of  $MXL_2$ , as well as XL and MutantXL algorithms, is that it can solve only systems having a unique solution. The standard method for solving polynomial systems which have finite number of solutions is to compute a Gröbner basis for the ideal generated by the input polynomials. In order to solve systems with a finite number of solutions, we modify the termination criteria of  $MXL_2$  such that it terminates once a Gröbner basis is found. This is done by using a new sufficient condition for a set of poly-

nomials to be a Gröbner basis. The new algorithm, called  $\text{MXL}_3$  [37], introduces further improvements to the enlargement step of  $\text{MXL}_2$ . Our experimental results show that in both classical cryptographic challenges as well as randomly generated polynomial systems,  $\text{MXL}_3$  performs better than the Magma's implementation of  $\mathbb{F}_4$  in terms of memory and time consumption.

*Flexible Partial Enlargement Strategy.* We introduce an efficient algorithm, called Mutant-based Gröbner Basis algorithm, MGB. We modify  $\text{MXL}_3$  using a more flexible partial enlargement strategy to omit some parts of the enlarged system and still satisfy the  $\text{MXL}_3$  criterion [11]. This new strategy provides an important improvement, thus allowing us to save even more memory and time consumption. Our experimental results show that the MGB algorithm outperforms both  $\text{MXL}_3$  and  $\mathbb{F}_4$  in all studied cases. Finally, we present a preliminary study for the complexity of solving multivariate polynomial systems over the finite field with two elements ( $\mathbb{F}_2$ ). This study suggests a new upper bound for the complexity of computing Gröbner bases which may lead us to think of a new paradigms for estimating the complexity of Gröbner bases computation.

## Author's Contribution

The work presented in this thesis is a result of an original research carried out by our research group (Darmstadt), in collaboration with professor Ding's group (Cincinnati). We can specify the exact contribution of the author as follows.

**MutantXL:** is a new efficient algorithm based on the XL algorithm and Ding's idea (*mutants*). The author took part in designing and implementing the MutantXL algorithm. Also, the author proposed a formula to determine the necessary number of mutants needed to successfully terminate the process and adapting the implementation.

**MXL2:** is an improved MutantXL algorithm that uses a new partial enlargement strategy. The author introduced the idea and the methodology of this partial enlargement technique, designed the  $\text{MXL}_2$  algorithm, and fully implemented it.

**MXL3:** is the first variant of the XL algorithm that computes Gröbner bases. The author proposed the idea of the algorithm, improved  $\text{MXL}_2$ , and fully implemented the algorithm. He also shared in justifying and proving a new checkable condition for computing Gröbner basis that is used as a termination condition for the algorithm.

**MGB:** is an improved  $\text{MXL}_3$  algorithm that uses a more flexible partial enlargement technique. The author shared in designing the MGB algorithm and he fully implemented it. The author also made a preliminary study of the complexity of solving multivariate systems over  $\mathbb{F}_2$  using the MGB algorithm.

## 1 Introduction

**Applications:** The author made an efficient algebraic attack on the MQQ public-key cryptosystem using an adapted version of MutantXL [39]. Finally, the author presented experimental results for solving some scaled versions of the HFE challenge 2 using MXL<sub>3</sub>. These results are a considerable step towards algebraic cryptanalysis of the full challenge [38].

## 2 Preliminaries

This chapter recalls definitions and basic notations that are needed to understand the rest of this thesis.

### 2.1 Basic notations

We give the fundamental definitions for the thesis: polynomial rings, ideals, and term orders.

#### 2.1.1 Polynomial ring

We define polynomials, the main object of this thesis. We are interested in polynomials defined over finite fields. Let  $\mathbb{F}_q$  be a finite field of size  $q$  and  $X := \{x_1, \dots, x_n\}$  be a set of variables. A term  $t$  in  $x_1, \dots, x_n$  is defined as a product of the form  $t = x_1^{\alpha_1} \cdot x_2^{\alpha_2} \dots \cdot x_n^{\alpha_n}$ , where  $\alpha_i \geq 0$ ,  $1 \leq i \leq n$ . Let  $\alpha = (\alpha_1, \dots, \alpha_n)$ , the total degree of  $t$  is equal to  $|\alpha| = \alpha_1 + \dots + \alpha_n$ , denoted by  $\deg(t) = \alpha$ . A polynomial  $p$  in  $X$  is a finite linear combination of terms. We write it in the form

$$p = \sum_{\alpha} a_{\alpha} x^{\alpha}, \quad a_{\alpha} \in \mathbb{F}_q,$$

where  $x^{\alpha} = x_1^{\alpha_1} \cdot x_2^{\alpha_2} \dots \cdot x_n^{\alpha_n}$ . The set of all such polynomials with coefficients in  $\mathbb{F}_q$  forms a commutative ring, w.r.t. polynomial addition and multiplication, denoted by  $\mathbb{F}_q[x_1, \dots, x_n]$  and called the polynomial ring over  $\mathbb{F}_q$ .

The set of terms in  $\mathbb{F}_q[x_1, \dots, x_n]$  is defined as:

$$T(\mathbb{F}_q[x_1, \dots, x_n]) = \{x_1^{\alpha_1} \cdot x_2^{\alpha_2} \dots \cdot x_n^{\alpha_n} \mid \alpha_i \geq 0, 1 \leq i \leq n\},$$

where  $1 \in T(\mathbb{F}_q[x_1, \dots, x_n])$  with  $\alpha_1 = 0, \dots, \alpha_n = 0$ .

Let  $p \in \mathbb{F}_q[x_1, \dots, x_n]$ ,  $p = \sum_{t \in T} a_t t$ , where  $a_t \in \mathbb{F}_q$  is the coefficient of  $t$  in  $p$ . The set of terms that forms the polynomial  $p$  is given by  $T(p) := \{t \in T \mid a_t \neq 0\}$ , the degree of  $p$  by  $\deg(p) := \max\{\deg(t) \mid t \in T(p)\}$ .

#### 2.1.2 Ideals

Polynomial ideals are one of the important ingredients in the thesis. A subset  $I$  of  $\mathbb{F}_q[x_1, \dots, x_n]$  is an ideal if for all  $f, g \in I$ ,  $h \in \mathbb{F}_q[x_1, \dots, x_n]$ ,  $f + g \in I$  and  $hf \in I$ . Let  $f_1, \dots, f_m$  be a subset of  $\mathbb{F}_q[x_1, \dots, x_n]$ . The ideal generated by  $f_1, \dots, f_m$  is

## 2 Preliminaries

defined as

$$I := \langle f_1, \dots, f_m \rangle := \{f \mid f = h_1 f_1 + h_2 f_2 + \dots + h_m f_m, h_i \in \mathbb{F}_q[x_1, \dots, x_n], 1 \leq i \leq m\}.$$

Let  $I = \langle f_1, \dots, f_m \rangle$  be an ideal, the variety of  $I$  is defined as follows

$$V(I) := \{(a_1, \dots, a_n) \in \mathbb{F}_q^n \mid f_i(a_1, \dots, a_n) = 0, 1 \leq i \leq m\}.$$

We call  $V(I)$ , the set of solutions of the equations  $f_i = 0, 1 \leq i \leq m$ . In practice, the attention often goes to equation systems that have a finite number of solutions. Assume  $I = \langle f_1, \dots, f_m \rangle$ , if the set of variety of  $I$  ( $V(I)$ ) is a finite set then  $I$  is called a zero-dimensional ideal. The variety of polynomials  $f_1, \dots, f_m$  depends on the ideal generated by them. If  $\langle f_1, \dots, f_m \rangle = \langle g_1, \dots, g_s \rangle$ , then we have  $V(f_1, \dots, f_m) = V(g_1, \dots, g_s)$ .

For an ideal  $I$  in  $\mathbb{F}_q[x_1, \dots, x_n]$ , the sub-ring

$$\mathbb{F}_q[x_1, \dots, x_n]/I \tag{2.1}$$

is called a quotient ring of  $\mathbb{F}_q[x_1, \dots, x_n]$ . In this thesis, we are interested in the following quotient ring  $R$ :

$$R = \mathbb{F}_q[x_1, \dots, x_n]/\langle x_1^q - x_1, \dots, x_n^q - x_n \rangle, \tag{2.2}$$

where  $x_i^q - x_i = 0, 1 \leq i \leq n$  are the so-called field equations. Assuming that  $t = x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdot \dots \cdot x_n^{\alpha_n} \in T(R)$ , then  $\alpha_i < q, 1 \leq i \leq n$  and  $T(R)$  is the set of all terms in  $R$ . A quotient ring  $R$  is a finite dimensional vector space and its dimension is equal to the cardinality of  $T(R)$ . For simplicity, we will call  $R$  a polynomial ring in  $X$  over the finite field  $\mathbb{F}_q$ . The Boolean polynomial ring  $B$  is the polynomial ring over  $\mathbb{F}_2$  that consists of only idempotent elements,

$$B = \mathbb{F}_2[x_1, \dots, x_n]/\langle x_1^2 - x_1, \dots, x_n^2 - x_n \rangle. \tag{2.3}$$

### 2.1.3 Term orderings

A polynomial  $p$  in the polynomial ring  $R$  is ordered with respect to a term ordering. For most of computational algorithms on a set of polynomials, a term ordering has to be declared together with the polynomial ring.

A term ordering on  $R$  is a total ordering  $<$  on  $T(R)$  such that:  $1 < t, \forall t \in T(R), t \neq 1$  and  $\forall s, t_1, t_2 \in T(R)$  with  $t_1 < t_2$  then  $st_1 < st_2$ . There are several term orderings used in the polynomial algebra. Let  $t_1, t_2 \in T(R), t_1 = x^\alpha, \alpha = (\alpha_1, \dots, \alpha_n)$  and  $t_2 = x^\beta, \beta = (\beta_1, \dots, \beta_n)$ . We can order  $t_1, t_2$  using one of the following methods:

**Lexicographical ordering (*lex*)**  $t_1 >_{lex} t_2$  if and only if the leftmost non zero entry of  $\alpha - \beta = (\alpha_1 - \beta_1, \dots, \alpha_n - \beta_n)$  is positive. For example, define  $t_1 = (x_1^2 x_2^3 x_3)$  and  $t_2 = (x_1^2 x_2^3 x_4^3)$ . Then  $t_1 >_{lex} t_2$  since  $\alpha - \beta = (0, 0, 1, -3)$  and the leftmost

entry is  $1 > 0$ .

**Graded lexicographical ordering (*glex*)**  $t_1 >_{glex} t_2$  if and only if  $\deg(t_1) > \deg(t_2)$  or  $\deg(t_1) = \deg(t_2)$  and  $t_1 >_{lex} t_2$ . For  $t_1, t_2$  defined above  $t_1 <_{glex} t_2$ , since  $\deg(t_1) < \deg(t_2)$ .

**Graded reverse lexicographical ordering (*grlex*)**  $t_1 >_{grlex} t_2$  if and only if  $\deg(t_1) > \deg(t_2)$  or  $\deg(t_1) = \deg(t_2)$  and the rightmost non zero entry of  $\alpha - \beta$  is negative. For example, let  $t_1 = (x_1^2 x_2^3 x_3)$ ,  $t_2 = (x_1^2 x_2 x_4^3)$ ,  $t_1 >_{grlex} t_2$  since  $\alpha - \beta = (0, 2, 1, -3)$  and the rightmost entry is -3.

Let  $p \in \mathbb{F}_q[x_1, \dots, x_n]$  and the terms in  $p$  is ordered by  $\leq$ . The leading term of  $p$  is defined by  $LT(p) := \max_{\leq} T(p)$ ,  $T(p)$  the set of terms of  $p$ . The leading coefficient of  $p$ , denoted by  $(LC(p))$ , is the coefficient of  $LT(p)$  and the leading monomial of  $p$ , denoted by  $LM(p)$ , is  $LM(p) = LC(p) LT(p)$ .

## 2.2 The MP problem

We call the problem of solving systems of non-linear simultaneous multivariate equations the *MP problem*. It is defined as follows. Let  $P(x_1, \dots, x_n) = \{p_1, \dots, p_m\}$  be a system of  $m$  multivariate polynomials in the polynomial ring  $R$  (defined as in (2.2) over a finite field  $\mathbb{F}_q$ ). The problem is to find at least one vector  $(v_1, \dots, v_n) \in \mathbb{F}_q^n$  such that

$$\begin{aligned} p_1(v_1, \dots, v_n) &= 0 \\ p_2(v_1, \dots, v_n) &= 0 \\ &\vdots \\ p_m(v_1, \dots, v_n) &= 0. \end{aligned}$$

The MP problem is NP-hard and hard on average even when the polynomials are quadratic and over  $\mathbb{F}_2$  [29, 28]. The worst case of solving such quadratic systems is when the number of equations ( $m$ ) is close to the number of variables ( $n$ ) and easier when the difference is significant. Moreover, the complexity of solving a structured MP system is often easier than a randomly generated MP system with the same degree and the same size.

### 2.2.1 Multivariate cryptosystems

Multivariate cryptosystems are public key cryptosystems that are based on the hardness of the MP-problem and is called MPKC. The first construction of a MPKC is due to Matsumoto and Imai [35]. This section reviews the structure of MPKCs and gives the HFE cryptosystem as an example.

The generic construction of multivariate cryptosystems works as follows. As any public key cryptosystem, an MPKC uses two keys, one is public and the other is

## 2 Preliminaries

private. The private key consists of a map  $\varphi$ , the basic part in the secret key, which transforms a vector  $(x_1, \dots, x_n) \in \mathbb{F}_q^n$  to a vector  $(y_1, \dots, y_m) \in \mathbb{F}_q^m$ . We can reconstruct  $\varphi$  to  $m$  multivariate equations  $q_1 \dots, q_m$ , where  $q_i : \mathbb{F}_q^n \rightarrow \mathbb{F}_q$ ,  $1 \leq i \leq m$ . Different multivariate schemes have different definitions of  $\varphi$ . Two affine transformations  $S$  and  $T$  that are usually generated randomly to hide the construction of  $\varphi$ .  $S$  transforms from  $\mathbb{F}_q^n$  to  $\mathbb{F}_q^n$  and  $T$  transforms from  $\mathbb{F}_q^m$  to  $\mathbb{F}_q^m$ . The transformations  $S, \varphi, T$  are invertible.

The public key is defined as

$$P = S \circ \varphi \circ T.$$

It can be defined by a set of multivariate polynomials  $p_1, \dots, p_m$ .

The plaintext space is  $\mathbb{F}_q^n$ . A plaintext  $x = (x_1, \dots, x_n) \in \mathbb{F}_q^n$  is encrypted by computing

$$\begin{aligned} p_1(x_1, \dots, x_n) &= c_1 \\ p_2(x_1, \dots, x_n) &= c_2 \\ &\vdots \\ p_m(x_1, \dots, x_n) &= c_m, \end{aligned}$$

and  $c = (c_1, \dots, c_m)$  is the corresponding ciphertext.

If the secret key is known, then we can decrypt  $c$  using the form

$$x = S^{-1} \varphi^{-1} T^{-1}(c),$$

where  $S^{-1}, \varphi^{-1}$ , and  $T^{-1}$  are easy to evaluate. So to break a multivariate scheme one has to decrypt  $c$  by solving the multivariate system  $P(x) = c$ . This means that, the security of MPKCs is strongly related to solving the MP problem.

### 2.2.2 HFE cryptosystems

The hidden field equation (HFE) cryptosystem is a good example of MPKCs. This cryptosystem was introduced by Patarin at EuroCrypt96 [43] after breaking the Matsumoto-Imai cryptosystem [42]. We use HFE systems for benchmarks in our experiments during this thesis.

In the case of HFE cryptosystem, the transformation  $\varphi$  is a univariate polynomial of degree  $d$  in a variable  $x$  over an extension field  $\mathbb{F}_{q^n}$ , where  $x = (x_1, \dots, x_n)$  and  $x_i \in \mathbb{F}_q$ ,  $1 \leq i \leq n$ . Usually HFE is defined over  $\mathbb{F}_{2^n}$ . The inverse  $\varphi^{-1}$  of  $\varphi$  is easily evaluated over  $\mathbb{F}_{q^n}$  by finding a solution for the equation  $\varphi(X) = Y$ . The map  $\varphi$  is chosen such that it can be expressed as a system of  $n$  multivariate quadratic polynomial equations over  $\mathbb{F}_q$ . In this case each coordinate of  $\varphi(x)$  is expressed by a polynomial in  $x_1, \dots, x_n$ . HFE hides its secret polynomial using two randomly chosen invertible affine transformations  $(S, T)$  from  $\mathbb{F}_q^n$  to  $\mathbb{F}_q^n$ . The public key is defined by a system of quadratic equations  $P = (p_1, \dots, p_n)$  over  $\mathbb{F}_q$ ,  $P = T \circ \varphi \circ S$ .

As any MPKC, the HFE security is based on solving a polynomial system  $P(x) = c$ , where  $x$  is an input plaintext and  $c$  is the output ciphertext. An HFE system has two parameters that affect the complexity of solving its system. The first parameter is the number of variables ( $n$ ) and the other is the degree of its secret polynomial ( $d$ ). The hardness of solving HFE systems is close to solving random systems when  $d$  is very big, say  $d > 512$ . However, the univariate degree  $d$  should be small enough to obtain an efficient HFE cryptosystem in practice. In the extended version of [43], Patarin introduced two HFE challenges with a prize US \$500 for attacking any of them. The first challenge (HFE challenge 1) with parameters  $n = 80, d = 96$  was broken by Faugère and Joux in [26]. The second challenge (HFE challenge 2) with parameters  $n = 144$  and  $d = 4352$ , where 16 of the 144 equations are not given public is still not broken.

## 2.3 Techniques for solving multivariate systems

### 2.3.1 Linearization

The linearization method is commonly used in many algorithms for solving multivariate nonlinear polynomial systems. It is based on using linear algebra techniques in the process of solving multivariate systems. The idea of this method is described as follows.

Let  $P = \{p_1, \dots, p_m\}$  be a finite set of polynomials in  $R$  and its maximal degree is  $d$ . Let  $T_d = (t_1, \dots, t_r)$  be an ordered set of all terms in  $R$  such that  $\deg(t) \leq d$  for all  $t \in T_d$  and  $t_1 > t_2 > \dots > t_r$ . We can write a polynomial  $p_i$  in  $P$  as

$$p_i = \sum_{j=1}^r a_{i,j} t_j, \quad 1 \leq i \leq m.$$

By considering each term in  $T_d$  as a new variable, the polynomial equations ( $p_i = 0, 1 \leq i \leq m$ ) represent a system of linear equations in the new variables (terms). One can write the set of polynomials  $P$  in the following coefficient matrix:

$$\mathcal{M}_d = \begin{matrix} & t_1 & t_2 & t_3 & \dots & t_r \\ \begin{matrix} p_1 \\ p_2 \\ p_3 \\ \vdots \\ p_m \end{matrix} & \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,r} \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,r} \\ a_{3,1} & a_{3,2} & a_{3,3} & \dots & a_{3,r} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \dots & a_{m,r} \end{pmatrix} \end{matrix}$$

In this case, each column represents a term and each row represents a polynomial. Each entry  $\mathcal{M}_d[i, j]$  is the coefficient of the term  $t_j$  in the polynomial  $p_i$ . The matrix  $\mathcal{M}_d$  is considered a bridge between multivariate polynomial systems and

## 2 Preliminaries

linear algebra.

We apply now a linear algebra method (say Gaussian elimination) to construct the eliminated linear system  $(\widetilde{\mathcal{M}}_d)$ ,

$$\widetilde{\mathcal{M}}_d = \begin{pmatrix} \times & \times & \dots & \times & \times & \times & \dots & \times \\ 0 & \times & \dots & \times & \times & \times & \dots & \times \\ 0 & 0 & \dots & \times & \times & \times & \dots & \times \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \times & \times & \dots & \times \\ 0 & 0 & \dots & 0 & 0 & \times & \dots & \times \end{pmatrix}$$

where the matrix  $\widetilde{\mathcal{M}}_d$  is the row echelon form of  $\mathcal{M}_d$ . By reversing the above operation, the matrix  $\widetilde{\mathcal{M}}_d$  can be interpreted back to a polynomial system  $\widetilde{P}$ . The resulting system  $\widetilde{P}$  is called the row echelon form of  $P$  and is defined as follows:

Let  $P = \{p_1, \dots, p_m\}$ , a finite subset  $\widetilde{P}$  of  $R$  is a *row echelon form* of  $P$  w.r.t.  $\leq$  (term order) if  $\text{span}(\widetilde{P}) = \text{span}(P)$  and elements of  $\widetilde{P}$  have pairwise different leading terms, where

$$\text{span}(P) = \{f \mid f = b_1p_1 + b_2p_2 + \dots + b_m p_m, b_i \in \mathbb{F}, 1 \leq i \leq m\}.$$

Using only linearization technique to solve the polynomial system ( $P = 0$ ) is based on the rank of  $\widetilde{P}$ . In other words, to solve the system the rank should be equal or at least very close to the number of terms in  $T_d$ . However, we can use an exhaustive search on the solutions space when the dimension of the kernel of  $\widetilde{P}$  is not too large.

### 2.3.2 EXtended Linearization (XL)

Let  $P = (p_1, \dots, p_m)$  be a set of linearly independent polynomials in  $R$ , as we explained above, we can solve the system  $P = 0$  using the linearization technique only when  $m$  is approximately equal to the number of terms that appear in the system. However, in most cases, polynomial systems contain only few equations compared to the number of their terms. The re-linearization [32] is an algorithm for solving multivariate polynomial equations. It generates extra non-linear equations from the ideal  $\langle P \rangle$  and includes them to the system. After that it uses the linearization technique. The re-linearization algorithm could solve many systems that are not be solvable by the linearization method. However, its complexity is not understood and the success rate is not clear.

The XL algorithm was proposed in [13] as a simple and powerful algorithm for solving polynomial systems. XL is designed to solve over-determined systems that have only one solution. The main idea of XL is to generate polynomials of degree  $\leq d$  in the ideal as many as the number of terms in  $T_d$ . We explain this idea as follows.

---

**Algorithm 2.1 XL**

---

**Input:**  $P$  is a quadratic finite polynomials of  $R$ ,  $d \geq 2$

- 1:  $U \rightarrow \emptyset$  // the set of univariate polynomials
- 2:  $S \rightarrow \emptyset$  // the set of solved variables and values
- 3: **repeat**
- 4:    $H_d \rightarrow \{t \cdot p \mid t \text{ is a term in } R, p \in P, \deg(tp) \leq d\}$
- 5:    $\tilde{H}_d \rightarrow \text{Echelonize}(H_d)$
- 6:    $U \rightarrow \{p \mid p \in \tilde{H} \text{ and } p \text{ is univariate } \}$
- 7:   **if**  $U \neq \emptyset$  **then**
- 8:      $P \leftarrow \text{Substitute}(P, U)$   
       //  $\forall u(x) \in U$ , find  $a \in \mathbb{F}$ ,  $u(a) = 0$ , replace  $x$  with its value  $a$  in  $P$
- 9:      $S \rightarrow S \cup \text{Solve}(U)$  //  $\forall u(x) \in U$ , insert the pair  $(x, a)$  into  $S$ ,  $u(a) = 0$
- 10:     $U \rightarrow \emptyset$
- 11:   **else**
- 12:      $d \rightarrow d + 1$
- 13:   **end if**
- 14: **until**  $P = \emptyset$
- 15: **return**  $S$

---

Let  $P = (p_1, \dots, p_m)$  be quadratic polynomials in  $R$  (2.2). Assuming a degree bound  $d$ , the XL algorithm (Alg. 2.1) is based on extending  $P$  by multiplying each polynomial in  $P$  with all the possible terms in  $T(R)$  such that the resulting set of polynomials ( $H_d$ ) has degree less than or equal to  $d$ . We define the set  $H_d$  of degree  $\leq d$  polynomials as follows

$$H_d := \{t \cdot p \mid t \in T(R), p \in P, \deg(tp) \leq d\} \quad (2.4)$$

Then XL applies the linearization technique as explained above and computes the row echelon form  $\tilde{H}_d$ . XL uses a term ordering  $\leq$  such that terms of degree one are eliminated last. In case of the number of linearly independent polynomials  $I = |\tilde{H}_d|$  is equal to  $|T_d| - 1$ , then XL terminates and returns the solution. Otherwise, XL simplifies the original system  $P$  with solved variables (if any) and repeats the process. As an illustration, we consider the equations

$$\begin{aligned} p_1 &= x_1x_2 + x_1 + 1 \\ p_2 &= x_1x_2 + x_2 \end{aligned}$$

in the Boolean ring  $B$  (2.3) and consider the degree bound  $d = 3$ . XL generates the

## 2 Preliminaries

set  $H_3$  by multiplying each polynomial with  $x_1$  and  $x_2$ , then we have

$$\begin{aligned} p_1 &= x_1x_2 + x_1 + 1 \\ p_2 &= x_1x_2 + x_2 \\ p_3 &= x_1x_2 \\ p_4 &= x_2 \end{aligned}$$

where the two field equations  $x_1^2 + x_1$  and  $x_2^2 + x_2$  are used. Then XL uses Gaussian elimination to compute  $\tilde{H}_3$ ,

$$\begin{aligned} \tilde{p}_1 &= x_1x_2 + x_1 + 1 \\ \tilde{p}_2 &= x_1 + x_2 + 1 \\ \tilde{p}_3 &= x_2 \\ \tilde{p}_4 &= 0 \end{aligned}$$

which leads to solving the system.

The authors of [13] presented a preliminary complexity analysis of XL that hoped to be asymptotically subexponential over small fields. However, in [49] and [18], the complexity of XL has been carefully studied that made this hope unlikely. They explained that the complexity of XL is exponential in the degree bound  $d$  such that XL succeeds. Since the number of terms of degree  $\leq d$  in  $R$  is equal to  $\binom{n+d}{d}$ .

A number of variants of XL have been introduced that exploit certain properties of polynomial systems. A first variant was presented by the XL authors in [13], FXL. The ‘‘F’’ stands for ‘‘Fix’’. We assign random values to some variables and simplify the system before we pass it to XL. FXL aims to decrease the degree bound  $d$  needed for XL to solve. The XSL algorithm [14] is a variant of XL that uses the sparsity or the special structure of the polynomial system. XSL extends the system by multiplying the polynomial system with some selected terms instead of multiplying them by all. However, the XSL authors did not explain a strategy to select terms that used in the multiplication.  $XL_2$  was presented in [16] as a variant of XL over  $\mathbb{F}_2$ . The idea of  $XL_2$  is to generate at least one extra equation under certain condition. By including the new equations with the initial equations, we may obtain another new equations. This will be repeated as many time as possible. This method was reformulated in [50]. Other variants of XL can be found in [16, 50, 12, 51, 36].

### 2.3.3 Gröbner bases

The standard way to represent the polynomial ideals is to compute a Gröbner basis of it. Gröbner bases (standard bases) were introduced by Buchberger in [8]. We give the necessary definitions of Gröbner bases for this thesis and refer to [17, 6] for more details. Let  $P = \{p_1 \dots, p_m\}$  be a subset of  $\mathbb{F}[x_1, \dots, x_n]$ , we denote by  $I = \langle P \rangle$  the ideal generated by  $P$ , and by  $LT(P)$  the set of leading terms of elements in  $P$ . A finite subset  $G$  of an ideal  $I$  of the polynomial ring  $R$  is called a Gröbner basis for

## 2 Preliminaries

$I$  (w.r.t the term order  $\leq$ ) if

$$\langle \text{LT}(I) \rangle = \langle \text{LT}(G) \rangle.$$

In other words,  $G$  is a Gröbner basis of the ideal  $I$  if for any polynomial  $f$  in the ideal  $I$ , we find a polynomial  $g$  in the set  $G$  such that  $\text{LT}(f)$  is divisible by  $\text{LT}(g)$ . It was proved in [17] that, every ideal  $0 \neq I \subset \mathbb{F}[x_1, \dots, x_n]$  has a Gröbner basis. Moreover, each ideal  $I \subset \mathbb{F}[x_1, \dots, x_n]$  has a uniquely defined Gröbner basis called the reduced Gröbner basis. A finite subset  $G \subset I$  is the reduced Gröbner basis of  $I$  if for all  $g \in G$ , no term of  $g$  belongs to  $\langle \text{LT}(G \setminus \{g\}) \rangle$ .

One of the most useful applications of Gröbner bases is to compute the variety of the ideal. This leads to solving the polynomial system induced by the ideal. Since we are interested on solving polynomial systems over the polynomial ring  $R$  (2.2), we add the field equations to the set of initial polynomials. So, we compute the Gröbner basis of  $\langle p_1 \dots, p_m, x_1^q - x_1, \dots, x_n^q - x_n \rangle$ .

### The Buchberger's algorithm

The Buchberger's algorithm [8] was the first algorithm for computing Gröbner bases. It is based on the following fact: Let  $G = \{g_1, \dots, g_m\}$  be a set of  $m$  polynomials in  $\mathbb{F}[x_1, \dots, x_n]$  and  $f \in I = \langle G \rangle$ . By the definition of Gröbner basis, if there exists  $f$  such that  $\text{LT}(f) \notin \langle \text{LT}(G) \rangle$  then  $G$  is not a Gröbner basis. So we should include all such  $f$  to the set  $G$  until  $G$  satisfies the Gröbner basis definition. More precisely, the Buchberger algorithm computes the so-called s-polynomials  $S(g_i, g_j)$  for every pair  $g_i, g_j$ . Let  $t = \text{LCM}(\text{LT}(g_i), \text{LT}(g_j))$ , LCM stands for the least common multiple, then

$$S(g_i, g_j) = \frac{t}{\text{LT}(g_i)} \cdot g_i - \frac{t}{\text{LT}(g_j)} \cdot g_j.$$

Then the algorithm computes the remainders of the polynomials  $S(g_i, g_j)$  by  $G$  and includes the non-zero remainders to  $G$ . It repeats that until all remainders are zero.

The complexity of the Buchberger algorithm is based on the degree of the polynomials appearing during the computation and it is known to run doubly exponential time in the worst case [44].

### The $F_4$ algorithm

Following the idea of Lazard [34], the  $F_4$  algorithm was introduced by Faugère in [24].  $F_4$  uses linear algebra techniques similar to explained above and Buchberger's s-polynomial techniques to compute Gröbner bases. It simultaneously reduces a large number of S-polynomial pairs in one step, by constructing the coefficient matrix of the products  $(t_i p_i)$  obtained from the selected pairs (critical pairs). Classically,  $F_4$  selects all pairs  $(p_i, p_j)$  such the the degree of  $\text{LCM}(p_i, p_j)$  is minimal. By computing the reduced row echelon form of the constructed matrix, we get the reduced s-polynomials of the considered pairs. Combining the strategy of  $F_4$ , as well as the XL algorithm, with any efficient implementation of a linear algebra step,

leads to a good performance. In this thesis, we use Magma's implementation of  $F_4$  in our  $F_4$  experiments.

**The  $F_5$  algorithm**

One of the main drawbacks of  $F_4$  and Buchberger's algorithms is that many of computed pairs are linearly dependent and waste a long time to be reduced to zero. Faugère introduced the  $F_5$  algorithm in [25] that can be used to avoid the redundant computations occurred during the linear algebra step. The  $F_5$  criterion [25] tracks the construction of all new polynomials and detects all rows in the coefficient matrix that coming from the relations  $p_i p_j = p_j p_i$  (trivial syzygies [41]). The matrix versions of  $F_5$  [5] and [1] construct incrementally in the degree, the constructed matrix of  $F_5$  is full rank for regular systems and semi-regular systems (random systems) as the maximal degree is smaller than the so-called degree of regularity ( $d_{reg}$ ) [4].

