TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computing Science

MASTER'S THESIS

Algebraic Attacks from a Gröbner Basis Perspective

by

A.J.M. Segers

Eindhoven, October 2004

Supervisor:
Prof. dr. ir. H.C.A. van Tilborg

Advisors:
Dr. B.M.M. de Weger
Drs. G. Schmitz
Dr. ir. P.A.H. Bours

Abstract

Recently, a special kind of cryptanalysis coined as the algebraic attack has gained a lot of attention. In this thesis, we clarify this attack and discuss the threat to common ciphers. Among the known attacks, one can roughly distinguish between two classes. The first consists of structural attacks that focus on specific properties of a certain cipher. The second includes inversion attacks, which are general purpose algorithms that solve multivariate systems of equations. In this thesis we focus on the latter.

The different methods appear to be reducible to Gröbner Basis techniques, an area of algebraic geometry that is understood reasonably well. This report focusses on three topics. Firstly, we introduce a variety of common ciphers and discuss advanced techniques based on Gröbner Bases to cryptanalyze these ciphers. These are implemented in Magma and presented to the reader. Secondly, the much discussed cryptanalytic tool XL is shown to be related to methods based on Gröbner Bases. And lastly, this report discusses a complexity approximation of XL based on Hilbert series.

Preface

This report is the result of my graduation project in completion of the Master of Science program Industrial and Applied Mathematics at the Eindhoven University of Technology. The project has been carried out at the Netherlands National Communications Security Agency, which is part of the General Intelligence and Security Service in Leidschendam.

The algebraic attack has drawn a lot of attention in recent years and is still a very active area of research in cryptology. Papers that discuss the potential threat to cryptosystems range from very thorough to sketchy. This means that it is hard to get a complete overview on the matter. This thesis aims to explain algebraic attacks from the ground up, while paying attention to both the theoretical and the practical aspects.

The thesis shows that algebraic attacks are strongly related to Gröbner Basis techniques and studies the algorithms F4, F5 and XL. These are regarded as some of the most promising tools to solve an algebraic system coming from cryptography but their complexity has not yet been understood completely.

The contribution of this thesis consists of three parts. Firstly, the complexity of various algebraic attacks are illustrated on the basis of examples. Secondly, the relation between XL and Gröbner Basis techniques is clarified and made explicit. We show that the algorithm XL is a cumbersome way to compute Gröbner Bases. However, it does provide the cryptanalyst with a tool to distinguish some algebraic systems, like HFE, that are easier to solve than random systems. Lastly, we discuss a complexity estimation of XL based on Hilbert series and place some remarks about its applicability.

The thesis is organized as follows:

- Chapter 1 summarizes the recent claims regarding the potential threat to cryptosystems;
- Chapter 2 covers the mathematical prerequisites from computer algebra and algebraic geometry;
- Chapter 3 introduces several common cryptosystems and explains their algebraic cryptanalysis;
- Chapter 4 discusses advanced algebraic techniques to solve cryptographic systems of equations;
- Chapter 5 relates XL to Gröbner Basis algorithms and discusses the complexity of promising algebraic attacks;
- The appendices mainly consist of Magma implementations of the considered algorithms.

At this point the author wishes to thank the people who were involved in this project. It is a pleasure to thank Henk van Tilborg for overall supervision and arrangement of current and previous projects. Furthermore, many improvements are due to reviews by Martijn Maas and Roderick Rodenburg. Finally, special thanks are due to my committee, Benne de Weger, Gido Schmitz, Patrick Bours and Hans Sterk, for fruitful discussions and feedback to improve this report.

Toon Segers

Contents

CHAPTER 1

Recent attention to algebraic attacks

Recent speculation on the potential threat of algebraic attacks to common cryptosystems has drawn a lot of attention. In this cryptanalysis, ciphers are rewritten to systems of multivariate equations that are solved for variables representing, for example, key bits. Algebraic attacks apply to a variety of ciphers, ranging from blockciphers, like AES and Serpent [**CP02**], to streamciphers, like Toyocrypt [**Cou03**] and Bluetooth [**Arm02**], and asymmetric cryptosystems, like HFE [**FJ03**]. In this chapter we summarize two of the most promising claims and explain the focus of this thesis.

An optimistic evaluation by Courtois and Pieprzyk [**CP02**] shows that an algebraic attack might threaten, among other block ciphers, 128 bit AES. This claim is discussed in Section 1.1. Other authors analyzed asymmetric cryptosystems in a similar way. In Section 1.2 we introduce briefly how Faugère broke the first HFE challenge [**Pat96a**]. The challenge consisted of a system of 80 multivariate, quadratic equations over $GF(2)$ and was proposed by Patarin. The HFE cipher is based on the assumption that solving a random system of multivariate, quadratic equations is NP-complete. The NP-completeness of random quadratic equations is explained in Section 1.3. We begin by introducing these claims and treat the cryptosystems in more detail later on in Chapter 3.

1.1. AES claimed to be less secure

On November 26th 2001, Rijndael [**DR99**] was chosen by the U.S. National Institute of Standards and Technology as the Advanced Encryption Standard, *AES*. Often credited as the successor of the Data Encryption Standard, AES was designed to resist standard block cipher attacks such as differential and linear cryptanalysis.

In a famous paper from 1949, Shannon [**Sha49**] proposes a design criterion for cryptosystems. He states that breaking a good cipher should require

> *"as much work as solving a system of simultaneous equations in a large number of unknowns of a complex type."*

Being confident with the fact that solving systems of nonlinear equations becomes intractable very quickly, designers do not seem to take Shannon's criterion into account explicitly. Unfortunately, when cryptanalysts express common ciphers as systems of equations it seems that these systems are not necessarily as difficult as, say, a random system of multivariate equations. In general, such random systems are not solvable in polynomial time, as we shall see in Section 1.3.

Courtois et al. introduce a technique called Extended Linearization or XL [**CKPS00**]. In a nutshell, XL solves a system of equations by adding new linearly independent equations, created from the original ones by multiplying them with monomials up to a given degree. Then, the coefficients of this expanded system are

represented by a matrix in which the rows represent the equations and the columns represent the monomials. The final step is to reduce this matrix to row echelon form and solve the corresponding system. This becomes possible when the number of linearly independent equations approaches the number of monomials, so that the matrix has nearly full rank. The complexity of the algorithm mainly depends on the time it takes to row reduce the final matrix. Therefore, the number of equations and distinct monomials in the expanded system will determine the complexity.

The authors of XL give a complexity estimation and claim that their algorithm solves a randomly generated system of polynomial equations in sub-exponential time when the number of equations slightly exceeds the number of variables. These claims are based on simulations and are still impractical but better than the theoretical worst case.

In [**CP02**] the authors introduce an improvement of XL called Extended Sparse Linearization, XSL, and express AES in a large number of quadratic equations as depicted in Table 1.1. Despite their claim about the sub-exponential behavior of the algorithm, the complexity estimation for their attack on 256 bit AES represented by quadratic equations is still harder than exhaustive search.

TABLE 1.1. Dimensions of different ciphers.

cipher	*equations*	*variables*	*field*
AES 128	8,000	1,600	$GF(2)$
AES 256	22,400	4,480	$GF(2)$
BES 128	5,248	3,948	$GF(2^8)$

AES encryption rounds combine operations over two different finite fields, $GF(2)$ and $GF(2^8)$. To overcome the difficulty of analyzing a system working in two different fields, Murphy and Robshaw introduced a new algebraic representation of the 128 bit AES over $GF(2^8)$ coined the Big Encryption System, BES [**MR02b**]. AES can be regarded as identical to BES with restricted message space and key space. Hence, messages in the AES space encrypted by BES with an AES key are equal to the corresponding AES encryptions. The BES cipher is described by 3840 very sparse quadratic equations and 1408 linear equations in 3948 variables.

Based on the method from [**CP02**] to estimate the number of linearly independent equations, attacking the 128 bit AES would roughly cost the equivalent of 2^{100} AES encryptions. This is still infeasible but promising. However, after noticing that this approximation method sometimes returns more linearly independent equations than distinct terms, Murphy and Robshaw refute the validity of these estimations in a follow-up paper [**MR02a**].

Summarizing, we see that the behavior of algebraic attacks applied to AES still remains unclear. Estimations of the number of equations and terms of the expanded system have not been substantiated and are certainly not universally accepted. However, the claims regarding the complexity of algebraic attacks similar to XL are serious enough to investigate.

1.2. First HFE challenge broken

From a different perspective, Faugère and Joux present a new result regarding algebraic attacks at Crypto 2003 [**FJ03**]. With their algorithm F5 implemented in

C, they tackle the first HFE challenge proposed by Patarin [**Pat96a**]. F5 is based on advanced Gröbner Bases techniques, which form the main topic of this report.

HFE or Hidden Field Equations [**Pat96b**] is an asymmetric cryptosystem based on the intractability of a random system of quadratic multivariate equations. The challenge consisted of 80 quadratic equations in 80 variables over $GF(2)$ and took the authors of [**FJ03**] two days and four hours on a computer with a 1 Ghz processor and 4 GB of RAM. HFE is treated in more detail in Section 3.2.

The F5 algorithm is the result of many improvements on the original Buchberger Algorithm dating from 1965 [**Buc65**]. Similar to XL, F5 tries to expand the original system of equations by selecting equations and monomials and converting the larger system into a matrix. The most interesting differences with XL are a more developed selection strategy and the fact that the degree of intermediate polynomials is kept to a minimum. After row reduction of the matrix, the resulting new equations are added to the system and a new selection is made. For this technique, the most time consuming part is the linear algebra step.

For the HFE challenge, the largest linear system involved was represented by a $307,126 \times 1,667,009$ matrix over $GF(2)$. To store such a matrix without compression requires roughly 64 GB of memory. The contribution by Faugère and Joux demonstrates the effectiveness of techniques developed with theory from Gröbner Bases. This field of algebraic geometry is understood reasonably well and seems helpful for explaining the behavior of algebraic attacks.

In early August 2004, Steel also solves the first HFE challenge using the implementation of Faugère's algorithm F4 in Magma 2.11. It took 25.4 hours on a 750 MHz processor using 15 GB of memory. His latest results are presented on his website [**Ste04**]. Steel shows that this implementation is roughly 2.7 times faster than Faugère's implementation of F5. The polynomials during computation did not exceed degree four. This is a curiosity that will be explained further in Section 3.2.

1.3. Complexity of quadratic equations

The previous sections suggest that cryptosystems expressed as multivariate, quadratic equations are not as complex as random systems of quadratic equations. To be more precise about the feasibility of systems of equations, we introduce common definitions regarding complexity following [**Kob97**]. Then, we explain why solving a random system of multivariate quadratic equations is hard and illustrate this with numerical results.

DEFINITION 1.1 (NP). A decision problem $\mathcal{P}$ is in the class of nondeterministic polynomial time problems, *NP*, if, given any instance of $\mathcal{P}$, a person with unlimited computing power not only can answer the question, but in case the answer is 'yes', he can supply evidence that another person could use to verify the correctness of the answer in polynomial time. His demonstration that his 'yes' answer is correct is called a polynomial time *certificate*.

EXAMPLE 1.2 (Reducing a problem). *Let $\mathcal{P}_1$ be the problem of deciding if a quadratic polynomial $p(x)$ has two distinct real roots. Suppose $p(x) = ax^2 + bx + c$ and set $N = b^2 - 4ac$ for a, b, c and $N \in \mathbb{Z}$. Then $\mathcal{P}_1$ reduces to the problem $\mathcal{P}_2$, being the decision whether the integer N is positive or not.*

The integer N in the above example can be computed in polynomial time, therefore we know that our original problem $\mathcal{P}_1$ is equally easy. A positive answer to $\mathcal{P}_1$ implies a positive answer to $\mathcal{P}_2$ and vice versa.

DEFINITION 1.3. A decision problem $\mathcal{P}$ in NP is said to be *NP-complete* if every other problem in NP can be reduced to $\mathcal{P}$ in polynomial time. A decision or search problem not necessarily in NP is said to be *NP-hard* if any NP-problem reduces to it.

Let P be the class of problems that have polynomial time complexity with respect to the size of the input. One of the most famous conjectures in computer science claims that $P \neq NP$. Solving a problem that is NP-complete in polynomial time, would mean that all problems in NP are solvable in polynomial time, $P = NP$. Deciding if a random system of multivariate equations has a solution is NP-complete. This decision problem can be reduced from the the Satisfiability problem for Boolean formulas, which is known to be NP-complete. This is shown below.

A Boolean variable x is a variable that can only adopt the values $true$ and $false$. Boolean variables can be combined by the logical *or* denoted by $\vee$, the logical *and* denoted by $\wedge$ and $not(x)$ denoted by $\overline{x}$ to form Boolean formulas. Boolean formulas satisfying *Conjunctive Normal Form* consist of Boolean expressions with *or* and *not* statements connected by *and* statements. In complexity theory, the expressions containing the *or* and *not* statements are called *clauses*, the variables *literals*.

EXAMPLE 1.4. *The Boolean formula* $(x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee \overline{x_3})$ *consists of two clauses with two respectively three literals.*

DEFINITION 1.5 (Satisfiability). The following problem is called the *Satisfiability* problem: "Given a Boolean formula, is it satisfiable?" If we only allow Boolean formulas to consist of clauses with three literals, the corresponding problem is called *3-Satisfiability*.

The noted computer scientist Cook formalized the notion of complexity and proved the NP-completeness of the Satisfiability problem, since then called Cook's Theorem [**Coo71**]. To derive the NP-completeness of an arbitrary system of multivariate, quadratic equations over a finite field, we use the NP-completeness of 3-Satisfiability. The following proof is taken from the literature to illustrate how one NP-complete problem reduces to another.

LEMMA 1.6. *The Satisfiability problem is reducible to the 3-Satisfiability problem, hence 3-Satisfiability is NP-complete.*

PROOF. [**Coo71**] Because 3-Satisfiability is a special case of Satisfiability, it is in NP. To show NP-completeness, we shall show that Satisfiability polynomially transforms to 3-Satisfiability.

Consider any Boolean formula F consisting of clauses $C_1, \ldots, C_m$. We shall construct a new formula F' with three literals per clause, such that F' is satisfiable if and only if F is. We shall examine the clauses of F one by one and replace each C_i by an equivalent set of clauses, each with three literals. We distinguish between three cases.

(1) If C_i has three literals, we do nothing.
(2) If C_i has more than three literals, say $C_i = (\lambda_1 \vee \lambda_2 \vee \ldots \vee \lambda_k)$, $k > 3$, we replace C_i by the $k-2$ clauses $(\lambda_1 \vee \lambda_2 \vee x_1) \wedge (\overline{x_1} \vee \lambda_3 \vee x_2) \wedge (\overline{x_2} \vee$

$\lambda_4 \vee x_3) \wedge \cdots \wedge (\overline{x_{k-3}} \vee \lambda_{k-1} \vee \lambda_k)$, where $x_1, \ldots, x_{k-3}$ are new variables. These new clauses are satisfiable if and only if C_i is.

(3) If $C_i = \lambda$, we replace C_i by $\lambda \vee y \vee z$, and if $C_i = \lambda \vee \lambda'$, we replace it by $\lambda \vee \lambda' \vee y$. We then add the clauses

$$(\overline{z} \vee \alpha \vee \beta) \wedge (\overline{z} \vee \overline{\alpha} \vee \beta) \wedge (\overline{z} \vee \alpha \vee \overline{\beta}) \wedge (\overline{z} \vee \overline{\alpha} \vee \overline{\beta}) \wedge$$
$$(\overline{y} \vee \alpha \vee \beta) \wedge (\overline{y} \vee \overline{\alpha} \vee \beta) \wedge (\overline{y} \vee \alpha \vee \overline{\beta}) \wedge (\overline{y} \vee \overline{\alpha} \vee \overline{\beta})$$

to the formula, where y, z, α and β are new variables. This addition forces the variables z and y to be $false$ in any assignment satisfying F', so that the clauses λ and $\lambda \vee \lambda'$ are equivalent to their replacements.

What we have just described is a polynomial-time transformation from Satisfiability to 3-Satisfiability. Hence 3-Satisfiability is NP-complete. □

THEOREM 1.7. *Deciding if an arbitrary system of multivariate, quadratic equations over a finite field is solvable is NP-complete.*

PROOF. We show how to reduce any 3-Satisfiability problem to a system of multivariate, quadratic equations over $GF(2)$ in polynomial time. It is easy to extend this system of equations to allow the equations to be over any finite field.

If we want show a one-to-one correspondence between a Boolean formula and a system of equations over $GF(2)$, we start by replacing the Boolean arithmetic. Substitute $\vee$ by the field addition $+$, $\wedge$ by the field multiplication $\cdot$, *true* by 1 and $false$ by 0. Suppose a Boolean formula $F = C_1 \wedge C_2 \cdots \wedge C_m$ in the 3-Satisfiability problem consists of clauses $C_i = (x_{i_1} \vee x_{i_2} \vee x_{i_3})$. Simply substituting the arithmetic would result in the following system of equations over $GF(2)$,

$$x_{1_1} + x_{1_2} + x_{1_3} = 1, \ \ldots, \ x_{m_1} + x_{m_2} + x_{m_3} = 1.$$

Unfortunately, this does not correspond to the original Boolean formula in general. To make every clause C_i equivalent to a subsystem of equations, we replace it by the following set of equations. A solution of the following system corresponds to a *true* clause C_i,

$$\begin{aligned} &x_{i_1} + x_{i_2} + x_{i_3} = x_{i_4}, \\ &x_{i_1} \cdot x_{i_2} + x_{i_2} \cdot x_{i_3} + x_{i_1} \cdot x_{i_3} = x_{i_5}, \\ &x_{i_4} + x_{i_5} + x_{i_4} \cdot x_{i_5} = 1, \end{aligned}$$

as can be checked easily by means of a table. If the clause C_i is *true* for a certain assignment of the literals x_{i_1}, x_{i_2} and x_{i_3}, then it corresponds to an equivalent assignment of the corresponding variables. In this way we are able to replace every clause by an equivalent subsystem, which reduces the 3-Satisfiability problem to a system of multivariate, quadratic equations over $GF(2)$.

Adding the equations $x_{i_j}(1 - x_{i_j}) = 0$ for all variables, allows the variables to be defined over a larger finite field while still being equivalent to the original Boolean formula. In this way, F is Satisfiable if and only if the given system of quadratic equations over a given finite field is solvable. □

In literature on algebraic attacks the NP-completeness is often mentioned but never formally referred to. For completeness, we would like to refer to [**GJ79**] and [**LRK79**] summarizing important results in this field. In particular, the NP-completeness of deciding whether systems of quadratic equations have a solution over finite fields is proven in [**HPS93**, p. 3].

REMARK 1.8. The 2-Satisfiability problem is solvable in polynomial time. We show that this implies that some systems of quadratic equations are not NP-complete. Suppose $F = C_1 \wedge \cdots \wedge C_m$ is a 2-Satisfiability problem. Since every clause $C_i = (x_i \vee y_i)$ is equivalent to $x_i + y_i + x_i \cdot y_i = 1$ over $GF(2)$, a system of quadratic equations with this form is not NP-complete.

NP-completeness is a definition for a class of problems that are hard to solve on average. Cryptosystems are usually disguised as such problems, like the knapsack problem. However, decryption necessarily demands that there has to be a trapdoor. This makes structure inevitable and these instances with additional structure within the class of NP-complete problems might be easier to solve than they seem.

In general, a system of multivariate, quadratic equations over a finite field is NP-complete. This means that it will be very unlikely that every quadratic system is solvable in polynomial time. To illustrate this, we generate several random systems consisting of n quadratic equations in n variables over $GF(2)$ and try to solve them using F4, which is one of the fastest algorithms available. Figure 1.1 shows the average time it took Magma 2.11 [**BCP97**] to compute the solutions of these systems on a Pentium 4 desktop computer. Every average corresponds to twenty samples.

REMARK 1.9. Unless indicated otherwise, all tests in this report are run on a Pentium 4, 2 GHz with 512 MB RAM.

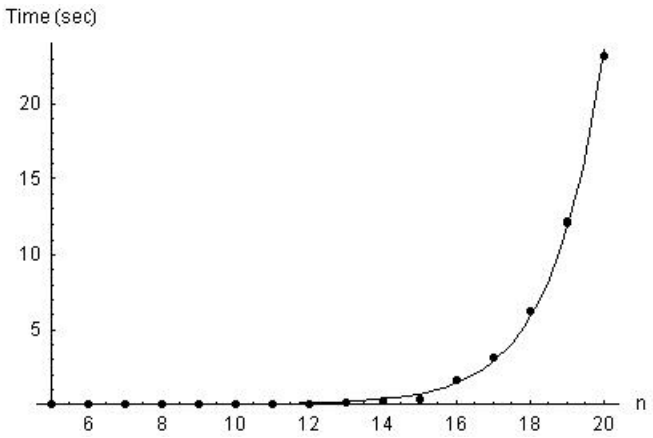

FIGURE 1.1. Average timings for Magma 2.11 to solve a random multivariate, quadratic system of n equations in n variables.

The time needed to solve the examples for n equations differs roughly at least a factor 2 from the systems of $n-1$ equations, which indicates the exponential behavior of the method. For growing n, this factor needs to be asymptotically less than two, for the algebraic attack to become better than exhaustive search. The figure also shows the least-square fit for an exponential function, being

$$0.0000193 * 2.02^n.$$

REMARK 1.10. The fit curve was created in Mathematica 4. For example, the command

```
D = Fit[d, {1, n, n^2}, n]
```

finds the least-square fit D for the data d as a linear combination of the functions 1, n and n^2. The goodness of a fit curve is measured by the least sum of squares between the data d_i and its approximation D_i, $\chi^2 = \sum_i |D_i - d_i|^2$.

The depicted curve for the time spent is close to optimal. This is due to the following observation. After a brief glance at the plotted averages, the behavior seems exponential. Therefore, we would like to apply a least-square fit with respect to an exponential function. To figure out the base number, we take the natural logarithm of the values in the data set and apply a least-square fit with a linear function. This is illustrated in Figure 1.2. Now, let c equal the derivative of the linear fit. Taking the natural exponent to the power c returns the base number of the exponential function approximating the data in the original figures.

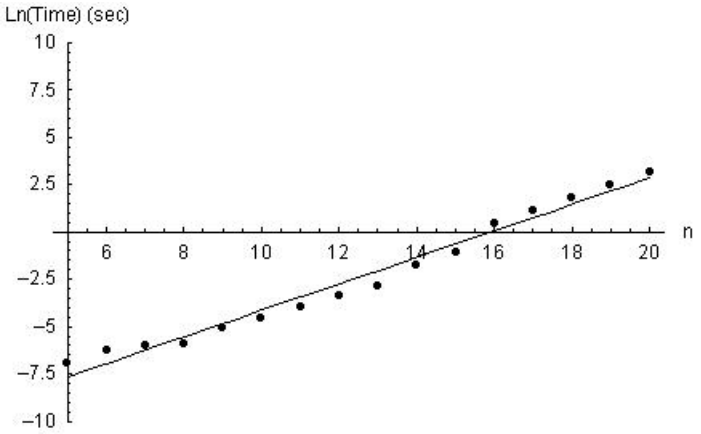

FIGURE 1.2. Ln(average timings) for Magma 2.11 to solve a random multivariate, quadratic system of n equations and n variables.

In cryptography you do not want a certain percentage of the problems to be easily solvable. The recent claims suggest that existing cryptosystems might not be as hard to solve as random systems of quadratic equations. They suggest that equations corresponding to cryptosystems contain more structure than random systems. From the well studied field of Gröbner Bases we shall try to understand the difference between common ciphers and random systems of multivariate, quadratic equations. Furthermore, the behavior of algebraic attacks is studied and expressed in measurable terms, such as time and memory consumption.

CHAPTER 2

Preliminaries on algebraic geometry

The theory of Gröbner Bases is an example of how the solution to one problem became the key for solving a great variety of other problems. This major step was taken by Buchberger in 1965 who formulated the concept as a solution to the *Ideal Membership Problem* and, extending a suggestion of his advisor Gröbner, found an algorithm to compute such a basis and proved the correctness and the termination of the algorithm.

The purpose of this chapter is to introduce the basic theory and notation necessary to discuss systems of multivariate equations. We shall introduce:

- Affine varieties and ideals in Section 2.1;
- Gröbner Bases in Section 2.2;
- Buchberger's Algorithm in Section 2.3;
- How to solve systems of equations in Section 2.4.

To conclude the introduction of this chapter, we would like to mention the following references as a good introduction to algebraic geometry and computer algebra, [**CLO96**], [**CLO98**], [**BW93**], [**KR00**] and [**Mis93**]. Many of the theorems in this chapter appear in one or more of these books. We limit ourselves to the proofs that are essential for understanding the operation of the Buchberger Algorithm.

2.1. Affine varieties and ideals

In this section we introduce ideals and affine varieties following the description of [**CLO96**]. However, at some points we slightly change the notation. A *monomial* in n variables $x_1, \ldots, x_n$ is a product of the form

$$x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdot \ldots \cdot x_n^{\alpha_n},$$

where all the exponents are in $\mathbb{Z}_{\geq 0}$, the nonnegative integers. The *total degree* of this monomial is the sum $\alpha_1 + \ldots + \alpha_n$. We simplify the notation by setting $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}_{\geq 0}^n$ and

$$x^\alpha = x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdot \ldots \cdot x_n^{\alpha_n}.$$

Denote the total degree of x^α by $|\alpha|$.

Definition 2.1. A *polynomial* f in $x_1, \ldots, x_n$ with coefficients in a field k is a finite k-linear combination of monomials,

$$f = \sum_\alpha c_\alpha x^\alpha, \quad c_\alpha \in k,$$

where the sum is over a finite number of n-tuples $\alpha = (\alpha_1, \ldots, \alpha_n)$. The set of all polynomials in $x_1, \ldots, x_n$ with coefficients in k is denoted by $k[x_1, \ldots, x_n]$. From this point, P refers to the *polynomial ring* over field k in n variables, $k[x_1, \ldots, x_n]$, if not explicitly stated otherwise.

We will use the following terminology.

DEFINITION 2.2. Let $f = \sum_\alpha c_\alpha x^\alpha$ be a polynomial in P.

- We call c_α the *coefficient* of the monomial x^α in the polynomial f;
- If $c_\alpha \neq 0$, then we call $c_\alpha x^\alpha$ a *term* of f;
- The *total degree* of f, denoted *deg(f)*, is the maximum $|\alpha|$ such that the coefficient c_α is non-zero.
- The set of monomials in P is denoted by $T(P)$ and $T(f)$ is the set of monomials of f, called the *support*.

The ability to regard a polynomial as a function is what makes it possible to link algebra and geometry. Therefore, the affine space is introduced in which the geometrical shapes exist.

DEFINITION 2.3. Given a field k and a positive integer n, we define the n-dimensional *affine space* to be the set

$$k^n = \{(a_1, \ldots, a_n) : a_1, \ldots, a_n \in k\}.$$

Evaluating a polynomial f at $(a_1, \ldots, a_n) \in k^n$ is a function

$$f : k^n \to k,$$

where every x_i is replaced by a_i, for $1 \leq i \leq n$.

As mentioned in the previous chapter, the purpose of this thesis is to study the complexity of solving systems of polynomial equations. The notion of a variety is therefore inevitable. The set of all solutions to a system of equations

$$f_1(x_1, \ldots, x_n) = \ldots = f_m(x_1, \ldots, x_n) = 0$$

is called an affine variety, explicitly defined as follows.

DEFINITION 2.4. Let k be a field and let $f_1, \ldots, f_m$ be polynomials in P. We define

$$\mathbf{V}(f_1, \ldots, f_m) = \{(a_1, \ldots, a_n) \in k^n : f_i(a_1, \ldots, a_n) = 0 \text{ for all } 1 \leq i \leq m\}.$$

We call $\mathbf{V}(f_1, \ldots, f_s)$ the *affine variety* defined by $f_1, \ldots, f_m$.

The following example gives an idea of the applications of affine varieties.

EXAMPLE 2.5. *Suppose we have a robot arm consisting of two parts, similar to a human arm. The parts are connected together like an elbow connects the lower and upper arm. The part anchored at the origin is of length* 2 *and the other of length* 1*. If the robot arm is able to freely control its elbow at coordinates* (x_1, x_2) *and hand at* (x_3, x_4) *in two dimensions, we can describe the subset of possible states by the affine variety in* $\mathbb{R}^4$ *defined by the equations*

$$x_1^2 + x_2^2 = 4,$$
$$(x_1 - x_3)^2 + (x_2 - x_4)^2 = 1.$$

Intersections and unions of affine varieties are again affine varieties as we shall see in the following lemma.

LEMMA 2.6. *If* $V = \boldsymbol{V}(f_1, \ldots, f_s)$, $W = \boldsymbol{V}(g_1, \ldots, g_t) \subset k^n$ *are affine varieties, then so are*

$$V \cap W = \boldsymbol{V}(f_1, \ldots, f_s, g_1, \ldots, g_t),$$
$$V \cup W = \boldsymbol{V}(f_i g_j : 1 \leq i \leq s, 1 \leq j \leq t).$$

PROOF. See [**CLO96**, p.11]. □

To compute with affine varieties, we need the notion of ideals.

DEFINITION 2.7. A subset $I \subset P$ is an *ideal* if it satisfies:

(1) $0 \in I$;
(2) If f, $g \in I$, then $f + g \in I$;
(3) If $f \in I$ and $h \in P$, then $hf \in I$.

Polynomial equations have a nice interpretation in terms of ideals. The ideal generated by a finite number of polynomials is defined as follows.

DEFINITION 2.8. Let $f_1, \ldots, f_m$ be polynomials in P. Define the ideal

$$\langle f_1, \ldots, f_m \rangle = \{\sum_{i=1}^{m} h_i f_i : h_1, \ldots, h_m \in P\}.$$

If there exists a finite set of polynomials in P that generates a given ideal, we call this set a *basis*. In Section 2.2 a fundamental theorem in commutative algebra, called the Hilbert Basis Theorem, is proven. This theorem states that every ideal in P is finitely generated. Note that a given ideal may have many different bases.

PROPOSITION 2.9. *If $f_1, \ldots, f_s$ and $g_1, \ldots, g_t$ are bases of the same ideal in P, so that $\langle f_1, \ldots, f_s \rangle = \langle g_1, \ldots, g_t \rangle$, then $\boldsymbol{V}(f_1, \ldots, f_s) = \boldsymbol{V}(g_1, \ldots, g_t)$.*

PROOF. Every $f \in \langle f_1, \ldots, f_s \rangle$ is also in $\langle g_1, \ldots, g_t \rangle$ and can therefore be expressed as $h_1 g_1 + \ldots + h_t g_t$. Hence, every $a = (a_1, \ldots, a_n) \in \mathbf{V}(g_1, \ldots, g_t)$ satisfies $f(a) = 0$ and vice versa for all $g \in \langle g_1, \ldots, g_t \rangle$. This shows that both varieties consist of the same points. □

EXAMPLE 2.10. *Set $k = GF(2)$ and*

$$f_1 = xy + xz + y^2 + yz + z^2, \quad f_2 = xz + z^2, \quad f_3 = y + z^2$$

in $k[x, y, z]$. Consider the variety $\boldsymbol{V}(f_1, f_2, f_3)$. The ideal spanned by $\langle f_1, f_2, f_3 \rangle$ equals

$$\langle xz + z^2, y + z^2, z^4 \rangle,$$

since $f_1 = (z{+}1)(xz{+}z^2){+}(x{+}y{+}z^2{+}z)(y{+}z^2){+}z^4$. The second variety is easier to determine, since the new-found basis elements allow backwards substitution. Hence, we find the common zeros of f_1, f_2 and f_3 being $\{(0, 0, 0), (1, 0, 0)\}$ in k^3.

The ability of changing the basis of an ideal without affecting the variety plays a key role in solving a system of polynomial equations. In the next section we introduce Gröbner Bases and Buchberger's Algorithm, which turn out to be powerful tools for understanding affine varieties. To study which polynomials vanish on the same variety, we introduce a new algebraic object.

DEFINITION 2.11. Let $V \subset k^n$ be an affine variety. Then we set

$$\mathbf{I}(V) = \{f \in P : f(a_1, \ldots, a_n) = 0 \text{ for all } (a_1, \ldots, a_n) \in V\}.$$

The crucial observation is that this is also an ideal ([**CLO96**, p.31]). Something that might not be completely intuitive is that $\mathbf{I}(\mathbf{V}(f_1, \ldots, f_m))$ does not equal $\langle f_1, \ldots, f_m \rangle$ in general.

LEMMA 2.12. *If $f_1, \ldots, f_m \in P$, then $\langle f_1, \ldots, f_m \rangle \subseteq \boldsymbol{I}(\boldsymbol{V}(f_1, \ldots, f_m))$, although equality need not occur.*

PROOF. See [**CLO96**, p.33]. □

This lemma is illustrated by an example.

EXAMPLE 2.13. *The ideal spanned by* $x^2 \in \mathbb{R}[x]$, $\langle x^2 \rangle$, *is a proper subset of* $\langle x \rangle = \boldsymbol{I}(\boldsymbol{V}(x^2))$.

2.2. Gröbner Bases

In the previous section, we have seen that the problem of solving a system of polynomial equations

$$f_1(x_1, \ldots, x_n) = \ldots = f_m(x_1, \ldots, x_n) = 0$$

is the same as finding the points of the affine variety $\mathbf{V}(f_1, \ldots, f_m)$. To help us in doing this, we introduce Gröbner Bases, which turn out to be a powerful tool to describe ideals in terms of a finite generating set. The main goal in this section is to describe the most important ideas to find such a set.

Let P be a polynomial ring. The set of monomials $T(P)$ is important to us and one can choose to arrange this set in many different ways. One way is of particular interest.

DEFINITION 2.14. A *monomial ordering* on P is any relation $>$ on $\mathbb{Z}^n_{\geq 0}$, or equivalently, any relation on the set of monomials x^α, $\alpha \in \mathbb{Z}^n_{\geq 0}$, satisfying:

(1) $>$ is a total ordering on $\mathbb{Z}^n_{\geq 0}$;
(2) If $\alpha > \beta$ and α, β, $\gamma \in \mathbb{Z}^n_{\geq 0}$, then $\alpha + \gamma > \beta + \gamma$;
(3) $>$ is a *well-ordering* on $\mathbb{Z}^n_{\geq 0}$. This means that every nonempty subset of $\mathbb{Z}^n_{\geq 0}$ has a smallest element under $>$.

A specific ordering σ for $>$ is denoted by $>_\sigma$. The third property of the previous definition, the well-ordering, is often referred to as *admissible ordering* and has an important implication. It is used to show that specific algorithms that make explicit use of a well-ordering, like the Buchberger Algorithm, eventually terminate.

LEMMA 2.15. *An order relation* $>$ *on* $\mathbb{Z}^n_{\geq 0}$ *is a well-ordering if and only if every strictly decreasing sequence in* $\mathbb{Z}^n_{\geq 0}$

$$\alpha(1) > \alpha(2) > \alpha(3) > \ldots$$

eventually terminates.

PROOF. See [**CLO96**, p.53]. □

We introduce the three most important monomial orderings for our applications.

DEFINITION 2.16. [*Lexicographic Order* (*lex*)] Let $\alpha = (\alpha_1, \ldots, \alpha_n)$ and $\beta = (\beta_1, \ldots, \beta_n) \in \mathbb{Z}^n_{\geq 0}$. We say $\alpha >_{lex} \beta$ if, in the vector difference $\alpha - \beta \in \mathbb{Z}^n$, the left-most non-zero entry is positive. We will write $x^\alpha >_{lex} x^\beta$ if $\alpha >_{lex} \beta$.

DEFINITION 2.17. [*Graded Lex Order*] Let α, $\beta \in \mathbb{Z}^n_{\geq 0}$. We say $\alpha >_{grlex} \beta$ if

$$|\alpha| = \sum_{i=1}^{n} \alpha_i > |\beta| = \sum_{i=1}^{n} \beta_i, \text{ or } \alpha >_{lex} \beta \text{ if } |\alpha| = |\beta|.$$

DEFINITION 2.18. [*Graded Reverse Lex Order (grevlex)*] Let α, $\beta \in \mathbb{Z}^n_{\geq 0}$. We say $\alpha >_{grevlex} \beta$ if

$$|\alpha| = \sum_{i=1}^{n} \alpha_i > |\beta| = \sum_{i=1}^{n} \beta_i,$$

or in $\alpha - \beta \in \mathbb{Z}^n$, the right-most non-zero entry is negative if $|\alpha| = |\beta|$.

Furthermore, there are also some common definitions for special terms of polynomials with respect to orderings.

DEFINITION 2.19. Let $f = \sum_\alpha a_\alpha x^\alpha$ be a non-zero polynomial in P and let $>$ be a monomial order.

- The *multidegree* of f is
$$multideg(f) = max_>(\alpha \in \mathbb{Z}^n_{\geq 0} : a_\alpha \neq 0).$$
- The *total degree* of f is
$$totaldeg(f) = \sum_{1 \leq i \leq n} multideg(f)_i.$$
- The *leading coefficient* of f is
$$LC(f) = a_{multideg(f)} \in k.$$
- The *leading monomial* of f is
$$LM(f) = x^{multideg(f)}.$$
- The *leading term* of f is
$$LT(f) = LC(f) \cdot LM(f).$$
- The polynomial f is called *monic* if
$$LC(f) = 1.$$

REMARK 2.20. For a subset F of the polynomial ring P, let $LT(F)$ and $LM(F)$ denote $\{LT(f) : f \in F\}$ and $\{LM(f) : f \in F\}$ respectively.

To test whether an element f is a member of a certain ideal $\langle f_1, \ldots, f_m \rangle \subseteq P$, one needs to show if it can be written as a combination of elements in $\{f_1, \ldots, f_m\}$ in the form

$$f = a_1 f_1 + \ldots + a_m f_m,$$

for $a_i \in P$. If so, f is in the given ideal. However, if this is not possible an element $r \in P$ remains, which can not be expressed in the specified polynomials. This test is done by the Division Algorithm for the multivariate case, which is introduced explicitly in the following theorem.

THEOREM 2.21. *Fix a monomial order $>$ on $\mathbb{Z}^n_{\geq 0}$, and let $F = (f_1, \ldots, f_m)$ be an ordered m-tuple of polynomials in P. Then every $f \in P$ can be written as*

$$f = a_1 f_1 + \ldots + a_m f_m + r,$$

where a_i, $r \in P$, and either $r = 0$ or r is a k-linear combination of monomials, none of which is divisible by any of $LT(f_1), \ldots, LT(f_m)$. We will call r a remainder of f on division by F. Furthermore, if $a_i f_i \neq 0$, then we have

$$multideg(f) \geq multideg(a_i f_i).$$

PROOF. See [**CLO96**, p.62]. This proof introduces the Division Algorithm for the multivariate polynomial ring P, which returns the $a_1, \ldots, a_m$ and r of the theorem and thereby shows the existence of these elements. □

DEFINITION 2.22. The *Division Algorithm* is defined as follows.

- Input: $f_1, \ldots, f_m, f \in P$
- Output: $a_1, \ldots, a_m, r \in P$ such that f can be expressed as in Theorem 2.21.

(1) $a_1 := 0; \ldots; a_m := 0; r := 0$
(2) $p := f$
(3) $WHILE\ p \neq 0\ DO$
(4) $\quad i := 1$
(5) $\quad divisionoccurred := false$
(6) $\quad WHILE\ i \leq m\ AND\ divisionoccurred = false\ DO$
(7) $\quad\quad IF\ LT(f_i)|LT(p)\ THEN$
(8) $\quad\quad\quad a_i := a_i + LT(p)/LT(f_i)$
(9) $\quad\quad\quad p := p - (LT(p)/LT(f_i))f_i$
(10) $\quad\quad\quad divisionoccurred := true$
(11) $\quad\quad ELSE$
(12) $\quad\quad\quad i := i + 1$
(13) $\quad IF\ divisionoccurred = false\ THEN$
(14) $\quad\quad r := r + LT(p)$
(15) $\quad\quad p := p - LT(p)$

In some situations permutations of the set $\{f_1, \ldots, f_m\}$ do lead to different remainders. As we shall see in Section 2.3, this is not the case when the set is a Gröbner Basis.

In the univariate case, the division algorithm includes a criterion on the degree of the remainder. Intuitively one would expect a simpler criterion on the total degree of the remainder in the multivariate case than the one in Theorem 2.21. However, this is not the case. The following example illustrates why.

EXAMPLE 2.23. *Let $x >_{lex} y$. The remainder on division of $x^5 + y^5 \in \mathbb{R}[x, y]$ by $x + y^2$ by the Division Algorithm is $-y^{10} + y^5$, since*

$$x^5 + y^5 = (x + y^2)(x^4 - x^3y^2 + x^2y^4 - xy^6 + y^8) - y^{10} + y^5$$

and y^{10} is not divisible by x.

One of the important characteristics of a Gröbner Basis is that it specifies a finite basis for the ideal of all the leading terms that occur in the ideal defined by a given set of polynomials. This *ideal of leading terms* is a monomial ideal, a subclass of ideals with special properties.

DEFINITION 2.24. An ideal $I \subset P$ is a *monomial ideal* if it can be spanned by monomials or, equivalently, if there is a subset $A \subset \mathbb{Z}^n_{\geq 0}$ (possibly infinite) such that I consists of all polynomials that are finite sums of the form $\sum_{\alpha \in A} h_\alpha x^\alpha$, where $h_\alpha \in P$. In this case, we write $I = \langle x^\alpha : \alpha \in A \rangle$.

LEMMA 2.25. *Let a monomial ideal $I = \langle x^\alpha : \alpha \in A \rangle \subset P$ and $f \in P$. Some basic properties of I are:*

- *A monomial x^β lies in I if and only if x^β is divisible by x^α for some $\alpha \in A$;*

- *$f \in I \Leftrightarrow T(f) \subset I \Leftrightarrow f$ is a k-linear combination of the monomials in I;*
- *Two monomial ideals are the same if and only if they contain the same monomials.*

PROOF. See [**CLO96**, pp. 67–69]. □

An important building block for the Hilbert Basis Theorem is the property that monomial ideals of P are finitely generated. This is stated explicitly in the following theorem called *Dickson's Lemma*.

THEOREM 2.26 (Dickson's Lemma). *A monomial ideal $I = \langle x^\alpha : \alpha \in A\rangle \subset P$ can be written in the form $I = \langle x^{\alpha(1)}, \ldots, x^{\alpha(m)}\rangle$, where $\alpha(1), \ldots, \alpha(m) \in A$. In particular, I has a finite basis.*

PROOF. See [**CLO96**, p. 69]. □

To answer the question how to describe a given ideal by a finite set of generators, we shall see that its *monomial ideal of leading terms* plays a key role.

DEFINITION 2.27. Let $I \subset P$ be an ideal other than $\{0\}$. We denote by $LT(I)$ the set of leading terms of elements of I. Thus

$$LT(I) = \{cx^\alpha : \exists f \in I \text{ with } LT(f) = cx^\alpha\} \subset P.$$

We denote by $\langle LT(I)\rangle$ the ideal generated by the elements of $LT(I)$.

With help of Dickson's Lemma the following property is derived.

PROPOSITION 2.28. *$\langle LT(I)\rangle$ is a monomial ideal. There are $g_1, \ldots, g_m \in I$ such that $\langle LT(I)\rangle = \langle LT(g_1), \ldots, LT(g_m)\rangle$.*

PROOF. See [**CLO96**, p.73]. □

Now, by the previous proposition and the Division Algorithm one can prove that every ideal has a finite generating set. This proof is included because it gives a clue of how an algorithm could test whether a polynomial is a member of a given ideal.

THEOREM 2.29 (Hilbert Basis Theorem). *Every ideal $I \subset P$ has a finite generating set. That is, $I = \langle g_1, \ldots, g_m\rangle$ for some $g_1, \ldots, g_m \in I$.*

PROOF. If $I = \{0\}$, we take our generating set to be $\{0\}$, which is certainly finite. If I contains some non-zero polynomial, then a generating set $g_1, \ldots, g_m$ for I can be constructed as follows. By Proposition 2.28, there are $g_1, \ldots, g_m \in I$ that generate $\langle LT(I)\rangle$. We show $I = \langle g_1, \ldots, g_m\rangle$.

It is clear that $\langle g_1, \ldots, g_m\rangle \subset I$. Conversely, let $f \in I$ be any polynomial. If we apply the Division Algorithm to divide f by the m-tuple $(g_1, \ldots, g_m)$ then we get an expression of the form

$$f = a_1 g_1 + \ldots + a_m g_m + r$$

where r is divisible by $LT(g_1), \ldots, LT(g_m)$. We claim that $r = 0$.

Note that $r = f - a_1 g_1 + \ldots + a_m g_m \in I$. If $r \neq 0$, then $LT(r) \in \langle LT(I)\rangle = \langle LT(g_1), \ldots, LT(g_m)\rangle$, and from the first property of Lemma 2.25 it follows that $LT(r)$ must be divisible by some $LT(g_i)$. This contradicts with the definition of a remainder and $r = 0$.

Thus,

$$f = a_1 g_1 + \ldots + a_m g_m \in \langle g_1, \ldots, g_m \rangle$$

and therefore $I \subset \langle g_1, \ldots, g_m \rangle$, which completes the proof. □

Now suppose we want to test if a given f is a member of the ideal $I = \langle g_1, \ldots, g_m \rangle$. The proof of the Hilbert Basis Theorem indicates that we can expand our basis of $\langle LT(I) \rangle$ until all minimal monomials are included. Hence, every polynomial $f \in I$ reduces to zero with respect to this basis since the remainder cannot be larger than the leading terms of the basis elements. To be more precise, there exists a finite basis G of I such that for every $f \in I$ there is a basis element $g \in G$ satisfying $LT(g)|LT(f)$. This crucial observation finally leads to the explicit idea of a Gröbner Basis.

Definition 2.30. Fix a monomial order. A finite subset $G = \{g_1, \ldots, g_m\}$ of an ideal I is said to be a *Gröbner Basis* or *standard basis* if

$$\langle LT(g_1), \ldots, LT(g_m) \rangle = \langle LT(I) \rangle.$$

Example 2.31. *Let $I = \langle f_1, f_2 \rangle \subset \mathbb{R}[x, y]$, where $f_1 = x^3 - 2xy$ and $f_2 = x^2 y - 2y^2 + x$. Suppose we are using the grlex ordering on monomials, then*

$$x \cdot f_1 - y \cdot f_2 = x^2.$$

Therefore, $x^2 \in I$ and $x^2 \in \langle LT(I) \rangle$. However, x^2 is not divisible by $LT(f_1)$ or $LT(f_2)$, so f_1 and f_2 do not form a Gröbner Basis of I.

Next, consider the ideal $J = \langle g_1, g_2 \rangle = \langle x + z, y - z \rangle$. For lex order in $\mathbb{R}[x, y, z]$, we claim that g_1 and g_2 form a Gröbner Basis of the ideal J. To prove this, one must show that every leading term in J lies in the ideal $\langle x, y \rangle$ or, equivalently, that every leading term is divisible by x or y. This is done as follows.

Every element $f \in J$ can be written as

$$f = a_1 g_1 + a_2 g_2 = a_1 x + a_1 z + a_2 y - a_2 z,$$

for a_1, $a_2 \in P$. If $a_1 x$ and $a_2 y$ do not cancel each other, then the leading term of f is in $\langle LT(J) \rangle$. Now, suppose $a_1 x$ and $a_2 y$ do cancel each other, then the leading terms of a_1 and a_2 are divisible by x or y, or $a_1 = a_2 = 0$ holds. If a_1 and a_2 are non-zero, then $a_1 z - a_2 z \neq 0$ and therefore the leading term of $a_1 z - a_2 z$ is divisible by x or y.

The Hilbert Basis Theorem has two important consequences. The first one concerns a nested increasing sequence of ideals in P,

$$I_1 \subset I_2 \subset I_3 \subset \ldots$$

and states that this so-called ascending chain stabilizes at a certain moment. This will be a necessary condition to show that algorithms for computing a Gröbner Basis terminate. The second consequence is that the variety corresponding to a set of polynomials equals the variety of the ideal generated by the set of polynomials. To be more precise about both, the following results are stated.

Theorem 2.32. *[The Ascending Chain Condition] Let*

$$I_1 \subset I_2 \subset I_3 \subset \ldots$$

be an ascending chain of ideals in P. Then there exists an $N \geq 1$ such that

$$I_N = I_{N+1} = I_{N+2} = \ldots.$$

PROOF. See [**CLO96**, p. 76]. □

DEFINITION 2.33. Let $I \subset P$ be an ideal. We will denote by $\mathbf{V}(I)$ the set

$$\{(a_1, \ldots, a_n) \in k^n : f(a_1, \ldots, a_n) = 0 \text{ for all } f \in I\}.$$

PROPOSITION 2.34. *$\boldsymbol{V}(I)$ is an affine variety. In particular, if $I = \langle f_1, \ldots, f_m \rangle$, then $\boldsymbol{V}(I) = \boldsymbol{V}(f_1, \ldots, f_m)$.*

PROOF. See [**CLO96**, p. 77]. □

2.3. Buchberger's Algorithm

In the previous section it was shown that an ideal can be described by a finite set of generators. After describing a cryptosystem by a set of equations, the problem is to find the common zeros of the corresponding set of polynomials, the affine variety. The affine variety is determined by the polynomials and, equivalently, by the ideal spanned by the polynomials. In this section Buchberger's Algorithm is introduced to find a Gröbner Basis. This special finite set of generators turns out to be more suitable for finding the common zeros.

Gröbner Bases have the useful property that if we apply the Division Algorithm to a polynomial f and a Gröbner Basis G, the remainder r will be unique. This is a crucial observation for testing whether an element is a member of a given ideal.

PROPOSITION 2.35. *Let $G = \{g_1, \ldots, g_m\}$ be a Gröbner Basis for an ideal $I \subset P$ and let $f \in P$. Then there is a unique $r \in P$ with the following two properties:*

1. *No term of r is divisible by any leading term of G;*
2. *There is a $g \in I$ such that $f = g + r$.*

In particular, r is the remainder on division of f by G no matter how the elements of G are listed when using the Division Algorithm.

PROOF. See [**CLO96**, p.79]. □

If, by the Division Algorithm, f reduces to r modulo a tuple $G = (g_1, \ldots, g_m)$, r is denoted by $\overline{f}^G$. Due to the previous proposition, G might also be a set if its elements form a Gröbner Basis, since the remainder r does not depend on the order of the elements in G in this case.

In the proof of the Hilbert Basis Theorem (Theorem 2.29) we observed that at some point any polynomial f in the ideal $I \subset P$ reduces to zero modulo a finite basis G. In constructing a Gröbner Basis, we search for basis elements in which the leading terms generate the ideal $\langle LT(I) \rangle$. The division of f by a set of generators G' that does not satisfy this criterion, results in an r with a leading term that does not exist in $\{LT(g) : g \in G'\}$. The idea of S-polynomials is introduced to find new elements in the ideal, like f, that could add leading terms to the intermediate basis G'.

DEFINITION 2.36. Let $f, g \in P$ be non-zero polynomials.

1. If $multideg(f) = \alpha$ and $multideg(g) = \beta$, then let $\gamma = (\gamma_1, \ldots, \gamma_n)$, where $\gamma_i = max(\alpha_i, \beta_i)$ for each i. We call x^γ the *least common multiple* of $LM(f)$ and $LM(g)$, written $x^\gamma = LCM(LM(f), LM(g))$;

(2) The *S-polynomial* of f and g is the combination
$$S(f,g) = \frac{x^\gamma}{LT(f)} \cdot f - \frac{x^\gamma}{LT(g)} \cdot g;$$
(3) The pairs (f,g) in the foregoing are commonly referred to as *critical pairs* and $totaldeg(LCM(LM(f), LM(g)))$ is the *degree of the critical pair* (f,g).

The name S-polynomial is actually an abbreviation for 'syzygy polynomial', a notion that will be studied later in Section 4.4. In astronomy the word syzygy is used for an alignment of three planets. Its root is a Greek word meaning 'yoke'. Hence, like a yoke aligns oxen, a syzygy aligns the leading terms of two polynomials. Using S-polynomials, the following lemma turns out to be a helpful tool to prove the correctness and termination of the Buchberger Algorithm and variations.

LEMMA 2.37. *Suppose we have a sum* $f = \sum_{i=1}^m c_i f_i$, *where* $c_i \in k$ *and* $multideg(f_i) = \delta \in \mathbb{Z}^n_{\geq 0}$ *for all* i. *If* $multideg(\sum_{i=1}^n c_i f_i) < \delta$, *then* f *is a* k*-linear combination of the S-polynomials* $S(f_j, f_k)$ *for* $1 \leq j, k \leq m$ *where* $S(f_j, f_k)$ *has multidegree* $< \delta$.

PROOF. See [**CLO96**, p. 81]. □

Based on Lemma 2.37, Buchberger formulated his *S-pair criterion* that enables to determine effectively whether a given generating set of an ideal is a Gröbner Basis. This criterion leads in a natural way to Buchberger's Algorithm. The concept of a standard representation is introduced to explain the proof.

DEFINITION 2.38. [**BW93**, p.218] Let $f \in P$ and $G = \{g_1, \ldots, g_m\}$ a finite subset of P. A representation
$$f = \sum_{i=1}^m b_i g_i$$
with polynomials $b_i \in P$ is called a *standard representation* of f w.r.t. G if
$$max_i(multideg(b_i g_i)) \leq multideg(f).$$
We say that this is a *t-representation* of f w.r.t. G if
$$max_i(multideg(b_i g_i)) \leq multideg(t).$$

The following result regarding S-polynomials and standard representations makes the theorem easier to prove. It states that when all S-polynomials reduce to zero, there is always a representation of $f \in I$ where the maximum multidegree of the terms equals the multidegree of f.

LEMMA 2.39. *Let* $G = \{g_1, \ldots, g_m\}$ *be a finite subset of* P *and the ideal* I *be generated by* G. *If all* g_i, $g_j \in G$ *satisfy* $\overline{S(g_i, g_j)}^G = 0$ *then every* $f \in I$ *has a standard representation*
$$f = \sum_{i=1}^m b_i g_i \tag{2.1}$$
satisfying $multideg(f) = max_i(multideg(b_i g_i))$.

PROOF. We follow the proof of [**CLO96**, p. 82]. Let $f \in I = \langle g_1, \ldots, g_m \rangle$, there are polynomials $h_i \in P$ such that

$$f = \sum_{i=1}^{m} h_i g_i \text{ where } multideg(f) \leq max_i(multideg(h_i g_i)). \tag{2.2}$$

If equality does not hold for the multidegree then some cancellation must occur among the leading terms of (2.2). Lemma 2.37 will enable us to rewrite this in terms of S-polynomials and the assumption that the S-polynomials have zero remainders will allow us to replace it by an expression that involve less cancellation and will eventually satisfy (2.1).

Now, let $\delta(i) = multideg(h_i g_i)$ and define $\delta = max(\delta(1), \ldots, \delta(m))$, then inequality (2.2) becomes

$$multideg(f) \leq \delta.$$

Since a monomial ordering is a well-ordering, according to Definition 2.14, we can select an expression for f such that δ is minimal. The equality $multideg(f) = \delta$ is proven by contradiction.

Suppose $multideg(f) < \delta$. Isolate the terms of multidegree δ in representation (2.2) as follows.

$$\begin{aligned} f &= \sum_{\delta(i)=\delta} h_i g_i + \sum_{\delta(i)<\delta} h_i g_i \\ &= \sum_{\delta(i)=\delta} LT(h_i) g_i + \sum_{\delta(i)=\delta} (h_i - LT(h_i)) g_i + \sum_{\delta(i)<\delta} h_i g_i. \end{aligned} \tag{2.3}$$

The monomials appearing in the second and third sums on the second line all have multidegree $< \delta$. Thus, the assumption that $multideg(f) < \delta$ means that $multideg(\sum_{\delta(i)=\delta} LT(h_i) g_i) < \delta$.

Write $LT(h_i) = c_i x^{\alpha(i)}$. Then the first sum in expression (2.3) has exactly the form described in Lemma 2.37. This implies that this sum is a linear combination of the S-polynomials $S(x^{\alpha(j)} g_j, x^{\alpha(j')} g_{j'})$. However,

$$\begin{aligned} S(x^{\alpha(j)} g_j, x^{\alpha(j')} g_{j'}) &= \frac{x^\delta}{x^{\alpha(j)} LT(g_j)} x^{\alpha(j)} g_j - \frac{x^\delta}{x^{\alpha(j')} LT(g_{j'})} x^{\alpha(k)} g_{j'} \\ &= x^{\delta - \gamma_{jj'}} S(g_j, g_{j'}), \end{aligned}$$

where $x^{\gamma_{jj'}} = LCM(LM(g_j), LM(g_{j'}))$. Thus there are constants $c_{jj'} \in k$ such that

$$\sum_{\delta(i)=\delta} LT(h_i) g_i = \sum_{j,j'} c_{jj'} x^{\delta - \gamma_{jj'}} S(g_j, g_{j'}). \tag{2.4}$$

The hypothesis $\overline{S(g_i, g_j)}^G = 0$ and the Division Algorithm imply

$$S(g_j, g_{j'}) = \sum_{i=1}^{m} a_{ijj'} g_i,$$

where $a_{ijj'} \in P$ and

$$multideg(a_{ijj'} g_i) \leq multideg(S(g_j, g_{j'})) \tag{2.5}$$

for all indices i, j, $j' \in \{1, \ldots, m\}$. Now, multiply $S(g_j, g_{j'})$ by $x^{\delta-\gamma_{jj'}}$ to obtain

$$x^{\delta-\gamma_{jj'}} S(g_j, g_{j'}) = \sum_{i=1}^{m} b_{ijj'} g_i,$$

where $b_{ijj'} = x^{\delta-\gamma_{jj'}} a_{ijj'}$. Then (2.5) and Lemma 2.37 imply that

$$multideg(b_{ijj'} g_i) \leq multideg(x^{\delta-\gamma_{jj'}} S(g_j, g_{j'})) < \delta. \tag{2.6}$$

Substitution of the S-polynomials in the first sum of (2.3) gives

$$\sum_{\delta(i)=\delta} LT(h_i) g_i = \sum_{j,j'} c_{jj'} (\sum_i b_{ijj'} g_i) = \sum_i \widetilde{b_i} g_i$$

for $\widetilde{b_i} \in P$, which by (2.5) and (2.6) have the property that for all i,

$$multideg(\widetilde{b_i} g_i) < \delta.$$

Hence, every term in (2.3) has multidegree $< \delta$. This contradicts the minimality of δ and completes the proof. □

Lemma 2.39 simplifies the proof of the following characterization of Gröbner Bases due to Buchberger.

THEOREM 2.40 (Buchberger's S-pair criterion). *Let I be a polynomial ideal. Then a basis $G = \{g_1, \ldots, g_m\}$ for I is a Gröbner Basis for I if and only if for all pairs $i \neq j$, the remainder on division of $S(g_i, g_j)$ by G is zero.*

PROOF. $\Rightarrow$: If G is a Gröbner Basis, then since $S(g_i, g_j) \in I$, the remainder on division by G is zero by Proposition 2.35.

$\Leftarrow$: Let $f \in I$ be a non-zero polynomial. If the S-polynomials have zero remainders on division by G, then $LT(f) \in \langle LT(g_1), \ldots, LT(g_m) \rangle$ since f has a standard representation by Lemma 2.39. Hence, G is a Gröbner Basis. □

The criterion can be formulated somewhat more rigid. This is done in [**BW93**]. In [**Fau99**] and [**Fau02**] this criterion is used, hence it is included here.

THEOREM 2.41. *Let G be a finite subset of P with 0 not in G. Assume that for all g_1, $g_2 \in G$, $S(g_1, g_2)$ either equals zero or it has a t-representation with respect to G for some $t < LCM(LT(g_1), LT(g_2))$. Then G is a Gröbner Basis.*

PROOF. See [**BW93**, p.219]. □

Theorem 2.40 is a consequence of Theorem 2.41, since

$$LT(S(g_1, g_2)) < LCM(LT(g_1), LT(g_2)).$$

Now we are ready to state the Buchberger Algorithm, which forms the basis of our analysis of algebraic attacks.

THEOREM 2.42 (*Buchberger Algorithm*). *Let $I = \langle f_1, \ldots, f_{m'} \rangle \neq \{0\}$ be a polynomial ideal. Then a Gröbner Basis for I can be constructed in a finite number of steps by the following algorithm.*

- *Input: $F = \{f_1, \ldots, f_{m'}\}$*
- *Output: A Gröbner Basis $G = \{g_1, \ldots, g_m\}$ for I such that $F \subset G$.*

(1) $G := F$
(2) $REPEAT$
(3) $\quad G' := G$

(4) FOR *each critical pair* $(p, q), p \neq q$ *in* G' DO
(5) $\quad S = \overline{S(p,q)}^{G'}$
(6) $\quad IF\ S \neq 0\ THEN\ G := G \cup \{S\}$
(7) $UNTIL\ G = G'$

PROOF. Correctness and termination are proven by the following three observations.

(1) At every stage of the algorithm, $G \subset I$ and $\langle G \rangle = I$ hold;
(2) If $G = G'$ then $\overline{S(p,q)}^{G'} = 0$ for all p, $q \in G$ and, by Buchberger's S-pair criterion, G is a Gröbner Basis at this moment;
(3) The equality $G = G'$ happens in finitely many steps since the ideals $\langle LT(G') \rangle$, from successive iterations of the loop, form an ascending chain. Due to the Ascending Chain Condition (Theorem 2.32), this chain of ideals stabilizes after a finite number of iterations and at that moment $\langle LT(G) \rangle = \langle LT(G') \rangle$ holds.

□

This is the first one of a family of algorithms for computing a Gröbner Basis. A part of this thesis is dedicated to introducing other algorithms for this purpose. Many improvements on the Buchberger algorithm concern the order in which to choose and reduce the critical pairs, which is often called *selection strategy*. Other improvements concern the so-called *criteria* to avoid critical pairs that reduce to zero with respect to the intermediate basis G'. F4 is a well-known example of an algorithm with an improved selection strategy, which is treated in Section 4.3. Gebauer and Möller [**GM88**] and Faugère [**Fau02**] formulated criteria, which shall be treated in Chapter 4.

REMARK 2.43. Commonly, algorithms for the computation of a Gröbner Basis, like the Buchberger Algorithm, append reduced S-polynomials to a set G' to form an intermediate basis of the original ideal I, while simultaneously creating a larger monomial ideal spanned by its leading terms. From this point, the notion of *intermediate basis* refers to a set similar to G' in the Buchberger Algorithm.

There is another criterion for a set of generators to be a Gröbner Basis. This criterion is used for example in the proof of correctness of the algorithm F4. The following two important definitions are equivalent. They are both very common and are included for completeness.

DEFINITION 2.44. [**CLO96**, p. 100] Fix a monomial order and let $G = \{g_1, \ldots, g_m\} \subset P$. Given a polynomial $f \in P$, we say that f *reduces to zero modulo* G, written

$$f \to_G 0,$$

if f can be written in the form

$$f = a_1 g_1 + \ldots + a_m g_m,$$

such that whenever $a_i g_i \neq 0$, we have

$$multideg(f) \geq multideg(a_i g_i).$$

DEFINITION 2.45. [**BW93**, p. 196] The *Normal Form* r of f with respect to G is the reflexive-transitive closure of the reduction by elements of G, $\to_G$. This

means that if there is a reduction chain from f to r by division with elements of G and r can not be reduced any further by elements of G (with respect to some monomial ordering), this r is called the Normal Form of f with respect to G.

The new algorithmic criterion is formulated in the following theorem.

THEOREM 2.46. *Let G be a finite subset of P. G is a Gröbner Basis if and only if the Normal Form of $S(g_1, g_2)$ equals 0, or $S(g_1, g_2) \rightarrow_G 0$, for all g_1, $g_2 \in G$.*

PROOF. See [**CLO96**, p. 101] or [**BW93**, p. 211]. □

Many Gröbner Bases can span the same ideal. Nevertheless, the reduced Gröbner Basis stated in the following definition is a unique representation. For the uniqueness proof, see [**CLO96**, p. 90]. Whenever one has obtained a Gröbner Basis, it is not very hard to compute the reduced Gröbner Basis from it. See for example [**BW93**, p. 217] for an explanation of this algorithm.

DEFINITION 2.47. A *reduced Gröbner Basis* for a polynomial ideal I is a Gröbner Basis for G such that:

(1) $LC(p) = 1$ for all $p \in G$;
(2) For all $p \in G$, no monomial of p lies in $\langle LT(G - \{p\})\rangle$.

In [**Buc85**], Buchberger presents a criterion to cancel redundant intermediate basis elements during a Gröbner Basis algorithm. The details are not included here, but the reader is referred to [**GM88**, p. 283] for a brief explanation.

2.4. Solving systems of polynomial equations

This section explains how to use Gröbner Bases to solve a system of polynomial equations and it provides the algebraic subtleties that are often omitted in articles on algebraic attacks. We claim that for our purposes, a Gröbner Basis in lexicographic order will bring the system of polynomial equations in a 'triangular form'. This claim is stated explicitly by the Shape Lemma. To prove this lemma some fundamental properties concerning ideals are introduced.

DEFINITION 2.48. Given $I = \langle f_1, \ldots, f_m\rangle \subset k[x_1, \ldots, x_n]$, the l-th *elimination ideal* I_l is the ideal of $k[x_{l+1}, \ldots, x_n]$ defined by

$$I_l = I \cap k[x_{l+1}, \ldots, x_n]$$

DEFINITION 2.49. A field k is called a *perfect field* if either its characteristic is 0 or its characteristic is $p > 0$ and we have $k = k^p$, i.e. every element has a p-th root in k.

REMARK 2.50. Finite fields $k = GF(q)$, where $q = p^e$ and $e > 0$, are perfect since the map $x \mapsto x^{p^{e-1}}$ provides p-th roots, because $(x^{p^{e-1}})^p = x$ for all $x \in k$.

It turns out to be important whether the system of equations corresponding to the cryptographic problem describes a finite set of solutions. The ideal spanned by the corresponding polynomials of such a system will be called *zero-dimensional*. The following proposition provides an algorithmic criterion for finiteness.

PROPOSITION 2.51. *[Finiteness Criterion] Let $>$ be an ordering on the monomials $T(P)$ of the polynomial ring $P = \overline{k}[x_1, \ldots, x_n]$. For a system of equations corresponding to an ideal $I = \langle f_1, \ldots, f_m\rangle$, the following conditions are equivalent.*

(1) *The system of equations has only finitely many solutions;*
(2) *For* $i = 1, \ldots, n$, *we have* $I \cap \overline{k}[x_i] \neq \emptyset$;
(3) *The set of monomials* $T(P) \setminus \{LT_>(f) : f \in I\}$ *is finite;*
(4) *The* $\overline{k}$*-vector space* P/I *is finite-dimensional.*

PROOF. See [**KR00**, p. 243]. □

Notice that Buchberger's Algorithm is able to test condition 3 of Proposition 2.51. Furthermore, this criterion implies that for our cryptographic purposes, appending the so-called *field equations*

$$\{x_i^q - x_i : 1 \leq i \leq n\} \tag{2.7}$$

will assure that the ideal is zero-dimensional.

To say something about the possible polynomials occurring in the ideal described by a set of polynomials, the following theorem is of great importance. It states that a polynomial over an algebraically closed field having common zeros with the polynomials in $F = \{f_1, \ldots, f_m\}$, occurs to some power in the ideal spanned by F.