$F_5$  is primary intended for homogeneous regular sequences, all terms in the polynomial have the same degree [4]. There is no detailed description of how to adapt it for non-homogeneous sequences and moreover, no analysis exist on the behavior for such sequences. In the non-homogeneous case the  $F_5$  criteria can discover trivial syzygies of the leading forms but it says nothing about what to do with the non-trivial ones which in turn yield to useful polynomials as we will see in the next chapter. In addition, implementing  $F_5$  in a way that makes it works as claimed by Faugère is still an open problem. Therefore, in the absence of such algorithm one relies on Buchberger's or the  $F_4$  algorithm to compute a Gröbner Basis once these non-trivial syzygies appear.

## 3 Mutant Strategy

In this chapter, we present the first contribution of our research, called mutant strategy, which was first discovered by Ding at the end of 2006 [19]. It can be used to improve any algorithm for solving systems of multivariate polynomial equations that uses linear algebra (like XL,  $F_4$ ,  $F_5$ , ...). In order to understand how this strategy works, we split this chapter into five sections. Section 3.1 defines the concept of mutants and explores the potential of their application to improve polynomial system solving algorithms. Section 3.2 explains the steps of using the mutant strategy to improve XL and describes the MutantXL algorithm. Further improvements for MutantXL are shown in Section 3.3. Section 3.4 deals with the complexity of solving polynomial systems using MutantXL. Finally, Section 3.5 is dedicated to explain our implementation of the algorithms described in this chapter and to compare our MutantXL implementation to the widely used Magma's implementation of the  $F_4$  algorithm.

### 3.1 The basic idea

One of the main ideas that we explore in this thesis is the important role of mutant polynomials in solving systems of multivariate polynomial equations. To explain the idea behind mutant polynomials, we consider the ring

$$R = \mathbb{F}_q[x_1, \dots, x_n] / \langle x_1^q - x_1, \dots, x_n^q - x_n \rangle$$

of polynomials over  $R$  in the  $n$  variables  $x_1, \dots, x_n$  over a finite field  $\mathbb{F}_q$ . Here  $x_i^q - x_i = 0$ ,  $1 \leq i \leq n$  are field polynomials. In  $R$ , each element is uniquely expressed as a polynomial where each  $x_i$  has a power less than  $q$ .

Let  $P$  be a finite set of polynomials in  $R$ . Algorithms such as XL,  $F_4$ , and  $F_5$  for solving the system

$$p(x) = 0, \quad p \in P, \tag{3.1}$$

where

$$x = (x_1, \dots, x_n),$$

use the strategy that we explained in Chapter 2. They find additional polynomials of not so large degree in the ideal generated by the elements of  $P$  by multiplying them by monomials. Then they linearize the system (3.1) by replacing the monomials with new variables and apply, for example, Gaussian elimination.

Let the elements of the ideal be sorted by some degree ordering (graded or graded reverse lexicographic). In many experiments with those algorithms, we obtained

### 3 Mutant Strategy

polynomials of degree lower than expected after echelonizing (computing the row echelon form of) the enlarged system.

To be more precise, assume that the highest degree of the enlarged system is  $d$ . As explained in the previous chapter, we define and construct the coefficient matrix of the resulting degree  $d$  polynomials  $\mathcal{M}_d$  as

$$\mathcal{M}_d = \begin{pmatrix} \times & \times & \dots & \times & \times & \times & \dots & \times \\ \times & \times & \dots & \times & \times & \times & \dots & \times \\ \times & \times & \dots & \times & \times & \times & \dots & \times \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \times & \times & \dots & \times & \times & \times & \dots & \times \\ \times & \times & \dots & \times & \times & \times & \dots & \times \end{pmatrix}$$

$\underbrace{\hspace{15em}}$   
*degree  $d$*

$\underbrace{\hspace{15em}}$   
*degree  $< d$*

Let the echelonization matrix  $\widetilde{\mathcal{M}}_d$  of  $\mathcal{M}_d$  have some rows such that all degree  $d$  terms were canceled during the elimination process as follows.

$$\widetilde{\mathcal{M}}_d = \begin{pmatrix} \times & \times & \dots & \times & \times & \times & \dots & \times \\ 0 & \times & \dots & \times & \times & \times & \dots & \times \\ 0 & 0 & \dots & \times & \times & \times & \dots & \times \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \times & \times & \dots & \times \\ 0 & 0 & \dots & 0 & 0 & \times & \dots & \times \end{pmatrix}$$

$\underbrace{\hspace{15em}}$   
*degree  $d$*

$\underbrace{\hspace{15em}}$   
*degree  $< d$*

In this case, the last two rows of  $\widetilde{\mathcal{M}}_d$  represent new polynomials of degree less than  $d$  (the degree before echelonization). If those polynomials are univariate, then we know how to use them, namely for substitution as described in [42]. However, if they are not, they are just treated like any other polynomial in the algorithm. We call those polynomials *mutants*.

The mutant strategy is to distinguish mutants from the rest of polynomials and to give them a predominant role in the process of solving the system. Mathematically speaking, we define mutants as follows.

**Definition 3.1.** *Let  $I$  be the ideal generated by the finite set of polynomials  $P$ . An*

### 3 Mutant Strategy

element  $f$  in  $I$  can be represented as

$$f = \sum_{p \in P} f_p \cdot p \quad (3.2)$$

where  $f_p \in R$ . The maximum degree of  $(f_p \cdot p)$ ,  $p \in P$ , is the level of this representation. The level of  $f$  is the minimum level of all of its representations. The polynomial  $f$  is called mutant with respect to  $P$  if  $\deg(f)$  is less than its level.

The following toy example illustrates how we can generate mutants and how we use them to solve the system.

**Example 3.2.** Let  $P = \{p_1, p_2, p_3, p_4\}$  be a set of polynomials in the Boolean polynomial ring  $B$  as defined in (2.3) in the set of variables  $\{x_1, x_2, x_3, x_4\}$ :

$$\begin{aligned} p_1 &= x_1x_2 + x_2x_3 + x_2x_4 + x_3x_4 + x_1 + x_3 + 1 \\ p_2 &= x_1x_2 + x_1x_3 + x_1x_4 + x_3x_4 + x_2 + x_3 + 1 \\ p_3 &= x_1x_2 + x_1x_3 + x_2x_3 + x_3x_4 + x_1 + x_4 + 1 \\ p_4 &= x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + 1 \end{aligned}$$

We linearize  $p_1, p_2, p_3, p_4$  by representing each term as a new variable and then compute the row echelon form  $\text{Echelonize}(P)$ . We obtain the following equations:

$$\begin{aligned} \tilde{p}_1 &= x_1x_2 + x_2x_3 + x_2x_4 + x_3x_4 + x_1 + x_3 + 1 \\ \tilde{p}_2 &= x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + x_1 + x_2 \\ \tilde{p}_3 &= x_1x_4 + x_2x_3 + x_1 + x_2 + x_3 + x_4 \\ \tilde{p}_4 &= x_1 + x_2 + 1 \end{aligned}$$

The degree of  $\tilde{p}_4$  is 1, which is less than  $D = 2$ . So  $\tilde{p}_4$  is mutant.

We give another important definition for mutants that explains how mutants can be distinguished during the computations from the rest of polynomials.

**Definition 3.3.** Without loss of generality, let  $P$  be a finite set of degree 2 polynomials. For  $d \geq 2$  consider the set

$$H_d := \{t \cdot p \mid t \text{ is a term}, p \in P, \deg(tp) \leq d\}$$

a polynomial  $f$  is called a mutant, if  $\text{LT}(f) \in \text{LT}(\text{span}(H_d)) \setminus \text{LT}(\text{span}(H_{d-1}))$  and its degree is less than  $d$ .

### 3.2 The MutantXL algorithm

We describe the MutantXL algorithm and explain the steps of using mutants to improve the XL algorithm [13]. Let us consider the system (3.1) of multivariate polynomial equations. For simplicity, we assume that the system (3.1) is quadratic and has a unique solution. The set of polynomials generated by XL at degree  $d \geq 2$  is defined as in (2.4). We suppose the version of XL that enlarges  $P$  one degree incrementally as described in Algorithm 3.1. We call it stepwiseXL. It is a matrix-based algorithm, so it uses a coefficient matrix to represent the set of polynomials  $H_d$  as explained in Chapter 2 and includes them to  $H$ . It looks for univariate polynomials produced after computing the row echelon form of  $H$  (Alg. 3.1, step 5). In case of no univariates has been obtained, it generates the elements of  $H_{d+1}$  and includes them to  $H$  (Alg. 3.1, step 11).

---

#### Algorithm 3.1 StepwiseXL

---

**Input:**  $P(x_1, \dots, x_n)$  is a quadratic finite set of polynomials in  $R$ .

**Output:**  $U$  a set  $n$  univariate polynomials in  $x_1, \dots, x_n$ .

```

1:  $d \leftarrow 2$  // The highest degree in  $H$ 
2:  $H \leftarrow P$  // The set of generated polynomials
3:  $U \leftarrow \emptyset$  // the set of univariate polynomials
4: while  $|U| < n$  do
5:    $\tilde{H} \leftarrow \text{Echelonize}(H)$  // Compute a row echelon form of  $H$ 
6:    $U \leftarrow \{p \in \tilde{H} \mid \text{and } p \text{ is univariate}\}$ 
7:   if  $U \neq \emptyset$  then
8:      $H \leftarrow \text{Substitute}(\tilde{H}, U)$ 
       //  $\forall u(x) \in U$ , find  $a \in \mathbb{F}$ ,  $u(a) = 0$ , replace  $x$  with its value  $a$  in  $\tilde{H}$ 
9:   else
10:     $d \leftarrow d + 1$  // Extend the system
11:     $H \leftarrow \tilde{H} \cup \{t \cdot p \mid t \text{ is a term, } p \in P, \deg(tp) = d\}$ 
12:   end if
13: end while
14: return  $U$ 

```

---

Now we are going to use mutants. Let  $f \in \text{span}(H_d)$  such that  $\text{LT}(f)$  is an element in  $\text{LT}(\text{span}(H_d) \setminus \text{LT}(\text{span}(H_{d-1})))$  and its degree is less than  $d$ . Then  $f$  is a mutant. If  $x$  is a variable, then the polynomial  $(x \cdot f)$  belongs to  $\text{span}(H_{d+1})$ , however, since its degree is less than  $d + 1$  then we can reduce it using only elements from  $H_d$ . So if we have such mutants, we can enlarge them one degree more, include them to  $H_d$  and echlonize  $H_d$  again to reduce them instead of generating the whole  $H_{d+1}$ .

We describe a first version of XL that uses mutants,  $XL_{Mutant}$  (Alg. 3.2). We define the set  $M$  to store the mutants obtained during the process and  $L$  to store the leading terms of the elements of  $\tilde{H}$ . We use definition (3.3) to identify mutants and extract them from  $\tilde{H}$  (Alg. 3.2, step 9). In this case, all new polynomials of degree less than the maximal degree  $d$  are mutants. Steps 14-17 in (Alg. 3.2) represent our

### 3 Mutant Strategy

improvement over the XL algorithm described in ( Alg. 3.1). If some mutants are found, then they are used as explained above.

---

#### Algorithm 3.2 $XL_{Mutant}$

---

**Input:**  $P(x_1, \dots, x_n)$  is a quadratic finite set of polynomials in  $R$ .

**Output:**  $U$  a set  $n$  univariate polynomials in  $x_1, \dots, x_n$ .

```

1:  $d \leftarrow 2$  // The highest degree in  $H$ 
2:  $H \leftarrow P$  // The set of generated polynomials
3:  $L \leftarrow \emptyset$  // The set of leading terms of  $H$ 
4:  $U \leftarrow \emptyset$  // the set of univariate polynomials
5:  $M \leftarrow \emptyset$  // The set of mutants
6: while  $|U| < n$  do
7:    $\tilde{H} \leftarrow Echelonize(H)$ 
8:    $U \leftarrow \{p \in \tilde{H} \mid \text{p is univariate}\}$ 
9:    $M \leftarrow \{p \in \tilde{H} \mid \text{deg}(p) < d \text{ and } LT(p) \notin L\}$ 
10:   $L \leftarrow LT(\tilde{H})$ 
11:  if  $U \neq \emptyset$  then
12:     $H \leftarrow \text{Substitute}(\tilde{H}, U)$ 
13:  else
14:    if  $M \neq \emptyset$  then
15:       $H \leftarrow \tilde{H} \cup \{x \cdot p \mid x \text{ is a variable, } p \in M\}$  // Extend mutants
16:       $M \leftarrow \emptyset$ 
17:    else
18:       $d \leftarrow d + 1$ 
19:       $H \leftarrow \tilde{H} \cup \{t \cdot p \mid t \text{ is a term, } p \in P, \text{deg}(tp) = d\}$ 
20:    end if
21:  end if
22: end while
23: return  $U$ 

```

---

We use an HFE system of size  $n = 25$  and univariate degree 96 as an example. We use both Algorithm 3.1 and Algorithm 3.2 to solve it. Table 3.2 shows the number of linearly independent polynomials generated at each degree during the process of both algorithms. For this example,  $XL_{Mutant}$  can solve the system with the highest degree 4 while XL goes to degree 6.

$XL_{Mutant}$  generates a lot of redundant computations when it finds many mutants with different degrees. Clearly, these redundant computations reduced to zero during the *Echelonize* step. As an example, if we compare the generated polynomials in step 4 and step 5 in Table 3.2, the  $XL_{Mutant}$  algorithm finds 1669 mutants 1400 of them have degree 3, 249 have degree 2, and 20 have degree 1. After it multiplies those mutants the generated matrix has size  $55444 \times 15276$  which leads to 40169 redundant computations. Since our target is to generate a polynomials with low degree as possible, we can divide the set of mutants into subgroups based on the degree and start to multiply the lowest degree subgroup. In this case, we multiply

### 3 Mutant Strategy

step	XL						$XL_{Mutant}$			
	$d_1$	$d_2$	$d_3$	$d_4$	$d_5$	$d_6$	$d_1$	$d_2$	$d_3$	$d_4$
1	0	25	0	0	0	0	0	25	0	0
2	0	25	625	0	0	0	0	25	625	0
3	0	25	675	7125	0	0	0	25	675	7125
4	0	224	1474	9025	46450	0	0	25	875	8175
5	25	300	2300	12650	53130	177100	20	274	2275	11150
6							25	300	2300	12650

Table 3.1: Performance of  $XL_{Mutant}$  versus XL

$n$	XL			MutantXL			
	$D$	Matrix size	Rank	$D$	Matrix Size	Rank	$M$
7	4	$203 \times 99$	98	3	$63 \times 64$	62	14
10	4	$560 \times 386$	385	4	$560 \times 386$	175	0
13	4	$1196 \times 1093$	1092	4	$1196 \times 1093$	1092	0
15	5	$8520 \times 4944$	4943	4	$3045 \times 1941$	1760	90
17	5	$14025 \times 9402$	9401	4	$3349 \times 3214$	2737	255
20	5	$23105 \times 21700$	21699	4	$8010 \times 6196$	6195	200
22	5	$23105 \times 21700$	21698	4	$8010 \times 6196$	6195	200
25	6	$261928 \times 245506$	245506	4	$14218 \times 15276$	13750	250
30	6	$717666 \times 768212$	768211	4	$22515 \times 31931$	22087	300
35	ran out of memory			4	$33705 \times 59536$	33209	350

Table 3.2: Performance of MutantXL versus XL

the 20 mutants of degree one which leads to solving the system and generate only 469 reductions to zero.

We present MutantXL as an improvement to XL that gives the superior to the lowest degree mutants. Algorithm 3.3 describes the MutantXL algorithm. In step 15, we determine the lowest degree of mutants  $k$  then we multiply these mutants and remove them from the set of mutants  $M$ .

Table 3.2 shows the results of comparing the performance of MutantXL and XL algorithms. We use HFE systems with univariate degree 96 and sizes  $7, \dots, 35$ . For each example, we present the size of the initial system  $n$ , the maximum degree bound  $d$  used in the algorithm, the maximum matrix size, and the number of mutants found in case of the MutantXL algorithm.

Our comparison is based on the size of the system that is constructed by both algorithms. In terms of solving multivariate polynomial systems, the efficiency is to use an extended system with as small size as possible. Table 3.2 clearly shows that MutantXL outperforms XL in most of the cases presented and solves with smaller matrix size. They have the same maximum matrix size for the HFE system with size  $n = 13$ . In this case, MutantXL finds no mutants for  $d < d_{XL}$ .

---

**Algorithm 3.3** MutantXL

---

**Input:**  $P(x_1, \dots, x_n)$  is a quadratic finite set of polynomials in  $R$ .

**Output:**  $U$  a set  $n$  univariate polynomials in  $x_1, \dots, x_n$ .

```

1:  $d \leftarrow 2$  // The highest degree in  $H$ 
2:  $H \leftarrow P$  // The set of generated polynomials
3:  $L \leftarrow \emptyset$  // The set of leading terms of  $H$ 
4:  $U \leftarrow \emptyset$  // the set of univariate polynomials
5:  $M \leftarrow \emptyset$  // The set of mutants
6: while  $|U| < n$  do
7:    $\tilde{H} \leftarrow \text{Echelonize}(H)$ 
8:    $U \leftarrow \{p \in \tilde{H} \mid \text{and } p \text{ is univariate}\}$ 
9:    $M \leftarrow \{p \in \tilde{H} \mid \text{deg}(p) < d \text{ and } \text{LT}(p) \notin L\}$ 
10:   $L \leftarrow \text{LT}(\tilde{H})$ 
11:  if  $U \neq \emptyset$  then
12:     $H \leftarrow \text{Substitute}(\tilde{H}, U)$ 
13:  else
14:    if  $M \neq \emptyset$  then
15:       $k \leftarrow \min\{\text{deg}(p) \mid p \in M\}$  // The lowest degree mutants
16:       $H \leftarrow \tilde{H} \cup \{x \cdot p \mid x \text{ is a variable, } p \in M \ \& \ \text{deg}(p) = k\}$ 
17:       $M \leftarrow M \setminus \{p \in M \mid \text{deg}(p) = k\}$ 
18:    else
19:       $d \leftarrow d + 1$ 
20:       $H \leftarrow \tilde{H} \cup \{t \cdot p \mid t \text{ is a term, } p \in P \ \& \ \text{deg}(t \cdot p) = d\}$ 
21:    end if
22:  end if
23: end while
24: return  $U$ 

```

---

### 3 Mutant Strategy

$n$	$d$	Matrix Size	$M$	$Z$	$PZ$
7	3	$154 \times 64$	14	91	59%
15	5	$3045 \times 1941$	90	1105	36%
22	5	$77209 \times 35443$	1980	41767	54%
23	5	$57408 \times 44552$	736	12857	22%

Table 3.3: Reductions to zero of the MutantXL algorithm

### 3.3 Improved mutant strategy

This section explains an improvement to the mutant strategy as introduced in [40]. In many experiments with the MutantXL algorithm, we observed that MutantXL generates many reductions to zero after enlarging mutants. These results indicate that MutantXL finds a large number of mutants at certain degree  $d$ , and after it multiplies all of these mutants it generates many redundant polynomials.

Table 3.3 shows the results of solving some randomly generated multivariate quadratic systems over  $\mathbb{F}_2$  with MutantXL (Alg. 3.3). For each example, we present the number of variables and equations ( $n$ ), the maximum degree bound ( $d$ ) used in the algorithm, the maximum matrix size, the number of mutants found ( $M$ ), the number of zero reductions ( $Z$ ), and finally the percentage of zero computations to the total number of extended polynomials size ( $PZ$ ).

As we can see from Table 3.3, MutantXL generates many reductions to zero some times it generates more than 50% of the total number of polynomials. For these cases MutantXL consumes much memory to represent many redundant polynomials and spends much time to reduce those polynomials to zero.

Our proposed improvement to the mutant strategy is to avoid the above problem. It is based on multiplying a subset of the generated mutants that have the same degree instead of multiplying all of them. This leads to solving a system with smaller matrix size and generates fewer zero reductions. We need the following notation to explain the improvement.

Let  $T_d$  be a set of terms in  $T$  (the set of terms in  $R$ ) defined as

$$T_d = \{t \in R \mid \deg(t) \leq d\}.$$

Combinatorially, the number of elements of this set can be computed as

$$|T_d| = \sum_{\ell=1}^d \binom{n}{\ell}, 1 \leq d \leq n \quad (3.3)$$

where  $n$  is the number of variables.

Let  $d_m$  be the maximum degree of elements generated by MutantXL and  $k$  be the lowest degree of the mutants found. Assume that the number of linearly independent polynomials produced by MutantXL of degree  $\leq k + 1$  is  $I_{k+1}$ . MutantXL extends the set of degree  $k$  mutants by multiplying each one with all variables. The extended

### 3 Mutant Strategy

$n$	$d$	Matrix Size	$M$	$NM$	$Z$	$PZ$
7	3	$63 \times 64$	14	1	1	1%
15	5	$1950 \times 1941$	90	17	10	0.5%
22	5	$35453 \times 35442$	1980	82	11	0.0%
23	5	$44551 \times 44552$	736	177	2	0.0%

Table 3.4: Reductions to zero of the improved MutantXL algorithm

polynomials have the degree  $k + 1$ . In the worst case, MutantXL needs to generate all terms in  $T_{k+1}$  as leading terms to solve the system. The minimum number of mutants that MutantXL should use to generate these  $|T_{k+1}|$  linearly independent polynomials of degree  $\leq k + 1$  is

$$NM = \lceil (|T_{k+1}| - I_{k+1})/n \rceil, \quad (3.4)$$

where  $T_{k+1}$  is computed as in (3.3) and  $n$  is the number of variables. We call  $NM$  the *necessary mutants*. Therefore, by multiplying only the necessary number of mutants, the system can potentially be solved by a smaller number of polynomials and a minimum number of multiplication. This handles the problem of finding many mutants that have the same degree. In case the number of mutants found is smaller than the necessary mutants, this improved version will behave as normal MutantXL.

Table 3.4 explains how the new version of MutantXL that uses the necessary mutants concept is indeed improved over the normal one. The number of zero computations and the number of rows of the total matrix are significantly decreased. Here we add the column  $NM$  to report the necessary mutants as computed in (3.4).

From the experiments above, we can conclude that the MutantXL algorithm can indeed outperform the XL algorithm and can solve multivariate systems at a lower degree than the usual XL algorithm. Since the total degree bound that the XL algorithm needs to go up is typically the bottle neck of this algorithm, this is quite a considerable improvement.

**Remark 3.4.** *The set of polynomials generated by any of MutantXL versions described in this chapter has the same maximal degree. Since they are only different in the number of mutants used during their process which will not affect on the maximal degree of the generated polynomials.*

### 3.4 Complexity bounds

We give complexity bounds for solving multivariate polynomial systems using MutantXL. Our aim is to establish an upper bound for the complexity of solving polynomial systems by MutantXL. For this we use the version of MutantXL that enlarges all mutants produced during the computation,  $XL_{Mutant}$  (3.2). The complexity of the XL algorithm has been used to estimate the hardness of solving polynomial systems in many studies. In [49] Yang et al, introduced a formula to compute the

---

**Algorithm 3.4** ImprovedMutantXL

---

**Input:**  $P(x_1, \dots, x_n)$  is a quadratic finite set of polynomials in  $R$ .

**Output:**  $U$  a set  $n$  univariate polynomials in  $x_1, \dots, x_n$ .

```

1:  $d \leftarrow 2$  // The highest degree in  $H$ 
2:  $H \leftarrow P$  // The set of generated polynomials
3:  $L \leftarrow \emptyset$  // The set of leading terms of  $H$ 
4:  $U \leftarrow \emptyset$  // the set of univariate polynomials
5:  $M \leftarrow \emptyset$  // The set of mutants
6:  $SM \leftarrow \emptyset$  // The set of necessary mutants
7:  $NM \leftarrow 0$  // The number of necessary mutants
8: while  $|U| < n$  do
9:    $\tilde{H} \leftarrow \text{Echelonize}(H)$ 
10:   $U \leftarrow \{p \in \tilde{H} \mid \text{and } p \text{ is univariate}\}$ 
11:   $M \leftarrow \{p \in \tilde{H} \mid \text{deg}(p) < d \text{ and } \text{LT}(p) \notin L\}$ 
12:   $L \leftarrow \text{LT}(\tilde{H})$ 
13:  if  $U \neq \emptyset$  then
14:     $H \leftarrow \text{Substitute}(\tilde{H}, U)$ 
15:  else
16:    if  $M \neq \emptyset$  then
17:       $k \leftarrow \min\{\text{deg}(p) \mid p \in M\}$  // The lowest degree mutants
18:       $NM \leftarrow$  right hand side of (3.4)
19:       $SM \leftarrow \{p \in M \mid \text{deg}(p) = k, |SM| < NM\}$ 
//  $NM$  is computed as in (3.4)
20:       $M \leftarrow M \setminus SM$ 
21:       $H \leftarrow \tilde{H} \cup \{x \cdot p \mid x \text{ is a variable, } p \in SM \ \& \ \text{deg}(p) = k\}$ 
22:    else
23:       $d \leftarrow d + 1$ 
24:       $H \leftarrow \tilde{H} \cup \{t \cdot p \mid t \text{ is a term, } p \in P \ \& \ \text{deg}(t \cdot p) = d\}$ 
25:    end if
26:  end if
27: end while
28: return  $U$ 

```

---

### 3 Mutant Strategy

number of linearly independent equations produced by XL of a maximal degree  $d$ . This formula has been verified by Rønjem and Raddum in [45].

Let us extend  $P$  up to degree  $d$ , the number of such linearly independent polynomial equations is computed as follows:

$$I_d = |\text{LT}(\text{span}(H_d))| = \sum_{i=0}^{\lfloor \frac{d_k}{d_e} \rfloor} (-1)^i \binom{m+i}{i+1} \sum_{j=0}^{d_k-i \cdot d_e} \binom{n}{j}, \quad (3.5)$$

where each polynomial in  $P = (p_1, \dots, p_m)$  has degree  $d_e$  and  $d_k = d - d_e$  is the highest degree of monomials used to extend  $P$ .

However, to give an upper bound on the complexity of the XL algorithm, we have to estimate the degree of its extended system. Let  $d_{XL}$  be the highest degree of the polynomials generated by XL. In the worst case, the constructed set of polynomials by XL,  $\tilde{H}_{d_{XL}}$ , satisfies

$$\forall t \in \bigcup_{i=1}^{d_{XL}} T_i, \exists p \in \tilde{H}_{d_{XL}} \text{ and } \text{LT}(p) = t. \quad (3.6)$$

Using (3.5) and (3.6), an upper bound for the maximal degree of XL computations ( $d_{XL}$ ) is given by

$$d_{XL} = \min\{d \mid I_d = (\sum_{i=1}^d |T_i|)\} \quad (3.7)$$

Let MutantXL terminates its computations successfully with a maximal degree  $d_m$ . Since MutantXL behaves like XL until it starts to generate mutants as described in Alg. (3.2) and Alg. (3.3). Then it enlarges the system up to  $d_m$  using the XL strategy and constructs the set  $H_{d_m}$ . We can use formula (3.5) to compute the number of linearly independent polynomials generated by MutantXL at degree  $d_m$ ,  $I_{d_m}$ , without using any mutants.

Now, we are going to illustrate an upper bound on the number of linearly independent polynomials produced by mutants. Let us start with understanding the results that we have obtained in Table 3.2. We assume that  $\tilde{H}_d$  is a row echelon form of  $H_d$ . Then  $\tilde{H}_d$  is a basis of the span generated by  $H_d$  and the set of leading terms  $\text{LT}(\tilde{H}_d) = \text{LT}(\text{span}(H_d))$ . The XL algorithm needs to construct the set of polynomials  $\tilde{H}_6$  to solve the system. In other words, XL cannot find the 25 different leading terms of degree one in  $\text{LT}(\text{span}(H_5))$ . While,  $XL_{Mutant}$  finds these 25 leading terms when it enlarges the system only up to degree 4 using mutants. However, it investigates a subset of  $\text{span}(H_6) \setminus \text{span}(H_5)$ . At step 3,  $XL_{Mutant}$  finds new 50 polynomials of degree 3 with leading terms belong to  $\text{LT}(\text{span}(H_4)) \setminus \text{LT}(\text{span}(H_3))$ .  $XL_{Mutant}$  enlarges these mutants by multiplying them with terms of degree one and reduces the resulting polynomials using the elements of  $\tilde{H}_4$ . It finds new 200 mutants of degree 3. The leading terms of them

### 3 Mutant Strategy

are elements in  $\text{LT}(\text{span}(H_5)) \setminus \text{LT}(\text{span}(H_4))$ . Enlarging those mutants means that  $XL_{Mutant}$  investigates some elements of  $\text{span}(H_6) \setminus \text{span}(H_5)$  and so on. To be more precise, at least one of the 25 linearly independent linear polynomials is an element in  $\text{span}(H_6) \setminus \text{span}(H_5)$ . This means that  $XL_{Mutant}$  must investigate at least this polynomial to solve. This discussion leads us to the following proposition.

**Proposition 3.5.** *Let  $P$  be a set of finite elements in  $R$  and the system  $(P = 0)$  have only one solution. Let  $H_d$  be defined as in (2.4), then  $XL_{Mutant}$  must evaluate some elements with maximal degree  $\leq d_m$  of  $\text{span}(H_{d_{XL}}) \setminus \text{span}(H_{d_{XL}-1})$ , where  $d_m$  and  $d_{XL}$  are the highest degree of the system generated by  $XL_{Mutant}$  and  $XL$ , respectively.*

*Proof.* In case of  $XL_{Mutant}$  does not find mutants, it solves as well as  $XL$  and generates all the elements of  $H_{d_{XL}}$ .

Let  $XL_{Mutant}$  terminate and return the solution with maximal degree  $d_m < d_{XL}$ . Let  $d_{XL} = d_m + 1$  and  $g$  be a mutant of degree  $< d_m$ , then  $g \in \tilde{H}_{d_m} \setminus H_{d_m-1}$ . Let  $g$  be obtained after reducing  $f \in H_{d_m}$ , where  $\deg(f) = d_m$ . Then  $\exists f_1, \dots, f_s \in H_{d_m}$  such that  $g = f + f_1 + \dots + f_s$ . Let  $t$  be a term of degree one and  $t \cdot g = t \cdot f + t \cdot f_1 + \dots + t \cdot f_s$ . Since  $t \cdot f, t \cdot f_1, \dots, t \cdot f_s \in H_{d_m+1}$ , then  $t \cdot g \in (\text{span}(H_{d_m+1}) \setminus \text{span}(H_{d_m}))$ . Therefore  $XL_{Mutant}$  generates some elements of  $\text{span}(H_{d_{XL}}) \setminus \text{span}(H_{d_{XL}-1})$ .

Let  $d_{XL} > d_m + 1$  and  $M$  be the set of mutants obtained after constructing  $\tilde{H}_{d_m}$ . Assume that  $XL_{Mutant}$  terminates and returns the solution using only the mutants in  $M$ . Therefore,  $XL_{Mutant}$  can compute  $n$  linearly independent linear polynomials by representing only a subset of  $\text{LT}(\text{span}(H_{d_m+1}))$  as leading terms. However,  $XL$  represents all terms of  $\text{LT}(\text{span}(H_{d_m+1}))$  as leading terms, since  $d_m + 1 < d_{XL}$ , and cannot generate these linear polynomials. This is a contradiction. Then  $XL_{Mutant}$  cannot solve the system without generating some elements of  $\text{span}(H_{d_{XL}}) \setminus \text{span}(H_{d_{XL}-1})$ .  $\square$

Figure 3.1 explains the results reported in Table 3.5. It shows how  $XL$ ,  $XL_{Mutant}$ , and  $MutantXL$  algorithms investigate  $\text{span}(H_d)$  of the HFE system with  $n = 25$ . Here we report the newly constructed leading terms (in other words, leading terms of the new linearly independent polynomials) of  $(\text{LT}(\text{span}(H_d)) \setminus \text{LT}(\text{span}(H_{d-1})))$  by each algorithm (in other words the linearly independent polynomials). Up to the degree of generating mutants all algorithms evaluate all leading terms in  $\text{LT}(\text{span}(H_d)) \setminus \text{LT}(\text{span}(H_{d-1}))$ . However, after  $XL_{Mutant}$  and  $MutantXL$  starting to generate some mutants, they construct only a few leading terms. Moreover, when they find different degrees mutants  $MutantXL$  uses only the lowest degree one which leads to investigating fewer elements from  $\text{LT}(\text{span}(H_d)) \setminus \text{LT}(\text{span}(H_{d-1}))$  than  $XL_{Mutant}$ .