THEOREM 2.52 (*Hilbert's Nullstellensatz*). *Let* k *be an algebraically closed field. If* f, $f_1, \ldots, f_m \in P$ *are such that* $f \in \boldsymbol{I}(\boldsymbol{V}(f_1, \ldots, f_m))$, *then there exists an integer* $e \geq 1$ *such that*

$$f^e \in \langle f_1, \ldots, f_m \rangle$$

and conversely.

PROOF. See [**CLO96**, p. 170]. □

DEFINITION 2.53. Let $I \subset P$ be an ideal. The radical of I denoted by $\sqrt{I}$, is the set

$$\{f : f^e \in I \text{ for some integer } e \geq 1\}.$$

Let $\overline{k}$ denote the algebraic closure of k. Assume we are working with a cryptosystem over $k = GF(q)$, for q the power of a prime p. Suppose $F = \{f_1, \ldots, f_m\} \subset \overline{k}[x_1, \ldots, x_n]$ and the equations

$$\begin{aligned} y_1 &= f_1(x_1, \ldots, x_n) \\ y_2 &= f_2(x_1, \ldots, x_n) \\ &\vdots \\ y_m &= f_m(x_1, \ldots, x_n) \end{aligned}$$

describe the relations between the output characters $y_1, \ldots, y_m \in k$ and the message characters $x_1, \ldots, x_n \in k$. This would be similar to a HFE encryption but note that (a subset of) $x_1, \ldots, x_n$ could also represent the key bits of a blockcipher. Since the message bits are elements of k, we are not interested in the possible solutions existing in $\overline{k} \setminus k$. Therefore, appending the set

$$\{x_i^q - x_i : 1 \leq i \leq n\}$$

to F, creates a radical ideal from which the message bits are still solvable. This is due to Seidenberg's Lemma.

PROPOSITION 2.54 (*Seidenberg's Lemma*). *Let k be a field, let $P = k[x_1, \ldots, x_n]$, and let $I \subseteq P$ be a zero-dimensional ideal. Suppose that for every $i \in \{1, \ldots, n\}$ there exists a non-zero polynomial $g_i \in I \cap k[x_i]$ such that the greatest common divisor (GCD) of g_i and its derivative equals 1. Then I is a radical ideal.*

PROOF. See [**KR00**, p.250]. □

By adding the field equations, there exist g_i as defined in the foregoing proposition, which are relatively prime to their derivative. Therefore, the ideal I is radical and, due to the Finiteness Criterion (Proposition 2.51), zero-dimensional. Furthermore, since $x_i^q - x_i$ factorizes completely over k, the corresponding variety V does not contain points $p \in V$ with coordinates in $\overline{k} \setminus k$.

Now we are ready to state the Shape Lemma, which shows that the lexicographic Gröbner Basis of the ideal I has a triangular form "after most changes of coordinates" [**BMMT94**].

THEOREM 2.55. *[The Shape Lemma] Let k be a perfect field, let $I \subseteq P$ be a zero-dimensional radical ideal such that the x_n coordinates of the points in $\boldsymbol{V}(I)$ are distinct. Let $g_n \in k[x_n]$ be the monic generator of the elimination ideal $I \cap k[x_n]$, and let $d = deg(g_n)$.*

1. *The reduced Gröbner Basis of the ideal I with respect to the lexicographic ordering $x_1 > \ldots > x_n$ is of the form*
$$\{x_1 - g_1, \ldots, x_{n-1} - g_{n-1}, g_n\},$$
where $g_1, \ldots, g_n \in k[x_n]$;
2. *The polynomial g_n has d distinct zeros $a_1, \ldots, a_d \in \overline{k}$, and the set of zeros of I is*
$$\{(g_1(a_i), \ldots, g_{n-1}(a_i), a_i) : i = 1, \ldots, d\}.$$

PROOF. See [**KR00**, p.257]. □

Gröbner Bases turn out to be one of the most important tools for solving algebraic systems. The Gröbner Basis of an algebraic system strongly depends on the choice of the ordering. Different orderings have different advantages. From a complexity point of view, the best ordering is the Graded Reverse Lex Order from Definition 2.18. Computing a Gröbner Basis from a Lexicographic Order is often intractable in large examples where Graded Reverse Lex Order is just able to return a Gröbner Basis. However, the Lexicographic Order is better suited for computing solutions to algebraic systems, as we have explained in the Shape Lemma.

There are several ways to convert a Gröbner Basis efficiently from one ordering to another, like for example the FGLM Algorithm [**FGLM93**] and the Gröbner Walk Algorithm [**CKM97**]. For a zero-dimensional ideal I, the FGLM Algorithm is proven to have polynomial time complexity in the number of monomials not in the ideal. Furthermore, it is considered practical. It is also implemented in Magma as the algorithm of choice for basis conversion.

If the Gröbner Basis G of the ideal I has a triangular form according to the Shape Lemma, then this means that we have found a univariate polynomial g_n in the intersection of the basis and the $(n-1)$-th elimination ideal $G \cap I_{n-1}$. Factorization of g_n returns solutions for x_n, which could be entered in the polynomials of the intersection $G \cap I_{n-2}$. Repeating this for all elimination ideals $I_{n-3}, \ldots, I_1$ gives the variety describing the message bits.

REMARK 2.56. Univariate polynomial factorization is a challenging problem on itself. However, we focus on the Gröbner Basis step in the algebraic system solving since we have reasonable assurance that this will be the most time-consuming step in the process. Univariate factorization over a small finite field is feasible up to large degree. Table 2.1 illustrates the average time it takes Magma to factorize random univariate polynomials into irreducible factors of high degree over $GF(2^8)$ by the Berlekamp algorithm, see for example [**GG99**]. Over small finite fields, one can also consider partial factorization by exhaustive search on the zeros of the univariate polynomial. In the case of the BES for example, this would mean trying all elements of $GF(2^8)$, which can be done very rapidly.

TABLE 2.1. Average timings of Magma factorizing a random univariate polynomial of degree d over $GF(2^8)$, in seconds.

d	400	500	600	700	800	900	1000
time	0.8132	1.1316	1.8767	2.3414	3.3589	4.4704	6.0687

CHAPTER 3

Algebraic attacks on common cryptosystems

The term algebraic attack normally refers to variations on the XL algorithm by Courtois, Klimov, Patarin and Shamir [**CKPS00**]. This technique was inspired by the Relinearization technique adopted by Kipnis and Shamir in the cryptanalysis of the quadratic systems from HFE [**KS99**]. The authors prove that XL does the same computations as Relinearization in a smaller amount of time. XL was not tested on HFE but simulations were done on a toy system of quadratic equations by the authors of [**CKPS00**]. Our simulations with XL applied to HFE lead to interesting observations, which are explained in Chapter 5.

Two years later, a number of ciphers were described in terms of polynomial equations by Courtois and Pieprzyk [**CP02**]. These ciphers included AES. The authors described a way to compute the complexity of the technique based on an approximation of the independent equations and monomials needed to extract a solution. Now, to explain algebraic attacks on the basis of common examples, HFE and Rijndael are introduced in Sections 3.2 and 3.3 respectively. The algebraic attack XL is described in Section 3.4. First, we introduce a small, non-trivial blockcipher with an S-box similar to AES that can be attacked on a desktop computer.

3.1. A small blockcipher

By tackling the HFE challenge, Faugère and Joux [**FJ03**] showed that such a cryptosystem cannot be compared to a random multivariate system in the same number of unknowns. To illustrate on a desktop computer the differences between a random system of equations and a system used in cryptology, the algebraic attack is simulated on HFE and a four byte blockcipher. HFE is scalable in the key length and the small blockcipher in the number of rounds.

Our blockcipher is depicted in Figure 3.1. Let p be the so-called *Rijndael polynomial* $x^8 + x^4 + x^3 + x + 1 \in GF(2)[x]$ and $k = GF(2)[x]/\langle p \rangle$. The variables $x_1, \ldots, x_4 \in k$, $y_1, \ldots, y_4 \in k$ and $k_1, \ldots, k_4 \in k$ are respectively the plaintext, ciphertext and key bytes. S-boxes are denoted by circles with an S, key addition by circles with a '+' and intermediate values are depicted by squares with their corresponding variable. The variables a_i, b_i, c_i, d_i and e_i, for $1 \leq i \leq 4$, represent these values.

Remark 3.1. Notice that the key bytes are denoted by indexed k. Please do not be confused with our standard notation for the finite field, k.

Let us describe the blockcipher in more detail. The blockcipher is depicted as one round but can be scaled to multiple rounds by copying the states $d_1, \ldots, d_4$ to $a_1, \ldots, a_4$ and thus feeding it to the S-box, the Mixing Layer and the key addition step several times. This is illustrated by the dashed arrow in Figure 3.1. Similar

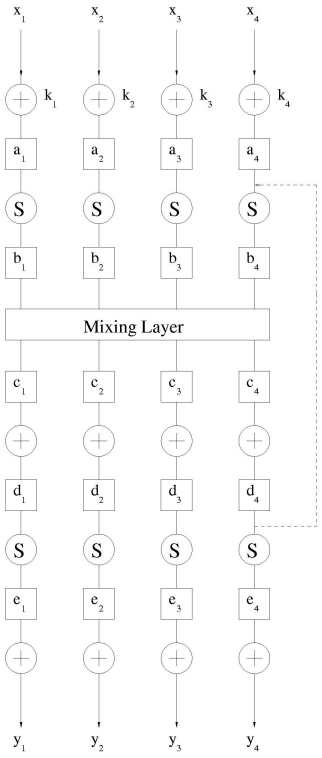

FIGURE 3.1. A small blockcipher.

to AES, our S-box satisfies

$$x \mapsto \begin{cases} x^{-1} & \text{if } x \neq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

The linear diffusion layer, or Mixing Layer, is described by $c_i = \sum_{j \neq i} b_j$ for $0 \leq i \leq 4$. Key addition is straightforward.

Suppose our blockcipher consists of one round. Under the assumption that the S-boxes do not have to 'invert' a zero, the equations describing the dependencies between the intermediate values follow directly from the above. For $1 \leq i \leq 4$, these are given by

$$\begin{aligned} a_i &= x_i + k_i, \\ a_i b_i &= 1, \\ c_i &= \sum_{j \neq i} b_j, \\ d_i &= c_i + k_i, \\ d_i e_i &= 1, \\ y_i &= e_i + k_i. \end{aligned}$$

An encryption can therefore be described as a multivariate quadratic system of 24 equations over $GF(2^8)$, 8 of which are sparse quadratic equations and 16 are linear equations.

Now, let us simulate an algebraic key recovery attack on one round of encryption by this small blockcipher. A key recovery attack assumes the knowledge of the plaintext and ciphertext and tries to extract the key. Usually, the cipher encrypts bytes, which are represented by elements in the finite field according to the following correspondence.

REMARK 3.2. Let $0xh$ be the hexadecimal representation of an integer h, represented in bits by $h_0, \ldots, h_7 \in GF(2)$, p an irreducible polynomial of degree 8, for example the Rijndael polynomial, and θ a root of p. A *byte*, denoted by $0xh$, is represented by an element in the finite field $k = GF(2)/\langle p \rangle$ as

$$\sum_{i=0}^{7} h_i \theta^i.$$

EXAMPLE 3.3. *The binary representation of byte* $0x05$ *is* $(0,0,0,0,1,0,1,0)$. *This implementation appends the representation of* 5 *to the representation of* 0. *In our* $GF(2^8)$ *arithmetic, this byte corresponds to the finite field element* $\theta^4 + \theta^6$.

The assumption regarding the S-box holds with approximated probability $(1 - 1/256)^8 \approx 0.969$, for one round of encryption. Hence, it is likely that a key recovery attack on these equations returns a sensible solution. Consider the following example.

EXAMPLE 3.4. *Set the 4-byte key string* $(0x13, 0x24, 0x57, 0xac)$ *and plaintext* $(0x43, 0x54, 0xfd, 0x23)$. *The 4-byte blockcipher described above returns the ciphertext* $(0x8e, 0xa6, 0x14, 0x34)$. *Hence, the equations over* k *containing only the plaintext and ciphertext bytes are*

$$\begin{aligned}
a_1 &= \theta^6 + \theta + 1 + k_1, \\
a_2 &= \theta^6 + \theta^4 + \theta^2 + k_2, \\
a_3 &= \theta^7 + \theta^6 + \theta^5 + \theta^4 + \theta^3 + \theta^2 + 1 + k_3, \\
a_4 &= \theta^5 + \theta + 1 + k_4, \\
\theta^7 + \theta^3 + \theta^2 + \theta &= e_1 + k_1, \\
\theta^7 + \theta^5 + \theta^2 + \theta &= e_2 + k_2, \\
\theta^4 + \theta^2 &= e_3 + k_3, \\
\theta^5 + \theta^4 + \theta^2 &= e_4 + k_4.
\end{aligned}$$

The reduced lexicographic Gröbner Basis G *of the complete system of equations has 24 elements and takes Magma* 0.35 *second to compute. It has a triangular form as described in the Shape Lemma, Theorem 2.55, and contains a univariate polynomial,* g_{24}, *of degree 41 in the variable* k_4.

To illustrate this, the set $\{LT(g) : g \in G\}$ *consists of all the variables to the first power and the monomial* k_4^{41}. *Since* G *is a reduced Gröbner Basis, no monomials of elements of* G *are divisible by leading terms of other basis elements.*

Factorizing the univariate polynomial g_{24} *gives four irreducible components of respective degrees 1, 6, 16 and 18, all with multiplicity* 1. *The linear factor* $k_4 +$

$\theta^7+\theta^5+\theta^3+\theta^2$ corresponds to the last byte of the key string, $0xac$. Evaluating the other elements of the Gröbner Basis at $k_4=\theta^7+\theta^5+\theta^3+\theta^2$, returns the complete solution in the affine space k^{24}. To obtain the complete key, one only needs to evaluate the three other basis elements that correspond to the key bytes.

Scaling this blockcipher to multiple rounds increases the complexity and the number of equations in the algebraic representation. Therefore, it is of great interest to us to approximate the complexity of an algebraic attack as a function of its number of rounds r or the number of equations and variables involved. For r rounds, there are $12+12r$ equations and equally many unknowns. Using the same key for another encryption would result in four more equations than variables, since the key variables stay unchanged.

We conclude the description of our blockcipher with the equations for multiple rounds. Let $1\leq i\leq 4, 1\leq r'\leq r$ and $v_{i_{r'}}$ denote the variable v_i in the r'-th round, then, under the assumption that no zero is inverted, the r-round cipher satisfies the equations,

$$\begin{aligned}
a_{i_1} &= x_i + k_i,\\
d_{i_1} &= a_{i_1},\\
d_{i_{r'}} b_{i_{r'}} &= 1,\\
c_{i_{r'}} &= \sum_{j\neq i} b_{j_{r'}},\\
d_{i_{r'}} &= c_{i_{r'}} + k_i,\\
d_{i_r} e_i &= 1,\\
y_i &= e_i + k_i.
\end{aligned}$$

Let us conclude this section with the demonstration of a key recovery attack for one round of this blockcipher in Magma. The factorization at the end of this example returns one linear factor, represented as a finite field element where `t` is the root of the Rijndael polynomial. This factor corresponds to the solution of the fourth key bit, which can be substituted in the other three polynomials corresponding to the other key bits.

EXAMPLE 3.5 (Analysis of the blockcipher in Magma).

```
> load "fbbc.m";
Loading "fbbc.m"
> In:=[[ f, f ],    [ f, f ],    [ f, f ],    [ f, f ]];
> k :=[[ 1, 3 ],    [ 2, 4 ],    [ 5, 7 ],    [ a, c ]];
> Out:=fbbc(In,k,1);
********** 4 byte blockcipher ************
Plaintext [
    [ f, f ],
    [ f, f ],
    [ f, f ],
    [ f, f ]
]
Key bytes [
    [ 1, 3 ],
    [ 2, 4 ],
    [ 5, 7 ],
    [ a, c ]
]
```

```
First step a= [ t^7 + t^6 + t^3 + t^2 + t, t^7 + t^5 + t^4 + t^3 +
t^2 + 1, t^7 + t^3 + t, t^5 + t^4 + t^2 + 1 ]

Round 1 Last step e= [ t^6 + t^5 + t^3 + t^2+ t, t^5 + t^4 + t^2 +
t + 1, t^6 + t^4 + 1, t^3 + 1 ]

Return the ciphertext
> F:=fbbcequations(In,Out,1);
> G:=GroebnerBasis(F);
> #G;
24
> // Factorize the last element in the lex Groebner Basis
> f:=Factorization(G[24]);
> // Show f (some lines are omitted)
> f;
[
    <k_4 + t^7 + t^6 + t^3 + t, 1>,
    <k_4^2 + (t^7 + t^3 + t^2 + 1)*k_4 + t^7 + t^5 + t^4 + t^3 + 1, 1>,
    <k_4^2 + (t^7 + t^5 + t^2 + t + 1)*k_4 + t^6 + t^4 + t^3 + t^2, 1>,
    <k_4^16 + (t^3 + t^2 + t)*k_4^15 + (t^7 + t^5 + t^2 + 1)*k_4^14 + (t^7 + t^6
        ...
        t^2, 1>,
    <k_4^20 + (t^5 + t^2 + t + 1)*k_4^19 + (t^6 + t^5 + 1)*k_4^18 + (t^7 + t^4 +
        ...
        + 1)*k_4^2 + (t^7 + t^5 + t^4 + 1)*k_4 + t^6 + t^5 + t^4 + t^3 + t^2, 1>
]
> Evaluate(f[1,1],24,0);
t^7 + t^6 + t^3 + t
> // Set variable a equal to the root in GF(2^8) of basis element, G[24]
> a := Evaluate(f[1,1],24,0);
> a;
t^7 + t^6 + t^3 + t
> // Evaluate the last-but-one polynomial, G[23], at k_4=a
> Evaluate(G[23],24,a);
k_3 + t^6 + t^5 + t^4 + t^2 + 1
```

The complexity of encryption and decryption is linear in the number of rounds, since one simply needs to apply the S-box and the linear diffusion layer more often. However, the algebraic key recovery attack becomes much more difficult as is illustrated in Table 3.1. On our desktop computer, attacking this blockcipher with the F4 implementation of Magma 2.11 becomes infeasible for more than three rounds of encryption.

TABLE 3.1. Time and memory consumption for an algebraic key recovery attack on multiple rounds of the blockcipher in Figure 3.1. We used the F4 implementation of Magma 2.11.

rounds	*time* (*sec*)	*memory* (*kb*)
1	0.021	400
2	62.419	26,261
3	25,533.155	1,319,550

3.2. Hidden Field Equations

The *HFE* cipher was introduced as an improvement of the asymmetric C^* scheme by Matsumoto and Imai [**MI88**] to create signatures and perform short encryptions of short messages. What makes these asymmetric schemes particularly nice is that they do not rely on the unproven intractability assumption of integer factorization or the discrete log problem. Therefore, if these assumptions turn out to be invalid, cryptographers have an alternative problem to base there schemes on. To further stress its importance, a variation of HFE forms the basis of the signature scheme Quartz, which is approved by the NESSIE consortium and is able to produce very short signatures.

This section introduces the HFE scheme, following [**Pat96b**] and [**Kob97**], and explains how to use our Magma implementation. It finishes with statistics describing an attack with the F4 algorithm, which has been included in Magma 2.11 since May 2004.

Suppose Alice wants to send Bob a secure message and Eve tries to eavesdrop on their communication. The public key in C^* and HFE is a set of polynomial equations over a finite field k in the plaintext variables $x_1, \ldots, x_n$,

$$\begin{aligned} y_1 &= p_1(x_1, \ldots, x_n), \\ y_2 &= p_2(x_1, \ldots, x_n), \\ &\vdots \\ y_n &= p_n(x_1, \ldots, x_n). \end{aligned}$$

To encrypt a message $(a_1, \ldots, a_n) \in k^n$, Alice evaluates the polynomials and obtains the encrypted tuple $y = (y_1, \ldots, y_n)$, which she can send to Bob. If Eve intercepts y, she can try to solve the system of equations following from the public key to recover the plaintext. Initially this should discourage Eve, since this is believed to be intractable.

Due to a property specific to C^*, Patarin broke the scheme and improved the setup, which finally lead to HFE. The alphabet of HFE is a finite field, $k = GF(q)$, of characteristic p for prime p. A part of the secret key in HFE is a polynomial with a specific shape.

Definition 3.6. If a polynomial $f \in GF(q^n)[x]$ of degree d satisfies

$$f(x) = \sum_{i,j} \beta_{ij} x^{q^{\theta_{ij}} + q^{\phi_{ij}}} + \sum_l \alpha_l x^{q^{\epsilon_l}} + \mu,$$

for certain parameters β_{ij}, $\alpha_l \in GF(q^n)$ and θ_{ij}, ϕ_{ij} and $\epsilon_l \in \mathbb{Z}_{\geq 0}$, then this polynomial is referred to as an *HFE polynomial*.

For an irreducible $g(x) \in k[x]$ of degree n, $GF(q^n)$ is isomorphic to $k[x]/\langle g(x) \rangle$ and elements of $GF(q^n)$ may be represented as n-tuples over k. A basis, like $\left\{\theta^i : 0 \leq i \leq n-1\right\}$ where θ is a root of polynomial g, determines a correspondence between the extension field $GF(q^n)$ and the vector space k^n. Furthermore, the HFE polynomial f may be represented as a polynomial in n variables $x_1, \ldots, x_n \in k$,

$$f(x_1, \ldots, x_n) = (p'_1(x_1, \ldots, x_n), p'_2(x_1, \ldots, x_n), \ldots, p'_n(x_1, \ldots, x_n)),$$

where $p'_i \in k[x_1, \ldots, x_n]$, for $i = 1, 2, \ldots, n$. The p'_i are quadratic polynomials due to the choice of f and the fact that $x \mapsto x^q$ is a linear function on the extension field $GF(q^n)$.

To prevent ambiguity during decryption, it is necessary to introduce some redundancy in the message to distinguish the correct plaintext from all other possible solutions. This is due to the following observation. Suppose $f(x) = a \in GF(q^n)$. If f consists of only one monomial, it permutes $GF(q^n)$, in which case each a gives rise to exactly one solution x. However, if f consists of more than one monomial in x, it seems to be difficult to choose f such that it is a permutation and each a corresponds to one solution. Obviously, there are at most d solutions to $f(x) = a$. HFE with one monomial is equivalent to the C^* scheme by Matsumoto and Imai.

To complete the main ingredients for the basic version of HFE used for encryption, two functions are introduced. Let s and t be two affine bijections between vector spaces, $k^n \to k^n$. These functions can be represented as n-tuples of polynomials in n variables over field k of total degree 1. Therefore it makes sense to write

$$t(f(s(x_1, \ldots, x_n))) = (p_1(x_1, \ldots, x_n), p_2(x_1, \ldots, x_n), \ldots, p_n(x_1, \ldots, x_n)),$$

with $p_i(x_1, \ldots, x_n) \in k[x_1, \ldots, x_n]$ for $i \in \{1, \ldots, n\}$. The polynomials p_i are again quadratic, due to the choice of s and t.

Hence, the complete setup of the basic HFE scheme is as follows:

- Together with the field k and the extension degree n, these polynomials p_i will form the public key of the HFE scheme and can be computed efficiently;
- The polynomials f, s and t, and the representation of $GF(q^n)$ over k make up the secret key. In [**Pat96b**, p. 35], Patarin advices to choose f to have degree at least 17.
- To encrypt the plaintext $x = (x_1, \ldots, x_n) \in k^n$, compute the ciphertext as described above for the C^* scheme,
$$y = (p_1(x_1, \ldots, x_n), \ldots, p_n(x_1, \ldots, x_n));$$
- To decrypt the ciphertext y, find all solutions z to the equation $f(z) = t^{-1}(y)$ and compute the possibly multiple solutions $x' = s^{-1}(z)$. Lastly, use the redundancy to determine which solution x' corresponds to the real plaintext x.

To conclude the introduction to HFE, this section finishes with an example. It illustrates the execution of HFE and its algebraic cryptanalysis. In Chapter 4 on advanced methods based on Gröbner Bases, Faugère's F5 technique, which tackles the first HFE challenge, is discussed in more detail.

EXAMPLE 3.7. *Suppose the field k equals $GF(q)$ for $q = 2$, set $n = 5$ and $g(x) = x^5 + x^4 + x^3 + x + 1$, then $GF(q^n)$ is isomorphic to $k[x]/\langle g(x)\rangle$. Let θ be a root of g. The basis $\{1, \theta, \theta^2, \theta^3, \theta^4\}$ determines the correspondence between the extension field $GF(q^n)$ and the vector space k^n. Furthermore, define*

$$A = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}, \quad c = (1, 0, 1, 1, 1),$$

$$B = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad d = (1, 0, 1, 0, 0),$$

and the affine transformations s and t by

$$s(x) = Ax + c \text{ and } t(x) = B^{-1}(x - d) \text{ for } x \in k^n.$$

Lastly, set the secret HFE polynomial $f(x) = x * x^8 + x^4 * x^{16}$ for elements x in the extension field $GF(2^5)$. Note that the terms are written as multiplications of x having some power of $q = 2$ in the exponent.

For a plaintext vector $x = (x_1, \ldots, x_5) \in k^5$, compute the affine transformation $s(x)$. Use the basis to represent $s(x) \in k^5$ as an element of the extension field and evaluate f at this point. Next, the public key is created by converting the element back to a vector and applying the affine transformation t. This leads to the following equations for the public key $y = (y_1, \ldots, y_5) = (p_1(x), \ldots, p_5(x))$,

$$\begin{aligned}
y_1 &= x_1^2 + x_1x_2 + x_1x_3 + x_1 + x_2x_5 + x_3 + x_4^2 + x_5 + 1, \\
y_2 &= x_1x_2 + x_1x_3 + x_1x_4 + x_1x_5 + x_2x_3 + x_2 + x_3^2 \\
&\quad + x_3x_5 + x_3 + x_4^2 + x_4 + x_5^2 + x_5 + 1, \\
y_3 &= x_1^2 + x_1x_4 + x_1x_5 + x_1 + x_2x_3 + x_2x_5 + x_2 + x_3^2 \\
&\quad + x_3x_5 + x_4 + x_5^2, \\
y_4 &= x_1^2 + x_1x_2 + x_1x_4 + x_1 + x_2x_3 + x_2x_4 + x_2 \\
&\quad + x_3x_5 + x_5^2 + 1, \\
y_5 &= x_1^2 + x_2^2 + x_2x_3 + x_2x_5 + x_2 + x_3x_4 + x_3x_5 + x_5.
\end{aligned}$$

Assume Alice wants to encrypt plaintext $(1, 1, 1, 1, 1) \in k^5$. All she has to do is evaluate the public key equations at this point. This gives her ciphertext vector $(1, 0, 1, 0, 0)$, which she can send to Bob.

The algebraic attack by Eve is simple for this small example but encounters the subtleties discussed in Section 2.4. All Eve has to do is solve a system of equations for the known $(y_1, \ldots, y_5)$. Computing a reduced lexicographic Gröbner Basis G' of $F = \{p_i - y_i : 1 \leq i \leq 5\}$, will not give her one that satisfies the Shape Lemma, since it is not zero-dimensional. This is due to the Finiteness Criterion defined in Proposition 2.51 and the observation that $T(k[x_1, \ldots, x_n]) \setminus \langle LT(F) \rangle$ is infinite, since the set $\{LT(g) : g \in G\}$ equals $\left\{x_1, x_2, x_3^4, x_3^2x_4, x_3^2x_5, x_3x_4^2, x_3x_5^3, x_4^6\right\}$.

Appending the field equations $\left\{x_i^2 - x_i : 1 \leq i \leq 5\right\}$ to the set F will create a zero-dimensional radical ideal. Hence, the reduced lexicographic Gröbner Basis, G, of this ideal is

$$\begin{aligned}
g_1 &= x_1 + x_5, \\
g_2 &= x_2 + 1, \\
g_3 &= x_3 + x_5, \\
g_4 &= x_4 + x_5, \\
g_5 &= x_5^2 + x_5
\end{aligned}$$

and the solutions $(0,1,0,0,0)$, $(1,1,1,1,1)$ *in* k^5 *follow immediately. Notice that in this case redundancy is necessary to determine which solution corresponds to the original plaintext, since the* x_5 *coordinates of the points in the affine variety are not distinct.*

Since May 2004 a very efficient implementation of F4 has been included in Magma 2.11. We ran a simulation on HFE equations for key lengths varying from 10 bits to 30 bits. The secret HFE polynomial was chosen randomly with average degree 40 and maximal degree 64. For every key length, twenty samples were taken. The average time and memory consumption on the desktop computer of Remark 1.9 are depicted in Figures 3.2 and 3.3 respectively.

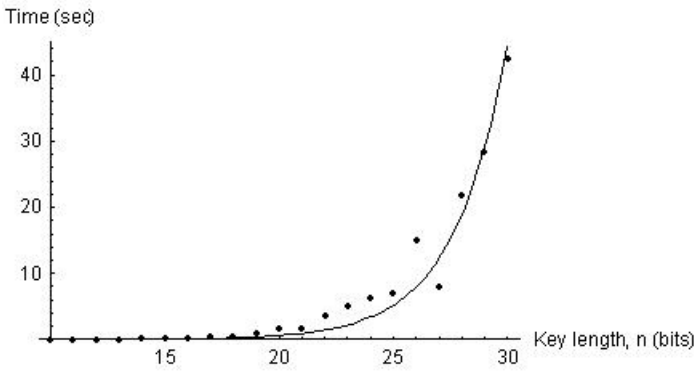

FIGURE 3.2. Average time to compute a Gröbner Basis with F4.

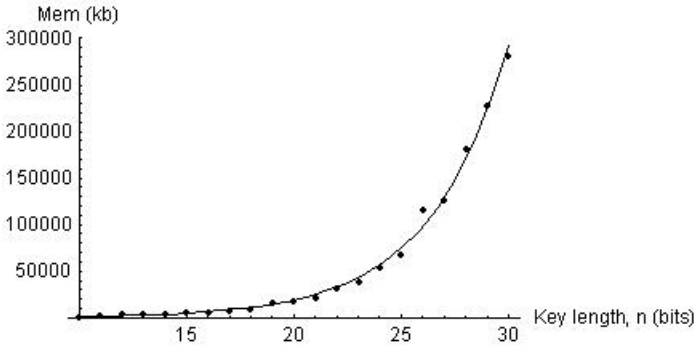

FIGURE 3.3. Average process memory usage to compute a Gröbner Basis with F4.

The least-square fits are also plotted for both data sets. The least-square fits for respectively the time and memory data equal

$$0.000118 * 1.53^n \text{ and } 78.6 * 1.32^n.$$

REMARK 3.8. The depicted curves for time and memory consumption are close to optimal. This is due to the following observation similar to Remark 1.10. For both data sets, we took the natural logarithm of the values and applied a least-square fit with a linear function. The base number of the exponential fit followed from the derivative of the linear function.

The simulations suggest that finding a Gröbner Basis with F4 has exponential complexity with respect to the key length. The algorithm used is due to Allen Steel, who explains on his website [**Ste04**] that the performance of the improved F4 implementation in Magma 2.11 is comparable to the performance of the implementation described in [**Fau99**]. Therefore, one could say that this latest Magma version currently includes one of the fastest Gröbner Basis implementations.

One important claim regarding the complexity of HFE is the following. In [**FJ03**], Faugère and Joux show that with their algorithm F5 the degree of intermediate polynomials during the Gröbner Basis computation is kept to a minimum. This claim is supported by their attack on the 80 bit HFE challenge [**Pat96a**]. The F5 algorithm did not exceed polynomials of degree 4, while the degree of the secret HFE polynomial equalled 96. Based on a combinatorial argument specific to HFE, Faugère and Joux computed the maximal degree of polynomials during the application of F5 to HFE. This argument was founded with simulations and the results are summarized in Table 3.2.

TABLE 3.2. Maximal degree of intermediate polynomials during the execution of F5, d_{F5}, for secret HFE polynomial of degree d.

d	$3 \leq d \leq 12$	16	$17 \leq d \leq 96$	128	$129 \leq d \leq 512$	$513 \leq d \leq 1280$
d_{F5}	3	3	4	4	5	6

In Section 5.2, we discuss a theoretical lower bound for the largest degree during Gröbner Basis computations for random systems of equations. This approach to approximate a lower bound for algebraic systems of equations seems to become more widespread. The result by Patarin and Joux exposes the structure hidden in non-homogeneous HFE equations and does not obey the theoretical lower bound from Section 5.2. Therefore, it remains questionable in which generality this approximation is applicable.

Faugère and Joux show [**FJ03**, p. 56] that F5 applied to HFE has polynomial complexity in the number of key bits, n. In Section 5.1 we argue that XL is a cumbersome way of computing a Gröbner Basis for general algebraic systems. However, we also show that the degree for which XL finds a HFE plaintext stays constant as the number of key bits increases, a claim derived from the analysis of d_{F5} in [**FJ03**]. This makes the complexity of XL applied to HFE polynomial, as follows from the discussion regarding the complexity of XL in Section 3.4.

3.3. AES and BES

As mentioned earlier, in Section 1.1, the blockcipher Rijndael [**DR99**] was chosen by the National Institute of Standards and Technology as the Advanced Encryption Standard, *AES*, as a successor to the Data Encryption Standard (*DES*). At this moment, AES is being adopted on a large scale.

At Eurocrypt 2000, Courtois, Klimov, Patarin and Shamir presented the algorithm XL [**CKPS00**], which was explained later as a possible threat to AES

[**CP02**]. The next section deals with XL and its predecessor, Relinearization. In this section, AES and its simpler algebraic description, BES, are introduced. BES stands for Big Encryption System.

Murphy and Robshaw list the significant encryption steps clearly in [**MR02b**, Chpt. 3]. We shall treat the 128 bit AES according to this description. This section focusses on a typical round of the cipher. The first and last rounds of AES have a different but related form. For the full description, the reader is referred to FIPS 197 [**Nat01**].

The field $k = GF(2^8)$ can be constructed as $GF(2)[x]/\langle p\rangle$, defined by the Rijndael polynomial $p(x) = x^8 + x^4 + x^3 + x + 1$. The basic 128 bit AES encrypts a 16 byte block using a 16 byte key with 10 encryption rounds. The input of AES can be viewed as a square array of bytes. One round of encryption in AES is specified by the following three transformations:

(1) **The AES S-box** The value of each byte in the array is substituted according to a table look-up S, which is a combination of three transformations:
   (a) The input w is 'inverted' as in Equation 3.1 in Section 3.1. The result is denoted by x;
   (b) The intermediate value x is regarded as a $GF(2)$ vector of dimension 8 and transformed using a square matrix L_A. The transformed vector $L_A x$ is then regarded in the natural way as an element of k;
   (c) The output of the S-box is $L_A x + 0x63$, where addition is with respect to $GF(2)$.

(2) **The AES linear diffusion or mixing layer**
   (a) Each row of the array is rotated by a certain number of byte positions;
   (b) Each column of the array is considered to be a vector $y \in k^4$ and is transformed to the column Cy, by the square matrix C.

(3) **The AES subkey addition** Each byte of the array is added to a byte from the corresponding array of round subkeys.

Similar to our small blockcipher, the AES S-box sometimes needs to invert a zero. For the standard 128-bit AES, this happens in approximately 53% of the encryptions. Therefore, like in Section 3.1, an algebraic description of AES that neglects the zero mapping by the S-box, is unreliable with this probability, which is considered fair for the purpose of cryptanalysis.

As we have seen in the description of HFE, it is common in cryptography to switch between representations of a byte. At one moment, it is interpreted as an element of the finite field $GF(2^8)$, while at another moment the byte is represented by a vector over $GF(2)$. The 128 bit AES is a blockcipher that encrypts 16 bytes simultaneously, while switching back and forth between both representations of a byte. In particular the use of the $GF(2)$-linear operation in the S-box makes it harder for the cryptanalyst to translate the cipher in simple algebraic equations.

However, by mapping an element in the state space of AES to its vector conjugate, Murphy and Robshaw were able to create *BES*, a simpler equivalent representation of AES in terms of the algebraic description, than, for example, Courtois and Pieprzyk discussed in [**CP02**]. We give a brief description of BES and explain the correspondence between typical AES and BES encryptions.

DEFINITION 3.9. For any element $a \in k$, we can define the *vector conjugate* of a as the eight $GF(2)$-conjugates,

$$\phi(a) = (a^{2^0}, a^{2^1}, \ldots, a^{2^7}),$$

where $\phi : GF(2^8) \to GF(2^8)^8$ is called the *vector conjugate mapping*. For $a' \in k^n$ this definition extends in the natural way to a mapping from vector space k^n to k^{8n} componentwise,

$$\phi(a') = (\phi(a'_0), \phi(a'_1), \ldots, \phi(a'_{n-1})).$$

The vector conjugate mapping is additive and preserves inverses.

DEFINITION 3.10. When each successive set of eight components in $a \in k^{8n}$ forms an ordered set of $GF(2)$-conjugates, a has the *conjugacy property*.

Now, the vector conjugacy mapping gives a natural embedding of the AES state space in the BES state space. To be precise, the AES subset of states of BES is a subset of $\phi(k^{16}) \subset k^{128}$. The description of the BES rounds is actually an affine transformation of the componentwise inverse of an element b in the BES state space k^{128},

$$Round_{BES}(b, k_{BES_i}) = M_{BES} b^{-1} + k_{BES_i},$$

where M_B is a (128×128) matrix over k performing linear diffusion within BES. The variable k_{BES_i} is the i-th round key and inversion is interpreted as in expression (3.1). Now, an AES plaintext $a \in k^{16}$ can be encrypted equivalently by the BES cipher,

$$Round_{AES}(a, k_{AES_i}) = \phi^{-1}(Round_{BES}(\phi(a), \phi((k_{AES_i})))).$$

The key schedule uses the same operations as the AES encryptions and can be described by similar algebraic operations over k.

Each of the BES operations, namely the inversion and the affine transformation over k, are algebraic transformations of elements in vector space k^{128}. Let the plaintext and ciphertext be denoted by respectively p and $c \in k^{128}$ respectively and the state vectors before and after the i-th application of the inversion layer by w_i and $b_i \in k^{128}$ respectively. With the necessary notation in place, a multivariate, quadratic system for BES satisfies,

$$\begin{aligned}
w_0 &= p + k_0, \\
x_i &= w_i^{-1}, && \text{for } i = 0, \ldots, 9, \\
w_i &= M_{BES} x_{i-1} + k_i, && \text{for } i = 0, \ldots, 9, \\
c &= M'_{BES} x_9 + k_{10},
\end{aligned}$$

where M'_{BES} is slightly different from M_{BES}, since the final round in BES does not mix the columns similarly to the matrix C in the above description of AES.

Let the $(8j+m)$-th component of x_i, w_i and k_i be denoted by $x_{i,(j,m)}$, $w_{i,(j,m)}$ and $k_{i,(j,m)} \in k$ and the matrices M_{BES} and M'_{BES} over k^{128} by (α) and (β). Following from the algebraic description, the following system of very sparse multivariate quadratic equations over the finite field k describes a BES encryption. For

$j \in \{0, \ldots, 15\}$, $m \in \{0, \ldots, 7\}$ these are given by

$$0 = w_{0,(j,m)} + p_{(j,m)} + k_{0,(j,m)},$$

$$0 = x_{i,(j,m)} w_{i,(j,m)} + 1, \qquad \text{for } i = 0, \ldots, 9, \tag{3.2}$$

$$0 = w_{i,(j,m)} + k_{i,(j,m)} + \sum_{j',m'} \alpha_{(j,m),(j',m')} x_{i-1,(j',m')}, \qquad \text{for } i = 0, \ldots, 9, \tag{3.3}$$

$$0 = c_{(j,m)} + k_{10,(j,m)} + \sum_{j',m'} \beta_{(j,m),(j',m')} x_{9,(j',m')}.$$

When considering an AES encryption in the BES framework, one can add the equations

$$x_{i,(j,m)}^2 + x_{i,(j,m+1)} = 0, \quad w_{i,(j,m)}^2 + w_{i,(j,m+1)} = 0,$$

for $i \in \{0, \ldots, 9\}$, since those state variables satisfy the conjugacy property.

AES and BES are scalable. Simply increasing the number of rounds implies more equations defining the encryption. To simulate algebraic attacks on AES, one could scale down the number of rounds. However, there will still be a lot of equations to take into account since the mixing layer, corresponding to Expression (3.3), involves a large number of variables. This implies an increase in quadratic equations, because these variables correspond to S-box inversions defined in Expression (3.2). Since the corresponding algebraic systems are huge, the small blockcipher of Section 3.1 and a scalable version of HFE are implemented in Magma to investigate the behavior of various Gröbner Basis algorithms on cryptosystems. The Magma code is included in Appendix A.

3.4. Relinearization and XL

In their attack on HFE, Kipnis and Shamir [**KS99**] describe a technique called Relinearization to solve a system of equations. In this section, Relinearization is briefly described and shown to be related to XL. Next, XL is studied in more detail and the complexity estimations by the authors of [**CKPS00**] are presented.

Let k be a finite field and suppose we have a system of quadratic polynomials $F = \{f_1, \ldots, f_m\} \subset k[x_1, \ldots, x_n]$ representing a cryptosystem. The original *Linearization* technique regards all monomials $x_i x_j$ as new variables y_{ij}, thereby obtaining a linear system that does not have full rank in general. If the number of distinct monomials in the original system, $T(F)$, equals the number of linear independent equations, the system is solvable by Linearization.

Relinearization is based on the observation that it might be possible to solve this newly obtained linear system by adding trivial equations of the form

$$(x_a x_c)(x_b x_d) = (x_a x_d)(x_b x_c)$$

for $1 \leq a \leq b \leq c \leq d \leq n$ and thus obtaining a system that can be linearized again. Since only four variables are considered in these extra constraints, this is called fourth degree Relinearization. Similarly, one could choose to add the equations describing the associativity and commutativity in more variables. The following proposition states the number of equations added by this technique. Because it was omitted in [**KS99**], it is included here.

PROPOSITION 3.11. *The number of extra equations of the form*

$$(x_a x_c)(x_b x_d) = (x_a x_d)(x_b x_c),$$

for $1 \leq a \leq b \leq c \leq d \leq n$, that are created by Relinearization as described above, is

$$2\binom{n}{4} + 3\binom{n}{3} + \binom{n}{2} = \frac{1}{12}(n^4 - n^2).$$

PROOF. The number of ways to choose four different indices a, b, c, d equals $\binom{n}{4}$. Then $(x_a x_b)(x_c x_d)$ can be parenthesized in two other ways, namely

$$(x_a x_b)(x_c x_d) = (x_a x_c)(x_b x_d) = (x_a x_d)(x_b x_c).$$

Similarly, after choosing three different indices, this expression can be parenthesized in three more ways. After choosing two different indices, there remains only one other way to place the parentheses. □

In [**CKPS00**], Courtois, Klimov, Patarin and Shamir show that many of the equations generated by Relinearization are linearly dependent and that the algorithm is therefore less efficient. To solve this problem, they introduce Extended Linearization, or XL, which is proven to be simpler and more powerful than Relinearization. The following is a description of XL.

DEFINITION 3.12. [**CKPS00**, p. 6] Let $F = \{f_1, \ldots, f_m\} \subset k[x_1, \ldots, x_n]$ be the polynomials in the algebraic description of a cryptosystem. For a parameter $D \in \mathbb{Z}_{\geq 0}$, the *XL* algorithm is described as follows.

(1) **Multiply** Generate all the products $x^\alpha f_i$, for $\alpha \in \mathbb{Z}^n_{\geq 0}$ of total degree $\leq D$;
(2) **Linearize** Consider each monomial x^α, $\alpha \in \mathbb{Z}^n_{\geq 0}$, of degree $\leq D$ as a new variable and perform Gaussian elimination on the equations obtained in Step 1. The ordering on the monomials must be such that all the terms containing one variable, say x_n, are eliminated last;
(3) **Solve** If D is chosen such that Step 2 yields at least one univariate equation in the powers of x_n, then solve this equation over the finite field k, for example with Berlekamp's algorithm;
(4) **Repeat** Simplify the equations with the solutions from Step 3 and repeat the process to find the values of the remaining variables.

Let us illustrate this with an example. HFE encrypts the 4-bit plaintext $(1, 1, 0, 1)$ with a random secret key and returns the ciphertext y and public key F. XL is applied to F with the field equations included and $D = 3$. The columns of the matrix representing the monomials are ordered using Lexicographic ordering. The algorithm finds four univariate polynomials with single roots in $GF(2)$. These roots correspond to the plaintext. Lastly, it returns the set of original polynomials, with the solutions substituted into them.

EXAMPLE 3.13 (XL applied to HFE in Magma).

```
> load "XL.m";
> load "HFE.m";
> y, F:= HFEencrypt([1,1,0,1]);
> // Apply XL to F for degree 3
> S:=XL(F,3);
    ...
> // Print the solution found by XL
> S;
[ [ 4, 1 ],    [ 3, 0 ],    [ 2, 1 ],    [ 1, 1 ] ]
```

REMARK 3.14. The parameter D is kept fixed in the original description of XL. By doing this, the authors keep the description of the XL algorithm very general. However, using the set of equations corresponding to degree D obtained after Gaussian reduction in Step 2 as input of a run of the algorithm for degree $D' = D + 1$, would create an incremental version of the algorithm. Similar to the normal selection strategy introduced in Section 4.2, one could also choose the lowest possible D'. This is discussed in Section 5.1.

During the course of the algorithm, the most time-consuming step is Gaussian elimination of the matrix representing all polynomials generated in the first step of the algorithm. The parameter D mainly determines the size of this matrix, therefore being a very important variable in the expression for complexity. Until recently, the correct D that solves the system of equations was approximated by experiments.

DEFINITION 3.15. For a set of equations, the variable d_{min} refers to the smallest degree at which XL finds a complete or partial solution.

If the number of original equations m equals the number of variables n like with HFE, the authors suggest to choose $D \approx 2^n$. However, "if $m = n + 1$, then $D = n$ and if m is larger [the authors] expect to have $D \approx \sqrt{n}$", see [**CKPS00**, p. 10]. This statement is based on the following proposition.

PROPOSITION 3.16. *Let $F = \{f_1, \ldots, f_m\} \subset k[x_1, \ldots, x_n]$ be a set of m quadratic polynomials describing a problem in cryptography. Under the assumption that almost all of the equations of the form $x^\alpha f_i$ of degree $\leq D$ generated by XL are linearly independent, XL has estimated complexity*

$$\mathcal{O}((\frac{n^D}{D!})^\omega) \text{ with } D = \mathcal{O}(\frac{n}{\sqrt{m}}) \text{ and } \omega = 2.376.$$

PROOF. The number of monomials of degree $\leq s$ in $k[x_1, \ldots, x_n]$ is $\binom{n+s}{s}$, thus the number of generated equations of degree $\leq D$ in Step 1 of XL is of order

$$\mathcal{O}(\frac{n^{D-2}}{(D-2)!} \cdot m),$$

while there are about

$$\mathcal{O}(\frac{n^D}{D!}),$$

linearized variables in Step 2. Hence, under the assumption that the generated equations are linearly independent, Gaussian reduction in Step 2 is expected to return a univariate polynomial if the number of equations equals the number of monomials, or equivalently, for those D of order

$$\mathcal{O}(\frac{n}{\sqrt{m}}).$$

Gaussian reduction has exponential complexity in the size of the matrix. The complexity exponent equals $\omega = 2.376$ for the fastest method based on the matrix multiplication algorithm by Coppersmith and Winograd published in [**CW90**]. □

Two observations weaken the claims of Proposition 3.16. First, assuming that most of the equations generated by XL are linearly independent seems a misconception. Based on simulations, the designers of XL [**CKPS00**, pp.7,8] and Moh

[**Moh01**, p.6] present a ratio between the number of linearly independent equations and the total number of equations generated in the first step of XL. They show that this ratio decreases for increasing values of D at which XL returns a solution. This is summarized in Table 3.3.

TABLE 3.3. The ratio between the number of linearly independent equations and the total number of equations generated by the first step of XL.

D	4	5	6	7	10	14	16
average ratio	0.8625	0.7141	0.6146	0.5186	0.4856	0.4181	0.3900

The second observation comes from [**MR02b**] and [**Moh01**]. Murphy and Robshaw mention that the number of linearly independent equations can never exceed the number of monomials. A trivial notion that seems to be overlooked by the designers of XL. In practice, the difference between the number of linearly independent polynomials and the size of the support stalls at some point, since some monomials are not included in the ideal. This happens for example when HFE has multiple plaintexts corresponding to a ciphertext. In this case, there are variables, x_i, at which solutions differ and the monomial x_i is not included in the ideal.

Furthermore, Moh shows that XL probably finds a solution if the difference between the number of independent equations and the number of monomials becomes smaller than D or $D + 1$, depending on whether there are constant terms in the system of equations. This is discussed in detail in Section 5.2.

The conjectured sub-exponential behavior of XL for overdetermined systems is a strong claim. To sketch how much time it takes an efficient implementation in C or C^{++} to solve an algebraic system, we plotted the time it took Magma to do the matrix reduction during XL in Figure 3.4. The abbreviation $MQ(n, m)$ stands for m random multivariate, quadratic equations in n variables. These random quadratic equations are homogeneous and the corresponding matrices, therefore, sparser than with HFE.

The sub-exponential behavior of barely overdetermined systems, leads to a variation of XL called Fixed XL, or *FXL*. This algorithm is identical to XL but with some variables fixed to get an overdetermined system of equations. Similarly, Extended Sparse Linearization, or *XSL*, is another variation on the same theme. It is based on the suspicion that a good selection criterion is to use products of monomials that already appear in other equations. Similarly, during the construction of S-polynomials in the Buchberger Algorithm, one cancels two leading terms by multiplying with their least common multiple.

In Chapter 5, we show that applying XL is essentially a cumbersome way of calculating a Gröbner Basis. In this field of mathematics, the way to choose which monomials to multiply with the original polynomials is extensively studied. Therefore, this report continues with an explanation of advanced techniques for Gröbner Basis computations.

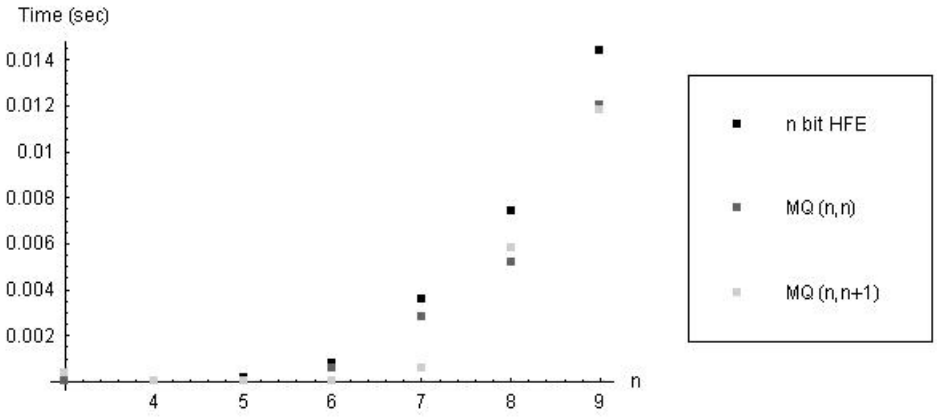

FIGURE 3.4. The average time it takes Magma 2.11 to row reduce a matrix generated by the XL algorithm (50 samples).

CHAPTER 4

Advanced Gröbner Bases techniques

To investigate the vulnerability of cryptosystems to algebraic attacks, we studied some of the most promising tools for solving systems of algebraic equations. We noticed that the cryptanalysis tool XL was very similar to existing Gröbner Basis techniques. This idea was later confirmed by the appearance of the article by Sugita, Kawazoe and Imai [**SKI04**] on the IACR e-print server in May 2004. The authors will present these results at AsiaCrypt 2004 in a joined work with Faugère and Ars.

In this chapter, we explain how algorithms for finding Gröbner Bases are related to Gaussian elimination. This is done by extending the application of linear algebra to find resultants in Section 4.1. Secondly, the normal selection strategy is discussed by means of the Homogeneous Buchberger Algorithm in Section 4.2. Then, in Section 4.3 Faugère's algorithm F4 is introduced, which is a leading algorithm for finding Gröbner Bases. Finally, two improved selection strategies are explained in Section 4.4, being the Gebauer and Möller Installation and the one introduced with F5.

4.1. Linking Gröbner Bases and linear algebra

The link between finding a Gröbner basis and linear algebra is introduced by Lazard in [**Laz83**]. A good way to understand the similarity is by considering the simplest case of two univariate polynomials,

$$f = a_0 + a_1 x + \ldots + a_l x^l, \quad a_l \neq 0,$$
$$g = b_0 + b_1 x + \ldots + b_m x^m, \quad b_m \neq 0.$$

In this case, the standard method for computing a Gröbner basis is exactly the Euclidean algorithm for GCD's and resultants, see for example [**CLO96**]. The second way to compute the resultant is by evaluating the determinant of the Sylvester matrix, as defined in Definition 4.1. For this purpose, Gaussian elimination has complexity $\mathcal{O}((deg(f) + deg(g))^3)$. The following simple modification gives an algorithm equivalent to Euclid's one with $\mathcal{O}(deg(f)deg(g))$ complexity.

DEFINITION 4.1. The following $m+l$ by $m+l$ matrix is called the *Sylvester matrix*.

$$\begin{pmatrix}
a_l & & & & b_m & & & \\
a_{l-1} & a_l & & & b_{m-1} & b_m & & \\
a_{l-2} & a_{l-1} & \ddots & & b_{m-2} & b_{m-1} & \ddots & \\
\vdots & & \ddots & a_l & \vdots & & \ddots & b_m \\
 & \vdots & & a_{l-1} & & \vdots & & b_{m-1} \\
a_0 & & & & b_0 & & & \\
 & a_0 & & \vdots & & b_0 & & \vdots \\
 & & \ddots & & & & \ddots & \\
 & & & a_0 & & & & b_0
\end{pmatrix}$$

This matrix has m columns with the coefficients a_i of f and l columns with coefficients of b_i of g. The columns of the Sylvester matrix can be interpreted as representations of the univariate polynomials $x^i f(x)$ for $i \in \{0, \ldots, m-1\}$ and $x^j g(x)$ for $j \in \{0, \ldots, l-1\}$.

The Gaussian method starts with subtracting the first column from the $(m+1)$-th one b_m/a_l times. If $m \geq l$, the same computations allow us to subtract the i-th column from the $(m+i)$-th one b_m/a_l times, for $i \in \{1, \ldots, l\}$. This operation can be interpreted as replacing the coefficients of g by $h = g - (b_m/a_l)x^{m-l}f$. In this case the Sylvester determinant of f and g is reduced to a_l times the Sylvester determinant of f and h. If we iterate this proces, we simulate Euclid's algorithm in the reduction of a Sylvester matrix. On the other hand, Euclid's algorithm can be viewed as a way to save memory and time in the reduction of a Sylvester matrix.

Now, the correspondence between ideals and infinite linear bases is put forward. Let P be the polynomial ring $k[x_1, \ldots, x_n]$ over a field k and $I = \langle f_1, \ldots, f_m \rangle$ an ideal generated by the polynomials $f_i \in P$, $0 \leq i \leq m$. As a k-vector space, I is generated by the

$$x^\alpha f_i \text{ for all } i \in \{1, \ldots, m\} \text{ and } \alpha \in \mathbb{Z}^n_{\geq 0}. \tag{4.1}$$

A property of Gröbner Bases is that they provide a finite description of the linear basis of I. This is stated in the following proposition. It turns out to be useful to compute the vector space dimension of elements in the ideal up to a given degree.

PROPOSITION 4.2. [**Laz83**, p.148] *The set $F = \{f_1, \ldots, f_m\} \subset P = k[x_1, \ldots, x_n]$ is a Gröbner Basis for the ideal $I = \langle F \rangle$ if and only if the following set of polynomials, B, is a basis of the vector space corresponding to I,*

$$B = \left\{ x^\alpha f_i : 1 \leq i \leq m, \alpha \in \mathbb{Z}^n_{\geq 0} \text{ and not } LM(f_j) | LM(x^\alpha f_i) \text{ for } j < i \right\}. \tag{4.2}$$

On the basis of P consisting of all monomials $T(P)$, the generating set from (4.2) defines a possibly infinite matrix, which has the following properties:

(1) The number of non-zero entries of each row are finite and consist of coefficients of one of the f_i's;
(2) Each column has a finite number of non-zero entries. If the column corresponds to some monomial x^α, then the non-zero entries must correspond to generators $x^\beta f_i$ where $x^\beta | x^\alpha$, $\alpha, \beta \in \mathbb{Z}^n_{\geq 0}$.

As in the Sylvester matrix, polynomials are represented as tuples of coefficients for every occurring monomial. However, we choose to switch the interpretation of

rows and columns, since the original description of the algorithm F4 uses matrices of this form.

In Section 4.3, this representation will be used to reduce polynomials during a Gröbner Basis computation. Before we continue with the description of F4, the Homogeneous Buchberger Algorithm is introduced.

4.2. Homogeneous Buchberger Algorithm

The Homogeneous Buchberger Algorithm forms a framework for many advanced algorithms to compute Gröbner Bases and enables us to calculate Gröbner Bases for parts of ideals up to elements of a certain degree.

DEFINITION 4.3. A polynomial f in $P = k[x_1, \ldots, x_n]$ is called homogeneous of total degree d if every term appearing in f has total degree d. For general $g \in P$, the *homogenous component* of degree d of g is the sum of terms having total degree d.

DEFINITION 4.4. I is said to be a *homogeneous ideal* if for each $f \in I$, the homogeneous components are in I as well.

While it seems straightforward to compute a Gröbner Basis degree by degree, it is remarkable that the problem of writing down the algorithm in full detail is hard to find in the literature. We follow the explanation of the draft of the book Computational Commutative Algebra 2 [**KR04**], which can be obtained from the authors Kreuzer and Robbiano.

PROPOSITION 4.5. *Let P be a polynomial ring and $I \subset P$ an ideal generated by the set of homogeneous polynomials $F = \{f_1, \ldots, f_m\}$.*

- *Buchberger's Algorithm applied to F, returns a homogeneous Gröbner Basis of I;*
- *The reduced Gröbner Basis of I consists of homogeneous vectors.*

PROOF. See [**KR04**, p. 81]. □

THEOREM 4.6 (*Homogeneous Buchberger Algorithm*). *Let $\{f_1, \ldots, f_m\}$ be a set of homogeneous polynomials spanning the ideal I. Then a Gröbner Basis for I can be constructed in a finite number of steps by the following algorithm.*

- *Input: $F = \{f_1, \ldots, f_m\}$*
- *Output: A tuple $G = (g_1, \ldots, g_{s'})$, the elements of which satisfy*

$$totaldeg(g_1) \leq totaldeg(g_2) \leq \ldots \leq totaldeg(g_{s'})$$

and form a Gröbner Basis of the ideal I.

(1) $B := \{\}, G = ()$ *and* $s' := 0$
(2) $REPEAT$
(3) $\quad d_1 := min\{totaldeg(f) : f \in F\}$
(4) $\quad d_2 := min\{totaldeg(LCM(LT(g_i), LT(g_j))) : (i,j) \in B\}$
(5) $\quad d := min\{d_1, d_2\}$
(6) $\quad B_d := \{(i,j) \in B : totaldeg(LCM(LT(g_i), LT(g_j))) = d\}$
(7) $\quad B := B \setminus B_d$
(8) $\quad F_d := \{f \in F : totaldeg(f) = d\}$
(9) $\quad F := F \setminus F_d$
(10) $\quad REPEAT$

(11) *IF* $B_d = \{\}$, *THEN choose* $f \in F_d$ *and delete it from* F_d
(12) *ELSE*
(13) *Choose a pair* $(i, j) \in B_d$, *delete it from* B_d
(14) $f := S(g_i, g_j)$
(15) *IF* $\overline{f}^G = 0$ *THEN continue with Step 10*
(16) *ELSE*
(17) $s' := s' + 1$
(18) $g_{s'} := \overline{f}^G$ *and append* $g_{s'}$ *to* G
(19) *Add pairs* $(1, s'), \ldots, (s' - 1, s')$ *to* B
(20) *UNTIL* $B_d = \{\}$ *or* $F_d = \{\}$
(21) *UNTIL* $B = \{\}$ *or* $F = \{\}$

PROOF. See [**KR04**, p.83]. □

REMARK 4.7. The description of the Homogeneous Buchberger Algorithm in [**KR04**, p.83] is more general in that it holds in the case of any positively graded module and chooses the degree of the critical pairs by lowest multidegree with respect to the lexicographic ordering, instead of lowest total degree.

Let the elements of an ideal I with degree less than or equal to d be denoted by $I_{\leq d}$. The Homogeneous Buchberger Algorithm is able to compute a basis with the same properties as a Gröbner Basis restricted to the elements in I of degree $\leq d$.

DEFINITION 4.8. Let G be the result of the Homogeneous Buchberger Algorithm for the ideal I. The elements in G of total degree $\leq d$ form the set $G_{\leq d}$, which shall be referred to as a *d-truncated Gröbner Basis* of the ideal I.

In [**BW93**] the distinction between the homogeneous and non-homogeneous case is made by calling Definition 4.8 a *D-Gröbner Basis* at page 455 and Definition 4.9 a *d-Gröbner Basis* at page 473. In a non-homogeneous context, a d-truncated Gröbner Basis can be characterized as follows.

DEFINITION 4.9. Let G be a finite subset of the ideal $I \subset P$. G is called a *d-truncated Gröbner Basis* of the ideal I if for all $f, g \in G$ with total degree of the corresponding critical pair less than or equal to d, the S-polynomial $S(f, g)$ reduces to zero by G.

If the Homogeneous Buchberger Algorithm is interrupted after some degree d is finished, the elements of G form a truncated Gröbner Basis $G_{\leq d}$. The vital property of such a basis is formulated in the following proposition.

PROPOSITION 4.10. *Let G be a set of non-zero homogeneous polynomials that generate the ideal $I \subset P$, and let $d \in \mathbb{Z}_{\geq 0}$. Then the following conditions are equivalent:*

(1) *$G_{\leq d}$ is a truncated Gröbner Basis of I;*
(2) *Every homogenous element $f \in I_{\leq d}$ satisfies*

$$\exists g \in G_{\leq d} : LT(g) | LT(f).$$

PROOF. See [**KR04**, p.87]. □

The strategy of selecting critical pairs in increasing degree, adopted in the description of the Homogenous Buchberger Algorithm, is called the *normal selection strategy*. This strategy helps to reduce the computations during the reduction of S-polynomials, due to the following theorem.

THEOREM 4.11. *Let I be an ideal in $k[x_1, \ldots, x_n]$, $F := \{f_1, \ldots, f_m\}$ a homogeneous basis of I. Let a Gröbner Basis of I be computed from F by an algorithm that treats critical pairs by increasing degree (e.g. the normal selection strategy). If during the algorithm an S-polynomial either has degree less than or is reduced to a degree less than the corresponding critical pair, then it eventually reduces to 0. As a consequence:*

(1) *whenever during a reduction-loop the degree is decreased, the reduction can be interrupted since it will eventually end with 0;*
(2) *new basis elements appear in increasing degrees.*

PROOF. See [**Tra97**, p.359]. □

The Homogeneous Buchberger Algorithm returns a correct Gröbner Basis for sets of polynomials that are not homogeneous either. Hence, it can be used as a framework for other improvements on the Buchberger Algorithm, like F4. Faugère advices to implement his F4 with the normal selection strategy. The following section introduces the algorithm with this strategy taken into account.

4.3. Reduction by linear algebra, F4

The algorithm F5 introduced in [**Fau02**] is based on an efficient selection criterion for critical pairs, combined with a reduction strategy based on linear algebra. This reduction strategy was introduced by Faugère in an earlier algorithm called F4 [**Fau99**]. In this section, the main ideas of F4 are explained in detail.

Normally, a reduction strategy consists of two parts. First of all, one needs an order in which to select the critical pairs to reduce. In the Buchberger Algorithm, this choice determines which reduced S-polynomials are added to the intermediate basis, which affects the new reductions during the course of the algorithm. Secondly, as in the Division Algorithm, one has a choice to which order of elements in the intermediate basis the new-found S-polynomials are reduced.

Instead of creating reduced S-polynomials one-by-one, the main idea of F4 is to reduce a set of critical pairs simultaneously, with respect to a preprocessed subset of the intermediate basis, called reductors. The theory behind the creation of reductors originally stems from the Normal Form subroutine in the FGLM algorithm [**FGLM93**].

For convenience, a correspondence between sets of polynomials and matrices is defined.

DEFINITION 4.12. Let $F = (f_1, \ldots, f_m)$ be a tuple of polynomials in $P = k[x_1, \ldots, x_n]$ and let $T(F)$ and $T_\sigma(F)$ denote the set of pairwise distinct monomials in F and its ordered equivalent, with respect to ordering σ, respectively. The number of distinct monomials in F, $|T(F)|$, is denoted by s.

Let a general polynomial $f \in P$ be written as

$$f = \sum_{i=1}^{s} c_i x^{\alpha_i}, \text{ with } \alpha_i \in \mathbb{Z}^n_{\geq 0} \text{ and } c_i \in k.$$

Define the vector representation map

$$\psi_{T_\sigma(F)} : P \to k^s$$

of f with respect to $T_\sigma(F)$ as follows

$$\psi_{T_\sigma(F)}(f) = (c_1, \ldots, c_s)$$

and the *matrix representation* of a tuple of polynomials F

$$\Psi_{T_\sigma(F)} : P^m \to Mat_{m,s}(k), \quad (f_1, \ldots, f_m) \mapsto \begin{pmatrix} \psi_{T_\sigma(F)}(f_1) \\ \vdots \\ \psi_{T_\sigma(F)}(f_m) \end{pmatrix}.$$

If it is clear from the context with respect to which ordering and support the polynomials are represented, the subscripts are omitted.

The following definition of the matrix $\widetilde{F}^+$ is the F4-equivalent of reduced S-polynomials in the Buchberger Algorithm.

DEFINITION 4.13. Let F be a subset of polynomial ring P. Define

- The set of polynomials corresponding to the row echelon form of $\Psi(F)$ is denoted by $\widetilde{F}$;
- Let $\widetilde{F}^+$ denote $\left\{ g \in \widetilde{F} : LT(g) \notin LT(F) \right\}$.

The elements of $\widetilde{F}^+$ are joined with a subset, H, of the original F, such that

$$LT(H) = LT(F) \text{ and } |H| = |LT(F)|$$

hold. As a consequence of the following theorem, the ideal $\langle F \rangle$ is spanned by $H \cup \widetilde{F}^+$.

THEOREM 4.14. [**Fau99**, p. 65] *Let k be a field, F a finite set of elements $P = k[x_1, \ldots, x_n]$ and let s denote the cardinality of the support $T_\sigma(F)$.*

For any subset $H \subseteq F$ such that $|H| = |LT(F)|$ and $LT(H) = LT(F)$, the vectors

$$\psi(g) \in k^s, \text{ for } g \in \widetilde{F}^+ \cup H,$$

form a triangular basis of the subspace of the vector space k^s spanned by the vectors $\psi(f)$ for $f \in F$.

PROOF. Write $G = \widetilde{F}^+ \cup H$. All elements g of G have distinct leading terms and are linear combinations of elements of F. Hence, the set $\{\psi(g) : g \in G\}$ is linearly independent and is included in the subspace spanned by the vectors corresponding to elements of F.