As we have explained above,  $MutantXL$  proceeds like  $XL$  up to the degree of finding mutants. It generates a sufficient number of mutants at the same degree it terminates  $d_m$ . Let  $I$  be the number of linearly independent polynomials generated by mutants. From Proposition 3.5,  $MutantXL$  must investigate some elements from  $\bigcup_{i=d_m+1}^{d_{XL}} H_i$ , where  $d_{XL}$  is the termination degree of  $XL$ . Since  $MutantXL$  terminates

### 3 Mutant Strategy

$d$	2	3	4	5	6	7
XL	25	625	7175	49348	188332	0
$XL_{Mutant}$	25	625	7175	1900	4644	1556
MutantXL	25	625	7175	1900	4644	31

Table 3.5: New elements generated from  $H_d$  by XL,  $XL_{Mutant}$ , and MutantXL

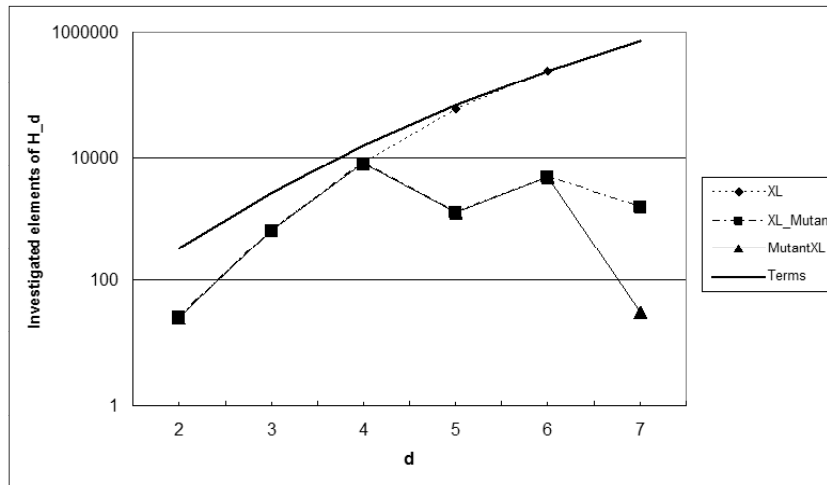


Figure 3.1: Compare the investigated parts of the set  $H_d$  for XL,  $XL_{Mutant}$ , and MutantXL.

### 3 Mutant Strategy

at the highest degree  $d_m$ , then all elements generated by it from  $\bigcup_{i=d_m+1}^{d_{XL}} H_i$  should be of degree  $\leq d_m$ . So, the number of linearly independent polynomials generated by mutants is upper bounded by the number of those elements.

The authors of [47] have presented an upper bound formula for  $I$ . This formula is working only when  $d_{XL} = d_m + 1$ . We verify and generalize it as follows. Let  $I_d^m$  is the maximum number of linearly independent polynomials constructed by MutantXL from  $\text{span}(H_d) \setminus \text{span}(H_{d-1})$  for  $d_m < d \leq d_{XL}$ . Then

$$I_{d_m+1}^m = I_{d_m+1} - I_{d_m} - |T_{d_m+1}|$$

and

$$I_{d_m+2}^m = I_{d_m+2} - I_{d_m+1} - |T_{d_m+2}|.$$

Then

$$I_{d_m+2}^m + I_{d_m+1}^m = I_{d_m+2} - I_{d_m} - (|T_{d_m+2}| + |T_{d_m+1}|)$$

and so on up to  $d = d_{XL}$ . Therefore, the total number of linearly independent polynomials generated by mutants  $I$  is bounded as in the following formula

$$I \leq (I_{d_{XL}} - I_{d_m}) - \sum_{i=d_m+1}^{d_{XL}} |T_i| \quad (3.8)$$

Another trivial upper bound can be obtained as follows. The maximum number of linearly independent polynomials that MutantXL needs to terminate at a degree ( $d_m$ ) is to compute all possible linear independents of degree less or equal than  $d_m$ . Since MutantXL has already  $I_{d_m}$  from them, then the upper bound of linear independents produced by MutantXL is

$$I \leq \left( \sum_{i=1}^{d_m} |T_i| \right) - I_{d_m} \quad (3.9)$$

Since we multiply mutants by all terms of degree 1 that are equal to  $n$ , then the necessary number of mutants needed for computing Gröbner basis is

$$N_m = \lceil I/n \rceil, \quad (3.10)$$

As an example, let the HFE-96 system of size  $n = 29$ , XL solves this system at degree  $d_{XL} = 6$  and MutantXL solves it at degree  $d_m = 4$ . The exact linearly independent polynomials generated by mutants is  $I = 15628$ . By computing the upper bound, we have obtained  $I < 16240$  by (3.8) and  $I < 15631$  by (3.9). Formula (3.9) is very close to the exact number of linearly independent polynomials generated by mutants for MutantXL described in Algorithm 3.2.

Figure 3.2 compares the maximum number of linearly independent polynomials generated by mutants in the computations of MutantXL. We used Formula (3.9) and (3.8) to estimate the number of linearly independent polynomials generated by

### 3 Mutant Strategy

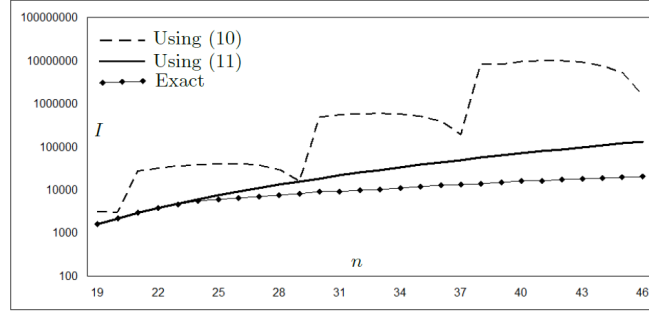


Figure 3.2: Compare upper bounds of  $I$  using (3.8) and (3.9) to the exact value of  $I$  for HFE-96 systems solved by MutantXL at degree 4.

mutants and the exact value when we use only the lowest degree mutants. For these experiments, we use HFE systems with univariate degree 96. These systems are solved at degree 4 as explained by Faugère and Joux in [26]. We used Formula (3.5) to estimate the number of linearly independent polynomials computed by XL at a certain degree  $d$ . XL solves these systems as it solves random, while MutantXL uses mutants appeared at degree 4 and terminates the computations at this degree.

**Remark 3.6.** *Let  $P$  be a set of finite elements in  $R$  and the system  $(P = 0)$  have only one solution. Let the maximal degree of the polynomials generated during the process of MutantXL be  $d_m$ . As described in Alg. (3.2), MutantXL enlarges the system using XL's method up to  $d_m$ . Moreover, it extends all polynomials of degree  $< d_m$  (mutants) obtained during its computations. Then any other algorithm that solves the system  $(P = 0)$  and uses a degree-based selection strategy should at least generate a subset of polynomials with a maximal degree  $d_m$ .*

The proof of this remark is very similar to the proof of Proposition 3.5. Since MutantXL enlarges  $P$  up to the degree  $d_m$  by multiplying it by all possible terms in  $R$  such that the resulting polynomials have degree  $\leq d_m$ . Moreover, MutantXL enlarges all lower degree polynomials (mutants) obtained after the linear algebra step (*Echelonize*) up to  $d_m$ . Therefore, any algorithm uses degrees of polynomials as the basic role for its selection strategy, enlarges the selected ones, and using linear algebra can not terminate successfully with maximal degree smaller than  $d_m$ .

We can reconstruct the above remark as follows. Since MutantXL enlarges all linearly independent lower degree polynomials obtained during its computations, then the maximal degree of MutantXL is the lowest among any other algorithms for solving multivariate equation systems that enlarge the system using a degree-based selection strategy.

As explained in the previous chapter, computing a Gröbner basis of the ideal generated by a finite set of polynomials  $P$  is one of the methods to solve the system  $P = 0$ . Let  $F$  be a finite set of randomly generated homogenous polynomials in  $R$ . Let  $T_d$  be the set of degree  $d$  terms. The degree of regularity [4], denoted by  $d_{reg}$ , is

### 3 Mutant Strategy

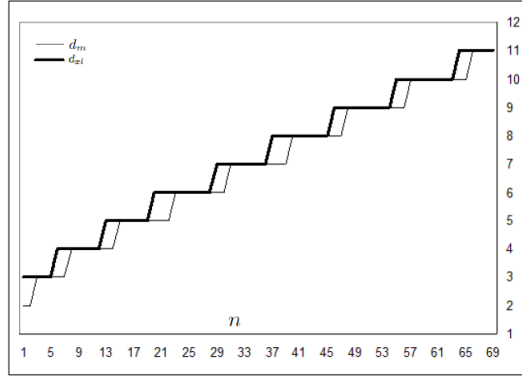


Figure 3.3: Comparison between XL and MutantXL maximal degrees for random systems of size  $n$ .

defined as

$$d_{reg} = \min\{d \geq 0 \mid \forall t \in T_d \exists p \in \langle F \rangle, \text{LT}(p) = t \text{ and } \deg(p) = d\}$$

Based on the assumption that, the degree of regularity ( $d_{reg}$ ) is the maximal degree appeared during Gröbner basis computations [4] and Remark 3.6, then MutantXL terminates with a maximal degree equal to  $d_{reg}$ .

Figure 3.3 compares the maximal degree of the polynomials generated by XL ( $d_{XL}$ ) and by MutantXL ( $d_m$ ) of a random system. It is based on the comparison between XL and Gröbner basis algorithms introduced in [4]. In this case, the maximal degree of MutantXL is equal to the degree of regularity ( $d_m = d_{reg}$ ). We used Formula (3.5) to estimate the number of linearly independent polynomials computed by XL at a certain degree  $d$ . The maximal degree in the XL computations is such a degree that satisfies (3.9).

### 3.5 Implementation and experimental results

This section explains our implementation for algorithms described in this chapter and compares the implementation of the version of MutantXL described in Alg. (3.4) to the widely used Magma's implementation of the  $F_4$  algorithm. MutantXL and all algorithms presented in this thesis have been fully implemented in C++ and over the Boolean ring  $B$ , as defined in (2.3).

Our implementation is working only for systems that have only one solution. The set of variables is represented as an ordered vector of variables indexes. Let for example the set of variables be  $X = \{x_1, \dots, x_n\}$ , then  $X$  is represented by the following vector  $(1, \dots, n)$ . A term  $t = x_{i_1} \dots x_{i_l}$  is represented by the combination  $(i_1, \dots, i_l)$ . The set of terms up to certain degree  $d$  is constructed as an ordered list of such combinations of variables indexes. We use the graded lexicographic ordering

### 3 Mutant Strategy

		MutantXL			F <sub>4</sub>			
$n$	$d$	Matrix size	Time	Mem	$d$	Matrix Size	Time	Mem
18	5	15340×12616	5	23	5	15187×8120	2	43
19	5	19193×16664	8	38	5	18441×11041	4	64
20	5	23708×21700	12	61	5	22441×14979	7	96
21	5	28970×27896	21	96	5	26860×19756	17	139
22	5	35343×35443	47	149	5	63621×21855	54	340
23	5	44252×44552	89	235	5	41866×29010	83	475
24	6	231581×190051	4394	5246	6	207150×78637	642	3069
25	6	287992×245506	7335	8428	6	248495×108746	1842	5348
26	6	355082×313912	12557	13287	6	298592×148804	3641	8744
27	6	436602×397594	26164	20990	6	354189×197902	6430	13601
28	6	528844×499178	39248	31846	6	420773×261160	13904	20434

Table 3.6: Performance of MutantXL versus Magma’s F<sub>4</sub> on random system

( $\langle_{lex}$ ) to sort the set of terms in our implementation. Let for example we have the 3 variables  $x_1, x_2, x_3$ , then the set of terms up to degree 2 ordered by ( $\langle_{lex}$ ) is  $T_2 = \{x_1x_2, x_1x_3, x_2x_3, x_1, x_2, x_3, 1\}$ . The set  $T_2$  is represented by the following ordered list of combinations:

$$T_2 = [(1, 2), (1, 3), (2, 3), (1), (2), (3), (0)].$$

We use the dense matrix representation to store the polynomial system during the process as explained in Chapter 2. In this matrix, columns are presenting the ordered list of terms  $T_d$  and rows are presenting the entries of each polynomial. Each term is assigned to a column index. We used the method of ranking and un-ranking combinations that have been explained in [33] to connect a column and its corresponding term. In this case we divide the set of column indexes into subsets based on the degree of the corresponding terms. Each subset is numbered from 1 to  $\binom{n}{k}$  and  $k \leq d$ , where  $d$  is the maximal degree of terms. Let for example,  $n = 5$  and maximal degree  $d = 3$ , then the matrix columns are indexed as follows Each number is un-ranked to its combination (term) using the un-ranking algorithm [33] and vice versa. We use this method to perform the multiplication and the simplification steps. For the linear algebra step (*Echelonize*), we used the M4RI package [2]. It is a library for fast arithmetic with dense matrix over  $\mathbb{F}_2$ . It was started by Gregory Bard and is maintained by Martin Albrecht. It is based on the Method of the Four Russian inversion algorithm described by Gregory Bard in [3]. We adapt it for our implementation by changing the strategy of selecting a pivot during Gaussian elimination to keep the old elements in the system intact.

In terms of identifying mutants after the *Echelonize* step, we use a Boolean array with size equal to the number of rows in the polynomial system matrix. We use this array to distinguish the new elements from the old ones in the system. After

### 3 Mutant Strategy

		MutantXL			F <sub>4</sub>			
$n$	$d$	Matrix size	Time	Mem	$d$	Matrix Size	Time	Mem
18	5	15342×12616	4	23	5	15134×8067	2	43
19	5	19179×16664	8	38	5	18441×11041	4	64
20	5	23708×21700	6	38	5	22513×15051	8	96
21	5	28970×27896	17	93	5	27018×19914	17	139
22	5	35343×35443	49	149	5	63758×21992	56	341
23	5	44252×44552	70	237	5	41963×29107	91	475
24	5	54252×55455	116	366	5	46593×37620	157	740
25	5	67275×68406	217	558	5	54028×47734	258	1136
26	5	81523×83682	420	813	5	62446×59925	462	1692
27	5	99036×101584	630	1199	5	72727×75270	656	2507
28	5	113302×122438	961	1654	5	83467×92603	934	4036

Table 3.7: Performance of MutantXL versus Magma’s F<sub>4</sub> on HFE-288 system

the system has been enlarged, we tag new rows by 1 and old rows by 0. During the *Echelonize* step the entries of that array are swapped as well as the corresponding rows in the matrix. We extract mutants as follows. Let the column that contains the leading entry of a row  $r$  be connected to a term with a degree less than the system degree and  $r$  be tagged by 1, then  $r$  is a mutant.

In the previous sections, we have presented some experiments that explain the performance of MutantXL over XL and the steps of our improvements. Now, we built experiments to compare the performance of our implementation to the Magma’s implementation of F<sub>4</sub>. We run all the experiments on a Sun X4440 server, with four “Quad-Core AMD Opteron™ Processor 8356” CPUs and 128 GB of main memory. Each CPU is running at 2.3 GHz. We used only one out of the 16 cores.

Table 3.6 and Table 3.7 report the results on some random generated systems and HFE systems of univariate degree 288. There we compare the highest degree, the maximum matrix size, the time, and the memory consumed of both algorithms.

The selection strategy used by the version of F<sub>4</sub> implemented in Magma is based on the total degrees of polynomials together with the s-polynomials as explained in the previous chapter. This leads F<sub>4</sub> to construct a maximal matrix of smaller size compared to the matrix constructed by MutantXL, as shown in Table 3.6 and Table 3.7. However, the simple selection strategy of MutantXL enables us to represent the whole polynomials in one matrix and we didn’t need to perform any pre-processing process. We have no details about Magma’s implementation To F<sub>4</sub>. Our implementation consumes a fewer memory space than Magma’s implementation of F<sub>4</sub> in many cases. For all HFE systems that we use in the experiments reported in Table 3.7, our implementation used less memory than F<sub>4</sub>. However, as the maximal degree  $d$  is increased, the difference in the matrix size becomes bigger.

Figure 3.4 compares the maximum number of terms generated by MutantXL and F<sub>4</sub> which affects the maximal number of polynomials needed to terminate the com-

### 3 Mutant Strategy

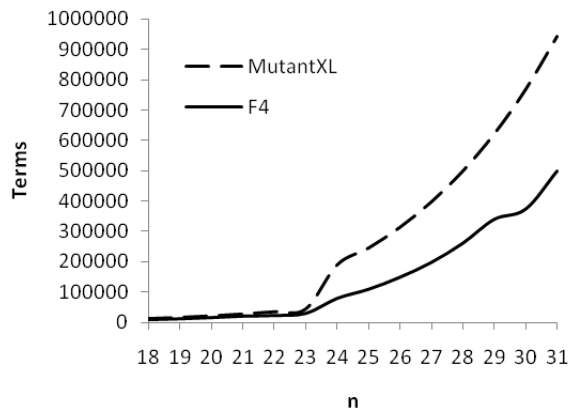


Figure 3.4: Comparison between the maximum number of terms generated by MutantXL and  $F_4$  for random systems of size  $n$ .

putations and return the solution. Figure 3.4 is based on the experiments explained in Table 3.6.

## 4 Partial Enlargement Strategy

In this chapter, our second contribution (the leading-variable-based partial enlargement strategy) of our research will be explained. We present MutantXL<sub>2</sub> (MXL<sub>2</sub>) as an improved version of MutantXL by applying the partial enlargement strategy. We use the so-called leading variable to divide a set of polynomials with elements having the same degree into subsets. We incrementally enlarge one subset at a time instead of enlarging the whole set at once, this may lead to solve systems with less number of polynomials. Experiments on HFE systems and randomly generated systems show that, in many cases MXL<sub>2</sub> solves systems with significantly smaller matrix than MutantXL. Moreover, experiments demonstrate that for a significant number of cases, there is a reduction in the size of the largest matrix of MXL<sub>2</sub> compared to Magma's implementation of the F<sub>4</sub> algorithm in a significant number of cases.

### 4.1 The basic idea

In this section, we introduce a partial enlargement strategy and explain how it improves MutantXL in terms of no mutants found. The generic concept of that strategy is described as follows. Let  $P$  be a finite set of polynomials in the polynomial ring  $R$ . Let the set  $H_d^+$  of degree  $d$  polynomials in  $R$  be defined as:

$$H_d^+ := \{t \cdot p \mid t \text{ is a term}, p \in P, \deg(tp) = d\}. \quad (4.1)$$

Assume that XL enlarges  $P$  degree-by-degree. In this case, XL uses the polynomials of  $H_d^+$  to construct the degree  $d + 1$  polynomials. Our strategy is to divide  $H_d^+$  into subgroups and explore the degree  $d + 1$  polynomials partially. We incrementally enlarge one subgroup of  $H_d^+$  at a time, until the elements of one iteration are sufficient to solve the system. We call this strategy *partial enlargement* which was first introduced in [40].

As we explained in the previous chapter, lower degree polynomials (mutants) that are produced during the process of the XL algorithm can be used to improve the algorithm. We also showed that MutantXL only outperforms XL in the case where it finds a sufficient number of mutants of degree  $< d_m$  such that  $d_m < d_{XL}$ , where  $d_{XL}$  is the highest degree of XL. Therefore, MutantXL could improve over XL only when mutants appeared in lower degree than  $d_{XL}$ . However, in many experiments with MutantXL, specially on randomly generated multivariate systems, we have noticed that MutantXL does not generate any mutants. In this case, the mutant strategy does not work, and MutantXL behaves like XL.

Table 4.1 shows the results of solving some randomly generated multivariate

#### 4 Partial Enlargement Strategy

$n$	$d$	Matrix size	Rank
16	5	9447×6885	6884
17	5	12120×9402	9402
18	5	15345×12616	12615
19	5	19181×16664	16663
20	5	23722×21700	21699
24	6	231499×190051	190050
25	6	287875×245506	245504
26	6	355082×313912	313910

Table 4.1: Experiments of MutantXL and XL on random systems.

quadratic systems over  $\mathbb{F}_2$  using MutantXL. We select some systems that produced no mutants during the process of MutantXL which give us the same results as XL. For each case, we present the size of the initial system ( $n$ ), the maximal degree of the polynomial generated by MutantXL ( $d$ ), the maximum matrix size, the rank of the matrix.

We proved in the previous chapter that MutantXL should investigate a subset of the span generated by  $H_{d_{XL}}^+$ , where  $d_{XL}$  is the maximal degree of the polynomials generated by XL. MutantXL uses mutants that have been found at degree  $d_m < d_{XL}$  to explore only a subset of polynomials of the span

$$\text{span}\left(\bigcup_{k=d_m+1}^{d_{XL}} H_k\right).$$

However, it needs to represent all terms in  $\text{LT}(\text{span}(H_{d_{XL}}^+))$  as leading terms to solve the system when it finds no mutants as the cases that are reported in Table 4.1. For these cases, we use a partial enlargement strategy to improve XL. In order to specify this strategy, we need the following additional notation.

Let  $X := \{x_1, \dots, x_n\}$  be a set of variables, for which we impose the following order:

$$x_1 > x_2 > \dots > x_n.$$

Assume the terms of  $R$  have been ordered by the graded lexicographical order  $<_{\text{lex}}$ . By an abuse of notation, we call the elements of  $R$  polynomials. The leading variable of  $p \in R$ ,  $\text{LV}(p)$ , is defined according to the order defined on  $X$  as:

$$\text{LV}(p) := \max\{x \mid x \in \text{LT}(p)\}.$$

For example, let  $p = x_1x_2x_4 + x_1x_3x_5 + \dots$ , then  $\text{LV}(p) = x_1$ .

We define the subset  $\text{LV}(P, x) \subset P$ , the *variable partition*, as follows:

$$\text{LV}(P, x) = \{p \in P \mid \text{LV}(p) = x\}.$$

#### 4 Partial Enlargement Strategy

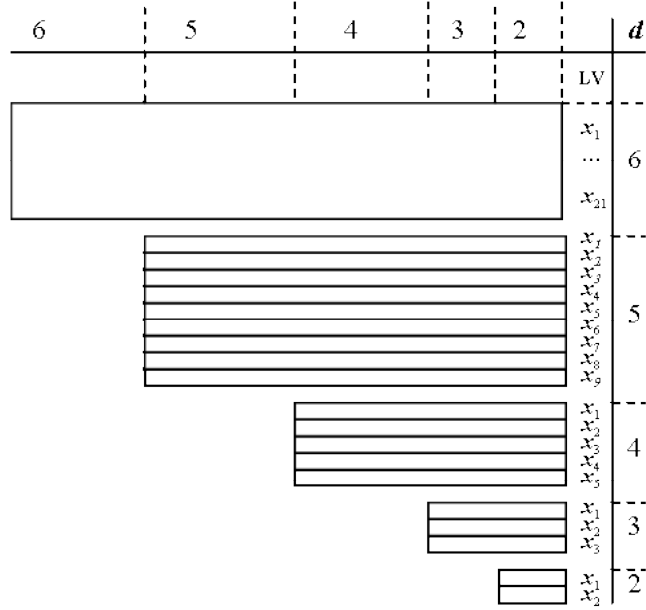


Figure 4.1: Variable partitions of polynomials generated by XL for a random system of size  $n = 26$ .

Regarding the results of Table 4.1, we have studied the total system of polynomials that are generated by XL. We have observed that each degree part can be partitioned using the leading variable and construct the so called *variable partitions*. For each degree  $d$  and after the linear algebra step (*Echelonize*), the set  $H_d^+$  (4.1) is divided into subsets based on the leading variable of its polynomials. Since the polynomials of  $H_d^+$  are ordered using the graded lexicographical order, then  $H_d^+$  are partitioned from up to down by  $x_1, x_2, \dots, x_n$  partitions. Only some of these partitions are not empty.

Figure 4.1 shows the structure of the total system generated by XL for a random system of size  $n = 26$ . Horizontal stripes represent non-empty variable partitions. For example, at  $d = 5$ ,  $H_5^+$  is divided into 9 partitions ( $x_1$ -partition,  $\dots$ ,  $x_9$ -partition). Let the set  $H_d^+$  be in the row echelon form, we assume a version of XL that uses  $H_d^+$  to enlarge the system and construct  $H_{d+1}^+$  as we will explain in the next section. In this case, XL enlarges the system to the degree 6 level by generating all elements of  $H_6^+$  using all partitions of  $H_5^+$  at once. Our new strategy suggests to stepwise constructing  $H_6^+$  by enlarging one partition of  $H_5^+$  per time. In this case, the partition with the smallest leading variable ( $H_5^+, x_9$ ) is enlarged first, then the next smallest ( $H_5^+, x_8$ ), and so on.

Let the XLP algorithm be the version of XL that enlarges the system using the strategy explained above. Table 4.2 explains how the partial enlargement strategy improves the XL algorithm. There we report in addition to Table 4.1, the greatest

#### 4 Partial Enlargement Strategy

$n$	$d$	GLV	XLP matrix	Rank	XL matrix
16	5	$x_8$	2667×2744	2645	9447×6885
17	5	$x_6$	3984×4493	3984	12120×9402
18	5	$x_6$	7481×6898	6083	15345×12616
19	5	$x_5$	12280×9404	9403	19181×16664
20	5	$x_4$	14540×12384	12383	23722×21700
24	6	$x_{10}$	59737×60627	58460	231499×190051
25	6	$x_8$	91347×88670	80784	287875×245506
26	6	$x_7$	126530×128766	110806	355082×313912

Table 4.2: Experiments of XLP on some random systems.

leading variable (GLV) of the polynomials used to enlarge the last step, the size of the largest matrix produced by XLP, and the size of the XL matrix. The results show how this approach indeed improves over XL. XLP generates, in the worst case, less than 50% of the XL total polynomials, also the number of terms appeared in the system generated by XLP is significantly smaller than what appeared in the XL system. It is clear that XLP terminates with a maximal degree equal to the maximal degree of XL. Since in the worst case, XLP terminates after exploring all parts of degree  $d_{XL}$ .

**Example 4.1.** *As a toy example to illustrate how the partial enlargement technique works. Let us have  $P = \{p_1, p_2, p_3, p_4, p_5\}$ , a set of polynomials over  $\mathbb{F}_2$  in  $\{x_1, x_2, x_3, x_4, x_5\}$ , and let  $P$  be in the row echelon form:*

$$\begin{aligned}
 p_1 &= x_1x_3 + x_2x_3 + x_3x_4 + x_3 + x_5 + 1 \\
 p_2 &= x_1x_5 + x_2x_5 + x_3x_5 + x_4x_5 + x_1 + x_2 + x_4 \\
 p_3 &= x_2x_4 + x_3x_4 + x_1 + x_2 + 1 \\
 p_4 &= x_2x_5 + x_3x_5 + x_4x_5 + x_1 + x_2 + x_3 + x_4 + 1 \\
 p_5 &= x_3x_5 + x_1 + x_2 + x_4 + x_5 + 1
 \end{aligned}$$

*We can divide  $P$  using the leading variable concept to 3 different partitions,  $\text{LV}(P, x_1) = \{p_1, p_2\}$ ,  $\text{LV}(P, x_2) = \{p_3, p_4\}$ ,  $\text{LV}(P, x_3) = \{p_5\}$ . Using the partial enlarged technique, first we enlarge  $\text{LV}(P, x_3)$ . Multiplying  $p_5$  with all variables, we obtain the following polynomials:*

#### 4 Partial Enlargement Strategy

$$p_6 = x_1x_3x_5 + x_1x_2 + x_1x_4 + x_1x_5$$

$$p_7 = x_2x_3x_5 + x_1x_2 + x_2x_4 + x_2x_5$$

$$p_8 = x_1x_3 + x_2x_3 + x_3x_4 + x_3$$

$$p_9 = x_3x_4x_5 + x_1x_4 + x_2x_4 + x_4x_5$$

$$p_{10} = x_1x_5 + x_2x_5 + x_3x_5 + x_4x_5$$

The new polynomials  $p_6, p_7, p_8$  are included to  $P$ . After echelonizing  $P$ ,  $XLP$  has the following two linear equations:

$$\tilde{p}_6 = x_5 + 1$$

$$\tilde{p}_8 = x_1 + x_2 + x_4$$

By substituting with  $x_5 = 1$ , we have the following equations:

$$\tilde{p}_6 = x_5 + 1$$

$$\tilde{p}_2 = x_3$$

$$\tilde{p}_8 = x_2x_4 + x_2$$

$$\tilde{p}_4 = x_1 + 1$$

$$\tilde{p}_8 = x_2 + x_4 + 1$$

which leads to solving the system.  $XLP$  solves with the total number of equations 10, while  $XL$  solves it using 30 equations.  $XL$  extends the system by multiplying all polynomials in the initial set of polynomials  $P$  by  $x_1, x_2, x_3, x_4, x_5$  which leads to producing 25 equations that are included to the original system.

Our proposed partial enlargement strategy is applicable to any algorithm for solving a polynomial system  $P$  over the polynomial ring  $R$ , such that the following are satisfied:

- The graded lexicographical ordering is used during the computations.
- The system is enlarged by multiplying a selected set of polynomials in the system by a selected set of terms in  $R$ .
- A linear algebra technique is used to echelonize the enlarged system.

In the next section, we apply our partial enlargement strategy to the improved version of MutantXL that we described in the previous chapter. However, it can be applied as a selection strategy for the  $F_4$  algorithm [24] and some versions of the  $F_5$  algorithm [1].

## 4.2 The MXL<sub>2</sub> algorithm

We describe the algorithm that uses the partial enlargement strategy, explained in the previous section, to improve the MutantXL algorithm. We call it MXL<sub>2</sub>.

We use the notations of the previous section. So  $P$  is a set of polynomials in  $R$  such that the system of equations  $P = 0$  has a unique solution. Without loss of generality, we assume that  $P$  is quadratic.

In order to improve the MutantXL algorithm using our partial enlargement technique, we need to modify the enlargement step of the algorithm. First, we recall its enlargement strategy. Let MutantXL need to enlarge the system from a degree  $d - 1$  to  $d$ . It generates the set of exactly degree  $d$  polynomials  $H_d^+$  by constructing all products  $t \cdot p$ , where  $p \in P$  and a term  $t \in R$  such that  $\deg(t \cdot p) = d$ . The set  $H_d$  defined as in (2.4) can be redefined as

$$H_d = H_{d-1} \cup \{t \cdot p \mid t \text{ is a term, } p \in P, \deg(t \cdot p) = d\}. \quad (4.2)$$

Our modification to that method is based on using the echelonized  $H_{d-1}^+$  to construct the extended set of degree  $d$  polynomials  $H_d$  as follows

$$H_d := H_{d-1} \cup \{x \cdot p \mid p \in H_{d-1}^+ \text{ and } x \in X\}. \quad (4.3)$$

In this context, the performance of this version of MutantXL is worth than the original one. Since, when we use  $H_{d-1}^+$  to enlarge the system we have obtained a large number of trivial redundant computations. For example, let  $P$  be a quadratic finite set of polynomials over  $\mathbb{F}_2[x_1, \dots, x_n]/(x_1^2 - x_1, \dots, x_n^2 - x_n)$  and  $d = 4$ , we use  $H_3^+$  to construct  $H_4$ . Assume that  $p \in P$  then  $\exists q_1, q_2 \in H_3^+$  and  $x_i \neq x_j \in \{x_1, \dots, x_n\}$  such that:  $q_1 = x_i p$  and  $q_2 = x_j p$ , where  $\text{LT}(p)$  is not divisible by  $x_i$  and  $x_j$ . To construct  $H_4$ ,  $q_1, q_2$  should be multiplied by  $x_1, \dots, x_n$ . In this case we have the following trivial redundant pairs:  $(q_1, x_i q_1)$ ,  $(q_2, x_j q_2)$ ,  $(x_j q_1, x_i q_2)$ .