Furthermore, let r denote the rank of the subspace spanned by $\psi(f)$ for $f \in F$. Also,

$$LT(G) = LT(\widetilde{F}^+) \cup LT(H) = LT(\widetilde{F})$$

holds, which implies $|LT(G)| = |LT(\widetilde{F})| = r$ and the theorem follows. □

The main idea of F4 is rooted in the reduction of S-polynomials. Instead of computing the reduction of every S-polynomial individually, it creates a selection of critical pairs $b = (b_1, b_2)$, for b_1, b_2 in the intermediate basis G' and passes the two polynomials

$$\frac{LCM(LT(b_1), LT(b_2))}{LT(b_1)} b_1, \quad \frac{LCM(LT(b_1), LT(b_2))}{LT(b_2)} b_2$$

to the reduction function. Let us assume the normal selection strategy is adopted. The critical pairs corresponding to degree d are

$$B_d = \{(b_1, b_2) : b_1, b_2 \in G' \text{ where } totaldeg(LCM(LT(b_1), LT(b_2))) = d, b_1 \neq b_2\}.$$

Hence, the following set is passed to the simultaneous reduction routine of F4,

$$L_d = \bigcup_{(b_1,b_2)\in B_d} \left\{ \frac{LCM(LT(b_1), LT(b_2))}{LT(b_1)} b_1, \frac{LCM(LT(b_1), LT(b_2))}{LT(b_2)} b_2 \right\}. \tag{4.3}$$

The reduction in F4 uses preprocessed reductors of an intermediate basis G'. The addition of reductors is done by a routine called Symbolic Preprocessing.

DEFINITION 4.15. During the execution of an algorithm to compute Gröbner Bases, a *reductor* r of the set F is a polynomial satisfying

$$LT(r) \in T(F) \setminus LT(F).$$

DEFINITION 4.16. [*Symbolic Preprocessing*] The following algorithm appends reductors to the set F with respect to an intermediate basis G'.

- Input: A set $F \subset P$ and an intermediate basis G'.
- Output: The set $F \cup R$ for a set of reductors R.

(1) $D := LT(F)$, $R := \{\}$
(2) $WHILE\ T(F \cup R) \neq D\ DO$
(3) Select $m \in T(F \cup R) \setminus D$
(4) Append m to D
(5) $IF\ LT(m)$ is divisible by an element $g \in LT(G')$
(6) $m' := m/LT(g)$
(7) Append gm' to R

Now we are ready to formulate the function F4 that simultaneously reduces polynomials corresponding to several critical pairs.

DEFINITION 4.17 (ReductionF4). The subroutine ReductionF4 returns $\widetilde{F}^+$, where F is the output of the subroutine Symbolic Preprocessing for a set L_d, as defined in Equation (4.3), with respect to intermediate basis G'.

S-polynomials that do not reduce to zero in the Buchberger Algorithm, extend the ideal spanned by the leading terms of the intermediate basis. This way, an ascending chain of leading term ideals is obtained. Similarly, the leading terms of the elements of $\widetilde{F}^+$ contribute to the ideal spanned by the leading terms of the intermediate basis. This is formalized in the following lemma.

LEMMA 4.18. *Let $\widetilde{F}^+$ denote the output of ReductionF4 applied to L_d with respect to G'. For all $f \in \widetilde{F}^+$, $LT(f)$ is not an element of $\langle LT(G')\rangle$.*

PROOF. Let f be in $\widetilde{F}^+$ and the output of Symbolic Preprocessing of L_d with respect to G' be denoted by F. Suppose, to achieve a contradiction, that $LT(f) \in \langle LT(G')\rangle$. This assumption and $LT(f) \in T(\widetilde{F}^+) \subset T(F)$ implies that Symbolic Preprocessing must have added a reductor $\frac{LT(f)}{LT(g)} g$ to F for a suitable $g \in G'$. This would mean $LT(f) \in LT(F)$, a contradiction according to the definition of $\widetilde{F}^+$. Hence, $LT(f)$ is not an element of $\langle LT(G')\rangle$. □

The next lemma assures that the elements that are added to the intermediate basis, are members of the ideal $\langle G'\rangle$.

LEMMA 4.19. *Let $\widetilde{F}^+$ be as in Lemma 4.18. Then $\widetilde{F}^+ \subset \langle G'\rangle$.*

PROOF. Every $f \in \widetilde{F}^+$ is a linear combination of elements of L_d and reductors R, which are both subsets of $\langle G' \rangle$. □

The following lemma states that all S-polynomials in the set of possible k-linear combinations of L_d reduce to zero by a subset of $\widetilde{F}^+ \cup G'$. This is used to prove the correctness of the algorithm by the criterion stated in Theorem 2.46.

LEMMA 4.20. *Let $\widetilde{F}^+$ be as in Lemma 4.18. For all k-linear combinations, f, of elements of L_d, the Normal Form equals zero with respect to $\widetilde{F}^+ \cup G'$.*

PROOF. Let f be a linear combination of elements of L_d. Suppose F is the output of the Symbolic Preprocessing of L_d with respect to G'. By construction, L_d is a subset of F and, therefore due to Theorem 4.14, these elements are a linear combination of the triangular basis $\widetilde{F}^+ \cup H$ for a suitable subset $H \subset F$. Elements of H are either elements of L_d or of the form $x^\alpha g$, for $g \in G'$ and $\alpha \in \mathbb{Z}^n_{\geq}$, and f can thus be written as

$$f = \sum_i a_i f_i + \sum_j a_j x^{\alpha_j} g_j,$$

for $f_i \in \widetilde{F}^+$ and $g_j \in G'$, a_i, $a_j \in k$ and $\alpha_j \in \mathbb{Z}^n_{\geq 0}$. Thus the Division Algorithm gives a remainder equal to 0 for a suitable tuple of elements in $\widetilde{F}^+ \cup G'$, hence there exists a reduction chain to 0. □

Now we are ready to describe F4 and prove its correctness. The selection strategy in Step 5 of the algorithm is kept blank. One can choose to select all critical pairs available at that time, or for example the normal selection strategy from Section 4.2.

THEOREM 4.21 (*F4*). *The algorithm F4 computes a Gröbner Basis G of an ideal spanned by F, such that $F \subseteq G$, in a finite number of steps.*

- *Input: $F = \{f_1, \ldots, f_m\} \subset P$*
- *Output: A Gröbner Basis G for $\langle F \rangle$, satisfying $F \subseteq G$.*

(1) $G' := F$, $\widetilde{F_0}^+ := F$, $d := 0$
(2) $B := \{(b_1, b_2) : b_1, b_2 \in G' \text{ with } b_1 \neq b_2\}$
(3) $WHILE\ B \neq \emptyset\ DO$
(4) $\quad d := d + 1$
(5) $\quad B_d := Select(B)$
(6) $\quad B := B \setminus B_d$
(7) $\quad L_d = \bigcup_{(b_1,b_2) \in B_d} \left\{ \frac{LCM(b_1,b_2)}{LT(b_1)} b_1, \frac{LCM(b_1,b_2)}{LT(b_2)} b_2 \right\}$
(8) $\quad \widetilde{F}_d^+ := ReductionF4(L_d, G')$
(9) $\quad FOR\ f \in \widetilde{F}_d^+$
(10) $\quad\quad B := B \cup \{(f, g) : g \in G'\}$
(11) $\quad\quad G' := G' \cup \{f\}$
(12) $G := G'$

PROOF. Correctness and termination are proven by the following observations.

(1) Lemma 4.19 implies that during stage $d = d'$ of the algorithm, the intermediate basis satisfies,

$$G' = \cup_{d=1}^{d'} \widetilde{F}_d^+ \subset \langle F \rangle;$$

(2) Lemma 4.18 shows that

$$\langle LT(\widetilde{F}_1^+)\rangle \subset \langle LT(\widetilde{F}_1^+ \cup \widetilde{F}_2^+)\rangle \subset \ldots,$$

is an ascending chain of monomial ideals. The Ascending Chain Condition, Theorem 2.32, states that it should stabilize eventually. This implies that the while-loop should terminate since we run out of critical pairs at some point;

(3) Suppose the algorithm terminates at $d = d_{F4}$. Since every pair (g_1, g_2) for $g_1,\ g_2 \in G = \cup_{d=1}^{d_{F4}} \widetilde{F}_d^+$ is considered, $S(g_1, g_2)$ is in the linear span of elements of G. Lemma 4.20 states that its Normal Form equals zero. Hence, the Gröbner Basis criterion of Theorem 2.46 is satisfied.

□

Faugère suggests to implement F4 with a strong criterion to reject useless critical pairs, for example the one by Gebauer and Möller [**GM88**]. This criterion can be applied to any extension of the Buchberger Algorithm, which iteratively selects critical pairs and computes the corresponding reduced S-polynomials. It is also recommended to store the matrices in compressed form and use specially tailored techniques for row reduction of sparse matrices. The so-called Gebauer and Möller Installation will be explained in more detail in Section 4.4.

Furthermore, in [**Fau99**] an improvement of the Symbolic Preprocessing routine is described, based on the theory in [**FGLM93**]. Our implementation of F4 is based on the Homogeneous Buchberger Algorithm and includes the normal selection strategy, the Gebauer and Möller Installation and improved Symbolic Preprocessing.

Our implementation of F4 is particularly useful since it provides the user with a tool to determine various properties of the Gröbner Basis computation. These include the number of critical pairs considered, their largest degree, the number of rows of the largest reduction matrix and its density.

Remark 4.22. One might also consider a parallel implementation of the F4 algorithm. Attardi and Traverso describe parallelizations of the Buchberger Algorithm in [**AT96**]. These algorithms distribute the critical pairs to other processes that compute and reduce the S-polynomials. Similarly, one could distribute sets of critical pairs. During the normal selection strategy for example, the sets B_d or L_d from equation (4.3) could be handled by different processes for different d. Magma 2.11 allows the creation of TCP sockets on UNIX systems to establish communication channels between several instances of Magma on a network

During the execution of F4, no real S-polynomials $S(g_i, g_j)$ are created but their components,

$$\frac{x^{\gamma_{ij}}}{LT(g_i)} g_i \text{ and } \frac{x^{\gamma_{ij}}}{LT(g_j)} g_j \text{ with } x^{\gamma_{ij}} = LCM(LT(g_i), LT(g_j)),$$

are stored in the reduction matrix. The largest of the total degrees of these terms corresponds to the total degree of a fictitious S-polynomial. This does not necessarily have to correspond with the largest degree of a critical pair during the course of the algorithm, since an S-polynomial might reduce to zero while its components contribute to the reduction step. Conversely, the total degree of an intermediate polynomial during the lexicographic Gröbner Basis computation might exceed the

largest degree of a critical pair, because in this case an S-polynomial might reduce to an element in the ideal of larger total degree. As is shown in Chapter 5, the largest total degree of an intermediate polynomial is an upper bound for the smallest D for which the XL algorithm is able to find a Gröbner Basis.

DEFINITION 4.23. The largest total degree of an intermediate polynomial during the execution of F4 is called d_{F4}.

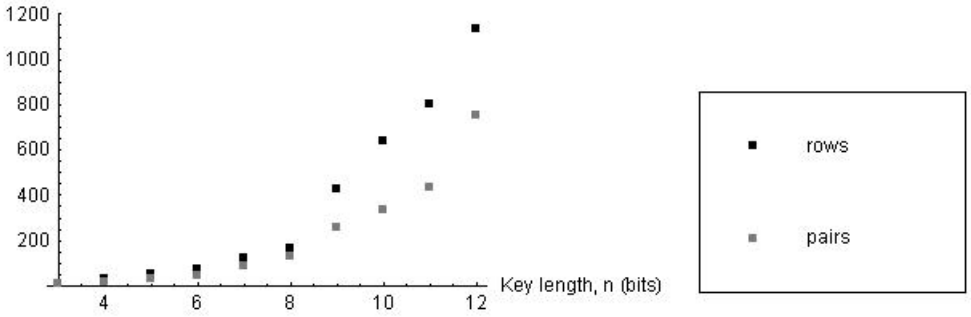

FIGURE 4.1. Average number of critical pairs considered (black) and rows in the reduction matrix (grey) for HFE plaintext attacks with F4.

For F4-runs on different HFE equations having *grevlex* order, the average number of rows in the largest reduction matrix and critical pairs considered are depicted in Figure 4.1. The key lengths varied from 3 to 12 bits. The average was taken over 100 samples for every key length. We also computed the exponential least-square fits for the number of rows and pairs, being respectively

$$3.44 * 1.63^n \text{ and } 2.01 * 1.64^n.$$

Notice how the base numbers are very close to each other. Intuitively one would expect this, since the largest reduction matrix considers a major part of the critical pairs.

The next section introduces the well-known and commonly adopted Gebauer and Möller Installation and the criterion of the algorithm F5 for discarding useless critical pairs. The latter is claimed to be optimal [**Fau02**], but does not seem to be widespread among the implementers of Gröbner Basis algorithms.

4.4. Gebauer and Möller Installation and F5

The Gebauer and Möller Installation is introduced in [**GM88**] and is an improvement of the two criteria proposed by Buchberger in [**Buc85**]. Recently, Caboara, Kreuzer and Robbiano [**CKR04**] showed how to compute a minimal set of critical pairs and concluded that the Gebauer and Möller criteria were almost optimal. This section introduces the Gebauer and Möller Installation and the criterion for avoiding useless critical pairs of F5.

The starting point of the Gebauer and Möller Installation is the observation that Gröbner Bases can be characterized using a basis of its module of syzygies. To define syzygies, we need the notion of modules. Modules are to rings what vector spaces are to fields: elements of a given module over a ring can be added to one another and multiplied by elements of the ring.

DEFINITION 4.24. For a ring R, an R-module M is a commutative group $(M, +)$ with an operation $\cdot : R \times M \to M$ called *scalar multiplication* such that $1 \cdot m = m$ for all $m \in M$ and such that the associative and distributive laws are satisfied. A commutative subgroup $N \subseteq M$ is called an R-submodule if we have $R \cdot N \subseteq N$.

In Section 2.2 we briefly touched upon the subject of syzygies and their relation to S-polynomials. This correspondence is further explained here.

DEFINITION 4.25. Let $F = (f_1, \ldots, f_m) \in P^m$. A *syzygy* on the leading terms $LT(f_1), \ldots, LT(f_m)$ is an m-tuple of polynomials $S = (h_1, \ldots, h_m) \in P^m$ such that

$$\sum_{i=1}^{m} h_i LT(f_i) = 0.$$

We let $S(F)$ be the subset of P^m consisting of all syzygies on the leading terms of F.

Let the canonical basis vector $\mathbf{e}_i = (0, \ldots, 0, 1, 0, \ldots, 0) \in P^m$, where the 1 is in the i-th place. Then a syzygy $S \in S(F)$ can be written as $S = \sum_{i=1}^{m} h_i \mathbf{e}_i$ with $h_i \in P$. Using this notation, a syzygy coming from S-polynomials is denoted as follows.

DEFINITION 4.26. Let F be the tuple $(f_1, \ldots, f_m) \in P^m$. The syzygy S_{ij} corresponding to the S-polynomial $S(f_i, f_j)$ is defined as,

$$S_{ij} = \frac{x^\gamma}{LT(f_i)}\mathbf{e}_i - \frac{x^\gamma}{LT(f_j)}\mathbf{e}_j, \tag{4.4}$$

where x^γ refers to the usual $LCM(LM(f_i), LM(f_j))$. These syzygies are referred to as *critical syzygies*, see for example [**CKR04**].

A nice fact about $S(F)$ is that it has a finite basis. There is a finite collection of syzygies such that every other syzygy is a linear combination with polynomial coefficients of the basis of syzygies. To show this, we define the notion of a *homogeneous syzygy*.

DEFINITION 4.27. An element $S \in S(F)$ is homogeneous of multidegree α, where $\alpha \in \mathbb{Z}_{\geq 0}$, provided that

$$S = (c_1 x^{\alpha(1)}, \ldots, c_m x^{\alpha(m)}),$$

where c_i is an element of the field k and $\alpha(i) + multideg(f_i) = \alpha$ whenever $c_i \neq 0$.

The syzygies of Definition 4.26 are homogeneous of multidegree γ according to the previous definition. The following proposition states that these S_{ij} form a basis of all syzygies on the leading terms $S(F) \subset P^m$.

PROPOSITION 4.28. *Given $F = (f_1, \ldots, f_m)$, every syzygy $S \in S(F)$ can be written as*

$$S = \sum_{i<j} u_{ij} S_{ij},$$

with $u_{ij} \in P$. Hence,

$$\{S_{ij} : 1 \leq i < j \leq m\},$$

with S_{ij} homogeneous of multidegree γ, is a homogeneous basis of $S(F)$.

PROOF. See [**CLO96**, p. 104]. □

Let us illustrate this proposition with an example.

EXAMPLE 4.29 (Basis of the module of syzygies in Magma). *The basis of the module of syzygies of the set $F = \left\{x_1^2 + x_2^2, x_2^2 + 1, x_1^2 + x_1, x_2^2 + x_2\right\} \subset P$ consists of module elements S_{ij} as in Definition 4.26.*

```
> // Create the polynomial ring P and set F
> P<[x]> := PolynomialRing(GF(2), 2, "lex");
> F := [ x[1]^2 + x[2]^2, x[2]^2 + 1, x[1]^2 + x[1], x[2]^2 + x[2] ];
> LTF := [LeadingTerm(f) : f in F];
> SyzygyModule(LTF);
Module of degree 4
TOP Order
Coefficient ring:
    Polynomial ring of rank 2 over GF(2)
    Lexicographical Order
    Variables: x[1], x[2]
Basis:
(     1      0      1      0)
(     0      1      0      1)
(     0      0 x[2]^2 x[1]^2)
> // Add to F the reduced S-polynomials corresponding to the above basis elements
> Include(~F, NormalForm(F[1]+ F[3], F));
> Include(~F, NormalForm(F[2]+ F[4], F));
> Include(~F, NormalForm(x[2]^2*F[3]+ x[1]^2*F[4], F));
> F;
[
    x[1]^2 + x[2]^2,
    x[2]^2 + 1,
    x[1]^2 + x[1],
    x[2]^2 + x[2],
    x[1] + 1,
    x[2] + 1,
    0
]
> // Check if the new F is a Groebner Basis, otherwise repeat the above steps
> IsGroebner(F);
true
```

The basis in Proposition 4.28 is called the *Taylor basis*. Methods to obtain a reduced basis from the Taylor basis give an advanced Gröbner Basis test, since less S-polynomials have to be considered. This follows from the next theorem, which is a new algorithmic criterion for Gröbner Bases.

THEOREM 4.30. *Let $G = (g_1, \ldots, g_m) \in P^m$. A basis $\{g_1, \ldots, g_m\}$ for an ideal I is a Gröbner Basis if and only if for every element $S = (h_1, \ldots, h_m)$ in a homogeneous basis for the syzygies $S(G)$, we have*

$$S \cdot G = \sum_{i=1}^{m} h_i g_i \to_G 0.$$

PROOF. See [**CLO96**, p. 105]. □

The syzygies S_{ij} correspond exactly to the S-polynomials $S(f_i, f_j)$ that one might consider during the course of the Buchberger Algorithm, or similar algorithms to compute Gröbner Bases. To exploit Theorem 4.30, one needs to know how to make smaller homogeneous bases of $S(G)$. We will show that, starting with the Taylor basis, there is a systematic way to predict when elements can be omitted. The following proposition is due to Buchberger [**Buc85**]. The proof from [**GM88**] is included since it illustrates an important idea on how to discard critical pairs. This is done by looking at the module of syzygies of the original module of syzygies on the leading terms.

PROPOSITION 4.31. *Given $G = (g_1, \ldots, g_m) \in P^m$, suppose that we have a subset $L \subset \{S_{ij} : 1 \leq i < j \leq m\}$ that is a basis of $S(G)$. In addition, suppose we have distinct elements g_i, g_j and $g_k \in G$ such that*

$$LT(g_k) | LCM(LT(g_i), LT(g_j)).$$

If S_{ij}, $S_{jk} \in L$, then $L \setminus \{S_{ij}\}$ is also a basis of $S(G)$. (Note: If $i > j$, we set $S_{ij} = S_{ji}$.)

PROOF. Without loss of generality, assume that $i < j < k$. Set

$$x^{\gamma_{ij}} = LCM(LM(g_i), LM(g_j))$$

and let $x^{\gamma_{ik}}$ and $x^{\gamma_{jk}}$ be defined similarly. Furthermore, set

$$x^{\gamma_{ijk}} = LCM(LM(g_i), LM(g_j), LM(g_k)).$$

Our hypothesis implies that $x^{\gamma_{ik}}$ and $x^{\gamma_{jk}}$ both divide $x^{\gamma_{ij}}$. We shall prove that

$$S_{ij} = \frac{x^{\gamma_{ij}}}{x^{\gamma_{ik}}} S_{ik} - \frac{x^{\gamma_{ij}}}{x^{\gamma_{jk}}} S_{jk}.$$

There are $m(m-1)/2$ syzygies S_{ij}. If there is some dependency in these generators of $S(G)$, then higher order syzygies in the module $P^{m(m-1)/2}$ exist that cancel them. To create a canonical basis for this module of syzygies, the $m(m-1)/2$ syzygies are ordered by $<_1$, which is defined as follows,

$$S_{ab} <_1 S_{cd} \Leftrightarrow x^{\gamma_{ab}} < x^{\gamma_{cd}} \text{ or } (x^{\gamma_{ab}} = x^{\gamma_{cd}}, b \leq d, \text{ where } b = d \Rightarrow a < c).$$

Using this order, we no longer denote the canonical i-th unit vector in $P^{m(m-1)/2}$ by $\mathbf{e}_i$ but by $\mathbf{e}_{ab}$ if S_{ab} is the i-th syzygy in this order.

The module of syzygies

$$S^{(2)}(G) = \left\{ \sum_{i,j=1, i<j}^{m} h_{ij} \mathbf{e}_{ij} \in P^{m(m-1)/2} : \sum_{i,j=1, i<j}^{m} h_{ij} S_{ij} = 0 \right\}$$

has the Taylor basis $L^{(2)} = \{S_{ijk} : 1 \leq i < j < k \leq m\}$ with

$$S_{ijk} = \frac{x^{\gamma_{ijk}}}{x^{\gamma_{ij}}} \mathbf{e}_{ij} - \frac{x^{\gamma_{ijk}}}{x^{\gamma_{ik}}} \mathbf{e}_{ik} + \frac{x^{\gamma_{ijk}}}{x^{\gamma_{jk}}} \mathbf{e}_{jk}.$$

The elements S_{ijk} are homogeneous of multidegree γ_{ijk} according to Definition 4.27.

Our hypothesis implies that S_{ijk} and S_{ij} are homogeneous of the same multidegree $\gamma_{ij} = \gamma_{ijk}$. In that case, S_{ij} can be expressed in terms of the syzygies S_{ik} and S_{jk}, because S_{ijk} is an element of the second module of syzygies $S^{(2)}(G)$. Hence,

$$S_{ij} = \frac{x^{\gamma_{ijk}}}{x^{\gamma_{ik}}} S_{ik} + \frac{x^{\gamma_{ijk}}}{x^{\gamma_{jk}}} S_{jk}.$$

This allows us to remove S_{ij} from L and the set $L \setminus \{S_{ij}\}$ still generates $S(G)$ because in every basis representation of an $S \in S(G)$, S_{ij} can be replaced by S_{ik} and S_{jk}. □

The Gebauer and Möller Installation is best described as a modification of the Buchberger Algorithm. This is presented below. An advantage of this installation is that any Gröbner Basis algorithm that selects critical pairs in a similar way as the Buchberger Algorithm, can adopt this selection criterion. The reader is referred to [**GM88**] for the details.

THEOREM 4.32 (*Gebauer and Möller Installation*). *Let $I = \langle f_1, \ldots, f_m \rangle \neq \{0\}$ be a polynomial ideal. Then a Gröbner Basis for I can be constructed in a finite number of steps by the following algorithm.*

- *Input: $F = \{f_1, \ldots, f_m\}$*
- *Output: A Gröbner Basis G for $\langle \{f_1, \ldots, f_m\} \rangle$.*

(1) $G := \{f_1\}$
(2) $D := \{\}$
(3) $FOR\ t := 2\ to\ m$
(4) $\quad UpdatePairs(D, t)$
(5) $\quad G := G \cup \{f_t\}$
(6) $r := m$
(7) $WHILE\ there\ exists\ (i,j) \in D\ REPEAT$
(8) $\quad h := \overline{S(f_i, f_j)}^G$
(9) $\quad IF\ h \neq 0\ THEN$
(10) $\quad\quad f_{r+1} := h$
(11) $\quad\quad D := UpdatePairs(D, r+1)$
(12) $\quad\quad G := G \cup \{f_{r+1}\}$
(13) $\quad\quad r := r + 1$
(14) $\quad D := D \setminus \{(i,j)\}$
(15) $RETURN\ G$

The procedure UpdatePairs is defined as follows.

- *Input: A set of pairs D and a positive integer t*
- *Output: An updated set of critical pairs D, such that $\{S_{ij} : (i,j) \in D\}$ together with some S_{ij} with $1 \leq i < j \leq t$ and $LT(f_i)LT(f_j) = x^{\gamma_{ij}}$, form a basis of the module of syzygies*

$$\left\{ (g_1, \ldots, g_t) \in P^t : \sum_{i=1}^{t} g_i LT(f_i) = 0 \right\}.$$

(1) *Cancel in D all pairs (i,j) which satisfy*

$$x^{\gamma_{ij}} = x^{\gamma_{ijt}} \text{ and } x^{\gamma_{it}} \neq x^{\gamma_{ij}} \neq x^{\gamma_{jt}};$$

(2) *Denote this set of remaining pairs by D';*
(3) *Set $D1 := \{(i,t) : 1 \leq i < t\}$;*
(4) *Cancel in $D1$ each pair (i,t) for which a $(j,t) \in D1$ exists, such that*

$$x^{\gamma_{jt}} | x^{\gamma_{it}} \text{ and } x^{\gamma_{jt}} \neq x^{\gamma_{it}};$$

(5) *In each non-void subset $\{(j,t) : x^{\gamma_{jt}} = \tau\} \subset D1'$ with monomial $\tau \in T(P)$, fix an element (i,t) satisfying*

$$LT(f_i)LT(f_j) = x^{\gamma_{it}}$$

or if no such (i,t) exists, fix an arbitrary (i,t). Cancel the other elements of $\{(j,t) : x^{\gamma_{jt}} = \tau\}$ in $D1'$. Finally delete in $D1'$ all (i,t) with

$$LT(f_i)LT(f_t) = x^{\gamma_{it}}$$

and denote again by $D1'$ this finally obtained subset of $D1'$;

(6) *RETURN* $D := D1' \cup D'$

PROOF. See [**GM88**, p. 283]. □

Suppose we want to compute a Gröbner Basis of the set $\{f_1, \ldots, f_m\}$. Similar to the Buchberger Algorithm, if we have a critical pair that does not reduce to zero, the reduced S-polynomial is called f_{m+1} and is appended to our original set. This process is repeated and S-polynomials are indexed by the order in which they are added to the original basis. Now, instead of trying all combinations of pairs (i,j) for $1 \leq i < j$, the Gebauer and Möller Installation runs through a selection of all possible combinations. This is illustrated in the following example.

EXAMPLE 4.33. *The implementation of the Homogeneous Buchberger Algorithm with the Gebauer and Möller Installation is able to return the list of selected critical pairs.*

```
> load "HomBuchGM.m";
> P<x,y,z> := PolynomialRing(GF(23), 3, "lex");
> F := [
> 6*x^2+12*x*y+4*y^2+14*x*z+9*y*z+7*z^2,
> 3*x^2+7*x*y+22*x*z+11*y*z+22*z^2+8*y^2,
> x^2+18*x*y+19*y^2+8*x*z+5*y*z+7*z^2];
> G, pairs := HomBuchGM(F);
> pairs;
[
    [ 1, 3 ],    [ 1, 4 ],    [ 1, 5 ],    [ 4, 5 ],    [ 1, 2 ],    [ 5, 7 ],
    [ 7, 8 ],    [ 4, 6 ],    [ 4, 8 ],    [ 8, 9 ],    [ 5, 9 ],    [ 6, 7 ]
]
```

In the case of a homogeneous set of generators of an ideal, Caboara, Kreuzer and Robbiano [**CKR04**] succeeded in describing an efficient way to compute a minimal set of generators of the module generated by the syzygies corresponding to the critical pairs. They noticed that the reduced basis of the module of syzygies of the leading terms was actually a reduced Gröbner Basis. From this basis, they computed a minimal set of generators.

Table 4.1 illustrates the performance of the Gebauer and Möller Installation for well-known examples, some of which were made homogeneous first. This table is taken from [**CKR04**, p.24] and compares the number of redundant critical pairs considered by the Gebauer and Möller Installation to the minimal set of generators of $S(G)$ called L^*, where G denotes the reduced Gröbner Basis. The binomium $\binom{\#G}{2}$ equals the total number of critical pairs. For benchmark $Kin1$ the Gebauer and Möller Installation considers the most redundant critical pairs, but this number is still relatively small compared to $\#L^*$.

We continue this section with the explanation of the F5 selection criterion. The author claims that if the input is a regular sequence, the algorithm generates no useless critical pairs.

DEFINITION 4.34 ([**KR00**], page 171). Let P be a polynomial ring and $I = \langle f_1, \ldots, f_m \rangle$ an ideal in P.

TABLE 4.1. Number of redundant pairs considered by the Gebauer and Möller Installation, GMI, compared to the minimal system of generators of the syzygies of the leading terms, L^*, for well-known benchmarks (see [**CKR04**]).

benchmark	$\#G$	$\binom{\#G}{2}$	GMI	$\#L^*$
$Alex3$	211	22,155	3	627
$Cyclic7$	443	97,903	0	681
$Kin1$	306	46,665	102	3,329
$Mora9$	4,131	8,530,515	7	44,458
$Wang$	317	50,086	8	1,389

- An ideal I is called *proper* if it is unequal to P.
- An element $f \in P$ is called a *non-zerodivisor* if $fg = 0$ implies $g = 0$.
- A sequence of elements $f_1, \ldots, f_m \in R$ is called a *regular sequence* if the ideal I is proper and the image of f_i is a non-zerodivisor in $P/\langle f_1, \ldots, f_{i-1}\rangle$ for $i = 1, \ldots, m$.

Let us illustrate the definition of a regular sequence with an example.

EXAMPLE 4.35. *Let P be the polynomial ring $GF(2)[x_1, \ldots, x_4]$ and define $f_1 = x_2x_4 - x_3^2$, $f_2 = x_1x_4 - x_2x_3$ and $f_3 = x_1x_3 - x_2^2$.*

The ideal is proper. Any two f_i, f_j, i, $j \in \{1, 2, 3\}$ and $i \neq j$, are coprime, so $f_ig = 0$ in $P/\langle f_j\rangle$ for $g \in P$ implies $g = 0$. Therefore, any two polynomials f_i, f_j form a regular sequence in P. However, $x_3 * f_3 = 0$ and $x_4 * f_3 = 0$ in $P/\langle f_1, f_2\rangle$ with x_3 and x_4 not in the ideal $\langle f_1, f_2\rangle$, hence f_1, f_2, f_3 is not a regular sequence in P.

To explain the F5 criterion we need to extend the original ordering $<$ to a new ordering $<_{P^m}$ for module P^m over polynomial ring P. As before, $\mathbf{e}_i$ refers to the canonical i-th unit vector $(0, \ldots, 0, 1, 0, \ldots, 0)$ in P^m.

DEFINITION 4.36. For two module elements $H = \sum_{i=j}^{m} h_i\mathbf{e}_i$ and $H' = \sum_{i=j'}^{m} h'_i\mathbf{e}_i$ in P^m, with non-zero h_j, $h'_{j'}$ in P, the module term ordering is defined as follows,

$$H <_{P^m} H' \Leftrightarrow j > j' \text{ or } (j = j' \text{ and } LT(h_j) < LT(h'_{j'})).$$

The description of the F5 criterion comes with many new definitions. Therefore, the final part of this chapter should not be considered as a common way to refer to certain objects.

With an ordering on the module elements, one can speak of a leading module term.

DEFINITION 4.37. The *leading module term* of an element $H = \sum_{i=j}^{m} h_i\mathbf{e}_i$, with non-zero $h_j \in P$, is defined as

$$LMT(H) = LT(h_j)\mathbf{e}_j.$$

The notion of a signature of a polynomial is essential and specific to the algorithm F5. Furthermore, Faugère introduces the index of a polynomial. Both are introduced in the following definition.

DEFINITION 4.38. During the computation of a Gröbner Basis of a tuple $F = (f_1, \ldots, f_m)$ by means of the algorithm F5, the *signature* of a polynomial f, $\mathcal{S}(f)$,

equals the leading module term of the smallest module element $H = \sum_{i=1}^{m} h_i \mathbf{e}_i$ that satisfies

$$LT(H \cdot F) = LT(\sum_{i=1}^{m} h_i f_i) = LT(f). \tag{4.5}$$

Hence, $\mathcal{S}(f)$ has the form $t\mathbf{e}_j$, for term $t \in P$ and integer $j \in \{1, \ldots, m\}$.

The *index* of the polynomial f is the index of the canonical unit vector in the signature, thus if $\mathcal{S}(f) = t\mathbf{e}_j$, then $index(f) = j$.

The *F5 criterion* is reformulated in the following theorem. The third criterion in this theorem is a relaxation of Theorem 2.41. To allow this relaxation the second criterion is added. If this second criterion holds for an element g in the Gröbner Basis G, then g is called *admissible* in [**Fau02**].