Fortunately, it is possible to avoid these redundant computations by using some information from the previous multiplication. More precisely, if we know the previous variable multiplier, we can avoid such computations. Let  $p \in H_{d-2}^+$  and  $(q = x_k p) \in H_{d-1}^+$ , we enlarge  $q$  by multiplying it by all  $x \in X$  such that  $x < x_k$ , according to the order defined on  $X$ . We call  $x_k$  in this case  $\text{prev}(q)$ , the previous variable multiplier of  $q$ . Let us go back to the above example,  $q_1 = x_i p, q_2 = x_j p$ . We enlarge  $q_1$  and  $q_2$  by using only variables smaller than  $x_i$  and  $x_j$  respectively. Assuming that  $x_i > x_j$ , then we compute only  $x_j q_1$  and avoid trivial redundant computations  $x_i q_1, x_j q_2$ , and  $x_i q_2$ .

In order to describe the algorithm presented in this chapter, we need to represent the set of polynomials as a set of pairs  $(p, x)$ , where  $p$  is a polynomial and  $x$  is  $\text{prev}(p)$ . In other words, if  $q = x_1 g$  then  $\text{prev}(q) = x_1$  and we construct the pair  $(g, x_1)$ . Let  $p$  be an original polynomial, then  $\text{prev}(p) = 1$  and  $p$  is enlarged by multiplying it with all  $x \leq x_1$ . A matrix representation of the set of pairs

#### 4 Partial Enlargement Strategy

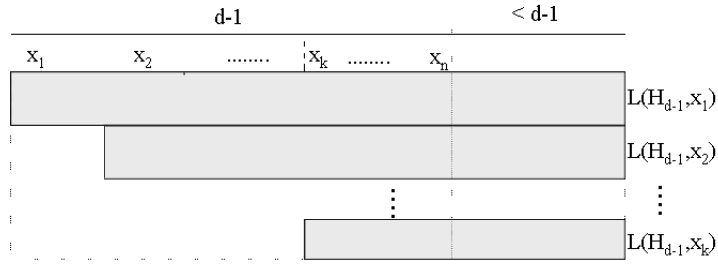


Figure 4.2: The partitions of  $H_{d-1}^+$

$H = \{(p_{i_1}, x_{i_1}), (p_{i_2}, x_{i_2}), \dots, (p_{i_m}, x_{i_m})\}$  in this case is as follows.

$$\left( \begin{array}{c} \overbrace{\begin{array}{c} x_{i_1} \\ x_{i_2} \\ \vdots \\ x_{i_m} \end{array}}^{Prev} \quad \overbrace{\begin{array}{c} p_{i_1} \\ p_{i_2} \\ \vdots \\ p_{i_m} \end{array}}^{polys} \end{array} \right)$$

We have now a version of the MutantXL algorithm that is ready for our improvement. The final step of such improvement is based on enlarging  $H_{d-1}^+$  one *variable partition* at a time. Figure 4.2 explains such variable partitions of the echelonized  $H_{d-1}^+$ .

Our technique enlarges the set  $H_{d-1}^+$  explained in Figure 4.2 as follows. For  $j = n, n-1, \dots, 1$  (in that order), we construct the set  $H_{d,x_j}$  using the subset  $LV(H_{d-1}^+, x_j)$  such that

$$H_{d,x_j} = \{x \cdot p \mid p \in LV(H_{d-1}^+, x_j) \text{ and } x < prev(p)\}.$$

The idea behind this enlarging order is to reduce the number of highest degree terms appeared in the system, which leads to increasing the probability of generating mutants during the linear algebra step (*Echelonize*).

As in Figure 4.2, we start with enlarging the partition  $LV(H_{d-1}^+, x_k)$  since

$$x_k = \min\{x \mid LV(H_{d-1}^+, x) \neq \emptyset\}$$

We construct the set  $H_{d,x_k}$ , merge it to  $H_d$ , and compute a row echelon form of  $H_d$ . If no mutants are obtained, it constructs  $H_{d,x_{k-1}}$  and repeats the process with the partition of the next smallest variable, that satisfies  $LV(H_{d-1}^+, x) \neq \emptyset$ , until mutants are found. Otherwise, it enlarges the system to the next maximal degree  $d+1$ . We call this algorithm  $MXL_2$ . Algorithm 4.1 provides a detailed description of the  $MXL_2$  algorithm. We improved over the MutantXL algorithm that we described in Algorithm (3.4).

---

**Algorithm 4.1** MXL<sub>2</sub>

---

**Input:**  $P(x_1, \dots, x_n)$  is a quadratic finite set of polynomials in  $R$ .

**Output:**  $U$  a set  $n$  univariate polynomials in  $x_1, \dots, x_n$ .

```

1:  $d \leftarrow 2$  // The highest degree in  $H$ 
2:  $i \leftarrow n$  // The index of the leading variable
3:  $H \leftarrow \{(p_1, 1), \dots, (p_m, 1)\}$  // The set of generated polynomial pairs
4:  $L \leftarrow \emptyset$  // The set of leading terms of  $H$ 
5:  $U \leftarrow \emptyset$  // The set of univariate polynomials
6:  $M \leftarrow \emptyset$  // The set of mutants
7:  $SM \leftarrow \emptyset$  // The set of necessary mutants
8:  $NM \leftarrow 0$  // The number of necessary mutants
9: while  $|U| < n$  do
10:  $\tilde{H} \leftarrow \text{Echelonize}(H)$ 
11:  $U \leftarrow \{p \in \tilde{H} \mid p \text{ is univariate}\}$ 
12:  $M \leftarrow \{p \in \tilde{H} \mid \deg(p) < d \text{ and } \text{LT}(p) \notin L\}$ 
13:  $L \leftarrow \text{LT}(\tilde{H})$ 
14: if  $U \neq \emptyset$  then
15:  $H \leftarrow \text{Substitute}(\tilde{H}, U)$ 
16:  $U \leftarrow \emptyset$ 
17: else
18: if  $M \neq \emptyset$  then
19:  $k \leftarrow \min\{\deg(p) \mid p \in M\}$  // The lowest degree mutants
20:  $NM \leftarrow$  right hand side of (3.4)
21:  $SM \leftarrow \{p \in M \mid \deg(p) = k, |SM| < NM\}$ 
//  $NM$  is computed as in (3.4)
22:  $M \leftarrow M \setminus SM$ 
23:  $H \leftarrow \tilde{H} \cup \{(xp, x) \mid x \text{ is a variable, } p \in M \ \& \ \deg(p) = k\}$ 
24: else
25: if  $i = n$  then
26:  $d \leftarrow d + 1$ 
27: end if
28: while  $i \geq 1$  and  $\mathbb{L}(H_{d-1}^+, x_i) = \emptyset$  do
29:  $i \leftarrow i - 1$ 
30: end while
31: if  $i > 0$  then
32:  $H \leftarrow \tilde{H} \cup \{(yp, y) \mid (p, x) \in H \text{ and } p \in \mathbb{L}(H_{d-1}^+, x_i) \text{ and } y < x\}$ 
33:  $i \leftarrow i - 1$ 
34: else
35:  $i \leftarrow n$ 
36: end if
37: end if
38: end if
39: end while
40: return  $U$ 

```

---

## 4 Partial Enlargement Strategy

Now, we are going to give a more detailed description to some subroutines of the  $\text{MXL}_2$  algorithm. The *Echelonize* function in line 10 of Algorithm 4.1 computes the row echelon form of the polynomial system of  $H$ . It represents the system in a matrix and represents the previous variable multiplier in an array with dimension equal to the number of polynomials. The entries of this array are swapped together with the corresponding rows of the polynomial matrix. The entries of the array are not changed in terms of adding two rows. Let for example  $q_1 = xp_1$  and  $q_2 = yp_2$  and  $q_3 = q_1 + q_2$ , *Echelonize* replaces  $q_2$  with  $q_3$ . In this case  $q_3$  inherits the previous computations of both  $q_1$  and  $q_2$ . However,  $q_1$  still remains in the system while  $q_2$  is removed. Therefore,  $\text{prev}(q_3) = \text{prev}(q_2)$ .

Lines 18-24 of Algorithm 4.1 enlarge the lower degree mutants. We use the concept of the necessary mutants  $NM$  (line 20) that are computed by (3.4). As we explained in the previous chapter, mutants generated at a maximal degree  $d$  are polynomials belongs to  $\text{span}(H_d) \setminus \text{span}(H_{d-1})$ . Then  $\text{MXL}_2$  deals with mutants as well as with the original polynomials. More precisely, we reset the previous variable multiplier to be one.

Let  $d_m$  be the degree at which  $\text{MXL}_2$  finds a sufficient number of mutants to terminate the process successfully. Since, as we explained in the previous chapter, mutants are elements in the span of  $H_{d_m}^+$ , then the obtained polynomials from enlarging mutants are elements in the span of  $H_{d_m+1}^+$ .  $\text{MXL}_2$  as well as *MutantXL* does not generate the entire  $H_{d_m+1}^+$ . It generates only those elements obtained from mutants.

Lines 25-37 of Algorithm 4.1 describe the main part of our improvement. Firstly,  $\text{MXL}_2$  selects the variable partition  $\mathbb{L}(H_{d-1}^+, x_i)$  that should be enlarged. The order of this selection is defined as we explained in the previous section. For each pair  $(p, x) \in H$  such that  $p \in \mathbb{L}(H_{d-1}^+, x_i)$ , it constructs a set of pairs  $(yp, y)$ ,  $y < x$  and includes these pairs to  $H$ .

We show that the system is partially enlarged, so  $\text{MXL}_2$  leads to the original *MutantXL* if the system is solved after the last partition is enlarged. Whereas  $\text{MXL}_2$  outperforms the original *MutantXL* if it solves after enlarging the system with avoiding at least one partition. Therefore,  $\text{MXL}_2$  terminates and returns the solution using smaller number of terms and polynomials than *MutantXL*, however, at the same degree. This will be clarified experimentally in the next section.

### 4.3 Implementation and experimental results

This section presents the  $\text{MXL}_2$  experimental results. The experiments are based on dense random systems with sizes (16,...,30) and some HFE systems with univariate degree 288 and sizes (16,...,30). We experimentally explain the steps of the improvements and compare the performance of  $\text{MXL}_2$  with *Magma*'s implementation of  $F_4$ .

We implemented the  $\text{MXL}_2$  algorithm in C++. As the *MutantXL* implementation we use the dense matrix representation to the system of polynomials. For the multi-

## 4 Partial Enlargement Strategy

plication step and the linear algebra step, we use the same method that we used in the implementation of MutantXL. We use a one dimension array *var-part* to store the row index of the last polynomials that has leading variables  $x \in \{x_1, \dots, x_n\}$  of degree  $d \geq 1$ . In this case, the first  $n$  entries of *var-part* store the variable partitions that have degree 1, the second  $n$  are for degree 2, and so on. For example, *var-part*[ $3n + 5$ ] stores the index of the last degree 4 polynomial that has leading variable  $x_5$ . The *var-part* array is very important to perform the partial enlargement technique explained in the previous sections. We use it to identify the enlarged partition. In case we have no polynomials of degree  $d$  that have leading variable  $x_i$ , then *var-part*[( $d - 1$ ) $n + i$ ] = -1. The entries of *prev-var* are updated after each echelonization step.

Another important one dimensional array used in the multiplication process is *prev-var*. The *prev-var* array is used to keep the previous variable multiplier. Let for example  $q = x \cdot p$ , then *prev-var*[ $q$ ] =  $x$ . The *prev-var* array is given to the echelonization process as an input. During the echelonization process the entries of *prev-var* are swapped in parallel with the corresponding rows in the matrix that represent the polynomial system. In case of we have obtained some mutants after the echelonization process we reset its corresponding entries in *prev-var* to -1 as the original polynomials.

When we enlarge the system, we estimate the number of polynomials generated after the multiplication. We extend the *prev-var* array by entries equal to the expected generated polynomials. We have two cases:

- Some mutants obtained, we select necessary number of mutants and determine the indexes of mutants rows in the system matrix. Each mutant is multiplied by all variables, adding a new row to the matrix for each new polynomials and set the corresponding entry in *prev-var* to the multiplied variable.
- No mutants obtained, in this case we determine the partition using the static variable  $i$  and use *var-part* to locate the first and the last indexes of the partition polynomials in both the system matrix and *prev-var*. Then we multiply each row  $r$  with all variables  $\leq$  *prev-var*[ $r$ ], as explained in the previous section.

Now, we present the experimental results that demonstrate the effectiveness of the proposed partial enlargement strategy and show how it improves the both XL and MutantXL algorithms. Also, we compare the performance of our MXL<sub>2</sub> implementation to the Magma's implementation of  $F_4$  (version V2.13-10). We run all the experiments on a Sun X4440 server, with four "Quad-Core AMD Opteron<sup>TM</sup> Processor 8356" CPUs and 128 GB of main memory. Each CPU is running at 2.3 GHz. We used only one out of the 16 cores.

Table 4.3 compares the maximum matrix size obtained by XL, XLP, MutantXL, and MXL<sub>2</sub> algorithms. MutantXL uses the mutant strategy to improve XL as explained in the previous chapter, while XLP uses the partial enlargement strategy explained in this chapter. The idea of this comparison is to shed light on which strategy is most effective in improving XL and to explain the effect of combining both

#### 4 Partial Enlargement Strategy

$n$	XL	XLP	MutantXL	MXL <sub>2</sub>
16	9447×6885	2667×2744	9447×6885	<b>2308×2675</b>
17	12120×9402	3984×4493	12120×9402	<b>3187×3831</b>
18	15345×12616	7481×6898	15345×12616	<b>5046×5332</b>
19	19181×16664	12280×9404	19181×16664	<b>7932×9359</b>
20	23722×21700	14540×12384	23722×21700	<b>14540×12384</b>
21	94857×82160	28000×27924	28970×27896	<b>17226×16115</b>
22	129823×110056	35663×35653	35343×35443	<b>20617×22629</b>
23	177694×145499	46061×45944	44252×44552	<b>26075×26407</b>
24	231499×190051	59737×60627	231499×190051	<b>52205×57171</b>
25	287875×245506	91347×88670	287875×245506	<b>70969×88235</b>
26	355082×313912	126530×128766	355082×313912	<b>97053×126190</b>

Table 4.3: Compare the effectiveness of mutant and partial enlargement strategies to improve XL and the impacts of combining both strategies in MXL<sub>2</sub>.

on the improvement. We use random systems of sizes  $(16, \dots, 26)$  for these experiments. We see that XLP constructs a smaller system in all cases when MutantXL has found no mutants  $(16, \dots, 20, 24, 25, 26)$ . In these cases the mutant strategy does not work. In the cases that MutantXL finds some mutants  $(21, 22, 23)$ , MutantXL is better than XLP. However, the difference is not so much since XLP solves at the same degree as XL with avoiding many parts during the enlargement step. Indeed, by combining the two strategies in one algorithm (MXL<sub>2</sub>) we do not only improve over XL but also over both MutantXL and XLP. MXL<sub>2</sub> solves with smaller number of polynomials and smaller number of terms in all the cases reported in Table 4.3.

Tables 4.4 and 4.5 explain how each strategy enriches the performance of the other one. Firstly, we use the partial enlargement strategy to improve XL (XLP) and then we use the mutant strategy to improve XLP (MXL<sub>2</sub>). For this comparison we report the maximal degree of polynomials generated by both algorithms ( $d$ ) and the leading variable of the last enlarged partition (GLV). To figure out the effects of using mutant strategy for XLP, we report the number of terms ( $T$ ) and the number of linearly independent polynomials (polys) generated by both algorithms with exactly degree equal to the maximal degree of XLP. It is clear that using mutants in the process of XLP (MXL<sub>2</sub>) enhances the partial enlargement strategy not only decrease the maximal degree  $d$  (cases: 21, 22, and 23) but also decrease the number of terms  $T$  and the number of linearly independent polynomials (cases: 16, 17, 18, 19, 24, 25, 26, and 27). Only two cases when  $n = 20, 28$  XLP and MXL<sub>2</sub> solves at the same highest degree and with the same number of polynomials. In all other cases that XLP and MXL<sub>2</sub> solves at the same degree level, MXL<sub>2</sub> solves with less number of parturitions than XLP.

As another point of view, assume we improve XL using mutants (MutantXL) and then improving MutnatXL using the partial enlargement strategy (MXL<sub>2</sub>). Since MutantXL and MXL<sub>2</sub> have the same maximal degree then in Table 4.5, we com-

#### 4 Partial Enlargement Strategy

$n$	XLP				MXL <sub>2</sub>			
	$d$	GLV	$T$	Polys	$d$	GLV	$T$	Polys
16	5	$x_8$	227	129	5	$x_9$	158	59
17	5	$x_6$	1279	774	5	$x_7$	617	460
18	5	$x_5$	2850	2036	5	$x_6$	1284	1280
19	5	$x_4$	4368	4368	5	$x_5$	4323	3036
20	5	$x_4$	6188	6188	5	$x_4$	6188	6188
21	6	$x_{14}$	28	28	5	$x_4$	0	0
22	6	$x_{13}$	210	210	5	$x_4$	0	0
23	6	$x_{13}$	1392	1018	5	$x_4$	0	0
24	6	$x_{11}$	5172	3006	6	$x_{12}$	3003	1874
25	6	$x_9$	20264	12380	6	$x_{10}$	19829	8190
26	6	$x_8$	45084	27126	6	$x_9$	42508	18572
27	6	$x_7$	82780	54366	6	$x_8$	81725	39357
28	6	$x_7$	116931	74818	6	$x_7$	116931	74818

Table 4.4: Compare the steps of the MXL<sub>2</sub> improvements, the mutant strategy over the partial enlargement strategy.

pare only the number of terms and the number of linearly independent polynomials generated by both algorithms that have degree equal to the maximal degree  $d$ . It is clear that MXL<sub>2</sub> generates less number of terms and polynomials than MutantXL in all cases that are reported in Table 4.5.

Now, we are going to compare the performance of our MXL<sub>2</sub> implementation to the Magma's implementation of  $F_4$ . In all experiments, the highest degree of the polynomials generated by MXL<sub>2</sub> as well as MutantXL is equal to the highest degree of S-polynomials generated by  $F_4$  implemented in Magma. As explained above, for MXL<sub>2</sub> implementation we use only one matrix to represent the system of polynomials during the process. When the system is enlarged, we extend the matrix horizontally as the number of terms increases and vertically as the number of polynomials increases. So, the largest matrix is an accumulative of the whole system of polynomials that are held in the memory. Unfortunately, in Magma we can not know the total accumulative matrices size because it is not an open source.

In Tables 4.6 and 4.7, we compare the maximum matrix size generated by MXL<sub>2</sub> to the matrix generated by  $F_4$  for solving random systems and HEF-288 systems, respectively. As we see from both tables, MXL<sub>2</sub> generates fewer polynomials compared to  $F_4$  in all cases reported in the two tables. Moreover, except for the two HFE-288 systems ( $n = 22, n = 28$ ), the number of terms appeared in the generated polynomials by MXL<sub>2</sub> is smaller than what are generated by  $F_4$ . This explains how our partial enlargement strategy enriched the performance of MutantXL.

Figure 4.3 compares the performance of MXL<sub>2</sub>, MutantXL, and Magma's implementation of  $F_4$  in terms of time and memory requires to solve random and HFE-288 systems. As we see from Figures 4.3(a) and 4.3(c), the combination of our improved

#### 4 Partial Enlargement Strategy

$n$	$d$	MutantXL		MXL <sub>2</sub>	
		$T$	Polys	$T$	Polys
16	5	4368	4368	158	59
17	5	6188	6188	617	460
18	5	8568	8568	1284	1280
19	5	11628	11628	4323	3036
20	5	15504	15504	6188	6188
21	5	20349	20349	8568	8568
22	5	26334	26334	13520	11630
23	5	33649	33649	15504	15504
24	6	134596	134596	3003	1874
25	6	177100	177100	19829	8190
26	6	230230	230230	42508	18572
27	6	296010	296010	81725	39357
27	6	376740	376740	116931	74818

Table 4.5: Compare the steps of the MXL<sub>2</sub> improvements, the partial enlargement strategy over the mutant strategy.

$n$	F <sub>4</sub>	MXL <sub>2</sub>
16	9995×4036	<b>2308×2675</b>
17	12415×5817	<b>3187×3831</b>
18	15187×8120	<b>5046×5332</b>
19	18441×11041	<b>7932×9359</b>
20	22441×14979	<b>14540×12384</b>
21	26860×19756	<b>17226×16115</b>
22	63621×21855	<b>20617×22629</b>
23	41866×29010	<b>26075×26407</b>
24	207150×78637	<b>52205×57171</b>
25	248495×108746	<b>70969×88235</b>
26	298592×148804	<b>97053×126190</b>
27	354189×197902	<b>135255×183309</b>
28	420773×261160	<b>197928×239369</b>
29	95169×112657	<b>290887×621616</b>
30	108233×136105	<b>346422×768212</b>

Table 4.6: Compare the efficiency of our MXL<sub>2</sub> implementation to the Magma's implementation of F<sub>4</sub> on random systems.

#### 4 Partial Enlargement Strategy

$n$	$F_4$	$MXL_2$
16	9995×4036	<b>2285×2573</b>
17	12415×5817	<b>3173×3992</b>
18	15134×8067	<b>4722×5332</b>
19	18458×11058	<b>7864×9353</b>
20	22513×15051	<b>9724×12383</b>
21	27018×19914	<b>17508×16999</b>
22	63758×21992	<b>20617×35443</b>
23	41963×29107	<b>26075×26407</b>
24	46593×37620	<b>32069×36577</b>
25	54028×47734	<b>40458×41610</b>
26	62446×59925	<b>49383×56735</b>
27	72722×75270	<b>69850×73978</b>
28	83467×92603	<b>79221×122438</b>
29	95169×119722	<b>89459×108571</b>
30	108233×136105	<b>100534×130211</b>

Table 4.7: Compare the efficiency of our  $MXL_2$  implementation to the Magma's implementation of  $F_4$  on HFE-288 systems.

strategies reduce significantly the memory requirements to solve multivariate systems compared to  $F_4$ . In terms of time, Figures 4.3(b) and 4.3(d) explain that  $MXL_2$  solves faster than MutantXL while Magma's  $F_4$  is still faster than  $MXL_2$ . This is due to the zero columns that still represented by  $MXL_2$ . This problem is solved with the improved version of  $MXL_2$  implementation as we will see in Chapter 6.

## 4 Partial Enlargement Strategy

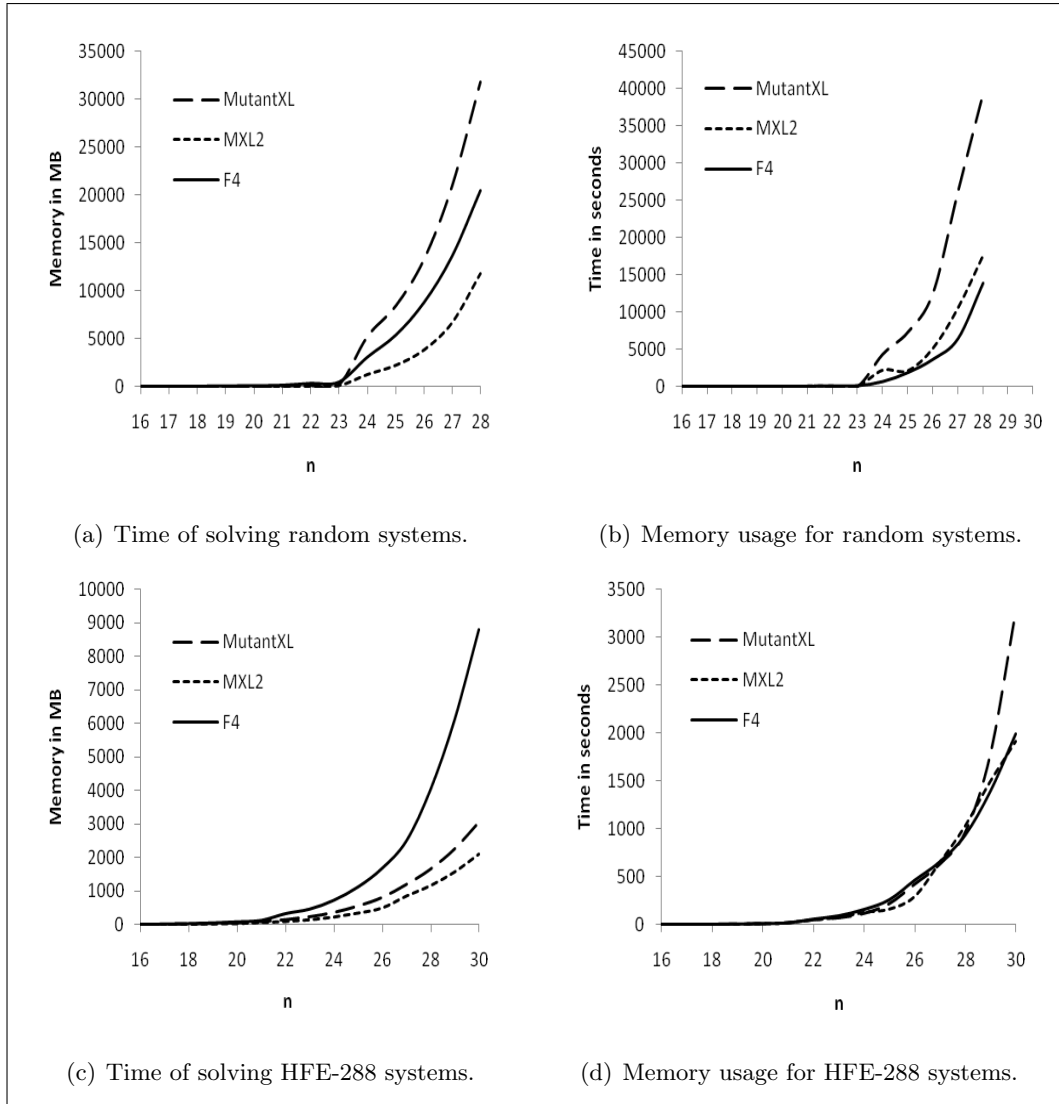


Figure 4.3: Compare the performance of  $MXL_2$  to MutantXL and  $F_4$ .

## 5 Algebraic cryptanalysis of MQQ

In this chapter, we present an efficient algebraic attack on the multivariate Quadratic Quasigroups (MQQ) public key cryptosystem. Our cryptanalysis breaks the MQQ cryptosystem by solving a system of multivariate quadratic polynomial equations. For this attack, we use the MutantXL algorithm and the  $F_4$  algorithm. We explain the interpretation of the efficiency of the MutantXL attack and we detail our experiments. Based on the obtained results, we show that MutantXL solves MQQ systems with much less memory than Magma's implementation of the  $F_4$  algorithm.

### 5.1 MQQ cryptosystems

This section explains the MQQ cryptosystem public key. The MQQ public-key cryptosystem is a standard multivariate public key cryptosystem that is constructed using quasigroup string transformation performed on a class of quasigroups. The security parameter is a positive integer  $n$  which is the number of variables and polynomials used in the public key. The authors of MQQ proposed the length of  $n \geq 140$  for a conjectured security level of  $2^{\frac{n}{2}}$ . In this Section we present an overview of the MQQ cryptosystem. A more detailed explanation is found in [31, 30].

**Definition 5.1.** *Let  $Q = \{a_1, \dots, a_n\}$  be a finite set of  $n$  elements. A quasigroup  $(Q, *)$  is a groupoid satisfying the law*

$$(\forall a, b \in Q)(\exists x, y \in Q)(a * x = b \quad \wedge \quad y * a = b) \quad (5.1)$$

The unique solutions to these equations are written  $x = a \backslash_* b$  and  $y = b /_* a$  where  $\backslash_*$  and  $/_*$  are called a left parastrophe and a right parastrophe of  $*$  respectively. The basic quasigroup string transformation, called e-transformation is defined as follows [31]:

**Definition 5.2.** *A quasigroup e-transformation of a string  $S = (s_0, \dots, s_{k-1}) \in Q^k$  with a leader  $l \in Q$  is the function  $e_l : Q \times Q^k \rightarrow Q^k$  defined as  $T = e_l(S)$ ,  $T = (t_0, \dots, t_{k-1})$  such that*

$$t_i = \begin{cases} l * s_0 & i = 0 \\ t_{i-1} * s_i & 1 \leq i \leq k-1 \end{cases} \quad (5.2)$$

Consider the case where each element  $a \in Q$  has a unique  $d$ -bit representation  $x_1, \dots, x_d \in \{0, 1\}$  such that  $a = x_1 x_2 \dots x_d$ . The binary operation  $*$  of the finite quasigroups  $(Q, *)$  is equivalent to a vector valued operation  $*_{vv} : \{0, 1\}^{2d} \rightarrow \{0, 1\}^d$

## 5 Algebraic cryptanalysis of MQQ

defined as:

$$a * b = c \Leftrightarrow *_{vv}(x_1, \dots, x_d, y_1, \dots, y_d) = (z_1, \dots, z_d)$$

where  $x_1 \dots x_d$ ,  $y_1 \dots y_d$ , and  $z_1 \dots z_d$  are the binary representations of  $a$ ,  $b$ , and  $c$  respectively.

**Lemma 5.3.** *For every quasigroup  $(Q, *)$  of order  $2^d$  and for each  $d$ -bit representation of  $Q$  there is a unique vector valued operation  $*_{vv}$  and  $d$  uniquely determined arrays of length  $2d$  of boolean functions  $f_1, \dots, f_d$  such that  $\forall a, b, c \in Q$*

$$a * b = c \Leftrightarrow *_{vv}(X^d, Y^d) = (f_1(X^d, Y^d), \dots, f_d(X^d, Y^d))$$

where  $X^d = x_1, \dots, x_d$ ,  $Y^d = y_1, \dots, y_d$ .

Each  $k$  – bit boolean function  $f(x_1, \dots, x_k)$  has the following algebraic normal form (ANF):

$$ANF(f) = c_0 + \sum_{1 \leq i \leq k} c_i x_i + \sum_{1 \leq i \leq j \leq k} c_{i,j} x_i x_j + \dots, \quad (5.3)$$

where  $c_0, c_i, c_{i,j}, \dots \in \{0, 1\}$ . The degrees of the boolean functions  $f_i$  are one of the complexity factors of the quasigroup  $(Q, *)$ .

**Definition 5.4.** *A quasigroup  $(Q, *)$  of order  $2^d$  is called multivariate quadratic quasigroup (MQQ) of type  $Quad_{d-k}Lin_k$  if exactly  $d - k$  of the polynomials  $f_i$  are quadratic and  $k$  of them are linear, where  $0 \leq k \leq d$ .*

The authors of [31] provide a heuristic algorithm to generate MQQs of order  $2^d$  and of type  $Quad_{d-k}Lin_k$ . The public and the private keys are constructed as follows.

A system  $P' = (p_1, \dots, p_n)$  of quadratic polynomials over  $\mathbb{F}_2$  in  $n$  variables is generated using uniformly and randomly selected quasigroups  $*_1, \dots, *_8$  as described in Table 5.1. That system represents a map  $P' : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ . The matrices  $T, S \in \mathbb{F}_2^{(n,n)}$  are selected uniformly at random. The public key is the map

$$P = T \circ P' \circ S$$

which can also be represented by  $n$  quadratic polynomials in  $n$  variables over  $\mathbb{F}_2$ . The secret key consists of the 10-tuple  $(T, S, *_1, \dots, *_8)$ .