THEOREM 4.39. *Let $F = \{f_1, \ldots, f_m\}$ and $G = \{g_1, \ldots, g_{m_G}\} \subset P$ span the ideal I. Define $x^{\gamma_{ij}} = LCM(LM(g_i), LM(g_j))$, for i, j in $\{1, \ldots, m_G\}$.*

The set G is a Gröbner Basis if the following criteria hold:

1. *$F \subset G$;*
2. *For every $g \in G$, there exists a module element $H = \sum_{i=1}^{m} h_i \mathbf{e}_i \in P^m$ with*

$$H \cdot F = \sum_{i=1}^{m} h_i f_i = g,$$

   such that $LMT(H)$ equals $\mathcal{S}(g)$;
3. *The S-polynomial $S(g_i, g_j)$ is 0 or has a t-representation $\sum_{l=1}^{m} b_l f_l$, with*
   (a) *$t < x^{\gamma_{ij}}$,*
   (b) *$\mathcal{S}(t) \leq_{P^m} \mathcal{S}(\frac{x^{\gamma_{ij}}}{LT(g_i)} g_i)$ and $\mathcal{S}(t) \leq_{P^m} \mathcal{S}(\frac{x^{\gamma_{ij}}}{LT(g_j)} g_j)$,*
   (c) *$\mathcal{S}(b_l f_l) \leq_{P^m} \mathcal{S}(S(g_i, g_j))$, for $1 \leq l \leq m$,*

   for all pairs (i, j) satisfying:
   (a) *$\mathcal{S}(g_j) <_{P^m} \mathcal{S}(g_i)$,*
   (b) *if $\mathcal{S}(\frac{x^{\gamma_{ij}}}{LT(g_i)} g_i) = t_i \mathbf{e}_{i'}$ and $\mathcal{S}(\frac{x^{\gamma_{ij}}}{LT(g_j)} g_j) = t_j \mathbf{e}_{j'}$, then t_i and t_j are not divisible by elements of respectively*

$$\{LT(f) : f \in \langle f_{i'+1}, \ldots, f_m \rangle\} \text{ and } \{LT(f) : f \in \langle f_{j'+1}, \ldots, f_m \rangle\}.$$

PROOF. The proof of the original criterion is included in [**Fau02**, p.78]. In spite of some mistakes, Faugère's colleague Ars assured us that the results are valid. However, we did not manage to verify them. □

The algorithm F5 takes a tuple $F = (f_1, \ldots, f_m)$ of polynomials as input and incrementally computes an intermediate Gröbner Basis G_i for every set $\{f_i, \ldots, f_m\}$ for $i = m$ down to $i = 1$. During the i-th round of the algorithm, the basis G_{i+1} and the polynomial f_i are taken as input and a Gröbner Basis G_i is calculated.

While computing the i-th intermediate basis G_i, the algorithm selects critical pairs in increasing degree. A pair (i, j) is rejected if one of the terms t_i or t_j occurring in the corresponding signatures (see Theorem 4.39, third criterion) is reducible with respect to G_{i+1}. Furthermore, the algorithm preserves the property that the new-found basis elements are admissible and keeps track of their signature.

REMARK 4.40. We have implemented an improvement of the Buchberger Algorithm including the selection criterion of F5 in the way it was described in [**Fau02**]. Although our implementation returns a correct Gröbner Basis for all binary HFE

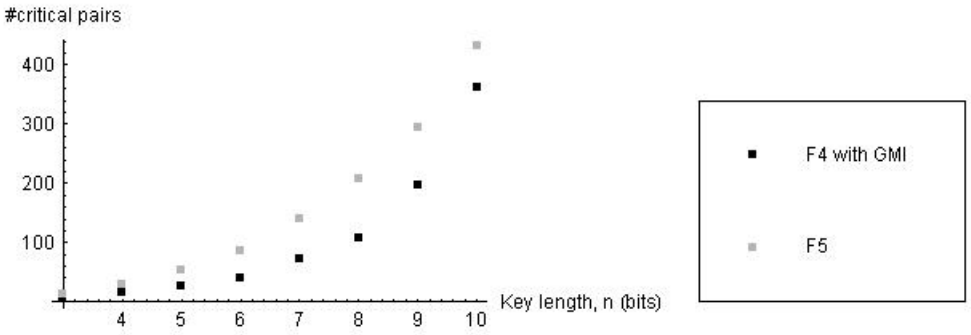

FIGURE 4.2. Comparison of the average number of critical pairs during the run of a Gröbner Basis algorithm applied to *lex* HFE equations (50 samples).

examples and almost all random systems over $GF(2)$, it is less reliable for systems over larger fields. Figure 4.2 compares the number of critical pairs considered during our implementations of F4, with the Gebauer and Möller Installation, and F5.

This chapter covered important variations on the original Buchberger Algorithm. Among these techniques, the Gebauer and Möller Installation and the reduction by means of linear algebra seem highly effective and most commonly adopted. Moreover, the theory regarding F4 turns out to be useful since it relates Gröbner Basis algorithms to XL. This relation will be the main topic of the following chapter.

CHAPTER 5

Analysis of XL and F4

In the final part of this thesis, we relate XL to Gröbner Basis algorithms and show that XL is a cumbersome way of solving a system of equations. This is mainly due to the observation that the matrices occurring in the XL computation are much larger than similar matrices in the F4 algorithm. This chapter also discusses a proposed theoretical lower bound for the minimal degree at which XL returns a solution.

Recently, two articles have appeared discussing the comparison of XL to Gröbner Basis algorithms. The first, [**SKI04**] by Sugita, Kawazoe and Imai, describes the incremental version of XL as an algorithm similar to F4 and proves that it computes a Gröbner Basis for degree d_{F4}. This topic shall be treated in more detail in Section 5.1.

The second article [**FA04**], relating XL to Gröbner Basis algorithms, is written by Faugère and Ars. The main claim of this paper is the following. Let $F \subset P$ be a set of polynomials. If the homogenized elements of F form a regular sequence and XL finds a Gröbner Basis for a certain degree, D, then $F5$ computes a Gröbner Basis for the *grevlex* ordering, without exceeding degree D during intermediate computations. Since their description of F5 is not completely unambiguous, we were not able to verify this result. However, their claims are discussed briefly in Section 5.3.

In [**Moh01**] Moh introduced a way of analyzing the complexity of XL. Based on the properties of regular series, he estimated the degree at which the number of independent polynomials generated by XL was close enough to the number of variables to find a partial solution. Diem [**Die04**] continued Moh's work and provided more theoretical background on the subject. Diem's intentions are to make this public in a short while. Section 5.2 summarizes his ideas and proposes a comment regarding the applicability.

5.1. Similarities between XL and F4

In [**SKI04**] F4 is shown to be very similar to the incremental version of XL. From the description of this version of XL, we shall see that the polynomials that occur during an F4 computation are a subset of those created by XL. The correctness and termination follow for that reason. It might happen that this XL version finds a Gröbner Basis for smaller D than the d_{F4} of F4, since the number of generated polynomials is larger. Consequently, this might lead to more reductions of leading terms. However, simulations show that in either case the matrices of F4 are smaller, which is logical due to the efficient selection criterion in place.

By three modest alterations in the description of the F4 algorithm (Theorem 4.21), we obtain an algorithm that corresponds to the incremental version of XL described in Remark 3.14. Firstly, let the selection function in Step 5 be the identity,

i.e. for a set B of possible critical pairs, $Select(B) = B$. Secondly, modify Step 7 such that not only the two terms

$$\frac{LCM(LT(b_1), LT(b_2))}{LT(b_1)} b_1 \text{ and } \frac{LCM(LT(b_1), LT(b_2))}{LT(b_2)} b_2$$

of the S-polynomial corresponding to the critical pair (b_1, b_2) in B are included, but all multiplications of b_1 and b_2, up to total degree d, of the form

$$x^\alpha b_i, \text{ for } i = 1, 2 \text{ and } \alpha \in \mathbb{Z}^n_{\geq 0},$$

as created in Step 1 of the original XL algorithm from Definition 3.12. Therefore, Step 7 becomes

$$XL_d = \bigcup_{(b_1,b_2)\in B_d, i=1,2} \left\{ x^\alpha b_i : \alpha \in \mathbb{Z}^n_{\geq 0}, totaldeg(x^\alpha b_i) \leq d \right\}. \tag{5.1}$$

The third modification is that we can omit the Symbolic Preprocessing subroutine from Definition 4.16. This is because all the polynomials that are appended to the reduction matrix in F4, are included in expression (5.1) and therefore automatically used for reduction.

Showing the correctness and termination is analogous to the proof of Theorem 4.21. Lemma 4.18 is applicable in this case since the modified set XL_d includes the polynomials in its F4 equivalent L_d from expression (4.3). Hence the intermediate bases during the incremental application of XL correspond to an ascending chain of ideals. This implies termination. Correctness follows from the fact that all generated elements are members of the ideal spanned by F. These observations lead to the following proposition.

PROPOSITION 5.1. *The incremental version of XL returns a Gröbner Basis for a degree smaller than or equal to the largest degree of an intermediate polynomial during the algorithm F4, d_{F4}.*

PROOF. During the course of F4, the largest degree of a polynomial that contributes to an intermediate basis or to the reduction of other polynomials equals d_{F4}. XL does not need polynomials of larger degree during the reduction, since the polynomials in L_d generated by F4 are included in the set XL_d and are sufficient to form a Gröbner Basis. Note that the incremental version of XL should be applied such that it agrees with the selection strategy of the F4 algorithm. □

The above description is illustrated by an example in Magma. We show that XL is able to find a partial solution for a degree smaller than the one at which the algorithm finds the Gröbner Basis. The set F is included in Appendix B.

EXAMPLE 5.2.

```
> // Encrypt a plaintext and generate the HFE equations
> y, F := HFEencrypt([1,1,1,1,1,1,1]);
> // Solve the system by XL for degree 3 and catch the polynomials F2
> // corresponding to the reduced matrix generated in Step 2
> s, F2 := XL(F, 3);
********** XL **************
Time of EchelonForm(Fmatrix) (s)   : 0
*** Summary ***
Memory size of process (kb)        : 200
Size of XL matrix                  : 112 rows and 120 columns
Size of row reduced XL matrix      : 109 rows and 120 columns
```

```
Number of univariate polynomials   : 17
Number of single root polynomials  : 15
Number of univariate monomials     : 22
Number of variables solved         : 5
Is the system a Groebner Basis     : false
Substitution sequence [[var,root]] :
[ [ 7, 1 ],    [ 3, 1 ],    [ 6, 1 ],    [ 2, 1 ],    [ 1, 1 ] ]

> // Now apply XL to the polynomials generated by XL for degree 3
> s, F3 := XL(F2, 4);
*********** XL **************
Time of EchelonForm(Fmatrix) (s)   : 0.02
*** Summary ***
Memory size of process (kb)        : 9772
Size of XL matrix                  : 2272 rows and 330 columns
Size of row reduced XL matrix      : 329 rows and 330 columns
Number of univariate polynomials   : 28
Number of single root polynomials  : 28
Number of univariate monomials     : 29
Number of variables solved         : 7
Is the system a Groebner Basis     : true
Substitution sequence [[var,root]] :
[ [ 4, 1 ],    [ 7, 1 ],    [ 3, 1 ],    [ 6, 1 ],
  [ 2, 1 ],    [ 5, 1 ],    [ 1, 1 ] ]
```

If one compares the performance of F4 and XL, there are two parameters one should take into account. The complexity of XL mainly depends on the smallest degree for which it returns a Gröbner Basis or a (partial) solution. Whether one chooses to look for one or the other, depends on the goals of the cryptanalyst. For example, when attacking HFE it is very common that the cryptanalyst encounters more plaintexts corresponding to the ciphertext. In the worst case it might even happen that there is no unique solution to any of the variables. This means that XL does not find a solution. Hence, searching for a Gröbner Basis returns more information, but usually needs a slightly higher degree.

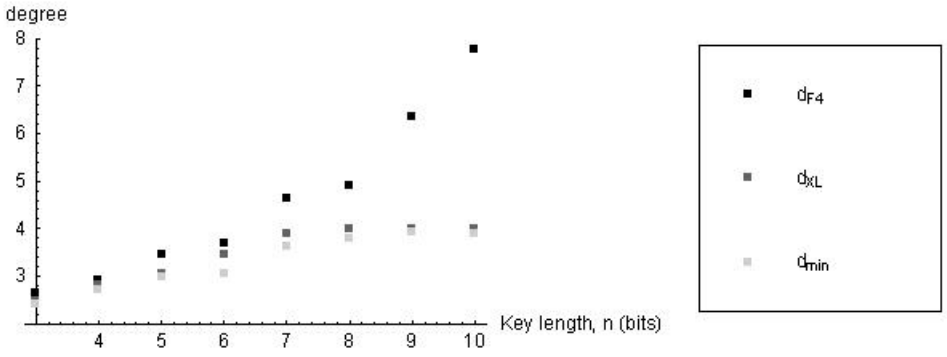

FIGURE 5.1. Comparison of the average degree to find a Gröbner Basis or a partial solution.

Figure 5.1 depicts the average degree to find a Gröbner Basis for a system of equations in *lex* order corresponding to a HFE plaintext. Fifty samples were taken per key length. The plotted d_{min} is the smallest degree for which XL finds a partial solution. Similarly, d_{XL} corresponds to the smallest D to find a Gröbner Basis with XL. Simulations with HFE for key sizes beyond 10 bits did not return a d_{XL} greater than 4, when the degree of the secret HFE polynomial was chosen smaller than 64. These simulations are very time and memory consuming and therefore small in number. Nevertheless, up to 17 bits we did not find instances of HFE with $d_{XL} > 4$.

XL applied incrementally could be regarded as an elaborate variation on the Buchberger Algorithm. It selects all critical pairs at once, multiplies with all possible monomials and reduces the new-found elements in the ideal with respect to an intermediate basis that consists of all elements in the ideal considered by the algorithm. The similarity is most striking when we compare XL to the algorithm F4. The extensiveness of XL implies many useless computations but seems optimal when one considers the largest degree of intermediate polynomials during the algorithm.

As mentioned above, XL is a cumbersome way to compute a Gröbner Basis. This is mainly due to the fact that there is no selection criterion in place. In that sense, algorithm XSL, briefly introduced in Section 3.4, is an improvement on the original XL, since it does not multiply the original polynomials with all possible monomials up to a certain degree. However, continuing on the discussed relation between F4 and XL, one could also easily embed a selection criterion in XL similarly to F4. Since the Gebauer and Möller Installation is much better understood than XSL, one could experiment with this criterion in combination with XL.

To illustrate the fact that XL is more time consuming than F4, the sizes of the matrices for both algorithms applied to HFE are depicted in Figure 5.2. The largest matrix that occurs during F4 is compared to the size of the matrices occurring in XL. Applying Gaussian reduction to these matrices forms the most time consuming part of both algorithms. The complexity equals $\mathcal{O}((\#rows)^{\omega})$ operations in $GF(2)$, where $\omega = 2.376$ is the best possible Gaussian exponent explained in Proposition 3.16.

5.2. Approximated lowest degree for XL

Approximating the complexity of Gröbner Basis algorithms is a difficult but important problem in computer algebra. The complexity is often expressed in terms of the largest degree of a polynomial during the computation. In this chapter we discuss the derivation of a lower bound for the lowest degree, d_{min}, for which XL finds a partial solution to a system of algebraic equations.

The method of estimating d_{min} was introduced by Moh in [**Moh01**]. Under the assumption that a set of polynomials F is a regular sequence, he derived an expression for the number of linearly independent polynomials generated by XL. In [**FA04**], Faugère and Ars used the Hilbert series of regular sequences to derive a lower bound for d_{F5}. Later, Diem [**Die04**] derived a lower bound for d_{min} based on the same Hilbert series and provided more background to the analysis.

Let $P = k[x_1, \ldots, x_n]$ be a polynomial ring, $F = \{f_1, \ldots, f_m\}$ a subset of P and $U_{\leq D}$ the k-vector space generated by the XL polynomials

$$\left\{x^{\alpha} f_i : \alpha \in \mathbb{Z}^n_{\geq 0}, 1 \leq i \leq m \text{ and } totaldeg(x^{\alpha} f_i) \leq D\right\}.$$

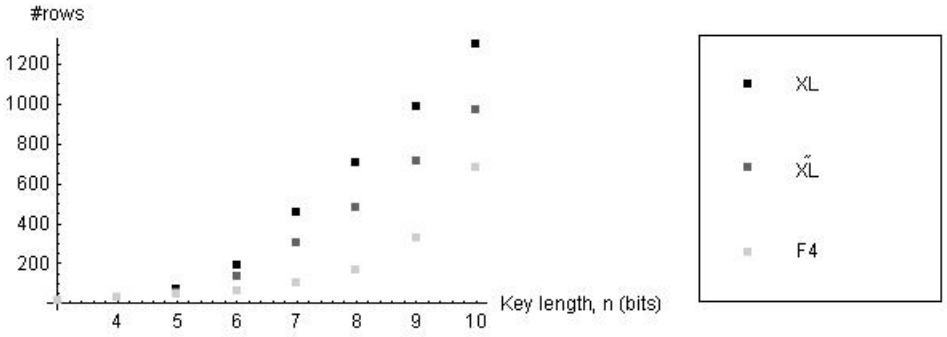

FIGURE 5.2. Comparison of the number of rows during the algorithms F4 and XL applied to HFE. $\tilde{XL}$ refers to the XL matrix after reduction.

Let $P_{\leq D}$ denote the elements in the polynomial ring P up to degree D. The *vector space dimension* of $P_{\leq D}$ and $U_{\leq D}$ are denoted by respectively $dim_k(P_{\leq D})$ and $dim_k(U_{\leq D})$.

In [**Die04**, p.03] a function $\chi(D)$ is introduced, satisfying

$$\chi(D) = dim_k(P_{\leq D}) - dim_k(U_{\leq D}).$$

Suppose the columns in the XL matrix are ordered in such a way that variable x_n is eliminated last. If the number of linearly independent polynomials, $dim_k(U_{\leq D})$, in the set U_D approaches the number of columns, or equivalently the number of distinct monomials in the system, it becomes more and more likely that one finds a univariate monomial.

Let us be more precise. If the monomials x_n^i, for $i = 0, 1, \ldots, D$, exist in the support and the relation

$$|T(U_{\leq D})| - dim_k(U_{\leq D}) \leq D + 1$$

holds, then it is certain that XL finds a univariate expression in the variable x_n that helps in solving the variable x_n. However, we are looking for the degree D that satisfies a slightly different relation, namely

$$\chi(D) \leq D + 1, \tag{5.2}$$

or $\chi(D) \leq D$ if we are sure that the support does not include constant terms. This is due to the following observation. For general systems of polynomial equations we do not know the size of the support, therefore we search for the difference between $|T(P_{\leq D})|$ and $dim_k(U_{\leq D})$, being $\chi(D)$. For equations coming from HFE and random quadratic systems of equations, the sizes of $|T(P_{\leq D})|$ and $|T(U_{\leq D})|$ are equal in most cases.

However, there are three important remarks concerning this simplification:

(1) For sparser systems of equations, the difference between the vector space dimension $dim_k(P_{\leq D})$ and $|T(U_{\leq D})|$ becomes larger. This might lead to distorted results that indicate that expression (5.2) is satisfied, while it is not.

(2) If the x_n coordinates of the different points in $\mathbf{V}(I)$ are distinct, the univariate polynomial in x_n can not give a decisive answer.
(3) If the field equations are included, it might happen that $x_n^2 - x_n$ is returned instead of a polynomial that corresponds to a unique solution for x_n.

For the purpose of an approximation of the degree for which XL returns a partial solution, the simplification $\chi(D) \leq D+1$ suffices for many instances. However, the condition is not necessary as we can see in Example 5.2.

Lower bounds for $\chi(D)$ are derived via homogeneous ideals, therefore the notion of homogenized polynomials is introduced.

DEFINITION 5.3. Let $g(x_1, \ldots, x_n) \in k[x_1, \ldots, x_n]$ be a polynomial of total degree d and $\sum_{i=0}^{d} g_i$ be the expansion of g as the sum of its homogeneous components, where g_i has total degree i. The *homogenization* of g is defined as

$$\begin{aligned} g^h(x_0, x_1, \ldots, x_n) &= \sum_{i=0}^{d} g_i(x_1, \ldots, x_n) x_0^{d-i} \\ &= g_d(x_1, \ldots, x_n) + g_{d-1}(x_1, \ldots, x_n)x_0 + \ldots + g_0(x_1, \ldots, x_n)x_0^d. \end{aligned}$$

REMARK 5.4. In the context of homogeneous ideals, the notation g^h refers to a homogenization of the polynomial g and not to a power of g.

The polynomial g^h is a homogeneous polynomial of total degree d in the polynomial ring $k[x_0, x_1, \ldots, x_n]$ including the extra variable x_0. Dehomogenizing g^h is done by evaluating g^h at $x_0 = 1$. Homogenization is an isomorphism [**CLO96**, p.454] and under this isomorphism, the k-vector space $U_{\leq D}$ corresponds to the k-vector space generated by the homogeneous polynomials

$$\left\{ x^\alpha f_i^h : \alpha \in \mathbb{Z}_{\geq 0}^{n+1}, 1 \leq i \leq m \text{ and } totaldeg(x^\alpha f_i^h) = D \right\}.$$

This space equals the D-th homogeneous component of the homogeneous ideal I spanned by $\left\{ f_1^h, \ldots, f_m^h \right\} \subset k[x_0, x_1, \ldots, x_n]$. Now we are ready to proceed to the following two important definitions. These definitions exist in greater generality of graded modules but for our purpose it suffices to restrict ourselves to homogeneous ideals.

DEFINITION 5.5. For a homogeneous ideal $I \subset k[x_0, x_1, \ldots, x_n]$, let the *Hilbert function* $HF_I(D)$ be defined as

$$HF_I(D) = dim_k((k[x_0, x_1, \ldots, x_n]/I)_D).$$

DEFINITION 5.6. The power series

$$HS(I) = \sum_{D=1}^{\infty} HF_I(D) T^D \in \mathbb{Z}[[T]]$$

is called the *Hilbert series* of I.

According to [**Die04**, p. 6], $\chi(D)$ satisfies

$$\begin{aligned}\chi(D) &= dim_k(P_{\leq D}) - dim_k(U_{\leq D}) \\ &= dim_k(P_{\leq D}/U_{\leq D}) \qquad (5.3) \\ &= dim_k(k[x_0, x_1, \ldots, x_n]_D/\langle f_1^h, \ldots, f_m^h\rangle_D) \\ &= dim_k((k[x_0, x_1, \ldots, x_n]/I)_D) \\ &= HF_I(D). \qquad (5.4)\end{aligned}$$

The derivation of a lower bound for D satisfying expression (5.2) is based on the observation that the coefficients of the Hilbert series of a homogeneous ideal are lower bounded by the Hilbert series of a so-called generic system of forms.

Definition 5.7. A homogeneous polynomial $g = \sum_i c_i x^i \in k[x_0, x_1, \ldots, x_n]$ of degree d is *generic* if all monomials of total degree d in $k[x_0, x_1, \ldots, x_n]$ have coefficients in g and these coefficients are algebraically independent over the prime field of k.

A system of generic homogeneous polynomials as above is a called *generic system of forms*. The reader is referred to [**Fro85**] for the definition in the general case of graded algebras. The lower bound for $HF_I(D)$ is formulated in the following two propositions.

Proposition 5.8. *Let k be any field and let $g_1, \ldots, g_m \in k[x_0, x_1, \ldots, x_n]$ be homogeneous polynomials of respective total degrees $d_1, \ldots, d_m$ spanning the homogeneous ideal I. Let $HS_{gen} \in \mathbb{Z}[[T]]$ be the Hilbert series corresponding to a generic system of forms and $HF_{gen}(D)$ its D-th coefficient. The Hilbert series corresponding to the homogeneous ideal I satisfies the following coefficient-wise inequality*

$$HF_I(D) \geq HF_{gen}(D), \text{ for } D \geq 0.$$

Proof. See [**Val96**, p. 317]. □

The Hilbert series of a generic system of forms is the consequence of a well known conjecture, known as the *maximal rank conjecture* [**Val96**, p. 317]. For m homogeneous polynomials in $k[x_0, x_1, \ldots, x_n]$, this conjecture is proven for $m \leq n$. Nevertheless, the correctness of the conjecture for general m and n is highly likely.

Proposition 5.9. *Let the finite field $k = GF(2)$. If the number of polynomials m in a generic system of forms $g_1, \ldots, g_m \in k[x_0, x_1, \ldots, x_n]$ with respective total degrees $d_1, \ldots, d_m$ satisfies $m \leq n$, then this generic system of forms is a regular sequence in $k[x_0, x_1, \ldots, x_n]$ and the Hilbert series of this system of forms equals the Hilbert series of a regular sequence, being*

$$\frac{(1+T)^{n+1}}{\prod_{j=1}^{m}(1+T^{d_j})}. \qquad (5.5)$$

Proof. See [**Val96**, p. 317]. □

Under the maximal rank conjecture, Proposition 5.9 holds for general m and n. Now, let d_{gen} denote the smallest D for which the expression $HF_{gen}(D) \leq D$ holds under this conjecture. For $1 \leq n \leq 32$ and $m = n$, the different values for d_{gen} are plotted in Figure 5.3. Figure 5.3 also illustrates that d_{gen} is very similar to d_{XL} for a random system of multivariate, quadratic equations including the field equations.

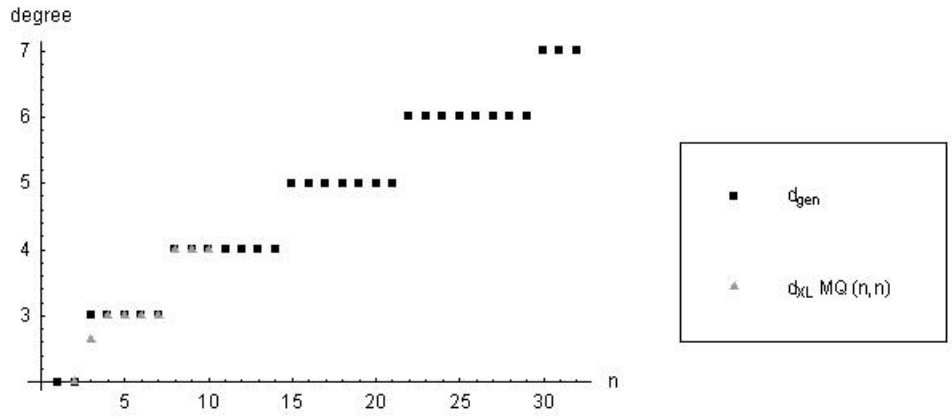

FIGURE 5.3. The first 32 values of d_{gen} and the average d_{XL} of a multivariate, quadratic system of n equations in n variables (10 samples). The field equations were included during the simulations.

REMARK 5.10. The development of d_{gen} is very close to linear for a larger domain. Its linear least-square fit for $1 \leq n \leq 512$ equals $3.89 + 0.0996n$.

Hence, for general ideals the value of D for which it is likely that XL finds a solution is claimed to be lower bounded by d_{gen}. Unfortunately, there are examples satisfying $D = d_{min} < d_{gen}$ while $\chi(D) > D$. Some equations, for example those corresponding to HFE plaintext attacks, have a lower d_{min} than the d_{gen} predicted by the foregoing analysis. The following example shows a HFE plaintext attack having a smaller d_{min} than the lower bound depicted in Figure 5.3.

EXAMPLE 5.11 (Incremental XL in Magma). *Applying XL two times for degree $D = 3$, returns a partial solution for the set of HFE equations stated in Appendix B while having $\chi(3) = 220 - 179 \not\leq 3 + 1$ and smaller $d_{min} = 3$ and $d_{XL} = 3$ than the $d_{gen} = 4$ predicted above.*

```
> y , F := HFEencrypt([1,1,1,1,1,1,1,1,1]);
> s, F2 := XL(F, 3);
********** XL, D=3 **************
Time of EchelonForm(Fmatrix) (s)   : 0

*** Summary ***
Memory size of process (kb)        : 502
Size of XL matrix                  : 180 rows and 220 columns
Size of row reduced XL matrix      : 179 rows and 220 columns
Number of univariate polynomials   : 17
Number of single root polynomials  : 15
Number of univariate monomials     : 28
Number of variables solved         : 5
Is the system a Groebner Basis     : false
Substitution sequence [[var,root]] :
[[ 6, 1 ], [ 9, 1 ], [ 5, 1 ], [ 1, 1 ], [ 8, 1 ]]
> // Apply XL to the set F2 of reduced polynomials returned by XL(F,3)
> s, F2 := XL(F2, 3);
********** XL, D=3 **************
```

```
Time of EchelonForm(Fmatrix) (s)   : 0.01

*** Summary ***
Memory size of process (kb)        : 2760
Size of XL matrix                  : 935 rows and 220 columns
Size of row reduced XL matrix      : 219 rows and 220 columns
Number of univariate polynomials   : 27
Number of single root polynomials  : 27
Number of univariate monomials     : 28
Number of variables solved         : 9
Is the system a Groebner Basis     : true
Substitution sequence [[var,root]] :
[[ 4, 1 ], [ 7, 1 ], [ 3, 1 ], [ 6, 1 ], [ 2, 1 ],
 [ 9, 1 ], [ 5, 1 ], [ 1, 1 ], [ 8, 1 ]]
```

Joux and Faugère derived that for a secret HFE polynomial of degree d satisfying $17 \leq d \leq 96$, the largest total degree of a critical pair during the Gröbner Basis computation did not exceed four, as was shown in Table 3.2 in Section 3.2. While trying to understand why, we found algebraic systems coming from cryptography with $d_{min} < d_{gen}$ where $\chi(D) \leq D + 1$ is not a necessary condition for XL to find a partial solution.

Hence, a system that is weaker than a random multivariate quadratic system of equations, for example HFE, is likely to have a d_{min} smaller than the smallest degree satisfying the lower bound derived with generic Hilbert series. Therefore, this way of analyzing a cipher might overestimate its strength.

5.3. F5 and the index of regularity

This section tries to shed some light on two important claims concerning the complexity of algebraic attacks. The proofs are hard to verify but simulations point in the same direction. The results enable the cryptanalyst to distinguish a random algebraic system from an algebraic system having a weaker structure, like HFE.

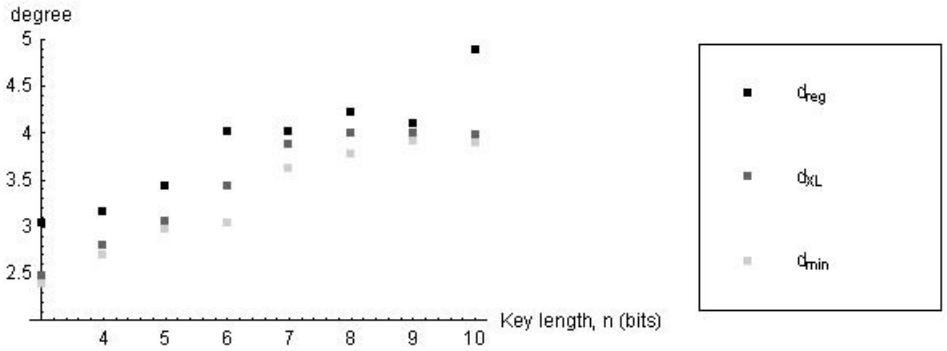

FIGURE 5.4. Comparison of d_{reg} and the average degree to find a *lex* Gröbner Basis or a partial solution with XL for HFE.

In [**FA04**], Faugère and Ars propose that their F4-like implementation of the F5 criterion is able to find a Gröbner Basis for a maximal degree d_{F5} that equals

d_{XL}. In a previous report [**MFS03**], Bardet, Faugère and Salvy assert that the so-called *index of regularity* forms an upper bound for d_{F5}, if F5 is applied to homogeneous regular sequences or sets of polynomials that forms a regular sequence after homogenization.

The authors conjecture in the same paper that the proportion of sets of n polynomials in n variables of this type tends to 1 as n goes to infinity. Hence, the index of regularity should be an upper bound for d_{XL} for large random algebraic systems.

The authors [**MFS03**] give a definition for degree of regularity in the special case of algebraic systems over the finite field of two elements and claim that this is equivalent to the index of regularity as defined in [**CLO96**, p. 449]. Both definitions are included for the sake of completeness. Firstly, we present the definition of Bardet, Faugère and Salvy in the special case of polynomials over $GF(2)$.

DEFINITION 5.12. Let $F = \{f_1, \ldots, f_m\}$ be a set of polynomials in the polynomial ring $GF(2)[x_1, \ldots, x_n]$ and $I = \langle f_1, \ldots, f_m \rangle$. The *degree of regularity* of I as defined in [**MFS03**] equals the minimal degree d such that

$$\left\{ LT(f) : f \in I^h,\ totaldeg(f) = d \right\},$$

equals the set of monomials of degree d in the quotient space of the polynomial ring containing I^h, modulo the field equations.

We define a function that stores information about the vector space dimension of the quotient space of an ideal I up to a certain degree, $P_{\leq D}/I_{\leq D}$, see for example [**CLO96**, p. 447].

DEFINITION 5.13. Let I be the ideal spanned by the set $F \subset P$. The *affine Hilbert function*, ${}^aHF_I$, of the intersection $I \cap P_{\leq D}$ is defined by

$$ {}^aHF_I(D) = dim_k(P_{\leq D}/I_{\leq D}) = dim_k(P_{\leq D}) - dim_k(I_{\leq D}). \tag{5.6}$$

Now, the common definition of index of regularity is as follows.

DEFINITION 5.14 (Index of regularity from [**CLO96**]). The polynomial which equals the affine Hilbert function ${}^aHF_I(d)$ for sufficiently large d is called the *affine Hilbert polynomial* of the ideal I and is denoted by ${}^aHP_I(d)$. The smallest integer d_{reg} such that

$$ {}^aHP_I(d) = {}^aHF_I(d), \text{ for all } d \geq d_{reg},$$

is called the *index of regularity*.

Unfortunately, our implementation of F5 does not seem to behave as nicely as claimed by Faugère. In spite of this, Figures 5.4 and 5.5 show that the index of regularity forms an upper bound for the degree at which XL is able to return a Gröbner Basis in the HFE and random multivariate, quadratic case. The latter shows that the smallest D for which XL finds a Gröbner Basis in the random multivariate, quadratic case equals d_{reg} for examples $n = 7$, 8, 9 and 10. Although XL may not be as efficient in terms of matrix size as for example F4, it does show that there exist Gröbner Basis algorithms which are substantially better in terms of degrees of intermediate polynomials.