The plaintext space is  $\mathbb{F}_2^n$ . A plaintext  $x = (x_1, \dots, x_n) \in \mathbb{F}_2^n$  is encrypted by computing

$$c = P(x)$$

Decrypting means solving the multivariate system  $P(x) = c$ . If the secret key is known then we can decrypt as follows

$$x = S^{-1}P'^{-1}T^{-1}(c)$$

## 5 Algebraic cryptanalysis of MQQ

where  $P'^{-1}$  is computed by using the left parastrophes  $\backslash_*$  of the quasigroups  $*_1, \dots, *_8$ .

The parameter that was suggested for the practical applications of the MQQ scheme is  $n(= 140, 160, 180, 200, \dots)$ , where  $n$  is the bit length of the encrypted block.

Table 5.1: Definition of the the nonlinear mapping  $P'$

Input: Integer $n$ , where $n = 5k, k \geq 28$
Output: Eight quasigroups $*_1, \dots, *_8$ and $n$ multivariate quadratic polynomials $P'$
<ol style="list-style-type: none"> <li>1. Randomly generate <math>n</math> Boolean functions <math>L = (f_1, \dots, f_n)</math> of <math>n</math> variables <math>x = (x_1, \dots, x_n)</math>;</li> <li>2. Represent a vector <math>L</math> as a string <math>L = X_1 \dots X_k</math>, where <math>X_i</math> are vectors of dimension 5;</li> <li>3. Generate several MQQs of type <math>Quad_4Lin_1</math> and <math>Quad_5Lin_0</math>; The algorithm of generating MQQs is described in [31, 30] Table 2.</li> <li>4. Randomly choose <math>*_1, *_2 \in Quad_4Lin_1</math> and <math>*_3, *_4, *_5, *_6, *_7, *_8 \in Quad_5Lin_0</math>;</li> <li>5. Define a <math>(k-1)</math>-tuple <math>I = (i_1, \dots, i_{k-1})</math> where <math>i_j \in \{1, \dots, 8\}</math> such that, 1, 2 are repeated 8 times in <math>I</math>, without loss of generality let <math>i_1, \dots, i_8 \in \{1, 2\}</math>.</li> <li>6. Compute <math>y = Y_1 \dots Y_k</math> where <math>Y_1 = X_1, Y_{j+1} = X_j *_i X_{j+1}</math>, for <math>j = 1, 2, \dots, k-1</math>;</li> <li>7. Set a vector <math>Z = Y_1    Y_{2,1}    Y_{3,1}    \dots    Y_{8,1}</math> that has 13 components as linear Boolean functions, where <math>Y_{j,1}</math> means the first coordinate of the vector <math>Y_j</math>;</li> <li>8. Transform <math>Z</math> by the bijection Dobbertin [23]: <math>W = Dob(Z)</math>;</li> <li>9. Set <math>Y_1 = (W_1, W_2, W_3, W_4, W_5), Y_{2,1} = W_6, \dots, Y_{8,1} = W_{13}</math>;</li> <li>10. Return <math>y</math> as <math>n</math> multivariate quadratic polynomials <math>P' = \{p'(x_1, \dots, x_n), i = 1, \dots, n\}</math> and the eight Quasigroups <math>*_1, \dots, *_8</math>.</li> </ol>

### 5.2 MQQ cryptanalysis

We performed several experiments to attack MQQ systems built using the algorithm described in the previous section. These systems come from decrypting ciphertexts using the public key but not the secret key. In other words, assume a ciphertext  $c$ , decrypting  $c$  using the public key  $P = (p_1, \dots, p_m)$  is equivalent to solving the system  $P(x) = c$ . The MQQ inventors supplied us with a few MQQ systems that are not sufficient for the analysis. However, we used them to confirm our implementation of the MQQ cryptosystem. We generated some systems for  $n = 60, 80, \dots, 300$  that were created using MQQs of type  $Quad_4Lin_1$  and  $Quad_5Lin_0$  as in Table 5.1. According to [31], these correspond to 30, 40,  $\dots$ , 150 bits of security.

We start with briefly describing the version of MutantXL that we used in our attack. We explain how and why we adapted MutantXL to MQQ. As we will see later in this section, MQQ systems produce some mutants of degree one. After using these mutants, MutantXL generates many new quadratic polynomials that are included to the system. In the next round, MutantXL enlarges these polynomials and generates a very large system. Many polynomials in this system are reduced to zero. However, enlarging only the original quadratic polynomials is not sufficient

for solving some MQQ systems. So, we adapt MutantXL to overcome this problem. We use the enlargement strategies of MutantXL and  $MXL_2$  algorithms as follows.

To enlarge the system, firstly we multiply the original polynomials or the polynomials produced by the originals. If the system can not be solved, the algorithm extends the polynomials obtained from the mutants using the partial enlargement technique that we explained in the previous chapter.

Our experiment setup has a Sun X4440 server, with four "Quad-Core AMD Opteron™ Processor 8356" CPUs and 128 GB of main memory. Each CPU is running at 2,3 GHz. MutantXL code at the moment uses only one out of the 16 cores. We used both our MutantXL variant and the Magma's implementation of the  $F_4$  algorithm (version V2.13-10).

Table 5.2 shows the results of our attacks. There we list the number  $n$  of variables and equations, the maximum required memory in Megabytes, the maximum matrix size, and the executed time in seconds. It is clear from Table 5.2 that all systems up to  $n = 300$  were successfully attacked by MutantXL as well as Magma's implementation of  $F_4$ .

For confirming our results, we applied our MutantXL implementation and Magma's implementation to three MQQ systems with 135, 150 and 160 variables as suggested by the author of the MQQ scheme. These systems are generated from attempts to improve MQQ. Table 5.3 shows the results of solving these three systems. As we see from the results, the maximum matrix generated by both MutantXL and  $F_4$  is larger than what is generated by the standard MQQ (explained in Table 5.1) as in Table 5.2. However, they are solved at the same degree bound 3.

Figure 5.1(a) compares the maximum number of polynomials used in case of MutantXL and Magma's  $F_4$ . We noticed from it that the MutantXL algorithm solves the MQQ systems with smaller number of polynomial equations than Magma's  $F_4$ . Conversely, Figure 5.1(b) shows that the number of monomials of Magma's  $F_4$  is smaller than MutantXL. This is due to the special selection strategy used by the  $F_4$  algorithm, while MutantXL multiplies polynomials of the initial system by all monomials up to certain degree  $D$ . For the MQQ systems, all the quadratic monomials appear in the initial system. In this case, all the monomials up to degree  $D$  will appear in the enlarged system.

Tables 5.4 and 5.5 show the steps of solving an MQQ system for  $n = 200$  using MutantXL and Magma's  $F_4$ , respectively. In Table 5.4, for each step we show the elimination degree (d), the matrix size, the rank of the matrix (Rank), the number of mutants found (NM), and the memory required in Megabyte (MB). In Table 5.5 we show, for each step, the step degree (SD), the number of pairs (NP), the matrix size, and the step memory in MB.

From Table 5.4 we see that MutantXL can easily solve the 200 variables MQQ system. In the first iteration of the algorithm, the *Eliminate* step created 37 mutant polynomial equations of degree 1. In the multiply step, 6697 linearly independent quadratic equations were generated from these 37 mutants. The resulting equations were then appended to the 163 quadratic polynomial equations produced by eliminating the original system. In the second iteration, no mutants were found.

5 Algebraic cryptanalysis of MQQ

$n$	MutantXL			$F_4$		
	Memory MB	Max Matrix	Time in in sec.	Memory MB	Max Matrix	Time in in sec.
60	1.6	$3714 \times 3605$	8	88.8	$18835 \times 35918$	4
80	70.1	$6830 \times 85401$	23	217.5	$33267 \times 32863$	10
100	212.7	$10649 \times 166751$	76	538	$63258 \times 62697$	55
120	498.2	$14387 \times 288101$	283	1819	$149077 \times 148234$	298
140	1109	$20192 \times 457451$	556	2909	$200397 \times 199391$	873
160	2281	$26937 \times 682801$	1283	4364	$262244 \times 261130$	1366
200	6437	$39497 \times 1333501$	16694	18198	$699138 \times 697280$	17186
300	47952	$88647 \times 4500251$	237362	111160	$2339710 \times 2336171$	387754

Table 5.2: Performance of MutantXL versus  $F_4$

$n$	MutantXL		$F_4$	
	Memory	Max Matrix	Memory	Max Matrix
135	2341	$47721 \times 410176$	18666	$395550 \times 392262$
150	3483	$51469 \times 562626$	24484	$501565 \times 497886$
160	12955	$157933 \times 682801$	44241	$556942 \times 551029$

Table 5.3: Performance of MutantXL versus  $F_4$

Step	D	Matrix Size	Rank	NM
1	2	$200 \times 20101$	200	37
2	2	$7600 \times 20101$	6897	0
3	3	$39497 \times 1333501$	39497	130
4	2	$7427 \times 20101$	7347	5
5	2	$8347 \times 20101$	8125	3
6	2	$8725 \times 20101$	8584	4
7	2	$9384 \times 20101$	9183	4
...	...	...	...	...
42	2	$20874 \times 20101$	20094	4

Table 5.4: Steps of solving MQQ-200 by MutantXL

## 5 Algebraic cryptanalysis of MQQ

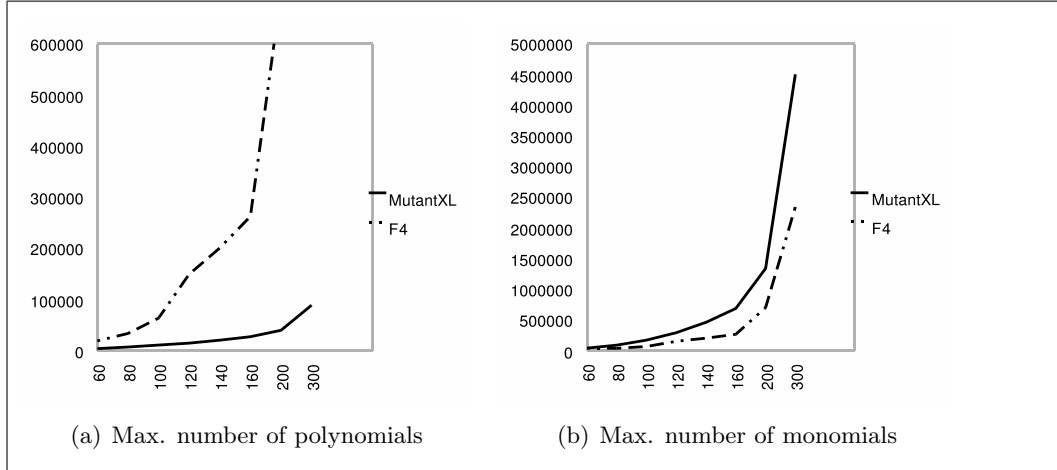


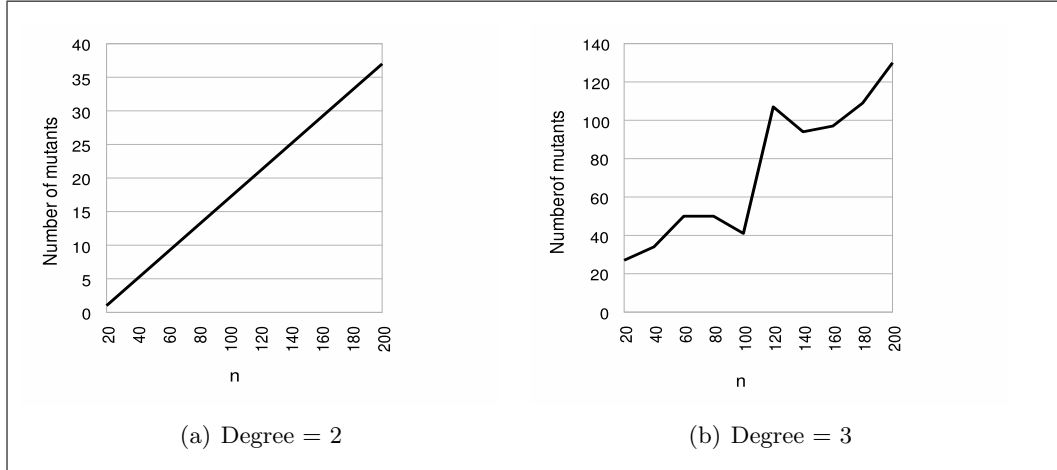
Figure 5.1: Comparison between MutantXL and  $F_4$  for MQQ

Step	SD	NP	Matrix Size
1	2	192	$200 \times 20101$
2	2	163	$6897 \times 20101$
3	3	4640	$26732 \times 721928$
4	2	84	$914 \times 13366$
5	2	182	$1948 \times 13366$
6	2	130	$2596 \times 13233$
7	2	148	$3409 \times 13189$
...	...	...	...
42	3	58891	$699138 \times 697280$

Table 5.5: Steps of solving MQQ-200 by Magma- $F_4$ .

Therefore, MutantXL extended the system by multiplying only the 163 quadratic equations producing 32600 cubic equations. In the third iteration, MutantXL eliminated the extended system thus generating 128 quadratic mutants and two linear mutants. Further iteration steps continuously generate linear mutants as shown in Table 5.4 until some of these mutants are univariate which finally leads to solving the system.

It was indeed observed that all MQQ systems offer enough algebraic information (in the sense that it finds enough mutants) to the MutantXL algorithm such that it was always able to solve the system having a critical degree of 3. Figure 5.2(a) shows the direct linear relation between the size of the initial system  $n$  and the number of linear mutants obtained from it. On the other hand Figure 5.2(b) shows the relation between  $n$  and the number of mutants obtained from the extended degree 3 system. Both figures point out two main drawbacks of the MQQ system: First, the initial systems contains linear equations, which is easily discovered in the first

Figure 5.2: Relation between  $n$  and the number of mutants obtained

step of MutantXL. Second, at degree 3, the system keeps producing mutants until the system is solved. This explains why MQQ systems are solved at degree 3 and therefore can be easily defeated.

We are going to estimate the security level of MQQ against attacks using MutantXL. The MQQ systems that we have broken are solved by MutantXL and  $F_4$  at degree  $D = 3$  as explained above. The author of MQQ together with Faugère et al in [27] confirmed our results and explained our attack from the point of view of Gröbner bases computation. They explained that the degree of regularity of MQQ  $d_{reg}$  is equal 3 which is the maximal degree of MutantXL computations. For MutantXL the memory resources are basically measured by the matrix size. From the linear relation explained in Figure 5.2(a), we can easily estimate the number of linear mutants obtained in the first step. This will enable us to calculate the maximum number of polynomials. For MutantXL, all the terms up to degree 3 are appeared. In this context, MutantXL can attack MQQ cryptosystems up to  $n = 365$  using the same memory resources of the architecture specified above (128 GB). Therefore, to be secure,  $n$  should be greater than 370. This affects on the performance of MQQ cryptosystem. So, we claim that MQQ specified in Table 5.1 is completely broken.

MQQ represents a good example that explains the differences of the behavior of MutantXL and The  $F_4$  implemented in Magma. Both algorithms generated linearly independent degree one polynomials (linear mutants) in the first echelonization step. They enlarge those polynomials in the next step which leads to generating many polynomials of degree 2. In this case we can divide the degree 2 polynomials into two groups as explained above. The first group is the set of remaining quadratic polynomials from the original system and the other group for the polynomials obtained from enlarging the linear mutants. MutantXL deals with the two groups differently, it enlarges the first group first and try to solve. In the case of MQQ, enlarging the first group is sufficient to solve the system. However,  $F_4$  deals with all

## 5 Algebraic cryptanalysis of MQQ

quadratic polynomials blindly and generates a list of pairs based of s-polynomials. In this case,  $F_4$  enlarges all pairs  $((p, t_1), (q, t_2))$  such that  $\deg(\text{LCM}(p, q)) = 3$  which leads to constructing a large number of degree 3 polynomials. This explains the results we have presented.

## 6 Computing Gröbner Basis

In this Chapter, we present a variant of the MutantXL algorithm for computing Gröbner bases of zero-dimensional ideals. We call it  $\text{MXL}_3$  [37]. The  $\text{MXL}_3$  algorithm is based on the XL algorithm, mutant strategy, and a new sufficient condition for a set of polynomials to be a Gröbner basis. We propose and prove a new termination criteria for  $\text{MXL}_3$  which yields a Gröbner basis of the ideal generated by the input set of polynomials. We describe the  $\text{MXL}_3$  algorithm, prove its correctness, and present experimental results that compare the performance of  $\text{MXL}_3$  to Magma's implementation of the  $F_4$  algorithm. For this comparison, we use some HFE and random systems. The experiments show that  $\text{MXL}_3$  can solve HFE systems that have a univariate degree 288 and number of variables up to 49, while  $F_4$  can not solve any system with more than 39 variables under the same memory conditions. Moreover, we show that  $\text{MXL}_3$  solves all systems faster and with less memory than  $F_4$ . Finally, we present experimental results towards algebraic cryptanalysis of HFE challenge 2. In Section 6.4, we describe and discuss experiments of solving some scaled versions of HFE challenge 2 using  $\text{MXL}_3$ .

### 6.1 The $\text{MXL}_3$ criterion

This section states and proves a new checkable condition to test whether a set of polynomials is a Gröbner basis. In addition to the notations introduced in Chapter 2 and Chapter 4, we need the following notations. Given a subset  $P$  of the polynomial ring  $R$  (as defined in (2.2)) over the finite field  $\mathbb{F}$ , we denote by  $\langle P \rangle$  the ideal generated by  $P$ . We denote by  $\text{span}_{\mathbb{F}}(P)$  the  $\mathbb{F}$ -linear span of  $P$ . Let the graded lexicographic ordering  $<_{\text{glex}}$  be used for ordering the elements of  $R$ . We will denote by  $P_{(op)d}$  the subset of all the polynomials of degree  $(op)d$  in  $P$ , where  $(op)$  is any of  $\{=, <, >, \leq, \geq\}$ .

**Definition 6.1.** *Let  $P$  be a finite subset of  $R$ ,  $0 \neq f \in \langle P \rangle$  and  $t \in T$ . A representation*

$$f = \sum_{i=1}^s a_i t_i p_i$$

*with  $a_i \in \mathbb{F}$ ,  $t_i \in T$ , and  $p_i \in P$  is called a  $t$ -representation of  $f$  w.r.t.  $P$  (and  $\leq$ ) if  $\text{LT}(t_i p_i) \leq t$  for  $i = 1, \dots, s$ . An  $\text{LT}(f)$ -representation of  $f$  w.r.t.  $P$  is called a standard representation.*

**Proposition 6.2.** [6] *Let  $G$  be a finite subset of  $R$  with  $0 \notin G$ , and assume that for all  $g_1, g_2 \in G$ ,  $\text{spol}(g_1, g_2)$  equals zero or has a standard representation w.r.t.  $G$ .*

## 6 Computing Gröbner Basis

Then  $G$  is a Gröbner basis.

We recall a result commonly known as Buchberger's second criterion. We paraphrase it in the following proposition.

**Proposition 6.3.** [9, 6] *Let  $P$  be a finite subset of  $R$  and  $g_1, f, g_2 \in R$  be such that  $\text{LT}(f) \mid \text{lcm}(\text{LT}(g_1), \text{LT}(g_2))$ , and for  $i = 1, 2$   $\text{spol}(g_i, f)$  has a standard representation w.r.t.  $P$ , then  $\text{spol}(g_1, g_2)$  also has a standard representation w.r.t.  $P$ .*

Now we present our new result that establishes a sufficient condition for a finite set to be a Gröbner basis.

**Proposition 6.4.** *Let  $G$  be a finite subset of  $R$  with  $D$  being the highest degree of its elements. Let  $<$  be an order on  $R$ . Suppose that the following holds:*

1.  $G$  contains all the terms of degree  $D$  as leading terms; and
2. if  $H := G \cup \{t \cdot g \mid g \in G, t \text{ a term and } \deg(t \cdot g) \leq D + 1\}$ , there exists  $\tilde{H}$ , a row echelon form of  $H$ , such that  $\tilde{H}_{\leq D} = G$ ,

then  $G$  is a Gröbner basis.

Note that condition 1 implies that  $\langle G \rangle$  is a zero-dimensional ideal; an ideal that has a finite number of solutions over the closure of the field. From now on, we concentrate on zero-dimensional ideals.

*Proof.* Let  $G = \{g_1, \dots, g_s\}$  with  $g_i \neq g_j$  for  $i \neq j$ . Suppose that the highest degree in  $G$  is  $D$  and that conditions 1 and 2 above hold. We want to show that for  $i, j \in \{1, \dots, s\}$ , with  $i \neq j$ ,  $f := \text{spol}(g_i, g_j)$  has a standard representation w.r.t.  $G$ . Without loss of generality, it suffices to show this for  $\text{spol}(g_1, g_2)$ .

If  $d := \deg(\text{lcm}(\text{LT}(g_1), \text{LT}(g_2))) \leq D + 1$ , then by condition 2

$$f \in \text{span}_{\mathbb{F}}(H) = \text{span}_{\mathbb{F}}(\tilde{H}) = \text{span}_{\mathbb{F}}(G) \oplus \text{span}_{\mathbb{F}}(\tilde{H}_{=D+1}).$$

If  $\deg(f) < D + 1$  then it is trivial to see that  $f \in \text{span}_{\mathbb{F}}(G)$  and hence has a standard representation w.r.t.  $G$ . Suppose that  $\deg(f) = D + 1$ . By condition 1, every term of degree  $D + 1$  appears as a leading term in  $H$ . Choose  $h_1 \in H$  such that  $\text{LT}(h_1) = \text{LT}(f)$  and define  $f_1$  by

$$f_1 := f - \frac{\text{LC}(f)}{\text{LC}(h_1)} h_1.$$

It is easy to see that  $f_1 \in \text{span}_{\mathbb{F}}(\tilde{H})$  and that  $\text{LT}(f_1) < \text{LT}(f)$ . If  $\deg(f_1) = D + 1$  we can repeat the same argument for  $f_1$  and by iterating the argument a finite number of times  $m$ , we obtain an expression

$$f = \sum_{i=1}^{m-1} a_i h_i + f_m \tag{6.1}$$

## 6 Computing Gröbner Basis

with  $a_i \in \mathbb{F}$ ,  $h_i \in H$ , for  $1 \leq i < m - 1$   $\text{LT}(h_i) > \text{LT}(h_{i+1})$  and  $\deg(f_m) < D + 1$ . Since  $f_m \in \text{span}_{\mathbb{F}}(\tilde{H})$  and  $\deg(f_m) < D + 1$ ,  $f_m \in \text{span}_{\mathbb{F}}(G)$  thus clearly (6.1) yields a standard representation of  $f$  w.r.t  $G$ .

For  $d > D + 1$ , we proceed by induction. Suppose that  $d > D + 1$  and that for  $i \neq j$ , if  $\deg(\text{lcm}(\text{LT}(g_i), \text{LT}(g_j))) < d$  then  $\text{spol}(g_i, g_j)$  has a standard representation w.r.t.  $G$ . Assume, without loss of generality, that  $\deg(g_1) \geq \deg(g_2)$  and note that  $\deg(g_1) > (D + 1)/2$ . Let  $t := \text{lcm}(\text{LT}(g_1), \text{LT}(g_2))$  and let  $t_1, t_2$  be terms such that for  $i = 1, 2$ ,  $t = t_i \text{LT}(g_i)$ . Note that  $\deg(t_i) \geq 2$  and that  $t_1$  and  $t_2$  are disjoint. Choose any terms  $t_{11}, t_{12}, t_{21}, t_{22}$  such that for  $i = 1, 2$ ,  $t_i = t_{i1}t_{i2}$  and  $\deg(g_1) + \deg(t_{12}) = D + 1$  and  $\deg(t_{21}) = 1$ . These choices are possible because  $(D + 1)/2 < \deg(g_1) \leq D$  thus  $1 \leq D + 1 - \deg(g_1) < D + 1 - (D + 1)/2 = (D + 1)/2 < \deg(g_1)$  and because  $\deg(t_2) \geq 2$ . It follows that  $\deg(t_{11}), \deg(t_{12}), \deg(t_{21})$  and  $\deg(t_{22})$  are all greater than or equal to 1. Also, if we let  $t^* := \frac{t}{t_{11}t_{21}}$ , by construction, for  $i = 1, 2$ ,  $\text{lcm}(t^*, \text{LT}(g_i)) = t/t_{i1}$  divides  $t$  properly,  $\deg(t^*) = D$  and since  $t_1$  and  $t_2$  are disjoint,  $t^*$  is different from both  $\text{LT}(g_1)$  and  $\text{LT}(g_2)$ . Then, by condition 1, there exist  $g \in G \setminus \{g_1, g_2\}$  with  $\text{LT}(g) = t^*$ . Also, for  $i = 1, 2$ , since  $\deg(\text{lcm}(\text{LT}(g), \text{LT}(g_i))) < \deg(t)$ , by the inductive hypothesis,  $\text{spol}(g, g_i)$  has a standard representation w.r.t.  $G$ . Moreover,  $\text{LT}(g)$  divides  $t$  and therefore, by the Buchberger's second criterion,  $\text{spol}(g_1, g_2)$  has a standard representation w.r.t.  $G$ .  $\square$

### 6.2 The $\text{MXL}_3$ algorithm

In this section, we describe the  $\text{MXL}_3$  algorithm, prove its correctness, and demonstrate the differences between  $\text{MXL}_3$  and  $\text{MXL}_2$  algorithms.

The main difference between the two algorithms is that  $\text{MXL}_2$  only works when a system of equations has a unique solution whereas  $\text{MXL}_3$  can handle any system of equations with a finite number of solutions. Any XL-type algorithm eventually computes a Gröbner basis, however it is uncertain for which degree bound it occurs. Proposition 6.4 provides an easy to check condition that guarantees a Gröbner basis that has been found. Experimental results show that in all the cases that we examined, using this alternative criterion reveals a Gröbner basis early.

Another important difference is explained as follows. Let  $X := \{x_1, \dots, x_n\}$  be a set of variables, upon which we impose the following order:  $x_1 > x_2 > \dots > x_n$ . The  $\text{MXL}_2$  algorithm enlarges the system from degree level  $d$  to  $d + 1$ , as described in Chapter 4, by extending one variable partition at a time. Let the selected partition have a leading variable  $x$  and  $p$  be a polynomial in that partition, so  $\deg(p) = d$  and  $\text{LV}(p) = x$ . The  $\text{MXL}_2$  algorithm multiplies  $p$  by all variables  $< \text{prev}(p)$ , where  $\text{prev}(p)$  is the previous variable multiplier of  $p$ . In this context, some polynomials of degree  $d + 1$  with leading variable  $> x$  may be generated. These polynomials will not play any role in the linear algebra step that follows this enlargement step, since the density of the degree  $d + 1$  terms expected to be high in the polynomials that have leading variable  $\leq x$  enough to eliminate them and generate mutants.

## 6 Computing Gröbner Basis

MXL<sub>3</sub> fixes this drawback of MXL<sub>2</sub> by generating all possible polynomials of degree  $d + 1$  that have leading variable  $\leq x$ . To explain the difference, assume that  $P = \{p_1, p_2, p_3, p_4, p_5\}$  is a set of polynomials over  $\mathbb{F}_2$  in  $\{x_1, x_2, x_3, x_4, x_5\}$ , and let  $P$  be in the row echelon form:

$$\begin{aligned} p_1 &= x_1x_3 + x_2x_3 + x_3x_4 + x_3 + x_5 + 1 \\ p_2 &= x_1x_5 + x_2x_5 + x_3x_5 + x_4x_5 + x_1 + x_2 + x_4 \\ p_3 &= x_2x_4 + x_3x_4 + x_1 + x_2 + 1 \\ p_4 &= x_2x_5 + x_3x_5 + x_4x_5 + x_1 + x_2 + x_3 + x_4 + 1 \\ p_5 &= x_3x_5 + x_1 + x_2 + x_4 + x_5 + 1 \end{aligned}$$

Since  $x_3$  is the smallest leading variable, then first we enlarge  $\text{LV}(P, x_3) = \{p_5\}$ . Multiplying  $p_5$  with all variables, we obtain the following polynomials:

$$\begin{aligned} p_6 &= x_1x_3x_5 + x_1x_2 + x_1x_4 + x_1x_5 \\ p_7 &= x_2x_3x_5 + x_1x_2 + x_2x_4 + x_2x_5 \\ p_8 &= x_1x_3 + x_2x_3 + x_3x_4 + x_3 \\ p_9 &= x_3x_4x_5 + x_1x_4 + x_2x_4 + x_4x_5 \\ p_{10} &= x_1x_5 + x_2x_5 + x_3x_5 + x_4x_5 \end{aligned}$$

The leading variables of  $p_6$  and  $p_7$  are  $x_1 > x_3$  and  $x_2 > x_3$ , respectively. These two polynomials have no effect in the process of solving the system. So MXL<sub>3</sub> enlarges  $\text{LV}(P, x_3)$  by multiplying  $p_5$  with all variables  $\leq x_3$ . In this case MXL<sub>3</sub> can solve the system by generating only  $p_8, p_9$ , and  $p_{10}$ .

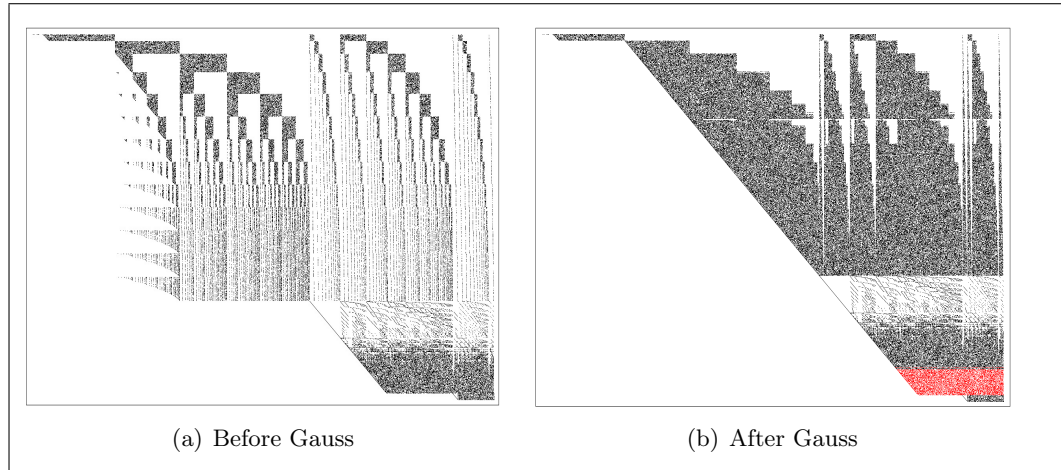


Figure 6.1: Matrix shape before and after the linear algebra step (*Echelonize*) for the generated degree 4 polynomials using MXL<sub>2</sub>

## 6 Computing Gröbner Basis

We give another example to explain the improved enlargement, Let  $P$  be a random system of size  $n = 14$ . Figure 6.1 shows the step when  $\text{MXL}_2$  starts to generate degree 4 polynomials. As we explained above,  $\text{MXL}_2$  enlarges the subset of degree-3 polynomials that have the smallest leading variable which is  $x_3$  in this case. Therefore,  $\text{MXL}_2$  multiplies these polynomials by all variables without generating trivial redundances as explained in Chapter 4. Figures 6.1(a) and 6.1(b) show the structure of the resulting matrix before and after the linear algebra step. The left upper block in the matrix is related to the degree 4 terms that have leading variable  $> x_3$  of the degree 4 polynomials. It is clear from both figures that during the elimination process this block of polynomials does not play any role in reducing degree 4 terms and producing mutants (the red part in the bottom).  $\text{MXL}_3$  avoids this block when it enlarges the system and produces the same number of mutants.

Before we are going to describe the  $\text{MXL}_3$  algorithm, we need the following additional notation. Let  $B$  be the Boolean polynomial ring in  $X$ , as defined in (2.3), and the monomials of  $B$  be ordered by the graded lexicographical order  $<_{\text{glex}}$ . We consider elements of  $B$  as polynomials over  $\mathbb{F}_2$  where degree of each term w.r.t any variable is 0 or 1. Let  $P = \{p_1, \dots, p_m\}$  be a finite set of polynomials in  $B$ .

As described in Chapter 4, we present a set of polynomials  $H$  as a set of pairs  $(p, x)$ , where  $p$  is a polynomial and  $x$  is  $\text{prev}(p)$ . Throughout the operation of the  $\text{MXL}_3$  algorithm (Alg. (6.1)), a degree bound  $D$  is used. This degree bound denotes the maximum degree of the polynomials generated during the  $\text{MXL}_3$  process. Also, we use  $d$  as a degree bound for the eliminated subset of  $H$ , ( $H_{\leq d}$ ). Note that the content of  $H$  is changed throughout the operation of the algorithm. As we defined in Chapter 4, the leading variable  $\text{LV}(p)$  for  $p \in H$  is the largest variable in  $\text{LT}(p)$ , according to the order defined on the variables set. Also, we define the subset  $\text{LV}(H, x)$  as the set of all polynomials of  $H$  with the leading variable  $x$ .

The  $\text{MXL}_3$  algorithm, Alg. (6.1), performs the following steps:

- *Initialization*, lines 1 – 6: The maximal and the elimination degrees ( $D, d$ ) are initialized by the maximum degree in the input set of polynomials  $p$ . The whole set of polynomials ( $H$ ) is initialized with the elements of the input set  $P$  and construct a pair  $(p, 1)$  for each  $p \in P$ . The index of the leading variable ( $i$ ) is initialized by 1. The set of leading terms  $L$  and the set of mutants  $M$  are set to empty.
- *Echelonize*( $H, d$ ), line 8: The row echelon form of  $H$  is computed such that all polynomials of  $H$  that have degree  $> d$  are not eliminated.
- Extract mutants, line 9: All polynomials of degree  $< d$  in  $\tilde{H}$  that have new leading term (not in  $L$ ).
- Update leading terms, Line 10: The set of leading terms is updated by adding the newly constructed leading terms appeared in  $\tilde{H}$ .
- *Gröbner*, lines 11 – 14: using Proposition 6.4 to check if a Gröbner basis of  $\langle P \rangle$  is computed.

## 6 Computing Gröbner Basis

- Enlarge mutants, lines 15 – 25: The system is enlarged by using the lowest degree ( $k$ ) mutants. In fact, we have two cases to be considered, namely: when  $k = D - 1$ , the resulting polynomials from enlarging mutants will be degree  $D$  polynomials. Let  $x_i$  be the largest leading variable of degree  $D$  polynomials, we enlarge mutants such that the resulting polynomial should have leading variable  $\leq x_i$ . Moreover, we use the leading variable of mutants in the multiplication process. Let  $x_j$  be the largest leading variable of degree  $k$  mutants, we enlarge mutants that have leading variable  $\leq x_i$  by all variables  $\leq \min\{x_i, x_j\}$ . The other case when  $k < D - 1$ , MXL<sub>3</sub> enlarges mutants in the same way as MutantXL and MXL<sub>2</sub> algorithms.
- Enlarge degree  $D$  polynomials, lines 26 – 40: We also have two cases here, namely: when we enlarge the first partition,  $i \leq 1$ . In this case, the first partition is identified by the smallest leading variable ( $x_i$ ) appearing in the degree  $D$  polynomials. The degree bound  $D$  is incremented by 1. Each polynomial  $p$  in  $\text{LV}(H_{=D-1}, x_i)$  is multiplied by all variables  $y < \text{prev}(p)$  such that  $y \leq x_i$ . The second case is to enlarge an intermediate partition. In this case  $1 < i < n$ . We determine the next smallest leading variable  $x_i$  and enlarge the partition  $\text{LV}(H_{=D-1}, x_i)$  as explained above. Moreover, we multiply each polynomial  $p$  in the previous partitions by  $x_i$  such that  $x_i < \text{prev}(p)$ . Finally, we assign the elimination degree  $d$  the degree bound  $D$ .

The following theorem establishes the correctness of the algorithm.

**Theorem 6.5.** *The MXL<sub>3</sub> algorithm computes a Gröbner basis  $G$  of the ideal generated by the set  $\{p_1, \dots, p_m\}$  of  $B$  with maximal degree  $D$ .*

*Proof.* Termination: MXL<sub>3</sub> terminates only when it enlarges all the polynomials of degree  $< d$  and when  $P$  contains all the terms of degree  $d - 1$  as leading terms, at a certain degree  $d \leq D$ . The worst case is to satisfy these two conditions at  $d = D = n + 1$ . Let the system be extended up to degree  $n$  without satisfying the termination conditions of the algorithm. In this case, MXL<sub>3</sub> extends the system to the next degree  $D = n + 1$ .  $P$  has only one polynomial of degree  $n$  which has leading variable  $x_1$ . In the *Enlarge* step, this polynomial is extended,  $i$  is set to 1, and  $d$  to  $n + 1$ . After the *Echelonize* step,  $P$  still contains only one polynomial of degree  $n = d - 1$  which is equal to the number of all terms in the Boolean ring  $B$  with degree  $n$ . If  $M \neq \emptyset$ , MXL<sub>3</sub> loops between *Eliminate* and *Enlarge* a finite number of times until the set  $M$  becomes empty. So all the conditions of *Gröbner* step are satisfied. Then MXL<sub>3</sub> returns  $G = P$  with highest degree  $n$  and terminates.

Correctness: Let the set of polynomials  $G$  with elements of maximum degree  $e < D$  satisfy the termination condition of MXL<sub>3</sub> (Alg. (6.1), line 11). The set  $G$  satisfies the first condition of Proposition 6.4 since it is in the row echelon form and contains all the terms of degree  $e$  as leading terms ( $T_{=e} \subseteq L$ ). Also, the *Gröbner* step returns  $G$  only when  $i = 1$  and  $M_{<e} = \emptyset$  which means that all the polynomials of degree  $\leq e$  are enlarged. Then  $G$  satisfies the second condition of Proposition

---

**Algorithm 6.1** MXL<sub>3</sub>

---

**Input:**  $P(x_1, \dots, x_n)$  is a set of finite polynomials in  $B$ .

**Output:**  $G$  is a Gröbner basis of  $\langle P \rangle$ .

```

1:  $D \leftarrow \max\{\deg(p) \mid p \in P\}$ 
2:  $d \leftarrow D$ 
3:  $H \leftarrow \{(p_1, 1), \dots, (p_m, 1)\}$ 
4:  $i \leftarrow 1$ 
5:  $L \leftarrow \emptyset$ 
6:  $M \leftarrow \emptyset$ 
7: loop
8:    $\tilde{H} \leftarrow \text{Echelonize}(H, d)$ 
9:    $M \leftarrow \{p \in \tilde{H}_{<d} \mid \text{LT}(p) \notin L\}$ 
10:   $L \leftarrow \text{LT}(\tilde{H})$ 
11:  if  $(d < D$  or  $i = 1)$  and  $(\exists e < D: M_{<e} = \emptyset$  and  $T_{=e} \subseteq L)$  then
12:     $G \leftarrow \tilde{H}_{\leq e}$ 
13:    return  $G$ 
14:  end if
15:  if  $M \neq \emptyset$  then
16:     $k \leftarrow \min\{\deg(p) \mid p \in M\}$     // The lowest degree mutants
17:    if  $k = D - 1$  then
18:       $x_j \leftarrow \max\{\text{LV}(p) \mid p \in M_{=k}\}$ 
19:       $H \leftarrow \tilde{H} \cup \{(xp, x) \mid x \leq x_j \text{ and } p \in M_{=k} \text{ and } \text{LV}(p) \leq x_i\}$ 
20:       $M \leftarrow M \setminus \bigcup_{s=j}^n \text{L}(M_{=k}, x_s)$ 
21:    else
22:       $H \leftarrow \tilde{H} \cup \{(xp, x) \mid x \leq x_1 \text{ and } p \in M_{=k}\}$ 
23:       $M \leftarrow M \setminus M_{=k}$ 
24:    end if
25:     $d \leftarrow k + 1$ 
26:  else
27:    if  $i \leq 1$  then
28:       $D \leftarrow D + 1$ 
29:       $i \leftarrow n + 1$ 
30:    end if
31:     $i \leftarrow i - 1$ 
32:    while  $i \geq 1$  and  $\text{L}(H_{=D-1}, x_i) = \emptyset$  do
33:       $i \leftarrow i - 1$ 
34:    end while
35:    if  $i > 0$  then
36:       $H \leftarrow \tilde{H} \cup \{(yp, y) \mid p \in \text{L}(H_{=D-1}, x_i) \text{ and } y \leq x_i \text{ and } y < \text{prev}(p)\}$ 
37:       $H \leftarrow \tilde{H} \cup \{(x_i p, x_i) \mid p \in \bigcup_{j=i+1}^n \text{L}(H_{=D-1}, x_j) \text{ and } x_i < \text{prev}(p)\}$ 
38:    end if
39:     $d \leftarrow D$ 
40:  end if
41: end loop

```

---

6.4. Therefore  $G$  is a Gröbner basis for the ideal generated by the input system  $\{p_1, \dots, p_m\}$ .  $\square$

### 6.3 Implementation and experimental results

In this section, we present experiments that compare the performance of our implementation of  $\text{MXL}_3$  and Magma’s implementation of  $F_4$ . The experiments based on random systems of sizes  $(25, \dots, 31)$  and HFE systems with univariate degree 288 and sizes  $(30, 35, \dots, 40, 45, 47, 48, 49)$ . Also, we show how  $\text{MXL}_3$  solves the HFE challenge 1 that has 3 solutions using maximum matrix size  $268840 \times 1666981$ . While it was solved by Faugère and Joux in [26] using  $F_5/2$  algorithm [25] with a maximum matrix size  $307126 \times 1667009$ .

The  $\text{MXL}_3$  implementation is based on the C++ implementation of  $\text{MXL}_2$ . Our modifications can be explained as follows. As in the  $\text{MXL}_2$  implementation, we use the dense matrix representation of the system of polynomials. When we enlarge the system from degree bound  $D - 1$  to  $D$  using the polynomials that have leading variable  $x_i$ , we extend the matrix by only terms of degree  $D$  and leading variable  $x_i$ . In the old implementation, all terms of degree  $D$  were represented since  $\text{MXL}_2$  may generate some terms with leading variable  $< x_i$ . In terms of relating the matrix columns to the list of terms as well as in the MutantXL and  $\text{MXL}_2$  implementations, we keep the number of not constructed degree  $D$  terms to use this value during the ranking and un-ranking methods explained in Chapter 3. We modified the implementation of  $\text{MXL}_2$  by checking after each Echelonization step if the new termination condition (Alg. (6.1)) is satisfied and returns the computed Gröbner basis. Additionally, we made many other changes based on the differences between  $\text{MXL}_2$  and  $\text{MXL}_3$  algorithms as explained in the previous section.

We built our experiments to compare the efficiency of  $\text{MXL}_3$  to the efficiency of  $F_4$  in solving some random systems generated by Courtois [15] as well as some HFE systems generated by the code of John Baena. We run all the experiments on a Sun X4440 server, with four “Quad-Core AMD Opteron™ Processor 8356” CPUs and 128 GB of main memory. Each CPU is running at 2.3 GHz. We used only one out of the 16 cores.

Tables 6.1 and 6.2 show the results of dense random systems with many solutions and the results of HFE systems of univariate degree 288, respectively. In both tables we denote the number of variables and equations by  $n$  and the highest degree of the iteration steps by  $D$ . The tables also show the maximum matrix size, the memory used in Megabytes, and the execution time in seconds. It is evident from Tables 6.1 and 6.2 that  $\text{MXL}_3$  solves the random generated systems and HFE systems faster and consumes less memory than  $F_4$ .

Table 6.1 shows that both  $\text{MXL}_3$  and  $F_4$  solve random systems up to a system of 31 variables. The solutions of  $\text{MXL}_3$  are consistent with the results of Magma. When  $\text{MXL}_3$  and  $F_4$  tried to solve a 32 variables system, both were able to enlarge the system up to degree 6. When the system was enlarged to degree 7, they ran out

## 6 Computing Gröbner Basis

$n$	MXL <sub>3</sub>				$F_4$			
	$D$	max. matrix	Mem.	Time	$D$	max. matrix	Mem.	Time
16	5	2295×2573	1	< 1	6	9995×4036	19	< 1
17	5	3211×3676	2	< 1	6	12415×5817	28	1
18	5	4477×5335	3	< 1	6	15187×8120	43	2
19	5	8150×8039	9	1	6	18441×11041	64	4
20	5	8719×9199	11	2	6	22441×14979	96	7
21	5	13152×13735	22	8	6	26860×19756	139	17
22	5	20332×20737	38	13	6	63621×21855	340	54
23	5	23415×26407	74	25	6	41866×29010	475	83
24	6	52215×57171	390	341	6	207150×78637	3069	642
25	6	66631×76414	698	704	6	248495×108746	5128	1341
26	6	88513×102246	1207	1429	6	298592×148804	8431	3325
27	6	123938×140344	2315	2853	6	354189×197902	13312	6431
28	6	201636×197051	4836	7982	6	420773×261160	20433	13810
29	6	279288×281192	9375	18796	6	499222×340254	30044	25631
30	6	332615×351537	15062	33331	6	1283869×374081	72258	92033
31	6	415654×436598	23078	94191	6	868614×489702	108738	162118

Table 6.1: Performance of MXL<sub>3</sub> versus  $F_4$  for dense random system

of memory. For the 30 variables system we get a strange matrix size from Magma. We created many 30 variables random system and we obtain approximately the same numbers.

$n$	MXL <sub>3</sub>				$F_4$			
	$D$	max. matrix	Mem.	Time	$D$	max. matrix	Mem.	Time
30	5	86795×130211	1389	3106	5	149532×136004	7105	3806
35	5	155914×296872	5737	10047	5	200302×321883	40480	11032
36	5	173439×344968	7310	14183	5	219438×382252	50846	15220
37	5	192805×399151	9288	20375	5	247387×444867	66623	20787
38	5	212271×459985	11351	27089	5	274985×512311	83445	27305
39	5	234111×528068	15070	36833	5	305528×588400	104135	38013
40	5	258029×604033	20881	63460	ran out of memory			
45	5	404940×1126819	55216	299355	ran out of memory			
47	5	457691×1417468	77967	371088	ran out of memory			
48	5	517642×1583807	98913	689235	ran out of memory			
49	5	561972×1765465	120524	751965	ran out of memory			

Table 6.2: Performance of MXL<sub>3</sub> versus  $F_4$  for HFE(288,n) systems

## 6 Computing Gröbner Basis

Table 6.2 shows that all the HFE systems of univariate degree 288 up to 49 variables are solved by using  $MXL_3$ , whereas  $F_4$  could only solve HFE systems up to 39 variables with the same memory resources.

In Tables 6.3 and 6.4 we compare the performance of the  $MXL_3$  algorithm against the  $F_4$  algorithm in computing a Gröbner basis of the random system  $n = 30$ . For  $MXL_3$ , we give the elimination degree ( $D$ ), the matrix size for each level, the rank of the matrix (Rank), the number of mutants found (NM), the number of used mutants (UM), and the lowest degree of mutants found (MD). For  $F_4$ , we give the step degree ( $D$ ), the matrix size, and the step memory in MB.

Step	$D$	Matrix Size	Rank	NM	UM	MD
1	2	$30 \times 466$	30	0	0	-
2	3	$930 \times 4526$	930	0	0	-
3	4	$13980 \times 31931$	13515	0	0	-
4	5	$131690 \times 174437$	121365	0	0	-
5	6	<b><math>332615 \times 351537</math></b>	329051	31060	665	5
6	6	$302981 \times 309033$	302981	3596,12340	0,191	5,4
7	5	$172945 \times 174437$	172945	2480,3160,90	0,0,11	4,3,2
8	3	$4510 \times 4526$	4510	315,15	0,1	2,1
9	2	$480 \times 466$	465	15	0	1

Table 6.3: Results for the system Random-30 by  $MXL_3$

Step	$D$	Matrix Size	Memory
1	2	$30 \times 466$	14.2
2	3	$937 \times 4526$	14.2
3	4	$13320 \times 30551$	207
4	5	$106603 \times 143547$	4318
5	6	$588160 \times 437262$	42843
6	6	<b><math>1283869 \times 374081</math></b>	72258
7	2	$722 \times 466$	72258
8	3	$4864 \times 3782$	72258
9	4	$22421 \times 19736$	72258
10	5	$103919 \times 62858$	72258

Table 6.4: Results for the system Random-30 by Magma

Table 6.3 shows that by using the mutant strategy,  $MXL_3$  can easily solve the 30 variables random system with a smaller matrix size compared to  $F_4$ .  $MXL_3$  starts to generate mutants at step 5. In this step 31060 mutants of degree 5 are generated, out of which only 665 are multiplied. Due to the degree of the generated mutants, the elimination degree remains the same in the next step, i.e.,  $D = 6$ . Starting from

## 6 Computing Gröbner Basis

step 7,  $D$  starts to decrease. In step 8, the system generates 315 quadratic mutants and 15 linear mutants. By using only one of the linear mutants,  $\text{MXL}_3$  generates additional 15 linear mutants in the next step, which in turn leads to solving the system.

Also, Table 6.3 shows that the number of reductions to zero is very few for each iteration step. This explains practically that our improved selection strategy has strictly increased the efficiency of the algorithm since it avoids the redundant computations.

Table 6.5 presents a comparison between the maximum matrix size constructed by  $\text{MXL}_3$  and  $\text{MXL}_2$  on some random systems that have only one solution. The results show that  $\text{MXL}_3$  solves with smaller number of polynomial equations and smaller number of terms than  $\text{MXL}_2$ . This due to the selection strategy of the multiplied variables that is used by  $\text{MXL}_3$ . Table 6.6 presents another comparison between  $\text{MXL}_3$  and Magma's  $F_4$  when the HFE parameter is setting to true; Magma's  $F_4$  avoids any s-polynomial pairs that produce polynomials of degree  $> 4$  when the secret degree  $d \leq 127$ . In this case  $\text{MXL}_3$  also solves with smaller number of polynomials than Magma's  $F_4$ . For example the HFE challenge 1  $n = 80$  that was first solved with maximum matrix size  $307126 \times 1667009$  by Faugère and Joux [26] using  $F_5/2$  algorithm [25] in May 2002, can be solved by  $\text{MXL}_3$  with maximum matrix size  $268840 \times 1666981$ , while Magma solves it with maximum matrix size  $293287 \times 1666981$ . In this case Magma is faster than our implementation since it uses a very fast linear algebra implementation.

$n$	$\text{MXL}_3$	$\text{MXL}_2$
16	2295×2573	2308×2675
17	3211×3676	3187×3831
18	4477×5335	5046×5332
19	8150×8039	7932×9359
20	8719×9199	14540×12384
21	13152×13735	17226×16115
22	20332×20737	20617×22629
23	23415×26407	26075×26407
24	52215×57171	52205×57171
25	66631×76414	70969×88235
26	88513×102246	97053×126190
27	123938×140344	135255×183309
28	201636×197051	197928×239369
29	279288×281192	280840×310704
30	332615×351537	346422×391811
31	415654×436598	493433×502378

Table 6.5: Performance of  $\text{MXL}_3$  versus  $\text{MXL}_2$  for dense random system

## 6 Computing Gröbner Basis

	MXL <sub>3</sub>	F <sub>4</sub>
<i>n</i>	max. matrix	max. matrix
20	5236×5227	7053×6196
25	9979×9941	12459×15276
30	22515×31931	20003×31931
35	33705×59536	30081×59536
40	37005×84516	43124×102091
50	67525×251176	79116×251176
60	144030×523686	130755×523686
70	181335×974121	201343×974121
80	268840×1666981	293287×1666981

Table 6.6: Performance of MXL<sub>3</sub> versus F<sub>4</sub> for HFE(96,*n*) systems

Another important comparison is to study the performance of MXL<sub>3</sub> and F<sub>4</sub> when they solve HFE systems of different univariate degrees (*d*). Table 6.7 shows the results of solving HFE systems with 25 variables and with different *d*. It is clear that MXL<sub>3</sub> is faster and uses less memory since it constructs smaller matrices than F<sub>4</sub>. It is important to point out that for F<sub>4</sub> we did not set the HFE parameter to true since we examine the general case of HFE systems whose univariate degrees can go above 128.

<i>d</i>	<i>D</i>	MXL <sub>3</sub>			F <sub>4</sub>		
		max. matrix	Mem.	Time	max. matrix	Mem.	Time
<i>d</i> < 17	3	2620×2626	1.5	0.16	3669×2500	15	0.35
16 < <i>d</i> < 64	4	7100×13252	12	1.56	8745×8101	68	2.25
63 < <i>d</i> < 96	4	9200×13252	15	2.34	10837×13913	137	6.9
95 < <i>d</i> < 128	4	10615×13252	17.6	3.28	12497×13648	153.5	15.1
<i>d</i> = 128	4	15050×15276	28.7	5.9	14344×15245	227	22.7
128 < <i>d</i> < 257	5	31400×41610	217	70.6	52333×52665	750	148
256 < <i>d</i> < 513	5	40458×41610	231	110	53388×52835	1135	273
<i>d</i> > 512	6	72456×76414	735	811	257316×108618	5716	1420

Table 6.7: Performance of MutantXL versus F<sub>4</sub> for HFE(*d*,25) systems

For the comparison with Faugère’s F<sub>4</sub> algorithm, we used Magma (version V2.13-10). We used the field equations  $x_i^2 = x_i$  and using the polynomial ring type for Magma that is defined over  $\mathbb{F}_2$  such that all the terms are reduced modulo the field equations. When we use the new version of Magma (V2.15) and the new Magma type BooleanPolynomialRing, we have worse results in terms of the matrix size and the memory, although we obtained better results in terms of the running times.

We chose to compare MXL<sub>3</sub> only with Magma’s implementation of F<sub>4</sub> because both Faugère’s algorithm and Steel’s implementation are widely recognized as bench-

marks for efficient Gröbner basis computation. A comparison with Faugère's  $F_5$  algorithm has been suggested by some researchers. We consider this infeasible due to the controversy about the algorithm and the lack of a well recognized implementation.

$F_5$  is primarily intended for homogeneous regular sequences. There is no detailed description of how to adapt it for non-homogeneous sequences and moreover, no analysis exist on the behavior for such sequences. In the non-homogeneous case the  $F_5$  criteria can discover trivial syzygies of the leading forms, but it says nothing about what to do with the non-trivial ones which in turn yield mutants. In [20, 40, 21] the importance of mutants in the computation of Gröbner Bases for non-homogeneous sequences has been shown.  $F_5$  says nothing about how to take advantage of those. Therefore, in the absence of such algorithm one relies on Buchberger's or the  $F_4$  algorithm to compute a Gröbner Basis once mutants appear.

## 6.4 Towards cryptanalysis of HFE challenge 2

In this section we explain our method towards cryptanalysis of the second HFE challenge and analyze our experimental results of solving some scaled versions of HFE challenge 2 systems. The HFE challenge 2 can be considered as a multivariate digital signature scheme that signs a message of length 128 bits and generates a signature of length 144 bits. It has 36 variables and 32 equations over  $\mathbb{F}_{2^4}$ . When we transfer the equations over  $\mathbb{F}_2$ , we have 144 variables and 128 equations. Since we initially construct HFE challenge 2 systems over  $\mathbb{F}_{2^4}$ , so we select to scale down the parameters of HFE Challenge 2 as follows:

- $\mathbb{F}_{2^4}$ :  $n = 36$ ,  $h = 4$ , and  $m = 32 \rightarrow \mathbb{F}_2$ :  $n = 144$ ,  $h = 16$ , and  $m = 128$ .
- $\mathbb{F}_{2^4}$ :  $n = 27$ ,  $h = 3$ , and  $m = 24 \rightarrow \mathbb{F}_2$ :  $n = 108$ ,  $h = 12$ , and  $m = 96$ .
- $\mathbb{F}_{2^4}$ :  $n = 18$ ,  $h = 2$ , and  $m = 16 \rightarrow \mathbb{F}_2$ :  $n = 72$ ,  $h = 8$ , and  $m = 64$ .
- $\mathbb{F}_{2^4}$ :  $n = 9$ ,  $h = 1$ , and  $m = 8 \rightarrow \mathbb{F}_2$ :  $n = 36$ ,  $h = 4$ , and  $m = 32$ .

Where  $n$  is the number of variables,  $m = n - h$  is the number of equations, and  $h$  is the number of hidden equations.

We can analyse these systems by applying the following steps on each one of the above systems:

1. Generate an HFE system of equations over  $\mathbb{F}_{2^4}$ .
2. Remove  $h$  equations from the system.
3. Convert the system of equations to  $\mathbb{F}_2$ .
4. Fix the first  $h$  variables  $(x_1, \dots, x_h)$ .
5. Guess more  $g$  variables.

## 6 Computing Gröbner Basis

6. Solve the resulting system with size  $(n - h - g) \times m$ , where  $(n - h - g)$  is the number of variables and  $m$  is the number of equations.
7. Repeat the previous two steps with  $g = g - 4$  until we reach to  $g$  such that the system of size  $(n - h - g) \times m$  could not be solved.

After converting the system to  $\mathbb{F}_2$ , we fix  $n - m$  variables to get a determined system. After that we guess a number of variables as many as enough for solving the resulting overdetermined systems. We decrease the number of guessing variables by 4 and repeat the previous step until we can not solve the resulting system. For the sixth step we use our  $\text{MXL}_3$  implementation to solve the systems. By this way we can estimate the complexity of solving the HFE Challenge 2 systems. Now we are going to present our experimental results and give more analysis. We built our experiments to explain the performance of  $\text{MXL}_3$  for solving some HFE challenge 2 systems. We use the code of D. Schmidt to generate HFE systems as explained in Chapter 2. We run all the experiments on a Sun X4440 server, with four “Quad-Core AMD Opteron™ Processor 8356” CPUs and 128 GB of main memory. Each CPU is running at 2.3 GHz. We used only one out of the 16 cores.

$g$	$n'$	max. matrix	D	Var	Time	Memory
88	40	$2600 \times 5781$	3	$x_9$	3	3.8
84	44	$6444 \times 10871$	3	$x_5$	12	13.2
80	48	$3668 \times 14421$	3	$x_5$	16	24.8
76	52	$8804 \times 23479$	3	$x_1$	100	61.5
72	56	$23452 \times 34162$	4	$x_{37}$	272	136
68	60	$24692 \times 127441$	4	$x_{21}$	14031	1855
64	64	$42964 \times 238325$	4	$x_{17}$	39547	4819
60	68	$196174 \times 419753$	4	$x_{13}$	44037	9817
56	72	$54772 \times 549904$	4	$x_{13}$	144173	19131
52	76	$286620 \times 887612$	4	$x_9$	365801	47366

Table 6.8: results of  $\text{MXL}_3$  for HFE Challenge 2 system ( $n = 144$ ,  $m = 128$  and  $h = 16$ )

Table 6.8 shows the results of solving HFE challenge 2 systems with  $n = 144$ ,  $m = 128$ , and  $h = 16$ . We used the method explained above. After fixing 16 variables we have a system of  $m = 128$  equations and variables. We guess more  $g$  variables. As the system is originally built over  $\mathbb{F}_{2^4}$ , then each sequential 4 variables  $x_{4i-3}, x_{4i-2}, x_{4i-1}, x_{4i}$  ( $i \in \{1, 2, \dots, m/4\}$ ) are related since they represent  $x_i$  over  $\mathbb{F}_{2^4}$ . In this case, we choose  $g$  such that  $4 \mid g$ . Moreover, we select the first  $g/4$  groups, for example when  $g = 40$ , we pass values for  $x_1, \dots, x_{40}$ . In this case  $n' = 128 - g$  is the number of variables in the obtained system after guessing  $g$  variables.

Also, we construct three scaled versions of HFE challenge 2, as explained in the previous section, with sizes  $(n = 108, h = 12, m = 96)$ ,  $(n = 108, h = 12, m = 96)$ ,

## 6 Computing Gröbner Basis

$g$	$n'$	max. matrix	D	Var	Time	Memory
56	40	$2172 \times 7961$	3	$x_5$	4	7.5
52	44	$6748 \times 14235$	3	$x_1$	22	21.9
48	48	$14112 \times 18473$	3	$x_1$	36	38.6
44	52	$28956 \times 82384$	4	$x_{17}$	629	523.8
40	56	$60292 \times 165068$	4	$x_{13}$	1092	1179.8
36	60	$51108 \times 230631$	4	$x_{23}$	30835	4775.6
32	64	$110368 \times 411035$	4	$x_9$	100976	16471.1
28	68	$630444 \times 866848$	4	$x_1$	222064	92566.8

Table 6.9: results of  $\text{MXL}_3$  for HFE Challenge 2 system ( $n = 108$ ,  $m = 96$  and  $h = 12$ )

$g$	$n'$	max. matrix	D	Var	Time	Memory
28	36	$3292 \times 7807$	3	$x_1$	6	6.7
24	40	$7100 \times 10701$	3	$x_1$	8	13.6
20	44	$11808 \times 50195$	4	$x_{13}$	868	238.6
16	48	$34772 \times 109863$	4	$x_9$	4828	1299.5
12	52	$42560 \times 159230$	4	$x_9$	9694	2750.7
8	56	$186304 \times 494887$	5	$x_{29}$	193740	14693
4	60	$219352 \times 1181694$	5	$x_{17}$	634592	59073

Table 6.10: results of  $\text{MXL}_3$  for HFE Challenge 2 system ( $n = 72$ ,  $m = 64$  and  $h = 8$ )

and ( $n = 108, h = 12, m = 96$ ). We apply the same solving strategy we explained above. Tables 6.9, 6.10, and 6.11 show the results of these systems.

Table 6.12 shows the results of solving some scaled versions of a HFE challenge 2 system with  $n = 144$ ,  $m = 128$ , and  $h = 16$  using Magma's implementation of  $\mathbb{F}_4$ . Magma can not solve any system greater than 128 equations in 72 variables. This explains how our improved  $\text{MXL}_3$  algorithm is more efficient than Magma's  $\mathbb{F}_4$  in terms of memory. However,  $\mathbb{F}_4$  is faster than our  $\text{MXL}_3$  implementation since it uses the advanced Magma's linear algebra techniques.

Table 6.13 shows how  $\text{MXL}_3$  solves a scaled version of HFE2 with  $n = 72$ ,  $m = 64$ , and  $h = 8$ . Let us fix 8 variables and guess another 8 variables, so the resulting system has 64 equations and 56 variables. As we see from the table at degree  $D = 4$ , we have nine rounds. Four of them come by enlarging degree 3 partitions of leading variables  $x_1, x_5, x_9, x_{13}$  that are generated from the original degree 2 partitions  $x_1, x_5$ . The other three partitions  $\{x_2, x_3, x_6\}$  are generated by reduction as shown in steps 5, 8. Also, in this level we found few mutants which are not sufficient to solve the system. At  $D = 5$ , we found some lower degree polynomials generated by reduction in rounds 2,3,4, and 5. While, at round 6 we found a lot of mutants of degree 3 and

## 6 Computing Gröbner Basis

$g$	$n'$	max. matrix	D	Var	Time	Memory
12	20	$692 \times 771$	3	$x_5$	0.0	0.06
8	24	$1820 \times 2325$	3	$x_1$	0.0	0.56
4	28	$2696 \times 3683$	3	$x_1$	1	1.6
0	32	$5984 \times 16115$	4	$x_9$	34	24
fix=0	36	$15204 \times 43767$	4	$x_5$	287	227.8

Table 6.11: results of  $\text{MXL}_3$  for HFE Challenge 2 system ( $n = 36$ ,  $m = 32$  and  $h = 4$ )

$g$	$n'$	D	Time	Memory
76	52	3	6	203
68	60	4	983	12288
64	64	4	8117	38912
60	68	4	12482	60416
56	72	4	73515	105472
52	76	ran out of memory		

Table 6.12: results of  $F_4$  for HFE Challenge 2 system ( $n = 144$ ,  $m = 128$  and  $h = 16$ )

4 that successfully solve the system with maximum matrix size  $186804 \times 494887$ .