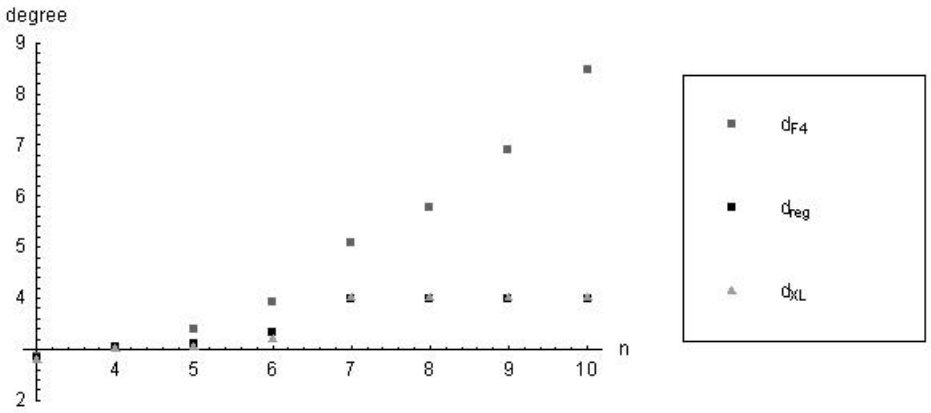

FIGURE 5.5. Comparison of the average degree to find a *lex* Gröbner Basis for a random multivariate, quadratic system of n equations in n variables over $GF(2)$, including the field equations (50 samples).

CHAPTER 6

Conclusion

Let us summarize the various ideas concerning the complexity of algebraic attacks. Throughout the report we have distinguished two important measures for complexity, the first being the size of the largest matrix involved and the second being the largest degree of a polynomial during the Gröbner Basis computation.

We have seen that the largest degree of a critical pair during the course of the algorithm F4 gives a reliable upper bound on the parameter D for a successful XL attack. Even for small D the matrices involved during XL computations are much larger since there is no efficient selection criterion installed. However, XL is useful in another sense.

In Section 5.1, the similarities between F4 and XL have been made explicit. We showed that XL is in fact a Gröbner Basis algorithm when applied incrementally. The smallest degree for which incremental XL finds a Gröbner Basis suggests the existence of a Gröbner Basis algorithm that is able to keep the total degree of intermediate polynomials to a minimum.

Faugère claims that his F5 is close to optimal in this sense and that it does not create critical pairs that reduce to zero in the case of a regular sequence of polynomials. Therefore, the F5 criterion deserves further investigation. In Section 4.4 the F5 criterion is explained. This section also introduces the Gebauer and Möller Installation, which is known to be close to optimal but does not excel in terms of lowest total degree during computations.

With XL or a similar algorithm that is able to keep the total degree of the intermediate computations as low as possible, the cryptanalyst has a tool to make a distinction between random algebraic systems and systems with more structure. Like HFE, the latter is less resistant to algebraic attacks and simulations show that for increasing number of equations and variables the largest total degree during computations is smaller on average.

The robustness of cryptosystems to XL attacks is sometimes based on a complexity analysis based on Hilbert series. Diem summarizes the results regarding this analysis in a clear paper that is accepted for AsiaCrypt in December 2004. In Section 5.2 we discuss this analysis. The stated condition at which XL returns a partial solution does not seem to be necessary for certain systems of equations coming from cryptography. The claimed polynomial complexity of the HFE attack by Faugère and Joux supports our idea.

Directions for further investigation might include the search for a degree optimal Gröbner Basis algorithm with an effective criterion to avoid useless critical pairs. Furthermore, the complexity analysis of XL based on Hilbert series requires further investigation since it seems that some discrepancies exist. Finally, we think it is interesting to find out whether there exist symmetric cryptosystems that show behavior similar to HFE in terms of d_{XL}.

APPENDIX A

Algorithms written for Magma

This chapter contains routines to simulate algebraic attacks on various cryptosystems. We implemented the following algorithms in the computer algebra language Magma [**BCP97**]:

- The four byte blockcipher from Section 3.1;
- The asymmetric cryptosystems HFE introduced in Section 3.2;
- The Homogeneous Buchberger Algorithm explained in Section 4.2;
- The algorithm F4 from Section 4.3;
- The algorithm F5 explained in Section 4.4.

The routines work as follows:

- Let *in* and k be sequences of four times two letters representing hexadecimal values, for example [["1","3"],["2","4"],["5","7"],["a","c"]]. The command `fbbc(in,key,r)` returns a similar sequence, representing four bytes, corresponding to the ciphertext after encrypting plaintext *in* with key *key* for r rounds of the four byte blockcipher.

  The routine `fbbcequations(in,out,r)` returns the set of polynomials describing an algebraic key recovery attack for plaintext *in*, output *out* and r rounds of encryption.
- `HFE(in)` returns a sequence of ciphertext bits and the equations for a plaintext recovery attack. The input is a binary sequence of arbitrary length. The routine randomly generates the secret affine transformations and HFE polynomial of maximal degree 64.
- `HomBuchGM(F)` returns the Gröbner Basis corresponding to a sequence of polynomials F and the set of critical pairs considered during the computation.
- `F4(F)` returns a Gröbner Basis, the largest degree of a critical pair, the number of critical pairs, the number of rows in the largest reduction matrix, the largest degree of an intermediate polynomial during computation and the density of the largest reduction matrix respectively.
- `F5(F)` returns the Gröbner Basis, a boolean value if the answer is a Gröbner Basis, the number of created S-polynomials, the largest degree of an S-polynomial and the largest degree of a contributing critical pair.
- Let F be a sequence of polynomials and D an integer. The command `XL(F,D)` returns a sequence of solved variables, the polynomials corresponding to the matrix after row reduction, a boolean variable if these polynomials form a Gröbner Basis, the number of created polynomials, the number of polynomials after reduction, the size of the support, the original polynomials evaluated at the partial solution and the the time it takes Magma to row reduce the matrix for degree D.

A.1. Shared routines

```
// Calculate an S-polynomial, used in HomBuchGM and F4
S := function(f,g)
    LTf := LeadingTerm(f);
    LTg := LeadingTerm(g);
    m := LCM(LTf,LTg);
    return (m div LTf) * f - (m div LTg) * g;
end function;

// A function to update the critical pairs without the Gebauer and Moeller criteria, used in HomBuchGM and F4
UpdatePairs := function(D, F)
    // Initialization
    s := #F;
    C := {[i,s] : i in [1..s-1]};
    D := D join C;
    return D;
end function;

// The Gebauer and Moeller criteria, used in HomBuchGM and F4
// See for an explanation Gebauer and Moeller (1987) On an Installation of Buchberger's Algorithm
// Last modifications 13-8-2004
UpdateGM := function(D, F)
    // Initialization
    t := #F;

    // Precompute all leading terms
    T := [ LeadingTerm(F[i]) : i in [1..t] ];

    // Rule B_t(i,j)
    D := {p : p in D | LCM(T[p[1]],T[p[2]]) ne LCM(LCM(T[p[1]],T[p[2]]), T[t])
              or LCM(T[p[1]],T[t]) eq LCM(T[p[1]],T[p[2]])
              or LCM(T[p[1]],T[p[2]]) eq LCM(T[p[2]],T[t])
              or LCM(T[p[1]],T[t]) eq LCM(T[p[2]],T[t])};

    // Create the set D1
    D1 := {[i,t] : i in [1..t-1]};

    // Rule M(i,t)
    for i in [1..t-1] do
        if exists(p1){ p : p in D1 | IsDivisibleBy(LCM(T[i],T[t]), LCM(T[p[1]],T[p[2]]))
                  and LCM(T[i],T[t]) ne LCM(T[p[1]],T[p[2]]) }
        then
            Exclude(~D1, [i,t]);
            print "Delete in D1 pair ", [i,t];
        end if;
    end for;

    // Modified criterion F
    // Create the set of all LCM monomials corresponding to the pairs in D1
    tauset := {LCM(T[p[1]],T[p[2]]) : p in D1};

    D1new := {};

    while not IsEmpty(tauset) do
        // Treat every subset of pairs of D1 with LCM equal to tau seperately
        tau := Rep(tauset);
        Exclude(~tauset, tau);
        subsetD1tau := {ptau : ptau in D1 | LCM(T[ptau[1]],T[ptau[2]]) eq tau};

        if exists(p1){ ps : ps in subsetD1tau | T[ps[1]]*T[ps[2]] eq LCM(T[ps[1]],T[ps[2]])}
        then
            subsetD1tau := {p1};
        else
            subsetD1tau := {Rep(subsetD1tau)};
        end if;

        D1new := D1new join subsetD1tau;
    end while;
    D1new := {p : p in D1new | T[p[1]]*T[p[2]] ne LCM(T[p[1]],T[p[2]])};

    return D join D1new;
end function;

// auxilary function to convert polynomial to sequence, used in F4 and XL
polytovector := function(poly, monseq)  // create vector
representing
                  // the poly in R^(#monseq) with
                  // respect to the sequence of
                  // monomials of the system.
    // check the parent object of the polynomial to create a vector over the right basering
```

```
    P := Parent(poly);
    vector := [BaseRing(P) | 0 : i in [1..#monseq]];
    M:=Monomials(poly);

    for i in [1..#M] do
        vector[Index(monseq, M[i])]:=MonomialCoefficient(poly,M[i]);
    end for;
    return vector;
end function;

// auxilary function to convert vector to polynomial, used in F4 and XL
vectortopoly := function(row, monseq)   // the inverse of
polytovector
    n := #monseq;
    if row eq [0 : x in [1..n]] then poly := 0;
    else poly := &+[monseq[i]*row[i] : i in [1..#monseq] | row[i] ne 0];
    end if;
    return poly;
end function;

// Create a function addwithcarry to assist in the generation of all the divisors of a term for Simplify.
// This function creates a list of all vectors with smaller values in all positions than a given maxvector.
// It is used in F4 and XL.
addwithcarry := function(tempvector, maxvector, pos);
    if tempvector[pos] lt maxvector[pos] then
        tempvector[pos] +:= 1;
    else
        tempvector[pos] := 0;
        // recursion
        tempvector := $$(tempvector, maxvector, pos + 1);
    end if;

    return tempvector, pos;
end function;
```

A.2. Four byte blockcipher from Section 3.1

```
// A four byte blockcipher
// Start: 13-5-2004, last update 14-6-2004
// Implementation of a small blockcipher with similarities to AES scalable in the number of rounds.

// Converts a sequence of hexidecimal characters to a binary representation
hextobinary := function(hex)
    bin := [];
    for i in [1..#hex] do
        case hex[i]:
            when "0": bin cat:= [0,0,0,0];
            when "1": bin cat:= [1,0,0,0];
            when "2": bin cat:= [0,1,0,0];
            when "3": bin cat:= [1,1,0,0];
            when "4": bin cat:= [0,0,1,0];
            when "5": bin cat:= [1,0,1,0];
            when "6": bin cat:= [0,1,1,0];
            when "7": bin cat:= [1,1,1,0];
            when "8": bin cat:= [0,0,0,1];
            when "9": bin cat:= [1,0,0,1];
            when "a": bin cat:= [0,1,0,1];
            when "b": bin cat:= [1,1,0,1];
            when "c": bin cat:= [0,0,1,1];
            when "d": bin cat:= [1,0,1,1];
            when "e": bin cat:= [0,1,1,1];
            when "f": bin cat:= [1,1,1,1];
        end case;
    end for;
    return bin;
end function;

// Converts the sequence of bits to its hexidecimal representation
bintohex := function(bin)
    hex := [];
    for i in [1..#bin] do
        case bin[i]:
            when [0,0,0,0]: hex cat:= ["0"];
            when [1,0,0,0]: hex cat:= ["1"];
            when [0,1,0,0]: hex cat:= ["2"];
            when [1,1,0,0]: hex cat:= ["3"];
            when [0,0,1,0]: hex cat:= ["4"];
            when [1,0,1,0]: hex cat:= ["5"];
            when [0,1,1,0]: hex cat:= ["6"];
            when [1,1,1,0]: hex cat:= ["7"];
```

```
            when [0,0,0,1]: hex cat:= ["8"];
            when [1,0,0,1]: hex cat:= ["9"];
            when [0,1,0,1]: hex cat:= ["a"];
            when [1,1,0,1]: hex cat:= ["b"];
            when [0,0,1,1]: hex cat:= ["c"];
            when [1,0,1,1]: hex cat:= ["d"];
            when [0,1,1,1]: hex cat:= ["e"];
            when [1,1,1,1]: hex cat:= ["f"];
        end case;
    end for;
    return hex;
end function;

// Subroutine for the S-box, it warns when an 'inversion' of zero occurs.
Sbox := function(In, F)
    Out := [F|];
    for i in [1..#In] do
        if In[i] eq 0 then
            print "Inversion of zero.";
            Out[i] := 0;
        else
            Out[i] := In[i]^(-1);
        end if;
    end for;
    return Out;
end function;

// A 4 byte blockcipher function 4bbc that encrypts plaintext In with key k using r rounds
// The plaintext and key bytes are char pairs representing hexadecimal numbers
fbbc := function(In, k, r)

    print "********** 4 byte blockcipher ***********";
    print "Plaintext";
    print In;
    print "Key bytes";
    print k;
    print "First step";

    P<x> := PolynomialRing(GF(2));
    // Use the Rijndael polynomial to specify the finite field isomorphism
    p := x^8+x^4+x^3+x+1;
    F<t> := ext< GF(2) | p >;

    // Initialize memore state variables
    b := [];
    c := [];
    d := [];

    // Express the 4 byte key and plaintext as elements of F
    // Example:     ["1","2"] is encoded as [1,0,0,0,0,1,0,0] by hextobinary
    // this corresponds to t^1+t^4 in field F
    In := [&+[hextobinary(In[i])[j+1]*t^j : j in [0..7]] : i in [1..4]];
    k  := [&+[hextobinary(k[i])[j+1]*t^j : j in [0..7]] : i in [1..4]];

    // Start actual encryption
    // Add key
    a := [k[i]+In[i] : i in [1..4]];
    print "a=",a;
    // Start applying multiple rounds of encryption
    d[1] := a;
    for round in [1..r] do
        print "Round", round;
        // S-box
        b[round] := Sbox(d[round], F);
        // Mixing layer
        c[round] := [&+[b[round][j] : j in [1..4] | i ne j] : i in [1..4]];
        // Add key
        d[round+1] := [k[i]+c[round][i] : i in [1..4]];
    end for;

    print "Last step";
    // S-box
    e := Sbox(d[r+1], F);
    print "e=",e;
    // Add key
    Out := [k[i]+e[i] : i in [1..4]];

    // Convert the finite field elements to hexadecimal representation
    Out := [bintohex([Eltseq(Out[i])[1..4], Eltseq(Out[i])[5..8]]) : i in [1..4]];

    print "Return the ciphertext";
    return Out;
end function;
```

```
// Create the equations corresponding to the 4bbc cipher for r rounds, input In and output Out
fbbcequations := function(In, Out, r)

    // Initialize the finite field F with Rijndael polynomial p
    P<x> := PolynomialRing(GF(2));
    p := x^8+x^4+x^3+x+1;
    F<t> := ext< GF(2) | p >;

    // Create a new polynomialring with 12+12r variables, the total number of variables
    // needed for r rounds
    Q<[z]> := PolynomialRing(F, 12+12*r, "lex");

    In  := [&+[hextobinary(In[i])[j+1]*t^j : j in [0..7]] : i in [1..4]];
    Out := [&+[hextobinary(Out[i])[j+1]*t^j : j in [0..7]] : i in [1..4]];

    // Create the correspondence between state variables and variables in polynomialring Q<[z]>
    // See Chapter 3 in the thesis for the description of the variables
    a := [z[1], z[2], z[3], z[4]];
    b := &cat[[[z[1+4*i], z[2+4*i], z[3+4*i], z[4+4*i]]] : i in [1..r]];
    c := &cat[[[z[4*r+1+4*i], z[4*r+2+4*i], z[4*r+3+4*i], z[4*r+4+4*i]]] : i in [1..r]];
    d := [[a[1],a[2],a[3],a[4]]] cat
        &cat[[[z[8*r+1+4*i], z[8*r+2+4*i], z[8*r+3+4*i], z[8*r+4+4*i]]] : i in [1..r]];
    e := [z[5+12*r],z[6+12*r],z[7+12*r],z[8+12*r]];

    // The key bytes are the last 4 variables of the 13+12r variables z[1],z[2],...
    k := [z[9+12*r],z[10+12*r],z[11+12*r],z[12+12*r]];

    // Create the set of polynials corresponding to the equations
    EQ :=   [In[i]+k[i] - a[i] : i in [1..4]] cat
        // Represent the different rounds by polynomials
        // Variable round is a counter for the rounds
        &cat[
            // S-box polynomials
            [d[round][i]*b[round][i] - 1 : i in [1..4]] cat
            // Mixing layer
            [c[round][i] - &+[b[round][j] : j in [1..4] | i ne j] : i in [1..4]] cat
            // Key addition
            [c[round][i]+k[i] - d[round+1][i] : i in [1..4]]
            : round in [1..r]
        ]
        cat
        // And the final S-box and key addition step
        [d[r+1][i]*e[i]-1 : i in [1..4]] cat
        [e[i]+k[i] - Out[i] : i in [1..4]];

    // Rename the intermediate veriables z in the polynomialring Q to the more common characters
    nc :=   ["a_1","a_2","a_3","a_4"] cat
        ["b_" cat IntegerToString(i) : i in [1..4*r]] cat
        ["c_" cat IntegerToString(i) : i in [1..4*r]] cat
        ["d_" cat IntegerToString(i) : i in [1..4*r]] cat
        ["e_1","e_2","e_3","e_4"] cat
        ["k_1","k_2","k_3","k_4"];
    AssignNames(~Q, nc);

    // EQ contains 4(3+r3)=12+12r polynomials
    return EQ;
end function;
```

A.3. Hidden Field Equations

```
// construction of HFE equations, 1-6-2004

// Convert a quadratic element of a quotient ring created in HFE() to an element of a polynomial ring
QRtoPR := function(qrpoly)
    QR := Parent(qrpoly);
    n := Rank(QR)-1;
    PR<[x]> := PolynomialRing(BaseRing(QR), n, "lex");

    M := Monomials(qrpoly);
    C := Coefficients(qrpoly);
    prpoly := 0;
    coeff := 0;

    for m in [1..#M] do
        if exists(t){ [i, j] : i, j in [1..n] | QR.i*QR.j eq M[m] }
        or exists(t){ [i] : i in [1..n] | QR.i eq M[m] }
        then
            if #t eq 2 then
```

```
                prpoly := prpoly + C[m]*PR.t[1]*PR.t[2];
            else
                prpoly := prpoly + C[m]*PR.t[1];
            end if;
        else
            c := [ coeff : coeff in BaseRing(QR) | coeff eq M[m] ][1];
            prpoly := prpoly + c;
        end if;
    end for;

    return PR ! prpoly;
end function;

// A HFE public key generator of size n bits
// The affine transformations are randomly generated, f=x*x^8 + x^4*x^16
HFE := function(n)

    P<[x]> := PolynomialRing(GF(2), n+1);

    // Create a irreducible polynomial of degree n
    // p := P.(n+1)^5 + P.(n+1)^4 + P.(n+1)^3 + P.(n+1) + 1;
    p := Evaluate(IrreduciblePolynomial(GF(2), n), P.(n+1));
    // Construct the quotient ring Q=P[X]/<p>
    Q := quo<P| p>;

    x := [P.i : i in [1..n]];

    // Initialize matrices A and B to zero
    A := [[GF(2) | 0 : i in [1..n]] : i in [1..n]];
    B := A;

    // Set the matrices and vectors defining the affine transformations s and t
    // Define A
    while not IsInvertible(Matrix(A)) do
        A := [ RandomSequenceRSA(100, n) : i in [1..n]];
    end while;

    // Define B
    while not IsInvertible(Matrix(B)) do
        B := [ RandomSequenceRSA(100, n) : i in [1..n]];
    end while;

    // And c and d
    c := RandomSequenceRSA(100, n);
    d := RandomSequenceRSA(100, n);

    // The affine transformation s(x) = A*x + c
    u := [&+[A[j][i]*x[i] : i in [1..n]] : j in [1..n]];
    u := [u[i] + c[i] : i in [1..n]];

    // HFE prescribes to use an f with more than one monomial
    // Otherwise the scheme is equivalent to C*

    // Apply secret polynomial f: v = u*u^8 + u^4*u^16
    // v := &+[u[i]*P.(n+1)^((i-1)) : i in [1..n]] * &+[u[i]*P.(n+1)^(8*(i-1)) : i in [1..n]]
    //   + &+[u[i]*P.(n+1)^(4*(i-1)) : i in [1..n]] * &+[u[i]*P.(n+1)^(16*(i-1)) : i in [1..n]];

    // Randomize f, for maximal exponent 2^e+2^e and 5 terms
    // This randomized secret HFE polynomial has average degree 40 and maximal degree 64
    e := 5;
    exp_v := [0 : i in [1..10]];
    // It might happen that no n-th powers of the root of p exist
    // Therefore we test if v has length n with boolean id
    id := false;
    while   not id
    //  and TotalDegree(&+[P.(n+1)^(exp_v[i])*P.(n+1)^(exp_v[i+1]) : i in [1..10 by 2]]) le 17
    do
        exp_v := [2^Random(e) : i in [1..10]];
        v :=    &+[&+[u[i]*P.(n+1)^(exp_v[j]*(i-1)) : i in [1..n]]
        * &+[u[i]*P.(n+1)^(exp_v[j+1]*(i-1)) : i in [1..n]] : j in [1..10 by 2]];
        v := Coefficients(Q ! v, P.(n+1));
        v := [v[i] : i in [1..#v]];

        // Test if v has length n
        id := IsDefined(v,n);
    end while;

    // The affine transformation t: y = B^-1(v-d)
    v := [v[i] - d[i] : i in [1..n]];
    Binv := RowSequence(Matrix(B)^-1);
    y := [&+[Binv[j][i]*v[i] : i in [1..n]] : j in [1..n]];

    // Convert the quadratic polynomials in the quotient ring to elements
```

```
    // of the polynomial ring by means of the function QRtoPR
    PR := Parent(QRtoPR(y[1]));
    PRy := [PR ! QRtoPR(y[i]) : i in [1..#y]];

/*  print "HFE Setup:";
    print "Number of bits n=", n;
    print "Affine transformation s: A=", A;
    print "c=", c;
    print "Affine transformation t: B=", B;
    print "d=", d;
*/
    print "HFE polynomial f=",
        &+[P.(n+1)^(exp_v[i])*P.(n+1)^(exp_v[i+1]) : i in [1..10 by 2]];
//  print "Degree f=", TotalDegree(&+[P.(n+1)^(exp_v[i])*P.(n+1)^(exp_v[i+1]) : i in [1..10 by 2]]);
//  print "Generated public key equations";

    return PRy;
end function;

HFEencrypt := function(plaintext)

    // Capute n, the number of plaintext bits and variables in the public key
    n := #plaintext;
    // Generate the public key
    PK := HFE(n);
    // Grab the properties of the polynomial ring of PK
    P := Parent(PK[1]);
    // Generate the ciphertext
    y := [Evaluate(PK[i], plaintext) : i in [1..n]];
    // Generate the basis of the ideal corresponding to the solutions
    F := [PK[i] - y[i] : i in [1..n]];
    F := F cat [P.i^2 - P.i : i in [1..n]];

    // Return the ciphertext y and a basis F of the ideal including the field equations
    return y, F;
end function;
```

A.4. Homogeneous Buchberger Algorithm

```
// Start: 23-4-2004
// Implementation of the Gebauer and Moeller criteria, BuchGM.
// Via the homogenization of BuchGM we try to create an F4 variant with the GM criteria.
// Eventually we try to create an F4 variant with minimal critical pairs.

// Buchberger algorithm with the Gebauer and Moeller criteria
// Last modifications 23-4-2004
HomBuchGM := function(F)

    // Initialization
    r := #F;
    G := [F[1]];
    B := {};

    Bdone := [];

    for t in [2..r] do
        Append(~G, F[t]);
        B := UpdateGM(B, G);
    end for;

    print "After Initialization, B = ", B;

    // Iteration
    while not IsEmpty(B) or not IsEmpty(F) do

        // The set of LCM's corresponding to the pairs in B
        LCMB := {LCM(LeadingTerm(G[p[1]]), LeadingTerm(G[p[2]])) : p in B};

        // The degree of the minimal element of LCMB and F w.r.t. the ordering
        d := Min([TotalDegree(i) : i in F cat Setseq(LCMB)]);

        // Select those critical pairs in B which have degree d
        Bd := { p : p in B | TotalDegree(LCM(LeadingTerm(G[p[1]]), LeadingTerm(G[p[2]]))) eq d};
        B := B diff Bd;

        // Select those elements in F of degree d
        Fd := [g : g in F | TotalDegree(g) eq d];
        F := [g : g in F | TotalDegree(g) ne d];
```

```
        // run through all critical pairs of degree d
        while not IsEmpty(Bd) do
            ExtractRep(~Bd, ~p);    // probably faster than Min and Exclude
            I := p[1];
            J := p[2];

            Append(~Bdone, p);

            h := S(G[I], G[J]);
            h := NormalForm(h, G);

            if h ne 0 then
                Append(~G, h);
                B := UpdateGM(B, G);
                // B := UpdatePairs(B,G);
                print "In Iteration after Update, B = ", B;
            end if;
        end while;

        // run through all original elements of degree d
        while not IsEmpty(Fd) do
            h := Fd[1];
            h := NormalForm(h, G);

            if h ne 0 then
                Append(~G, h);
                B := UpdateGM(B, G);
                // B := UpdatePairs(B,G);
                print "In Iteration after Update, B = ", B;
            end if;
            Remove(~Fd, 1);
        end while;

    end while;

    return G, Bdone;
end function;
```

A.5. Algorithm F4

```
/*
Start: 23-4-2004
(26-5-2004) Features of this version:
the Gebauer and Moeller-criteria,
suitable for homogeneous systems,
intermediate reduction w.r.t. G of the Fplus polynomials,
improved matrix to polynomial conversion (via RowSequence),
the improved F4 Symbolic Preprocessing.
(Since Magma 2.11 it is possible to pinpoint the performance bottlenecks in the implementation with a profiler.)
Last update: 13-9-2004
*/

// Set the outputprecision of the print statement to a maximum of 5 significant digits
AssertAttribute(FldPr, "OutputPrecision", 5);

F4Simplify := function(t, f, SPFred, SPFplus)

    // Create the set M of all divisors of a term t
    P := Parent(t);
    r := Rank(P);
    d := #SPFred;
    M := [];
    pos := 1;

    // maxvector is the multidegree of the term t
    maxvector := [Degree(t,i) : i in [1..r]];
    tempvector := [0 : i in [1..r]];

    // To create the set M we make use of the subroutine addwithcarry
    while tempvector ne maxvector and pos le r do
        tempvector, pos := addwithcarry(tempvector, maxvector, pos);
        Include(~M, &*[(P.i)^tempvector[i] : i in [1..r]]);
    end while;
    M := Reverse(Sort(M));
    s := #M;

    for j in [1..s] do
        for k in [1..d] do
            if IsDefined(SPFplus, k) and IsDefined(SPFred, k)
```

```
            and exists(fj){ g : g in SPFred[k] | M[j]*f eq g } then

                // exists(p){ g : g in SPFplus[k] | LeadingMonomial(g) eq LeadingMonomial(fj) };
                p := Rep({ g : g in SPFplus[k] | LeadingMonomial(g) eq LeadingMonomial(fj) });
                if j ne 1 then
                    return $$(t div M[j], p, SPFred, SPFplus);

                else
                    return 1, p;
                end if;
            end if;
        end for;
    end for;

    return t, f div LeadingCoefficient(f);
end function;

// Symbolic Preprocessing
SymbPrep := function(L, G, SPFplusd, SPFred)

    F := {};
    for l in L do
        t, f := F4Simplify(l[1], l[2], SPFplusd, SPFred);
        Include(~F, t*f);
    end for;

    D := {LeadingMonomial(i) : i in F};        // the set of leading terms
    M := &join{Seqset(Monomials(i)) : i in F}; // create the support of F

    while D ne M do
        m := Rep(M diff D); // pick one element m from M\D
        Include(~D, m);
        if exists(t){ g : g in G | IsDivisibleBy(m, LeadingTerm(g)) } then
            mf, f := F4Simplify(m div LeadingTerm(t), t, SPFplusd, SPFred);
            Include(~F, mf * f);
        end if;
    end while;
    return F;
end function;

FR := function(L, G, SPFplusd, SPFred, EFtimesum)
    // print "Number of elements in L (2x the number of critical pairs):", #L;
    // print "Number of elements in G (the number of basispolys so far):", #G;

    // print "Symbolic Preprocessing... ";
    F := SymbPrep(L, G, SPFplusd, SPFred);

    // create the ordered support of F
    // printf "Create the ordered support of F... ";
    M := Reverse(Sort(Setseq(&join{Seqset(Monomials(i)) : i in F})));

    // ask the characteristics of the polynomial ring
    // check whether this is too time consuming
    P := Parent(Rep(L)[1]);

    // create the matrix Fmatrix corresponding to the set of polynomials F
    // printf "Create the Fmatrix from F... ";
    Fmatrix := Matrix(BaseRing(P), [polytovector(i, M) : i in F]);

    densty := Density(Fmatrix);

    printf "Dimension of reduction matrix rows x columns        : %o x %o\n", Nrows(Fmatrix), Ncols(Fmatrix);
    printf "Density of Fmatrix                                  : %o\n", densty;

    // reduce to row echelon form and create the interreduced set of polynomials Fred
    EFtime := Cputime();
    Fmatrix := EchelonForm(Fmatrix);
    EFtimesum := EFtimesum + Cputime(EFtime);
    print "Time of EchelonForm(Fmatrix)                      :", Cputime(EFtime);

    // printf "Convert rows to polynomials... ";
    // my original choice was to cast the rows of the matrix as vectors
    // this was awefully slow so therefore I use the RowSequence operation
    Fmatrix := RowSequence(Fmatrix);
    nrows := #Fmatrix;
    Fred := [vectortopoly(Fmatrix[i], M) : i in [1..nrows]];
    Fred := [f : f in Fred | f ne 0];

    // create the set of polynomials Fplus with different leading terms than Fred
    DF := {LeadingMonomial(i) : i in F};  // the leading monomials of F
    // printf "Create the set Fplus of polys with new leading monomials... ";
    Fplus := [f : f in Fred | not LeadingMonomial(f) in DF ];
```

```
    return Fplus, Fred, EFtimesum, nrows, densty;

end function;

// Buchberger algorithm with the Gebauer and Moeller criteria
ImpHomF4GM := function(F)

    print "****************** F4 ******************";

    // Initialization
    F := [f : f in F | f ne 0];
    totaltime := Cputime();
    d := 0;         // the degree d of the LCM of the critical pairs
                    // or the degree of the original polynomials to process
    df4cp := 0;     // variable to track the maximum d during the algorithm
    df4 := 0;
    nrowsmax := 0;  // variable to track the largest number of rows in the reduction matrix
    denstynrowsmax := 0;

    r := #F;        // the number of original polynomials
    G := [F[1]];    // the intermediate set of basispolynomials
    B := {};        // the list of indices for the critical pairs

    SPFred := [];   // initialize sequences for improved reduction
    SPFplusd := []; // SPF and SPFplusd are thrown by Symbolic Preprocessing.

    nocp := 0;      // count the number of critical pairs
    redfr := 0;     // count the number of redundant polynomials spawned by FR
    FRtimesum := 0; // count the total time of the operation FR
    UpdateGMtimesum := 0; // count the total time of the operation UpdateGM
    EFtimesum := 0; // count the total time the EchelonForm takes


    // Iteration
    while (not IsEmpty(B) or not IsEmpty(F)) do

        // The set of LCM's corresponding to the pairs in B
        LCMB := {LCM(LeadingTerm(G[p[1]]), LeadingTerm(G[p[2]])) : p in B};

        // The degree of the minimal element of LCMB and F w.r.t. the ordering
        d := Min([TotalDegree(i) : i in F cat Setseq(LCMB)]);
        print "";
        print "Degree:", d;

        // Select those critical pairs in B which have degree d
        Bd := { p : p in B | TotalDegree(LCM(LeadingTerm(G[p[1]]), LeadingTerm(G[p[2]]))) eq d};
        B := B diff Bd;

        // Select those elements in F of degree d
        Fd := [g : g in F | TotalDegree(g) eq d];
        F := [g : g in F | TotalDegree(g) ne d];

        // run through all critical pairs of degree d
        if not IsEmpty(Bd) then
            // printf "Critical pairs of degree %o exist.\n", d;

            // create the set of all polynomials Ld corresponding to the critical pairs
            Ld := {};
            for p in Bd do
                LT1 := LeadingTerm(G[p[1]]);
                LT2 := LeadingTerm(G[p[2]]);
                Include(~Ld, [(LCM(LT1, LT2) div LT1), G[p[1]]]);
                Include(~Ld, [(LCM(LT1, LT2) div LT2), G[p[2]]]);
                df4 := Max(df4, TotalDegree((LCM(LT1, LT2) div LT1)* G[p[1]]));
                df4 := Max(df4, TotalDegree((LCM(LT1, LT2) div LT2)* G[p[2]]));
            end for;

            nocp := nocp + (#Ld div 2);
            // print "Number of critical pairs considered so far:", nocp;

            // Apply the reduction function FR which is specific to F4
            // measure the total duration of FR
            // and store the intermediate sets Fplusd and Fd
            // print "Reduce (FR) the critical pairs";
            FRtime := Cputime();
            SPFplusd[d], SPFred[d], EFtimesum, nrows, densty := FR(Ld, G, SPFplusd, SPFred, EFtimesum);
            FRtimesum := FRtimesum + Cputime(FRtime);
            Fplusd := SPFplusd[d];

            nrowsmax := Max(nrows, nrowsmax);
            if nrows eq nrowsmax then denstynrowsmax := densty;
            end if;
```

```
            // reduce the new "s-polynomials" created by subroutine FR w.r.t. G
            // print "Reduce (NF) the new found s-polynomials created by FR w.r.t. G";
            redfr := redfr + #[h : h in Fplusd | NormalForm(h,G) eq 0];
            // print "The total number of redundant polys created by FR so far:", redfr;
            Fplusd := [NormalForm(h, G): h in Fplusd | NormalForm(h,G) ne 0];

            printf "Number of polynomials with new leading terms found  : %o \n", #Fplusd;
            // print "Update the critical pairs (Gebauer and Moeller) ";

            // Remove all polynomials equal to zero from Fplus
            Fplusd := [f : f in Fplusd | f ne 0];

            // Grab the largest totaldegree of the critical pairs
            if not IsEmpty(Fplusd) then
                df4cp := Max(d, df4cp);
            end if;

            for h in Fplusd do
                Append(~G, h);
                UpdateGMtime := Cputime();
                B := UpdateGM(B, G);
                // Or without Gebauer and Moeller Installation
                // B := UpdatePairs(B,G);
                UpdateGMtimesum := UpdateGMtimesum + Cputime(UpdateGMtime);
            end for;

        end if;

        // run through all original elements of degree d
        // printf "%o original polynomials of degree %o exist\n", #Fd, d;
        while not IsEmpty(Fd) do

            h := Fd[1];
            h := NormalForm(h, G);
            df4 := Max(df4, TotalDegree(h));

            if h ne 0 then
                Append(~G, h);
                B := UpdateGM(B, G);
                // B := UpdatePairs(B,G);
            end if;
            Remove(~Fd, 1);
        end while;

        print "Size of the intermediate basis                    :", #G;
        print "Current df4                                       :", df4;
        // print "Intermediate basis a Groebner Basis                :", IsGroebner(G);

    end while;

    mtotdeg := Max([TotalDegree(g) : g in G]);

    printf "\n*** Summary ***\n";
    print "Total time spent on EchelonForm                  :", EFtimesum;
    print "Total time                                       :", Cputime(totaltime);
    print "Number of critical pairs considered              :", nocp;
    print "Largest number of rows in the reduction matrix   :", nrowsmax;
    print "Density of corresponding matrix                  :", denstynrowsmax;
    print "Highest degree of a contributing critical pair   :", df4cp;
    print "Highest degree of an S-polynomial (fictitious)   :", df4;
    print "Highest degree of a Groebner Basis element       :", mtotdeg;

    return G, df4cp, nocp, nrowsmax, df4, denstynrowsmax;
end function;

// Create the alias F4 for the function ImpHomF4GM
F4 := function(F)
    G, df4cp, nocp, nrowsmax, df4, denstynrowsmax := ImpHomF4GM(F);
    return G, df4cp, nocp, nrowsmax, df4, denstynrowsmax;
end function;
```