Figure 6.2 displays the experimental time complexity of solving scaled versions of HFE Challenge 2 system by  $\text{MXL}_3$  as in Table 6.8. In this case, after fixing 16 variables (the number of removed equations) we have a HFE system with 128 equations and 128 variables. We guess more  $g$  variables and solve the resulting systems with 128 equations and  $(128 - g)$  variables.

In Figure 6.2(a), X-axis represents the number of guessed variables  $g$  and Y-axis represents the time consuming to solve each system after guessing  $g$  variables. As we show, the time complexity increased as the number of guessing variables decreased. However, this does not give us a real feeling about the complexity of breaking the Challenge.

Figure 6.2(b) shows the complexity of breaking HFE Challenge 2 after guessing different  $g$  variables. Here X-axis as in Figure 6.2(a) represents  $g$ , while the values of Y-axis represent the logarithm of the time consuming to solve the scaled system with 128 equations and  $128 - g$  variables multiplied by  $2^g$ . For example, in the worst case we need  $10^{27}$  seconds to break HFE Challenge 2 when  $g = 88$  and around  $10^{21}$  seconds when  $g = 52$ . It is clear from Figure 6.2(b) that the time complexity for breaking HFE Challenge 2, in the worst case, decreased as the number of guessed variables decreased.

Another study of the complexity of solving HFE Challenge 2 is showed in Figure 6.3. Since the most time consuming part of  $\text{MXL}_3$  is the linear algebra step, we study the complexity of computing the row echelon form of the largest matrix

## 6 Computing Gröbner Basis

Step	D	Round	Matrix Size	Rank	Var	M	UM	MD
1	2	1	64×1597	64	$x_1$	0	0	-
2	3	1	688×23697	688	$x_5$	0	0	-
3	3	2	3612×29317	3612	$x_1$	0	0	-
4	4	1	7484×165068	7484	$x_{13}$	0	0	-
5	4	2	18132×223897	17780	$x_9$	1276	232	3
6	4	3	28916×223897	28916	$x_9$	0	0	-
7	4	4	51182×279217	51000	$x_6$	0	0	-
8	4	5	105942×300042	83762	$x_5$	36	24	3
9	4	6	85010×300042	84542	$x_5$	0	0	-
10	4	7	87230×345568	86582	$x_3$	0	0	-
11	4	8	161564×370372	135086	$x_2$	0	0	-
12	4	9	221456×396607	161320	$x_1$	0	0	-
13	5	1	161384×400975	161368	$x_{41}$	0	0	-
14	5	2	162332×412111	162256	$x_{37}$	152	0	4
15	5	3	163228×430256	163228	$x_{34}$	180	0	4
16	5	4	170040×439111	169820	$x_{33}$	2120	0	4
17	5	5	172352×477337	172352	$x_{30}$	480	0	4
18	5	6	186804×494887	186304	$x_{29}$	4344, 376	376	4, 3

Table 6.13: Results for HFE2 system ( $n = 72, m = 64, h = 8, g = 8$ ) by  $MXL_3$

## 6 Computing Gröbner Basis

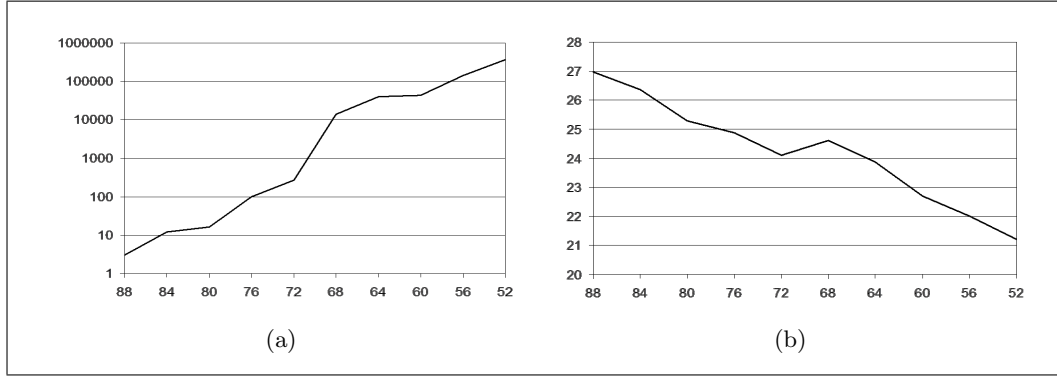


Figure 6.2: Explain time complexity in seconds ( $y$ -axis) and the number of guessing variables  $g$  ( $x$ -axis)

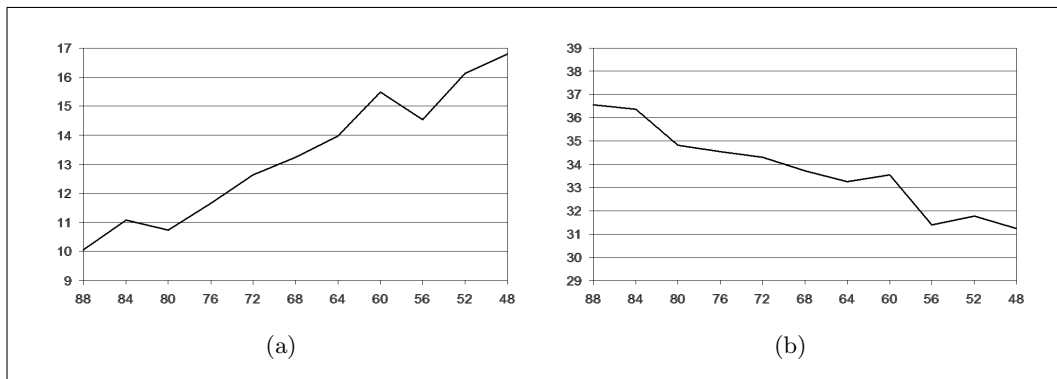


Figure 6.3: Relation between the O-Notation of the maximum matrix ( $y$ -axis) and the number of guessing variables  $g$  ( $x$ -axis)

computed by  $\text{MXL}_3$ . Our implementation of  $\text{MXL}_3$  uses the "Method of Four Russians" [3] in the linear algebra step. The complexity of this method is  $O(N \cdot M \cdot R / \log N)$  [3] where  $N, M$ , and  $R$  are the number of rows, the number of columns, and the rank respectively. In Figure 6.3(a), we compute the O-notation for the maximum matrix computed by  $\text{MXL}_3$  as in Table 6.8. In Figure 6.3(b), we multiply this O-notation by  $2^g$  when we guess  $g$  variables. In both figures, Y-axis represents the logarithm of the computed O-notation. Figures 6.3(a), 6.3(b) confirm the results that we showed in Figures 6.2(a), 6.2(b) respectively. The complexity of computing the row echelon form of the maximum matrix decreased as the number of guessing variables decreased.

Finally, we interpolate the results in Table 6.8 to estimate the memory needed to break the full challenge using  $\text{MXL}_3$  algorithm. Figure 6.4 shows the estimated memory consumed for solving scaled versions of HFE challenge 2 systems. We used Lagrange polynomial interpolation method in this computations. In this case, we need approximately 100 TB to break the full version of HFE challenge 2.

## 6 Computing Gröbner Basis

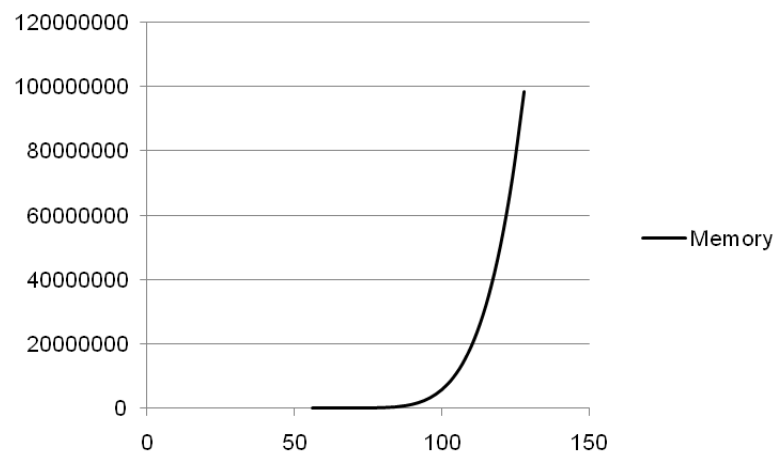


Figure 6.4: Relation between the memory usage of solving HFE challenge 2 systems ( $y$ -axis) and the number of variables  $n'$  ( $x$ -axis) after guessing  $g$  variables, while the number of equation is 128.

## 7 Flexible Partial Enlargement

This chapter introduces an efficient algorithm, called Mutant-based Gröbner Basis algorithm (MGB), that uses a more flexible partial enlargement to omit some parts of the  $\text{MXL}_3$  matrix and still satisfy the  $\text{MXL}_3$  termination criterion. We study the complexity of solving multivariate systems over  $\mathbb{F}_2$  using MGB. We present a preliminary study for the complexity of solving multivariate polynomial systems over  $\mathbb{F}_2$ . This study suggests a new upper bound for the complexity of computing Gröbner bases which may lead us to think of a new paradigms for estimating the complexity of Gröbner bases computation. The MGB algorithm is fully implemented in C++. We give experimental results that compare the performance of MGB with both  $\text{MXL}_3$  and the Magma's implementation of  $F_4$  algorithm [24]. Our experiments are based on randomly generated multivariate quadratic systems and HFE cryptosystems. We show that MGB computed a Gröbner basis of a degree 2 random system with 32 variables and equations in 2.3 days using only 30 Gigabytes which is never achieved by any other known Gröbner basis algorithm.

### 7.1 The basic idea

The partial enlargement technique introduced first in [40] is effective in reducing the number of polynomials needed at the highest degree  $d_m$  where mutants start to appear. However, it offers no advantage over generating the whole set  $H_d$  for  $d < d_m$  (as defined in equation (2.4)). We propose a heuristic method for systematically omitting subsets of  $H_d$  whenever a given condition is met. In this section we provide the theoretical foundation for this method. Consider the Boolean ring

$$B = \mathbb{F}_2[x_1, \dots, x_n] / \langle x_1^2 - x_1, \dots, x_n^2 - x_n \rangle.$$

as defined in (2.3). We assume that  $X := \{x_1, \dots, x_n\}$  is ordered as

$$x_1 > x_2 > \dots > x_n,$$

the terms of  $B$  have been ordered by the graded lexicographical order  $<_{\text{glex}}$ , and  $T_d$  the set of all terms of degree  $d$  in  $B$ . Let  $P = \{p_1, \dots, p_m\}$  be set of polynomials in  $B$ . We define the  $n$  sets,  $L_1(P), \dots, L_n(P)$ , by

$$L_j(P) := \{xp \mid x \leq x_j, \text{LV}(p) = x_j\}, 1 \leq j \leq n. \quad (7.1)$$

We should note that for  $q = xp \in L_j(P)$ ,  $\text{LV}(q) \leq x_j$  or  $\text{deg}(q) < d + 1$ .

We can describe the partial enlargement strategy used by  $\text{MXL}_3$  as follows: Let

## 7 Flexible Partial Enlargement

$P_2$  be a finite set of degree 2 polynomials in  $B$  and assume  $P_2$  is in row echelon form.  $\text{MXL}_3$  initializes the set  $H$  with  $P_2$  and enlarges  $P_2$  one variable partition at a time, i.e., for  $j = n, n-1, \dots, 1$  (in that order), construct the set  $L_j(P_2)$ , then append it to  $H$  and compute a row echelon form of  $H$ . If no mutants are found in  $H$ , consider  $P_3$  the set of new polynomials of degree 3 in  $H$  and repeat the process for  $P_3$  only.  $\text{MXL}_3$  proceeds in this way until mutants start to appear and compute a Gröbner basis at certain degree  $d$ . In this case, a sequence of sets  $P_2, P_3, \dots, P_d$  is produced such that for  $p \in P_d$ ,  $\deg(p) = d$ .

We have observed that often, all the largest variable-partitions of each  $P_k$ ,  $k < d$ , are *full*; all possible linearly independent polynomials exist. More precisely, there exist  $x_{j_k} > x_n$  such that for all terms  $t \in T_k$  such that  $\text{LV}(t) \geq x_{j_k}$ , there exist  $p \in P_k$ ,  $\text{LT}(p) = t$ . Note that if the  $j^{\text{th}}$  partition of  $P_d$  is full, necessarily the  $j^{\text{th}}$  partition for  $P_{d+1}$  is also full. Hence, the sequence  $j_2, j_3, \dots, j_d$  is non-decreasing. If the  $j^{\text{th}}$  partition of  $P_k$  is full, we propose to omit the  $j^{\text{th}}$  partition from any subsequent  $P_k$  ( $k > d$ ), under the certainty that it will be full.

To remedy this situation, we compute row reduced echelon forms instead of just any row echelon form, by means of which, we are able to perform reductions in advance, and ensure polynomials are fully reduced with respect to the omitted partitions. The following proposition states this result precisely

**Proposition 7.1.** *Let  $P_d$  be a finite subset of degree  $d$  polynomials from  $B$  in row echelon form. Suppose there exist  $x_j > x_n$  such that for all terms  $t \in T_d$  such that  $\text{LV}(t) \geq x_j$ , there exist  $p \in P_d$  such that  $t = \text{LT}(p)$ . Let*

$$G := P_d \cup \bigcup_{i=j+1}^n L_i(P_d),$$

$\tilde{G}$  the row reduced echelon form of  $G$  and

$$P_{d+1} := \{p \in \tilde{G} \mid \deg(p) = d + 1\}.$$

Then for  $i > j$ ,  $(x, p) \in L_i(P_{d+1})$  and any term  $s \in T_{d+1}$  such that  $\text{LV}(s) \geq x_j$ , the term  $s$  is not a term in  $xp$ .

*Proof.* Let  $i > j$ ,  $L_i(P_{d+1})$  and  $s \in T_{d+1}$  such that  $\text{LV}(s) \geq x_j$ . From the definitions of  $G$  and  $P_{d+1}$ , all terms of degree  $d + 1$  in  $p$  belong to  $T_{d+1}(x_{j+1}, \dots, x_n)$  thus  $s$  is not a term in  $p$ .

Since  $P_{d+1}$  is a subset of  $\tilde{G}$ , the row reduced echelon form of  $G$ , and all terms of degree  $d$  with leading variables  $\geq x_j$  appear as leading term of an element of  $P_d \subset G$ , then such terms do not appear in  $p$ , i.e. all terms of degree  $d$  in  $p$  are elements of  $T_d(x_{j+1}, \dots, x_n)$ . Moreover,  $x < x_j$  since  $i > j$ , hence, there is no term  $t$  in  $p$  with degree  $d$  that satisfies  $xt = s$ . Therefore,  $s$  is not a term in  $xp$ .  $\square$

The importance of this proposition is that the polynomials of degree  $d + 1$  with leading variables  $x_1, \dots, x_j$  are not needed to reduce polynomials coming from

$L_i(P_{d+1})$  for  $i > j$ . Hence, when enlarging  $P$  we may omit the polynomials coming from  $L_1(P), \dots, L_j(P)$ . This does not imply that these polynomials are not needed ever again, but just that their computation can be postponed yet obtaining sets of polynomials row reduced with respect to these missing polynomials. This observation leads to the heuristic algorithm described in the following section. Section 7.3 illustrates the operation of the algorithm.

## 7.2 The MGB algorithm

We introduce the MGB algorithm to compute Gröbner bases over the function ring  $B$  under the graded lexicographic order. The MGB is based on the  $MXL_3$  algorithm, and differs in a heuristic method to omit some variable-partitions based on the flexible partial enlargement explained in the previous section. The heuristic consists in enlarging up to the first full partition. By this way many partitions are omitted, yet enough polynomials are enlarged.

Algorithms 7.1, 7.2, 7.3 and 7.4 provide a detailed description of the MGB algorithm. The most important difference with  $MXL_3$  is the extra condition

$$|P_{=D-1}(x_1, \dots, x)| = |T_{=D-1}(x_1, \dots, x)|$$

in line 20 of the *Enlarge* subroutine (Algorithm 7.4). It means that all the terms of degree  $D - 1$  with leading variable  $\in x_1, \dots, x$  appear as leading terms in  $P$ . When the condition is met, the flag *newExtend* is set to true, which forces  $D$  to be increased in the next call to *Enlarge*.

Throughout the operation of the algorithm a degree bound  $D$  is used. This degree bound denotes the maximum degree of the polynomials contained in  $P$ . An elimination degree bound  $ED$  is used as a bound for the degrees of polynomials in  $P$  that are being eliminated.  $RREF(P, ED)$  means the reduced row echelon form of  $P_{\leq ED}$ . We mean by the new elements the set of all polynomials produced during the previous enlargement. The set  $M$  stores the mutants obtained during the *Echelonization* process. We define the array  $S$  to keep the the largest leading variable at each degree level. Note that the content of  $P$  is changed throughout the operation of the algorithm.

In the actual implementation we do not compute the full row reduced echelon form of the constructed matrix in each step. For the columns corresponding to highest degree terms we only clear entries under the pivot. For the rest of the columns we clear above and below the pivot. The idea of using the full reduction in this case is creating the vertical hole of the unextended partitions as explained in the previous section and illustrated in the next section.

The *Recover* function in line 6 of Algorithm 7.3 enlarges all partitions of  $P_{\leq ED}$  that were omitting during the enlargement process and echelonizes  $P$ . Also, it multiplies any mutants found.

The correctness of the MGB algorithm is guaranteed by Proposition 6.4, first stated and proved in [37]. The MGB algorithm terminates only when it returns the

---

**Algorithm 7.1** MGB

---

**Input:**  $P$  is a finite sequence from  $R$   
**Output:**  $G$  is a Gröbner basis of  $\langle P \rangle$ .

- 1:  $D = \max\{\deg(p) \mid p \in P\}$
- 2:  $ED = D$
- 3:  $M = \emptyset$
- 4:  $S = \{s_i = x_1 : 1 \leq i \leq D\}$
- 5:  $x = x_1$
- 6:  $newExtend = true$
- 7: **loop**
- 8:    $Echelonize(P, M, ED)$
- 9:    $G = Gröbner(P, M, D, ED)$
- 10: **if**  $G \neq \emptyset$  **then**
- 11:     **return**  $G$
- 12: **end if**
- 13:    $Enlarge(P, M, S, x, D, ED, newExtend)$
- 14: **end loop**

---



---

**Algorithm 7.2** Echelonize( $P, M, ED$ )

---

- 1: Consider each term in  $P$  as a new variable
- 2:  $P = RREF(P, ED)$
- 3:  $M = \{p \in P : P \text{ is a new element and } \deg(p) < ED \}$

---



---

**Algorithm 7.3** Gröbner( $P, M, S, D, ED$ )

---

- 1:  $G = \emptyset$
- 2: **if**  $M_{<ED} = \emptyset$  **and**  $(ED < D \text{ or } newExtend = true)$  **then**
- 3:    $s = S[ED - 1]$
- 4:   **if**  $|P_{=ED-1}| = |T_{=ED-1}(s, \dots, x_n)|$  **then**
- 5:     **if**  $s < x_1$  **then**
- 6:        $Recover(P, ED, S)$
- 7:     **end if**
- 8:      $G = P_{<ED}$
- 9:   **end if**
- 10: **end if**
- 11: **return**  $G$

---

---

**Algorithm 7.4** Enlarge( $P, M, S, x, D, ED, newExtend$ )

---

```

1: if  $M \neq \emptyset$  then
2:    $k = \min\{\deg(p) : p \in M\}$ 
3:   Select a necessary number of mutants  $NM$  from  $M_{=k}$ 
4:    $y = \max\{LV(p) : p \in NM\}$ 
5:   Multiply selected mutants by all variables  $\leq y$ 
6:   Remove the selected mutants from  $M$ 
7:   Add the new polynomials to  $P$ 
8:    $ED = k + 1$ 
9: else
10:  if  $newExtend$  then
11:     $D = D + 1$ 
12:     $x = \min\{LV(p) : p \in P_{=D-1}\}$ 
13:     $newExtend = false$ 
14:  else
15:     $x = \min\{LV(p) : p \in P_{=D-1} \text{ and } LV(p) > x\}$ 
16:  end if
17:   $S[D] = x$ 
18:  Multiply all  $p \in P_{D-1}$  that has leading variable  $x$  by all the variables  $\leq x$ 
  without redundancy
19:  Add the newly obtained polynomials to  $P$ 
20:  if  $|P_{=D-1}(x_1, \dots, x)| = |T_{=D-1}(x_1, \dots, x)|$  or  $(x = x_1)$  then
21:     $newExtend = true$ 
22:  end if
23:  Set  $ED = D$ 
24: end if

```

---

## 7 Flexible Partial Enlargement

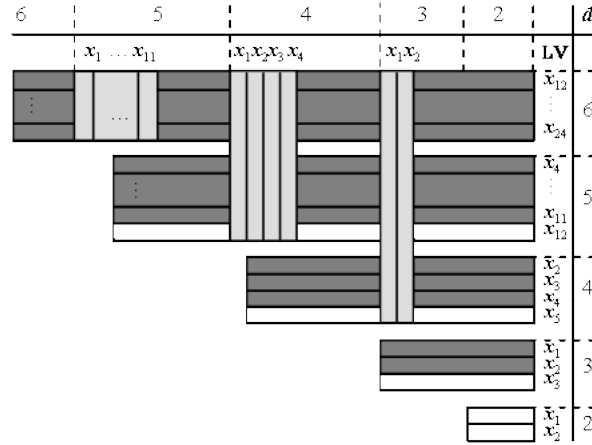


Figure 7.1: Behavior of the algorithm for a sequence of 24 degree 2 equations in 24 variables. Horizontal stripes represent variable-partitions, darker ones are full. Vertical stripes represent terms that do not appear in the given polynomials.

set  $G = P_{<ED}$  that is computed by the *Gröbner* subroutine. We are going now to prove that  $G$  is a Gröbner basis of the ideal generated by  $P$ .

The first if statement of Algorithm 7.3, line 2, guarantees the second condition of Proposition 6.4 since all the polynomials up to degree  $ED - 1$  are extended one degree more without producing any mutants ( $M_{<ED} = \emptyset$ ). The second if statement of Algorithm 7.3, line 4, represents a second difference from  $MXL_3$ . It means that all the terms of leading variable  $\in s, \dots, x_n$  appear as leading terms in  $P_{=Ed-1}$ . This will guarantee the first condition of Proposition 6.4 as follows.

In case of  $s = x_1$ ,  $P_{<ED}$  contains all terms of degree  $ED - 1$  as leading terms. In case of  $s < x_1$ , MGB needs to recover  $P_{\leq ED}$  as explained above. This leads to satisfying the two conditions of Proposition 6.4 since these unextended partitions have full rank.

Therefore MGB returns the set  $G = P_{<ED}$  that satisfies the two conditions of Proposition 6.4. Then it is a Gröbner basis. The worst case of MGB is to reproduce the  $MXL_3$  algorithm. So MGB terminates since  $MXL_3$  terminates. As an important note, for the experiments run so far, the recovering process was never necessary.

### 7.3 MGB in action

We describe the behavior of the MGB algorithm on a concrete example, a sequence of 24 degree-2 polynomials in 24 variables. We refer to Figure 7.1 for a schematic representation of the process.

After *Echelonization*, the leading variables of the original polynomials range from  $x_1$  to  $x_2$  as depicted at the bottom of Figure 7.1. Then, the  $x_1$  and  $x_2$  partitions

are enlarged and echelonized to obtain degree 3 polynomials with leading variables ranging from  $x_1$  to  $x_3$ . Here we encounter that the variable partitions for  $x_1$  and  $x_2$  are full and we represent it with the darker shading in Figure 7.1.

So in the next step, only the  $x_2$  and  $x_3$  partitions are enlarged to degree 4. After *Echelonization* we obtain polynomials of degree 4 with leading variables ranging from  $x_2$  to  $x_5$ . Note that since the two partitions  $x_1$  and  $x_2$  of degree 3 polynomials are full, no term with leading variable  $x_1$  or  $x_2$  of degree 3 appears in the degree 4 polynomials. This is depicted in Figure 7.1 with vertical stripes.

At degree 4, the variable partitions corresponding to  $x_2$ ,  $x_3$  and  $x_4$  are full, so only the  $x_4$  and  $x_5$  partitions are enlarged to degree 5. After *Echelonization*, we obtain polynomials of degree 5 with leading variables ranging from  $x_4$  to  $x_{12}$ . Note that by Proposition 7.1, no term with leading variable  $x_1$  of degree 4 appears in the degree 5 polynomials, and *Echelonization* clears  $x_2$  through  $x_4$ .

At degree 5, the variable partitions from  $x_4$  to  $x_{11}$  are full, so only the  $x_{11}$  and  $x_{12}$  partitions are enlarged to degree 6. In fact, once the  $x_{12}$  partition is enlarged and echelonized, mutants are produced and after a few steps of enlarging mutants and *echelonization*, we arrive at a situation in which the  $\text{MXL}_3$  criterion is satisfied and the algorithm terminates.

## 7.4 Experimental results

We present our experiments to compare the efficiency of MGB with both  $\text{MXL}_3$  and  $F_4$  algorithms. We tested them with random systems generated by Courtois [15] and HFE systems generated by the code of John Baena. We run all the experiments in a Sun X4440 server, with four “Quad-Core AMD Opteron™ Processor 8356” CPUs and 128 GB of main memory. Each CPU is running at 2.3 GHz. We used only one out of the 16 cores.

For  $F_4$ , we used Magma version V2.13-10 implementation of Faugère’s  $F_4$  algorithm which is considered the best available implementation of  $F_4$ . When we use the new version of Magma (V2.16), we found no big difference between them. the new version is worse in terms of memory, while it is a little bit faster. For both, the MGB algorithm and the  $\text{MXL}_3$  algorithm, we used our C++ implementation. For the *Echelonization* step, we used an adapted version of M4RI [2], the dense matrix linear algebra over  $\mathbb{F}_2$  library. Our adaptation was in changing the strategy of selecting the pivot during Gaussian elimination to keep the old elements in the system intact. We use the M4RI method that has complexity  $O(n^3/\log n)$  [2].

Tables 7.1 and 7.2 show the main experiments of dense random systems with many solutions and the experiments of HFE systems of univariate degree 288, respectively. We denote the number of variables and equations by  $n$  and the highest degree of the iteration steps by  $D$ . We also show the maximum matrix size. It is evident from Table 7.1 and 7.2 how the new strategy improves  $\text{MXL}_3$ .

Figure 7.2 displays a comparison between MGB,  $\text{MXL}_3$  and  $F_4$  in terms of space and time. It is clear from Figure 7.2(a) that the MGB algorithm uses less memory

## 7 Flexible Partial Enlargement

	F <sub>4</sub>		MXL <sub>3</sub>		MGB	
<i>n</i>	<i>D</i>	max. matrix	<i>D</i>	max. matrix	<i>D</i>	max. matrix
24	6	207150×78637	6	50367×57171	6	26409×33245
25	6	248495×108746	6	66631×76414	6	37880×47594
26	6	298592×148804	6	88513×102246	6	55063×67815
27	6	354189×197902	6	123938×140344	6	92296×99518
28	6	420773×261160	6	201636×197051	6	132918×148976
29	6	499222×340254	6	279288×281192	6	173300×224941
30	6	1283869×374081	6	332615×351537	6	265298×339236
31	6	868614×489702	6	415654×436598	6	349778×381382
32	ran out of memory		ran out of memory		7	437172×507294

Table 7.1: Experiments for dense random systems

	F <sub>4</sub>		MXL <sub>3</sub>		MGB	
<i>n</i>	<i>D</i>	max. matrix	<i>D</i>	max. matrix	<i>D</i>	max. matrix
30	5	149532×136004	5	86795×130211	5	68468×109007
35	5	200302×321883	5	155914×296872	5	116737×254928
36	5	219438×382252	5	173439×344968	5	125133×297503
37	5	247387×444867	5	192805×399151	5	142460×345635
38	5	274985×512311	5	212271×459985	5	153181×399855
39	5	305528×588400	5	234111×528068	5	171985×460727
40	ran out of memory		5	258029×604033	5	192506×528849
49	ran out of memory		5	561972×1765465	5	371368×1584984
50	ran out of memory		ran out of memory		5	382392×1766691
51	ran out of memory		ran out of memory		5	410169×1964756

Table 7.2: Experiments for HFE(288,n) systems

than both MXL<sub>3</sub> and Magma's F<sub>4</sub> since it constructs smallest matrices. However, it is not much faster than MXL<sub>3</sub> in terms of the size of the system becomes bigger as shown in Figure 7.2(b). The reason is that the new algorithm uses a row-reduced echelon form while MXL<sub>3</sub> uses only the row echelon form. Also, the gap between MGB and MXL<sub>3</sub> becomes a little smaller as the size of the system increased.

Table 7.3 shows the detailed result of computing a Gröbner basis of a dense random system with 32 variables by MGB. For each step we give the degree (*D*), the matrix size, the rank of the matrix (Rank), a set of leading variable of the level partitions, the number of variables in the degree *D* terms (*n<sub>D</sub>*), the number of lowest degree mutants found (NM), the number of used mutants (UM), and finally the lowest degree of mutants found (MD).

Table 7.3 explains how the MGB algorithm works. As the degree *D* is going up the number of variables in degree *D* terms goes down. Starting from step 3, the number

## 7 Flexible Partial Enlargement

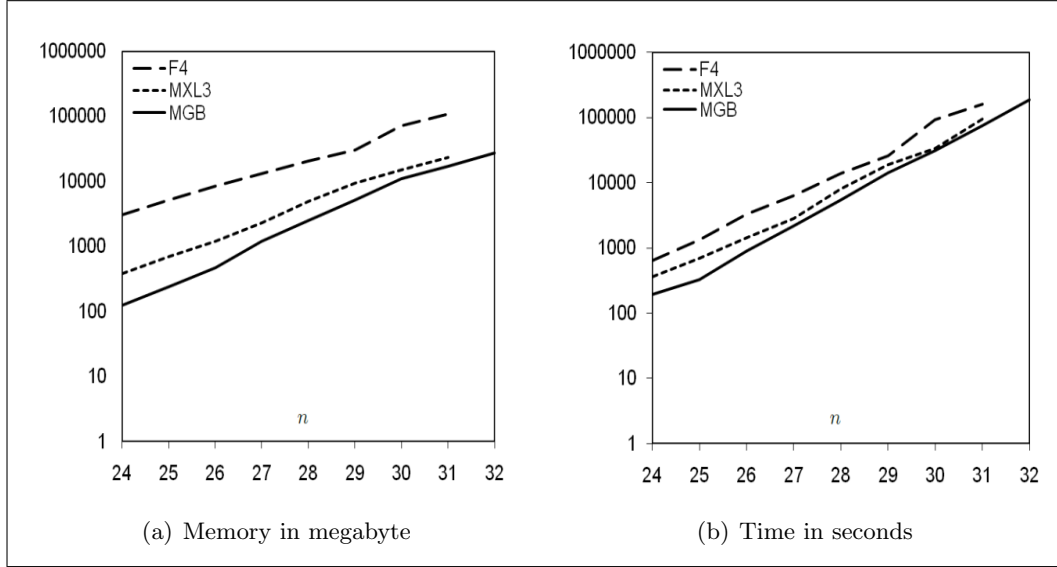


Figure 7.2: Comparison between MGB, MXL<sub>3</sub>, and F<sub>4</sub> for dense random systems

Step	D	Matrix Size	Rank	partitions	$n_D$	NM	UM	MD
1	2	$32 \times 529$	32	$\{x_1, x_2\}$	32	0	0	-
2	3	$1056 \times 5489$	1056	$\{x_1, x_2, x_3\}$	32	0	0	-
3	4	$11798 \times 36954$	11776	$\{x_2, x_3, x_4\}$	31	0	0	-
4	5	$93534 \times 179460$	91378	$\{x_3, \dots, x_7\}$	30	0	0	-
5	6	$389286 \times 475470$	372679	$\{x_6, \dots, x_{16}\}$	27	0	0	0
6	7	<b><math>437172 \times 507294</math></b>	437172	$\{x_{15}, \dots, x_{26}\}$	18	21445	2158	5
7	6	$305685 \times 314056$	305685	$\{x_9, \dots, x_{27}\}$	24	18589	199	4
8	5	$175490 \times 179460$	175490	$\{x_3, \dots, x_{28}\}$	30	3910	16	3
9	4	$36875 \times 36954$	36875	$\{x_2, \dots, x_{29}\}$	31	6	1	1
10	2	$535 \times 529$	528	$\{x_1, \dots, x_{31}\}$	32	25	0	1

Table 7.3: Results for the system Random-32 with the MGB algorithm

of variables of degree 4 terms starts to decrease. At step 6 the degree of the system reaches 7 by extending only two partitions of degree 6 polynomials  $(x_{15}, x_{16})$ , while the number of variables in degree 7 terms is only 18. The system starts to generate mutants. It generates 21445 mutants of degree 5. Only 2158 of them are used. All of these mutants have leading variable  $x_9$ . So by multiplying them with variables  $\geq x_9$ , we have new polynomials of degree 6 with leading variables at most  $x_9$ . We do not multiply mutants by  $x_6, x_7$ , and  $x_8$  since their partitions are not needed in the Gaussian elimination of step 7. This leads to decreasing the dimension of the matrix of the step. The system continuously generates low degree mutants until 6 linear mutants are produced at step 9. Another 25 linear ones are generated at step 10. By multiplying all mutants of degree  $\leq 2$ , the system does not produce more mutants which in turn leads to a Gröbner basis of the ideal generated by the initial 32 polynomials.

The experiments, presented in this section, confirm that the MGB algorithm is substantially better than  $\text{MXL}_3$  and  $\text{F}_4$  algorithms in both randomly generated and HFE systems and the experiment data also suggests that the complexity of the new algorithm challenges known theoretical estimates. In the next section, we present a preliminary complexity study for MGB that aim to a tight upper bound for the complexity of Computing Gröbner basis and solving polynomial systems.

## 7.5 Complexity bounds

In order to study the complexity of solving multivariate polynomial systems in the general case, we study the behavior of MGB when it solves random systems. This study based on solving a large number of random systems using both MGB and XL algorithms. We study the structure of enlarged systems produced during the process of these two algorithms. This study brought us with the following promising observation. Let  $P = \{p_1, \dots, p_m\}$  be a set of randomly generated quadratic polynomials in  $B$  and let the two sets  $H_d$  and  $H_d^+$  are defined on  $P$  as in (2.4) and (4.1), respectively. Let XL enlarges the set  $P$  up to a maximal degree  $d$ , as we described in Algorithm 2.1, which constructs  $H_d$ . After the  $\text{Echelonize}(H_d)$ , for each  $2 \leq k < d$ , the set of degree  $k$  polynomials,  $H_k^+$ , is filled as follows: Polynomials that have leading variable  $x_1$  is filled first, followed by polynomials that have leading variable  $x_2$ , and so on. This observation leads us to the following preliminary complexity study.

Let  $\text{LVT}(d, x_k) \subset B$  be the set of terms of degree  $d$  that has leading variables  $\geq x_k$ . The number of the elements of  $\text{LVT}(d, x_k)$  can be computed as

$$|\text{LVT}(d, k)| = \sum_{i=1}^k \binom{n-i}{d-1}, k \leq n \quad (7.2)$$

Let  $I_d$  be the number of linearly independent polynomials of degree  $\leq d$  generated

## 7 Flexible Partial Enlargement

$n$	Num_polys	partitions	<i>full</i>
24	41376	$x_1, \dots, x_{12}$	11
25	49375	$x_1, \dots, x_{10}$	9
26	58474	$x_1, \dots, x_9$	8
27	68769	$x_1, \dots, x_8$	7
28	80360	$x_1, \dots, x_8$	7
29	93351	$x_1, \dots, x_7$	6
30	107850	$x_1, \dots, x_7$	6
31	123969	$x_1, \dots, x_7$	6
32	141824	$x_1, \dots, x_7$	6
33	161535	$x_1, \dots, x_7$	6
34	183226	$x_1, \dots, x_7$	6

Table 7.4: Partitions generated by XL at degree 5 for Random systems

by XL, as defined in (3.7). Let  $x_k$  satisfy the following

$$|\text{LVT}(d, x_{k-1})| < I_d < |\text{LVT}(d, x_k)|. \quad (7.3)$$

Based on the assumption mentioned above, if the number of linearly independent polynomials of degree  $d$  is sufficient to fill  $k$  partitions with leading variables  $(x_1, \dots, x_k)$ , then at least  $k - 1$  of them are *full*.

Table 7.4 shows the number of linearly independent polynomials (Num\_polys) and the partitions generated by XL at degree 5 for random systems of size 24 ... 34. All partitions are *full* except the last one for each system.

Now we are going to study the complexity of the MGB algorithm in case of solving random systems. We need the following definition

**Definition 7.2.** *Let  $H$  be a set of polynomials such that all elements in  $H$  have degree  $d$ . Let*

$$H_x = \{h \in H \mid \text{LV}(h) = x\}$$

*and  $F_x$  be the set produced by enlarging  $H_x$  one degree more as follows*

$$F_x = \{y \cdot p : p \in H_x, y \leq x\}.$$

*We called  $H_x$  a leading variable reduced partition (LVR-partition), if there is  $q \in \tilde{F}_x$  such that  $\text{LV}(q) < x$ , where  $\tilde{F}_x$  is a row echelon form of  $F_x$ .*

Let  $x_j$  be the smallest leading variable with non empty partition  $(\tilde{H}_{d-1})_{x_j}$  that is *full*. Experimentally, we found that only the partitions that have leading variable  $\leq x_j$  are *LVR-partitions*. The MGB algorithm enlarges only that *LVR-partitions* of degree  $d - 1$  to generate the degree  $d$  polynomials of the extended system, while XL enlarges all partitions. We have noticed that, without using any mutants MGB generates all partitions of degree  $d$  polynomials with leading variable  $\leq x_j$  as well

## 7 Flexible Partial Enlargement

as XL generates. This leads to solving the system by MGB at the same degree as XL with fewer equations and fewer terms.

Table 7.5 compares the partitions generated by XL and MGB. For example, in the case of  $n = 24$ , at degree 4, XL has  $x_1, \dots, x_5$  partitions. It enlarges all of them up to degree 5. We obtain 12 partitions that are filled with 41376 polynomials. MGB has  $x_2, \dots, x_5$  partitions at degree 4 only  $x_4, x_5$  are *LVR – partitions*. So MGB enlarges only these two partitions. It generates 9 partitions at degree 5, which means that MGB skip  $x_1, x_2, x_3$  from the degree 5. MGB fills the 9 partitions by 19221 polynomials. By adding the number of skipped polynomials to them, we have exactly the same as XL generated at degree 5 (41376). Which explains that MGB generate all new partitions as well as XL.

$n$	XL			MGB		
	Num_polys	partitions	<i>full</i>	Num_polys	partitions	<i>full</i>
24	41376	$x_1, \dots, x_{12}$	11	19221	$x_4, \dots, x_{12}$	8
25	49375	$x_1, \dots, x_{10}$	9	22579	$x_4, \dots, x_{10}$	6
26	58474	$x_1, \dots, x_9$	8	26343	$x_4, \dots, x_9$	5
27	68769	$x_1, \dots, x_8$	7	30543	$x_4, \dots, x_8$	4
28	80360	$x_1, \dots, x_8$	7	35210	$x_4, \dots, x_8$	4
29	93351	$x_1, \dots, x_7$	6	40376	$x_4, \dots, x_7$	3
30	107850	$x_1, \dots, x_7$	6	46074	$x_4, \dots, x_7$	3
31	123969	$x_1, \dots, x_7$	6	52338	$x_4, \dots, x_7$	3
32	141824	$x_1, \dots, x_7$	6	82954	$x_3, \dots, x_7$	4
33	161535	$x_1, \dots, x_7$	6	94110	$x_3, \dots, x_7$	4
34	183226	$x_1, \dots, x_7$	6	106346	$x_3, \dots, x_7$	4

Table 7.5: Compare Partitions generated by XL and MGB at degree 5 for Random systems

Let us use formula (3.5) to compute the number of linearly independent polynomials by XL at certain degree  $d$ . By filling the partitions  $x_1, x_2, \dots, x_n$  (in this order). In case of the Semi-Regular systems, we fill all possible monomials that has leading variable  $x_1$  first, followed by the  $x_2$  partition and so on. In this way, we can compute the number of partitions generated by XL at this degree. Also, in this way we can determine the number of *full* partitions and the *LVR – partitions*. Then we can estimate the avoided partitions at each degree level.

Figure 7.3 explains the number of avoided variable partitions during the enlargement process (the number of unexpanded partitions). MGB enlarges different systems with size  $n \in (5, \dots, 65)$ . For example, in case of  $n = 50$ , MGB avoids (1, 2, 4, 6, 9, 17) variable partitions from the enlarged system at degree 9. This affects the size of the matrix generated by MGB, which is much smaller than the XL algorithm, even if we could represent only the linearly independent rows.

Based on the results in [4], the degree of regularity is the maximal degree appeared

## 7 Flexible Partial Enlargement

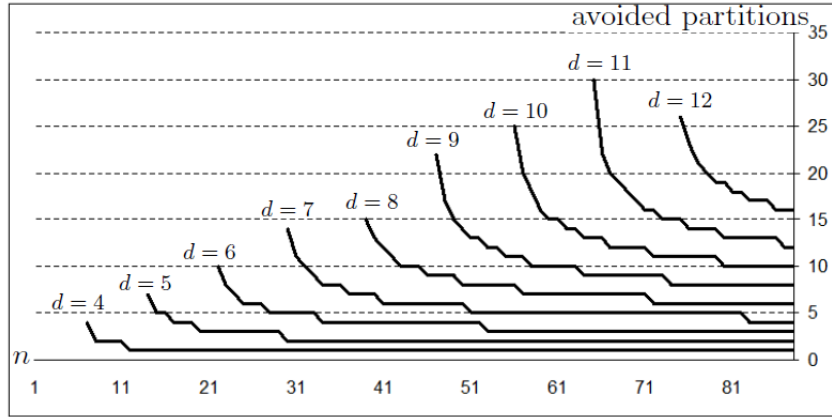


Figure 7.3: MGB different degree level number of variables.

in the computation of Gröbner bases. In this case, the size of the system can be upper bounded by the dimensional matrix that has a number of columns is equal to the number of terms up to the degree of regularity. The maximal number of linearly independent polynomials generated by XL up to the degree of regularity ( $I^{xl}$ ) is bounded from above by the number of terms as follows:

$$I^{xl} \leq \sum_{d=1}^{d_{reg}} \binom{n-1}{d}. \quad (7.4)$$

The MGB algorithm puts in question the sharpness of this bound. As we show in Figure 7.3, MGB avoids the full partitions from the enlarging step. For example, in case of  $n = 50$ , MGB avoids  $(1, 2, 4, 6, 9, 17)$  variable partitions from the enlarged system for the degree respectively. The complexity of MGB depends not only on the highest degree  $d$  of the system and the number of variables  $n$ , but also on the sequence  $n_1, n_2, \dots, n_d$  of variables appeared at each degree  $d$ . The maximal number of of linearly independent polynomials generated by MGB ( $I^{mgb}$ ) as follows:

$$I^{mgb} \leq \sum_{d=1}^{d_{reg}} \binom{n_d-1}{d}. \quad (7.5)$$

Now we are going to estimate the value of  $n_d$  for each degree  $d \leq d_{reg}$  without any reduction in the degree. The number of linearly independent polynomials generated by XL ( $I_d$ ) is computed by (3.5). Let  $x_{k_d} = \min\{\text{LV}(f) \mid f \in H_d\}$ ; the least leading variable of polynomials in  $H_d^+$ . The value of  $x_{k_d}$  can be estimated using the following formula.

$$x_{k_d} := \min\{x_i \mid (I_d - \sum_{j=1}^i \text{LV}(T_d, x_j)) \geq 0\}. \quad (7.6)$$

## 7 Flexible Partial Enlargement

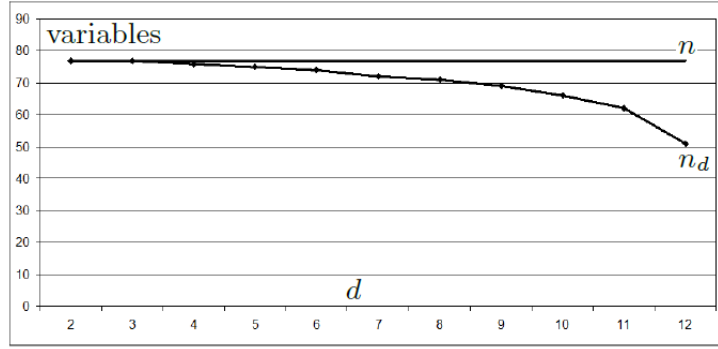


Figure 7.4: Compares  $n$  to  $n_d$  for a random system of size  $n = 77$ .

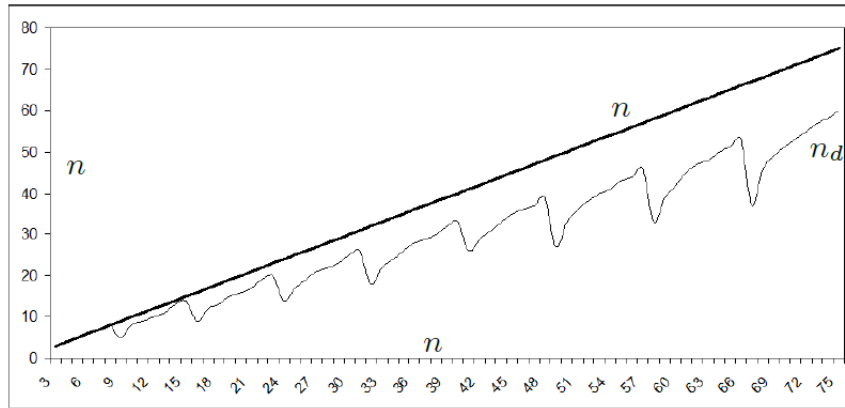


Figure 7.5: Compares  $n$  to  $n_d$  for a random system of sizes  $n = 3 \dots 77$  at the degree of regularity.

The value of  $x_{k_d}$  denotes the largest leading variable appeared at degree  $d + 1$ . In this case, the value of  $n_d$  is evaluated as follows:

$$n_d := n - k_{d-1} + 1. \tag{7.7}$$

Figure 7.4 shows the number of variables that are used during the process at each degree level  $d$ . We estimate the number of omitted partitions as explained above of a random system with size  $n = 77$ .

Figure 7.5 shows another kind of comparison to explain more the MGB improvements. It compares the initial number of variables  $n$  to the number of variables appeared in the polynomials of degree equal to  $d_{reg}$  generated by MGB. This analysis can be also used to estimate the complexity of the  $MXL_3$  algorithm. It is clear from the figure that as the initial number of variables ( $n$ ) of the system increased the number of avoided partitions ( $n - n_d$ ) is also increased.

To have more explanations about the efficiency of the MGB algorithm, we need to

## 7 Flexible Partial Enlargement

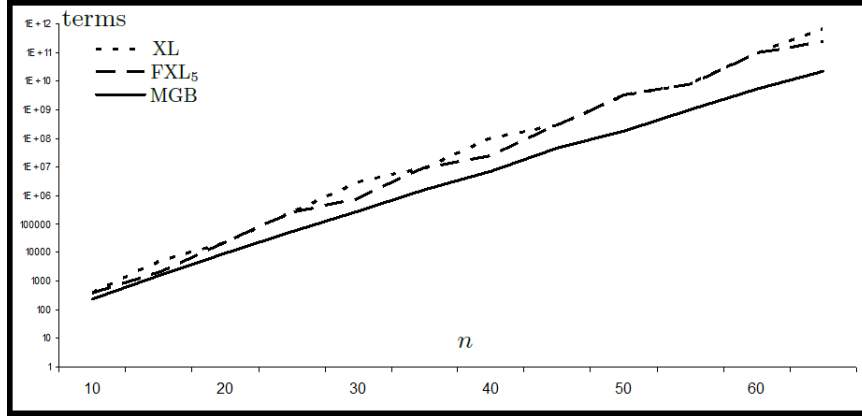


Figure 7.6: Experimental results compared with the number of terms at the degree of regularity.

define an algorithm called  $\text{FXL}_5$  algorithm. This algorithm is a variant of XL that generates only the linearly independent polynomials during the enlargement step using the advanced criteria of the  $F_5$  algorithm [25]. We assume that the maximal degree by  $\text{FXL}_5$  and MGB is the degree of regularity. Figure 7.6 compares the total number of terms generated in the system up to the degree of regularity between MGB and  $\text{FXL}_5$  for random systems of size  $n \in (10, 15, \dots, 65)$ . We compare these results with the total number of terms generated by XL.

As we see from the figure, the total number of terms appeared during the computation of a Gröbner basis is decreased when we use MGB. Since the maximal number of linearly independent polynomials generated to compute a Gröbner basis is bounded from above by the total number of terms, then the computations of MGB lead to a new tight upper bound for the complexity of computing Gröbner basis and solving polynomial systems.

## 8 Conclusion and Future Work

This thesis introduces several strategies for improving the XL algorithm, dramatically reducing the maximum matrix size produced by XL, and presents four algorithms: MutantXL,  $MXL_2$ ,  $MXL_3$ , and MGB.

MutantXL is the first algorithm which applies the Ding's concept of *mutant* to XL. It enlarges the system using the lowest degree mutants instead of enlarging the highest degree polynomials. MutantXL can indeed outperform XL and can solve multivariate systems at a lower degree than the usual XL algorithm. Moreover, an adapted version of MutantXL is used to attack the MQQ cryptosystem faster and uses less memory than attacks using  $F_4$ .

The  $MXL_2$  algorithm is an efficient improvement to MutantXL. It develops a partial enlargement strategy that reduces the size of the matrix constructed by MutantXL and still able to solve the system. This yields solving larger systems than MutantXL and outperforms  $F_4$  for many cases in terms of memory consumption.

The  $MXL_3$  algorithm introduces a new and efficient method for computing Gröbner bases on the Boolean polynomial ring. The experiments show that, both in classical cryptographic challenges and random systems,  $MXL_3$  performs better in terms of time and memory than the Magma' implementation of  $F_4$ , currently the best publicly available implementation of  $F_4$ . Our experimental results demonstrate the importance of both mutant and partial enlargement strategies for the computation of Gröbner bases. Also, the new criterion for determining the termination of  $MXL_3$ , proved to be efficiently checkable and sharp to detect a Gröbner basis.

Finally, the MGB algorithm is presented as a new algorithm for computing Gröbner bases which uses a new flexible partial enlargement strategy to avoid computing polynomials at different degrees. Our experiments confirm that MGB is substantially better than  $MXL_3$  and  $F_4$  algorithms in both random and HFE systems and the experimental data also suggests that the complexity of MGB challenges known theoretical estimates. Our preliminary complexity analysis of MGB suggested that this new strategy may change substantially our thinking of the hardness of computing Gröbner bases. This new strategy may leads to new paradigms in the computation of solving multivariate polynomial systems.

In summary, the results presented in this thesis demonstrate that the proposed strategies use the simple structure of XL and introduce several new algorithms that reduce time and memory consumption of the computation of solving multivariate polynomial systems and still having the same simple structure.

## **Future Work**

The results in this thesis bring us with the following research ideas:

1. We showed how the concept of mutant started and enriched several improvements introduced in this thesis. However, studying the impact of this concept on the complexity of attacking symmetric cryptosystems is an important research problem.
2. The results that can be obtained when we combine our improved strategies with other Gröbner basis algorithms, such as  $F_4$ , is an interested research question.
3. Finally, and not least, our observation of the complexity of solving multivariate systems using MGB provides a new upper bound that can be used to evaluate the security of many cryptographic primitives for awhile. However, the theoretical proof of this observation is still an open research problem.

## Bibliography

- [1] M. Albrecht. *Algorithmic Algebraic Techniques and their Application to Block Cipher Cryptanalysis*. Phd thesis, Royal Holloway, University of London, 2010.
- [2] M. Albrecht and G. Bard. M4RI – linear algebra over  $\text{GF}(2)$ . <http://m4ri.sagemath.org/index.html>, 2008.
- [3] G. V. Bard. Accelerating cryptanalysis with the Method of Four Russians. Report 251, Cryptology ePrint Archive, 2006.
- [4] M. Bardet, J.-C. Faugère, B. Salvy, and B.-Y. Yang. Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems. In *MEGA '05*, 2005. Eighth International Symposium on Effective Methods in Algebraic Geometry, Porto Conte, Alghero, Sardinia (Italy), May 27th – June 1st.
- [5] M. T. Bardet. *Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie*. Phd thesis, Université Paris VI, D`ecembre, 2004.
- [6] T. Becker, H. Kredel, and V. Weispfenning. *Gröbner bases: a computational approach to commutative algebra*. Springer-Verlag, London, UK, 0 edition, April 1993.
- [7] D. J. Bernstein, J. Buchmann, and E. Dahmen. *Post Quantum Cryptography*. Springer Publishing Company, Incorporated, 1st edition, 2008.
- [8] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal (An Algorithm for Finding the Basis Elements in the Residue Class Ring Modulo a Zero Dimensional Polynomial Ideal)*. PhD thesis, Mathematical Institute, University of Innsbruck, Austria, 1965. (English translation in *Journal of Symbolic Computation*, 2004).
- [9] B. Buchberger. A criterion for Detecting Unnecessary Reductions in the Construction of Gröbner Bases. volume 72, pages 3–21, London, UK, 1979. Johannes Kepler University Linz, Springer, Berlin - Heidelberg - New York.
- [10] J. Buchmann, S. Bulygin, J. Ding, W. S. A. E. Mohamed, and F. Werner. Practical Algebraic Cryptanalysis for Dragon-Based Cryptosystems. In *Proceedings of the 9th International Conference on Cryptology And Network Security, (CANS10)*, volume 6467 of *Lecture Notes in Computer Science*, pages 140–155, Kuala Lumpur, Malaysia, December 2010. Springer-Verlag, Berlin.

## Bibliography

- [11] J. Buchmann, D. Cabarcas, J. Ding, and M. S. E. Mohamed. Flexible Partial Enlargement to Accelerate Gröbner Basis Computation over  $\mathbb{F}_2$ . In *Proceedings of the 3rd Africacrypt Conference, (Africacrypt 2010)*, volume 6055 of *Lecture Notes in Computer Science*, pages 69–81, Stellenbosch, South Africa, May 2010. Springer-Verlag, Berlin.
- [12] N. Courtois. Algebraic attacks over  $\text{gf}(2^k)$ , application to hfe challenge 2 and sflash-v2. In *Public Key Cryptography*, pages 201–217, 2004.
- [13] N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In *Proceedings of International Conference on the Theory and Application of Cryptographic Techniques(EUROCRYPT)*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407, Bruges, Belgium, May 2000. Springer.
- [14] N. Courtois and J. Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, ASIACRYPT '02*, pages 267–287, London, UK, UK, 2002. Springer-Verlag.
- [15] N. T. Courtois. Experimental algebraic cryptanalysis of block ciphers. <http://www.cryptosystem.net/aes/toyciphers.html>, 2007.
- [16] N. T. Courtois and J. Patarin. About the xl algorithm over  $\text{gf}(2)$ . In *Proceedings of the 2003 RSA conference on The cryptographers' track, CT-RSA'03*, pages 141–157, Berlin, Heidelberg, 2003. Springer-Verlag.
- [17] D. A. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra, 3/e (Undergraduate Texts in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
- [18] C. Diem. The XL Algorithm and a Conjecture from Commutative Algebra. In *Proceedings of 10th International Conference on the Theory and Application of Cryptology and Information Security, (ASIACRYPT)*, volume 3329 of *Lecture Notes in Computer Science*, pages 323–337, Jeju Island, Korea, December 2004. Springer Berlin / Heidelberg.
- [19] J. Ding. Mutants and its impact on polynomial solving strategies and algorithms. Privately distributed research note, University of Cincinnati and Technical University of Darmstadt, 2006, 2006.
- [20] J. Ding, J. Buchmann, M. S. E. Mohamed, W. S. A. Moahmed, and R.-P. Weinmann. MutantXL. In *Proceedings of the 1st international conference on Symbolic Computation and Cryptography (SCC08)*, pages 16 – 22, Beijing, China, April 2008. LMIB.

## Bibliography

- [21] J. Ding, D. Carbarcas, D. Schmidt, J. Buchmann, and S. Tohaneanu. Mutant Gröbner Basis Algorithm. In *Proceedings of the 1st international conference on Symbolic Computation and Cryptography (SCC08)*, pages 23 – 32, Beijing, China, April 2008. LMIB.
- [22] J. Ding, J. E. Gower, and D. Schmidt. *Multivariate Public Key Cryptosystems (Advances in Information Security)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
- [23] H. Dobbertin. One-to-one highly nonlinear power functions on  $\text{gf}(2^n)$ . *Appl. Algebra Eng. Commun. Comput.*, 9(2):139–152, 1998.
- [24] J.-C. Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Pure and Applied Algebra*, 139(1-3):61–88, June 1999.
- [25] J.-C. Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Proceedings of the 2002 international symposium on Symbolic and algebraic computation (ISSAC)*, pages 75 – 83, Lille, France, July 2002. ACM.
- [26] J.-C. Faugère and A. Joux. Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases. In *Proceedings of the 23rd Annual International Cryptology Conference Advances in Cryptology - CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 44–60, Santa Barbara, California, USA, August 2003. Springer.
- [27] J.-C. Faugère, R. S. Ødegård, L. Perret, and D. Gligoroski. Analysis of the mqq public key cryptosystem. In *Cryptology and Network Security - 9th International Conference, CANS 2010, Kuala Lumpur, Malaysia, December 12-14, 2010. Proceedings*, volume 6467, pages 169–183, 2010.
- [28] A. Fraenkel and Y. Yesha. Complexity of problems in games, graphs and algebraic equations. In *Discrete Applied Mathematics*, volume 1, pages 15–30, 1979.
- [29] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.
- [30] D. Gligoroski. Candidate One-Way Functions and One-Way Permutations Based on Quasigroup String Transformations. Report 352, Cryptology ePrint Archive, 2005.
- [31] D. Gligoroski, S. Markovski, and S. J. Knapskog. Multivariate Quadratic Trapdoor Functions Based on Multivariate Quadratic Quasigroups. In *Proceedings of The AMERICAN CONFERENCE ON APPLIED MATHEMATICS, (MATH08)*, Lecture Notes in Computer Science, Cambridge, Massachusetts, USA, March 2008.

## Bibliography

- [32] A. Kipnis and A. Shamir. Cryptanalysis of the hfe public key cryptosystem by relinearization. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology, CRYPTO '99*, pages 19–30, London, UK, 1999. Springer-Verlag.
- [33] Z. Kokosinški. Algorithms for unranking combinations and their applications. In *Proceedings of 7th Int. Conference on Parallel and Distributed Computing and Systems, PDCS*, pages 216–224, Washington D.C., USA, IASTED, 1995.
- [34] D. Lazard. Gröbner-bases, gaussian elimination and resolution of systems of algebraic equations. In *Proceedings of the European Computer Algebra Conference on Computer Algebra*, pages 146–156, London, UK, 1983. Springer-Verlag.
- [35] T. Matsumoto and H. Imai. Public Quadratic Polynomial-Tuples for Efficient Signature-Verification and Message-Encryption. In *Workshop on the Theory and Application of Cryptographic Techniques Advances in Cryptology- EUROCRYPT*, volume 330 of *Lecture Notes in Computer Science*, pages 419–453, Davos, Switzerland, May 1988. Springer.
- [36] W. S. A. Moahmed, J. Ding, T. Kleinjung, S. Bulygin, and J. Buchmann. WXL: Widemann-XL Algorithm for Solving Polynomial equations over GF(2). In *Proceedings of the 2nd international conference on Symbolic Computation and Cryptography (SCC10)*, pages 89 – 100, London, UK, June 2010.
- [37] M. S. E. Mohamed, D. Cabarcas, J. Ding, J. Buchmann, and S. Bulygin. MXL3: An efficient algorithm for computing gröbner bases of zero-dimensional ideals. In *Proceedings of The 12th international Conference on Information Security and Cryptology (ICISC 2009)*, volume 5984 of *Lecture Notes in Computer Science*, pages 87–100, Seoul, Korea, December 2010. Springer-Verlag, Berlin.
- [38] M. S. E. Mohamed, J. Ding, and J. Buchmann. Towards algebraic cryptanalysis of hfe challenge 2 using MXL3. In *To appear in Proceedings of The 5th International Conference on Information Security and Assurance (ISA 2011)*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 2011.
- [39] M. S. E. Mohamed, J. Ding, J. Buchmann, and F. Werner. Algebraic Attack on the MQQ Public Key Cryptosystem. In *Proceedings of the 8th International Conference on Cryptology And Network Security, (CANS09)*, volume 5888 of *Lecture Notes in Computer Science*, pages 392–401, Kanazawa, Ishikawa, Japan, December 2009. Springer-Verlag, Berlin.
- [40] 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 *Proceedings of The Second international Workshop on Post-Quantum Cryptography (PQCrypto08)*, Lecture Notes in Computer Science, pages 203–215, Cincinnati, USA, October 2008. Springer-Verlag, Berlin.

## Bibliography

- [41] H. M. Möller, T. Mora, and C. Traverso. Gröbner bases computation using syzygies. In *Papers from the international symposium on Symbolic and algebraic computation*, ISSAC '92, pages 320–328, New York, NY, USA, 1992. ACM.
- [42] J. Patarin. Cryptanalysis of the matsumoto and imai public key scheme of eurocrypt'88. In *Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '95, pages 248–261, London, UK, 1995. Springer-Verlag.
- [43] J. Patarin. Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of Asymmetric Algorithms. In *Proceeding of International Conference on the Theory and Application of Cryptographic Techniques Advances in Cryptology- Eurocrypt*, volume 1070 of *Lecture Notes in Computer Science*, pages 33–48, Saragossa, Spain, May 1996. Springer.
- [44] L. Robbiano. Computer and commutative algebra. In *Proceedings of the 6th International Conference, on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 31–44, London, UK, 1989. Springer-Verlag.
- [45] S. Rønjom and H. Raddum. On the Number of Linearly Independent Equations Generated by XL. In *Sequences and Their Applications - SETA 2008*, pages 239–251, Lexington, Kentucky, USA, 2008. Springer-Verlag, Berlin.
- [46] C. Shannon. Communication Theory of Secrecy Systems. In *Bell System Technical Journal*, volume 28, pages 656–715, 1949.
- [47] E. Thomae and C. Wolf. Unravel xl and its variants. Cryptology ePrint Archive, Report 2010/596, version, 2010.
- [48] C. Wolf and B. Preneel. Taxonomy of public key schemes based on the problem of multivariate quadratic equations. Cryptology ePrint Archive, Report 2005/077, 2005.
- [49] B.-Y. Yang and J.-M. Chen. Theoretical Analysis of XL over Small Fields. In *Proceedings of 9th Australasian Conference on Information Security and Privacy (ACISP)*, volume 3108 of *Lecture Notes in Computer Science*, pages 277–288, Sydney, Australia, July 2004. Springer.
- [50] B.-Y. Yang, J.-M. Chen, and N. Courtois. On asymptotic security estimates in xl and gröbner bases-related algebraic cryptanalysis. In *ICICS*, pages 401–413, 2004.
- [51] B.-Y. Yang, O. C.-H. Chen, D. J. Bernstein, and J.-M. Chen. *Analysis of QUAD*, pages 290–308. Springer-Verlag, Berlin, Heidelberg, 2007.