A.6. Algorithm F5

```
/*
Implementation of F5.
The algorithm follows the structure of the Homogeneous Buchberger Algorithm as
explained in Faugre's article "A new efficient algorithm for computing Grbner
Bases without reduction to zero (F5)" from 2002.
Latest modifications: 31-8-2004.
```

```
Let P be the polynomial ring K[x_1, ..., x_n] and F = {f_1, ..., f_m} a basis of the ideal I.
During the algorithm, polynomials p are stored as so called rules r.
These rules are implemented as lists in Magma as follows:
    r = [* t, i, p *], where
    t is a term,
    i the index of the i-th canonical basisvector F_i of the module P^m,
    p the polynomal,
    such that S(r) = t*F_i = v_1(p).
Furthermore, (truncated) Groebner bases and (lists of) lists of rules are stored as sequences in Magma, like [ r_1, ..., r_l ]

(or [ [ r_1, ..., r_l ] ] ).
*/

// Bubblesort a sequence
bubblesort := function(P)
    changes := 1;

    if #P eq 1 then
        return P;
    end if;

    while changes eq 1 do   // If the for-loop doesn't change anything, stop.
        changes := 0;       // Reset the change indicator.

        for j in [#P..2 by -1] do
            if P[j] gt P[j-1] then    // If P[j] > P[j-1] then swap.
                changes := 1;       // Changes are made.
                Pjtemp := P[j];
                P[j] := P[j-1];
                P[j-1] := Pjtemp;
            end if;
        end for;
    end while;
    return P;
end function;

// Bubblesort a seq of rules by increasing Signature.
signaturebubblesort := function(P)
    changes := 1;

    P := [p : p in P | not IsEmpty(p)];

    if #P eq 1 then
        return P;
    end if;

    while changes eq 1 do
        changes := 0;
        for j in [#P..2 by -1] do
            if P[j, 1] lt P[j-1, 1] then    // If P[j,1] < P[j-1,1] then swap.
                changes := 1;
                Pjtemp := P[j];
                P[j] := P[j-1];
                P[j-1] := Pjtemp;
            end if;
        end for;
    end while;

    changes := 1;
    while changes eq 1 do
        changes := 0;
        for j in [#P..2 by -1] do
            if P[j, 2] gt P[j-1, 2] then    // If P[j,2] > P[j-1,2] then swap.
                changes := 1;
                Pjtemp := P[j];
                P[j] := P[j-1];
                P[j-1] := Pjtemp;
            end if;
        end for;
    end while;

    // the following for-loop removes double entries in a sequence of rules
    for j in [#P..2 by -1] do
    if P[j, 1] eq P[j-1, 1] and
    P[j, 2] eq P[j-1, 2] and
    P[j, 3] eq P[j-1, 3] then
        Remove(~P, j);
    end if;
    end for;
    return P;
end function;

// Bubblesort a seq of lists P, ordered on the totaldegree of the k'th element.
```

```
degreebubblesort := function(P, k)
    changes := 1;

    P := [p : p in P | not IsEmpty(p)];
    if #P eq 1 then
        return P;
    end if;
    while changes eq 1 do
        changes := 0;

        for j in [#P..2 by -1] do
            if TotalDegree(P[j, k]) lt TotalDegree(P[j-1, k]) then
                changes := 1;
                Pjtemp := P[j];
                P[j] := P[j-1];
                P[j-1] := Pjtemp;
            end if;
        end for;
    end while;
    return P;
end function;

Rewritten := function(u, r_k, r, i)
    ri := r_k[2];

    for j in [1..#r[i]] do
        t_j := r[i][j][1];
        if (not t_j in IntegerRing())
    and ri eq r[i][j][2]    // to test if we are comparing rules with the same F_i
    and IsDivisibleBy(u*r_k[1], t_j)
    then
            return [* u*r_k[1] div t_j, r[i][j] *];
        end if;
    end for;

    return [* u, r_k *];
end function;

RewrittenQ := function(u, r_k, r, i)

    test := Rewritten(u, r_k, r, i);
    r_l := test[2];
    IsRewrittenQ := (r_l[1] ne r_k[1])
        or (r_l[2] ne r_k[2])
        or not IsDivisibleBy(r_l[3], r_k[3])
        or TotalDegree(r_l[3]) ne TotalDegree(r_k[3]);

    return IsRewrittenQ;
end function;

CritPair := procedure(~P, r_1, r_2, i, phi)
    // Apply the F5 criterion to every new critical pair
    // print "Start procedure CritPair.";

    // Compare two signatures and normalize them like in Definition 2
    // The rules are ordered on Signature, largest Signature is called r_1
    if (r_1[2] gt r_2[2]) then      // F_k > F_l if k < l
        r_temp := r_2;
        r_2 := r_1;
        r_1 := r_temp;
    end if;
    if (r_1[2] eq r_2[2]) then
        if (r_1[1] lt r_2[1]) then
            r_temp := r_2;
            r_2 := r_1;
            r_1 := r_temp;
        end if;
    end if;

    // If the largest Signature in the possible critical pair is smaller
    // than F_i w.r.t. the module term ordering, we reject the pair
    if r_1[2] gt i then
        // print "Exception: r_1[2] > i.";
        // print "";
            return ;
    end if;

    p := [r_1[3], r_2[3]];
    // Set t equal to the LCM of the critical pair
    t := LCM(LeadingMonomial(p[1]), LeadingMonomial(p[2]));

    u := [t div LeadingTerm(p[1]), t div LeadingTerm(p[2])];
```

```
    // Apply the F5 criterion
    // Check if the new Signature does not reduce w.r.t. the previous Grbner Basis G[i+1]
    if NormalForm(u[1]*r_1[1], phi) ne u[1]*r_1[1] then
            return ;
    end if;

    if r_2[2] eq i and NormalForm(u[2]*r_2[1], phi) ne u[2]*r_2[1] then
            return ;
    end if;

    // Add the new critical pair to P
    P := P cat [[* t, u[1], r_1, u[2], r_2 *]];

end procedure;

Spol := procedure(~F_d, P_d, ~N, ~r, i)
    // Subroutine Spol calculates the S-polynomials of P_d if the critical pair can not be rewritten
    // Spol returns the new set F_d

    for l in [1..#P_d] do
    // Run through all critical pairs of degree d

    // The pair has elements (u_l, r_il) and (v_l, r_jl)
        u_l  := P_d[l][2];
        r_il := P_d[l][3];
        v_l  := P_d[l][4];
        r_jl := P_d[l][5];

    // printf "Candidate Spol=(%o, %o, %o)\n", u_l*r_il[1], r_il[2], u_l*r_il[3] - v_l*r_jl[3];

    // Test if the elements in the critical pair can not be rewritten by previous rules in r[i]
        if (not RewrittenQ(u_l, r_il, r, i))
        and (not RewrittenQ(v_l, r_jl, r, i)) then
            N := N + 1;
        // Create the rule corresponding to the new S-polynomial
            r_N := [* u_l*r_il[1] ,
                      r_il[2],
                      u_l*r_il[3] - v_l*r_jl[3] *];

        // We add the rules to r after the ReductionF5 procedure in the main-loop and not in Spol

        // The list-to-be-reduced is updated with the rule corresponding to the new S-polynomial
            F_d := F_d cat [r_N];
        // printf "Add s-polynomial.\nr_N: (%o, %o, %o).\n\n", r_N[1], r_N[2], r_N[3] ;
        else
            // printf "Reject Critical Pair nr %o.\n\n", l;
        end if;

    end for;

    // Sort F_d by increasing Signature
    F_d := signaturebubblesort(F_d);

end procedure;

IsReducible := function(r_i0, G_iUDone, i, phi, r)
    // Test if r_i0 is reducible with respect to G_iUDone
    // Again, i is the current step in the for-loop of IncrementalF5

    // printf "Enter IsReducible for r_i0 = (%o, %o, %o)\n", r_i0[1], r_i0[2], r_i0[3];
    for j in [1..#G_iUDone] do

    // Set r_j equal to the j-th element of G_iUDone
    r_j := G_iUDone[j];

        if IsDivisibleBy(LeadingMonomial(r_i0[3]), LeadingMonomial(r_j[3]))
        then
        // (a) Polynomial r_i0 satisfies the usual divisibility test for possible reductor r_j

            u := LeadingTerm(r_i0[3]) div LeadingTerm(r_j[3]);
            t_j := r_j[1];

        // (b) Test if the new rule is normalized
            if NormalForm(u * t_j, phi) eq u*t_j
        // (c) Test if we can use previous computations
            and not RewrittenQ(u, r_j, r, i)
        // (d) Remove identical rules
            and (u*t_j ne r_i0[1] or r_j[2] ne r_i0[2])
            then
        // All four criteria (a, b, c, d) in IsReducible are satisfied

        // printf "return G_iUDone[j]: (%o, %o, %o)\n", G_iUDone[j][1], G_iUDone[j][2], G_iUDone[j][3];
                return G_iUDone[j];
```

```
            end if;
        end if;
    end for;

    // print "r_i0 is irreducible";
    return [* *];
end function;


TopReduction := procedure(~h_1, ~ToDo_1, r_k0, G_iUDone, i, phi, ~r, ~N)
    // TopReduction reduces r_k0 with respect to G_iUDone and updates the Signature
    // The new Signature depends on the reductor

    // If we get a zero S-polynomial, print a warning
    if r_k0[3] eq 0 then
        // print "WARNING, the system is not a regular sequence: reduction to zero occurred";
        h_1 := [* *];
        ToDo_1 := [ ];
    return ;
    end if;

    // Test if r_k0 is reducible with respect to G_iUDone and store the reductor
    r_k1 := IsReducible(r_k0, G_iUDone, i, phi, r);

    // If r_k0 is irreducible then the reductor is empty and we return r_k0 with leading coefficient 1
    if IsEmpty(r_k1) then
        h_1 := [* r_k0[1], r_k0[2], r_k0[3] div LeadingCoefficient(r_k0[3]) *];
        ToDo_1 := [];
        return ;
    else
    // A reductor exists in this case
        u := LeadingTerm(r_k0[3]) div LeadingTerm(r_k1[3]);

        // Compare the Signatures S(r_k0) and S(r_k1).
        if (r_k1[2] gt r_k0[2]) or ((r_k1[2] eq r_k0[2]) and u*r_k1[1] lt r_k0[1])
        then
        // In this case the signature of the reductor is smaller
        // print "The signature of u*r_k1 is smaller ";

        // Compute the reduction
        r_k0[3] := r_k0[3]-u*r_k1[3];

            h_1 := [* *];

            if r_k0[3] eq 0 then
        // print "Reductor r_k1 cancels r_k0, so return an empty set";
                ToDo_1 := [];
                return;
            else
        // Return the new rule to test whether it is again reducible
        // printf "return r_k0=(%o, %o, %o)\n", r_k0[1], r_k0[2], r_k0[3];
                ToDo_1 := [r_k0];
                return;
            end if;

        else
        // The Signature of the r_k0 is smaller than that of the reductor
        // This means that we use the Signature of the reductor
        // print "The Signature of r_k0 is smaller ";

            N := N+1;
            r_N := [* u*r_k1[1], r_k1[2], u*r_k1[3]-r_k0[3] *];

        // In the description of the algorithm, the author adds the rules at this point
        // The rules are added after the Reduction step in the main loop of AlgorithmF5
        // This means we do not add the rules during the reduction

        // Return the new rule r_N
        // The article suggests to return r_k0 also but it will have the same reductor
        // printf "Return r_N=(%o, %o, %o)\n", r_N[1], r_N[2], r_N[3];
            // print "";
            h_1 := [* *];
        ToDo_1 := [r_N];    // removed r_k0 from this list.

            return ;
        end if;
    end if;
end procedure;

ReductionF5 := procedure(~R_d, ToDo, G_i, i, phi, ~r, ~N);
    // Reduce the polynomials in ToDo with respect to the polynomials in G_i
    // Return the set of rules corresponding to the reduced polynomials, R_d
```

```
    // The set phi corresponds to the set phi_iplus1 in the subroutine AlgorithmF5
    // The integer i indicates the current step of the for-loop of IncrementalF5

    while not IsEmpty(ToDo) do
    // Sort the rules in ToDo on increasing Signature
        ToDo := signaturebubblesort(ToDo);

    // Pick the rule with the smallest Signature from ToDo
        h := ToDo[1];
        ToDo := Remove(ToDo, 1);        // set ToDo := ToDo \ {h}

    // Test if the algorithm works if we reduce very rigorously
    // Modify phi temporarily, the result is that F5 usually works for larger systems than 3 variables
    // If phi is not a Grbner Basis then the Normal Form is not unique
    phi := phi cat [r_j[3] : r_j in (G_i cat R_d)];

    // Apply subroutine TopReduction, it returns an irreducible h_1 and a sequence of reducible ToDo_1
    h_1 := [* *];
        ToDo_1 := [];
        TopReduction(~h_1, ~ToDo_1, [*h[1], h[2], NormalForm(h[3], phi) *], G_i cat R_d, i, phi, ~r, ~N);

        if not IsEmpty(h_1) then
            R_d := R_d cat [ h_1 ];
        end if;

        ToDo := ToDo cat ToDo_1;
    end while;
end procedure;

AlgorithmF5 := procedure(i, f, ~G, ~r, ~N, ~nosp, ~dmax, ~df5)
    // Compute a Grbner Basis of the polynomials in G[i+1] joined with f=f_i
    // The integer i indicates the current step of the for-loop of IncrementalF5

    // Initiate the first rule for polynomial f_i of the original basis
    r_i := [* 1, i, f *];
    r[i] := Append(r[i], r_i );

    // Call the previous Grbner Basis phi_iplus1 to use in the calculation of Normal Forms
    phi_iplus1 := [G[i+1][j][3] : j in [1..#G[i+1]]];

    // Append the rule corresponding to the new polynomial to the previous Grbner Basis
    G[i] := Append(G[i+1], r_i);

    // Calculate the new critical pairs for the polynomial f_i
    // The new criterion of F5 is applied in the subroutine CritPair
    P := [];
    for j in [1..#G[i+1]] do
        CritPair(~P, r_i, G[i+1][j], i, phi_iplus1);
    end for;

    // Sort the critical pairs by increasing totaldegree
    P := degreebubblesort(P,1);

    print "Critical pairs after applying the F5 criterion         :", #P;

    while not IsEmpty(P) do
    // Set d, the smallest totaldegree in P
        d :=  TotalDegree(P[1][1]);

    // P_d is the subset of P with lowest totaldegree d
        P_d := [p : p in P | TotalDegree(p[1]) eq d];

        printf "Select the %3o critical pair(s) with degree d          : %o\n", #P_d, d;

    // Set P:=P\P_d (remove #P_d times the first element of P)
        for j in [1..#P_d] do
            P := Remove(P, 1);
        end for;

    // Create the set of S-polynomials F corresponding to the critical pairs in P_d
        F_d := [];
        Spol(~F_d, P_d, ~N, ~r, i);

    nosp := nosp + #F_d;

    // Reduce the new S-polynomials with respect to the current intermediate basis G[i]
        R_d := [];
        ReductionF5(~R_d, F_d, G[i], i, phi_iplus1, ~r, ~N);

    // The following line is to be exactly like the example in the article
    // This sorting is not mentioned in the pseudocode of F5
    R_d := Reverse(signaturebubblesort(R_d));
```

```
    if not IsEmpty(R_d) then
        print "Number of new elements in intermediate basis          :", #R_d;
        df5 := Max(df5, d);
        print "Current largest degree of a contributing critical pair :", df5;
    end if;

    // Add the new rules corresponding to the reduced polynomials
    for newrule in R_d do
           r[i] := Append(r[i] , newrule);    // Add rules
    end for;

        for j in [1..#R_d] do

        dmax := Max(TotalDegree(R_d[j,3]), dmax);

        // Create the critical pairs rising from the reduced S-polynomials
            for p in [1..#G[i]] do
                CritPair(~P, R_d[j], G[i][p], i, phi_iplus1);
            end for;

        // Append the reduced S-polynomials to the current intermediate basis G[i]
            G[i] := G[i] cat [ R_d[j] ];

        end for;

    /*  print "The new G[i] after adding the s-polynomials.";
        for j in [1..#G[i]] do
            printf "Elt. %o: (%o, %o, %o)\n", j, G[i][j][1], G[i][j][2], G[i][j][3];
        end for;
        print "";
    */
        // Sort the critical pairs by increasing totaldegree
        P := degreebubblesort(P,1);
        print "Number of new critical pairs                          :", #P;
    end while;
end procedure;

IncrementalF5 := function(F)
    print "********** F5 **************";
    // print "F=",F;

    N := #F;
    dmax := 0;
    df5 := 0;
    // Initiate the set G of intermediate Grbner Bases
    G := [[] : j in [1..N]];
    // Reset the simplification rules
    r := [ [ ] : j in [1..N] ];
    r[N] := [ [* 1, N, F[N] *] ];
    // The intermediate Grbner Basis elements are stored as rules
    G[N] := r[N];

    // Count the total number of s-polynomials created
    nosp := 0;

    for i in [N-1..1 by -1] do
        print "Start procedure AlgorithmF5, polynomials to go i       :",i ;

    AlgorithmF5(i, F[i], ~G, ~r, ~N, ~nosp, ~dmax, ~df5);

    print "Number of S-polys created                              :", nosp;
    print "Current maximal degree of S-polynomial                 :", dmax;
    print "";

    end for;

    gbasis := [G[1][j][3] : j in [1..#G[1]]];
    mtotdeg := Max([TotalDegree(g) : g in gbasis]);

    print "*** Summary ***";
    print "Total number of S-polys created                        :", nosp;
    print "Maximal degree of S-polynomial                         :", dmax;
    print "Maximal degree of a contributing critical pair         :", df5;
    print "Maximal degree in basis                                :", mtotdeg;
    print "Number of basis elements                               :", #gbasis;
    isgb := IsGroebner(gbasis);
    print "Is G a Groebner Basis                                  :", isgb;

    return gbasis, isgb, nosp, dmax, df5;
end function;

// Create an alias F5 for the official name IncrementalF5 as it is written in the article.
F5 := function(F)
```

```
    gbasis, IsGB, nosp, dmax, df5 := IncrementalF5(F);
    return gbasis, IsGB, nosp, dmax, df5;
end function;
```

A.7. XL

```
/*
Start 21-6-2004, the XL algorithm.
XL(F,D) tries to find univariate polynomials up to degree D.
Process data, like the solutions, reduced intermediate basis, d_xl and original equations with the partial
solution substituted, is returned.
*/

// Set the outputprecision of the print statement to a maximum of 5 significant digits
AssertAttribute(FldPr, "OutputPrecision", 5);

XL := function(F, D)
    printf "********** XL, D=%o **********\n", D;

    // Initialization
    F := [f : f in F | f ne 0];
    sd := Min([TotalDegree(f) : f in F]);
    // Grab the polynomial ring in which the elements of F lie
    P:=Parent(F[1]);
    r:=Rank(P);
    // Grab the current memory usage
    m1 := GetMemoryUsage();

    // Test if D is assigned a correct value
    if D lt sd then
        print "Choose a larger D";
        return [], F, false, #F, #F, 0, F;
    end if;

    F_D := [[] : i in [1..#F]];

    // This for-loop generates all monomials up to degree D-d for a degree d element of f
    // and includes all monomial multiplications of the form x^\alpha f in F_D[f]
    for f in [1..#F] do
        d := TotalDegree(F[f]);

        // Similar to F4, we use the function addwithcarry to create all monomials up to degree D-d
        M := [P ! 1];
        pos := 1;
        maxvector := [D-d : i in [1..r]];
        tempvector := [0 : i in [1..r]];
        while tempvector ne maxvector and pos le r do
            tempvector, pos := addwithcarry(tempvector, maxvector, pos);
            if &+tempvector le D-d then
                Include(~M, &*[(P.i)^tempvector[i] : i in [1..r]]);
            end if;
        end while;

        for m in M do
            Include(~F_D[f], F[f]*m);
        end for;
    end for;

    // All polynomials of the form x^alpha f_i up to degree D are put together in one sequence F_XL
    F_XL := &cat[F_D[f] : f in [1..#F]];

    // Create the ordered support of F_XL
    sup := Setseq(&join{Seqset(Monomials(i)) : i in F_XL});
    sup := ChangeUniverse(sup, ChangeOrder(P, "lex"));
    sup := Reverse(Sort(sup));
    // Store the number of univariate monomials in the support
    univsup := [s : s in sup | IsUnivariate(s)];
    noum := #univsup;

    // Create the matrix corresponding to set of polynomials F_XL
    Fmatrix := Matrix(BaseRing(P), [polytovector(i, sup) : i in F_XL]);

    EFtime := Cputime();
    Fmatrix := EchelonForm(Fmatrix);

    // If the matrix fits on the screen, print it
    if #sup lt 25 then
        print "Ordered support:";
        print sup;
        print "Reduced matrix:";
```

```
        print Fmatrix;
    end if;

    time_echform := Cputime(EFtime);
    print  "Time of EchelonForm(Fmatrix) (s)   :", time_echform;

    // Convert rows to polynomials
    Fmatrix := RowSequence(Fmatrix);
    nrows := #Fmatrix;
    Fred := [vectortopoly(Fmatrix[i], sup) : i in [1..nrows]];
    Fred := [f : f in Fred | f ne 0];

    // Create the set of univariate polynomiaals in Fred
    UP := [f : f in Fred | IsUnivariate(f)];

    // Create the set of univariate polynomials with a single root from UP
    R := [f : f in UP | #Roots(UnivariatePolynomial(f)) eq 1];

    // Collect the (intermediate) solutions from R, represented in a substitution sequence
    subst := {};
    for r in R do
        a, b, var := IsUnivariate(r);
        root := Roots(UnivariatePolynomial(r))[1,1];
        Include(~subst, [Integers() | var, root]);
    end for;
    subst := Setseq(subst);

    mem := (GetMemoryUsage() - m1) div 1000;
    isgb := IsGroebner(Fred);

    print "";
    print "*** Summary ***";
    print  "Memory size of process (kb)    :", mem;
    printf "Size of XL matrix          : %o rows and %3o columns\n", #F_XL, #sup;
    printf "Size of row reduced XL matrix      : %o rows and %3o columns\n", #Fred, #sup;
    print  "Number of univariate polynomials   :", #UP;
    print  "Number of single root polynomials  :", #R;
    print  "Number of univariate monomials     :", noum;
    print  "Number of variables solved         :", #subst;
    print  "Is the system a Groebner Basis     :", isgb;
    print  "Substitution sequence [[var,root]] :";
    print  subst;


    // Substitute the solved variables into the original system
    for f in [1..#F] do
        for s in subst do
            F[f] := Evaluate(F[f], s[1], s[2]);
        end for;
    end for;
    F := [f : f in F | f ne 0];

    return subst, Fred, isgb, #F_XL, #Fred, #sup, F, time_echform;
end function;
```

APPENDIX B

Data corresponding to Examples 5.2 and 5.11

The set F that is the result of the HFE encryption in Example 5.2 represented as Magma code.

```
P<[x]>:=PolynomialRing(GF(2),7,"lex");
F:=
[
    x[1]*x[2] + x[1]*x[5] + x[1]*x[6] + x[2]^2 + x[2]*x[4] + x[2]*x[5] +
        x[2]*x[6] + x[3]^2 + x[3]*x[4] + x[4]^2 + x[4]*x[5] + x[5]^2 + x[6]^2 +
        x[6]*x[7] + x[6] + x[7],
    x[1]^2 + x[1]*x[2] + x[1]*x[3] + x[1]*x[4] + x[1]*x[5] + x[1]*x[7] + x[1] +
        x[2]*x[3] + x[2]*x[5] + x[2]*x[6] + x[2]*x[7] + x[3]*x[5] + x[3]*x[7] +
        x[3] + x[4]^2 + x[4]*x[5] + x[4]*x[7] + x[5]*x[6] + x[5]*x[7] + x[5] +
        x[6]*x[7] + 1,
    x[1]*x[2] + x[1]*x[3] + x[1]*x[4] + x[1]*x[6] + x[1] + x[2]*x[4] + x[2]*x[7]
        + x[3]^2 + x[3]*x[6] + x[3]*x[7] + x[3] + x[4]^2 + x[4]*x[5] + x[4]*x[7]
        + x[4] + x[5]*x[7] + x[5] + x[6]*x[7] + x[6] + x[7],
    x[1]^2 + x[1]*x[3] + x[1]*x[4] + x[1]*x[5] + x[1]*x[6] + x[1]*x[7] + x[1] +
        x[2]^2 + x[2]*x[5] + x[2]*x[7] + x[2] + x[3]*x[4] + x[3]*x[5] + x[3] +
        x[4]*x[5] + x[4]*x[7] + x[4] + x[5] + x[6]*x[7] + x[7]^2 + x[7] + 1,
    x[1]^2 + x[1]*x[5] + x[2]*x[3] + x[2]*x[4] + x[2] + x[3]*x[4] + x[3]*x[5] +
        x[3] + x[4]*x[5] + x[4]*x[6] + x[4]*x[7] + x[5]^2 + x[5]*x[6] + x[5] +
        x[6]*x[7] + x[6] + x[7]^2 + x[7],
    x[1]^2 + x[1]*x[2] + x[1] + x[2]*x[5] + x[2] + x[4]*x[6] + x[4] + x[5]^2 +
        x[5]*x[6] + x[5]*x[7] + x[5] + x[6]^2 + x[6]*x[7] + x[6] + x[7] + 1,
    x[1]*x[3] + x[1]*x[4] + x[1]*x[5] + x[1]*x[7] + x[2]*x[3] + x[2]*x[5] +
        x[2]*x[6] + x[2]*x[7] + x[2] + x[3]*x[4] + x[4]^2 + x[4]*x[5] + x[5]^2 +
        x[5]*x[7] + x[5] + x[6]^2 + x[6] + x[7]^2,
    x[1]^2 + x[1],
    x[2]^2 + x[2],
    x[3]^2 + x[3],
    x[4]^2 + x[4],
    x[5]^2 + x[5],
    x[6]^2 + x[6],
    x[7]^2 + x[7]
];
```

Applying XL incrementally to the following set leads to a lower d_{min} and d_{XL} than predicted by the analysis based on Hilbert series. This set corresponds to Example 5.11.

```
P<[x]> := PolynomialRing(GF(2), 9, "lex");
F:=
[
    x[1]^2 + x[1]*x[2] + x[1]*x[4] + x[1]*x[5] + x[1] + x[2]*x[5] + x[2]*x[6] +
        x[2]*x[8] + x[2]*x[9] + x[2] + x[3]^2 + x[3]*x[6] + x[3]*x[7] +
        x[3]*x[8] + x[3]*x[9] + x[3] + x[4]^2 + x[4]*x[5] + x[4]*x[7] +
        x[4]*x[8] + x[4]*x[9] + x[5]^2 + x[5]*x[8] + x[5]*x[9] + x[5] + x[6]^2 +
        x[6]*x[9] + x[6] + x[8]*x[9] + x[9]^2 + x[9] + 1,
```

```
    x[1]^2 + x[1]*x[2] + x[1]*x[3] + x[1]*x[4] + x[1]*x[5] + x[1]*x[6] +
        x[1]*x[8] + x[1]*x[9] + x[2]*x[3] + x[2]*x[4] + x[2]*x[9] + x[3]^2 +
        x[3]*x[4] + x[3]*x[7] + x[3]*x[9] + x[4]^2 + x[4]*x[6] + x[4]*x[9] +
        x[4] + x[5]^2 + x[5]*x[7] + x[5]*x[9] + x[6]^2 + x[6]*x[7] + x[6]*x[8] +
        x[6]*x[9] + x[6] + x[8]*x[9] + x[8] + x[9]^2,
    x[1]*x[2] + x[1]*x[4] + x[1]*x[5] + x[1]*x[6] + x[2]^2 + x[2]*x[3] +
        x[2]*x[5] + x[2]*x[7] + x[2]*x[9] + x[2] + x[3]*x[5] + x[3]*x[6] +
        x[3]*x[8] + x[3]*x[9] + x[4]^2 + x[4]*x[5] + x[4]*x[7] + x[4]*x[9] +
        x[4] + x[5]*x[8] + x[6]^2 + x[6]*x[8] + x[6]*x[9] + x[6] + x[7]*x[9] +
        x[7] + x[9] + 1,
    x[1]*x[2] + x[1]*x[4] + x[1]*x[6] + x[1]*x[7] + x[1]*x[9] + x[2]*x[3] +
        x[2]*x[4] + x[2]*x[6] + x[2]*x[7] + x[2]*x[9] + x[2] + x[3]*x[5] +
        x[3]*x[7] + x[3]*x[8] + x[4]*x[7] + x[4] + x[5]*x[7] + x[5]*x[8] + x[5]
        + x[6]^2 + x[6]*x[8] + x[6]*x[9] + x[6] + x[7]^2 + x[7]*x[8] + x[7]*x[9]
        + x[8]^2 + x[8]*x[9] + x[8] + x[9],
    x[1]^2 + x[1]*x[6] + x[1]*x[7] + x[1]*x[8] + x[2]*x[4] + x[2]*x[5] +
        x[2]*x[9] + x[3]^2 + x[3]*x[4] + x[3]*x[5] + x[3] + x[4]*x[7] +
        x[5]*x[7] + x[5]*x[9] + x[5] + x[6]*x[7] + x[6]*x[8] + x[7]*x[8] +
        x[7]*x[9] + x[7] + x[8]^2 + x[8]*x[9] + x[8] + x[9]^2,
    x[1]^2 + x[1]*x[5] + x[1]*x[8] + x[1]*x[9] + x[1] + x[2]*x[4] + x[2]*x[9] +
        x[2] + x[3]*x[4] + x[3]*x[6] + x[3]*x[8] + x[3]*x[9] + x[3] + x[4]*x[7]
        + x[4]*x[8] + x[4]*x[9] + x[5]*x[6] + x[5]*x[8] + x[5]*x[9] + x[6]*x[8]
        + x[6]*x[9] + x[6] + x[7]^2 + x[7]*x[8] + x[7]*x[9] + x[8]*x[9],
    x[1]^2 + x[1]*x[2] + x[1]*x[4] + x[1]*x[5] + x[1]*x[6] + x[1]*x[7] +
        x[1]*x[9] + x[1] + x[2]*x[5] + x[2]*x[6] + x[2]*x[7] + x[2]*x[8] +
        x[3]*x[4] + x[3]*x[5] + x[3]*x[7] + x[3] + x[4]^2 + x[4]*x[5] +
        x[4]*x[6] + x[4]*x[8] + x[5] + x[6]*x[7] + x[6] + x[7]^2 + x[9]^2 +
        x[9],
    x[1]^2 + x[1]*x[3] + x[1]*x[4] + x[1]*x[5] + x[1]*x[6] + x[2]*x[3] +
        x[2]*x[4] + x[2]*x[5] + x[2]*x[9] + x[3]*x[8] + x[3]*x[9] + x[3] +
        x[4]*x[5] + x[4]*x[6] + x[4]*x[8] + x[5]^2 + x[5]*x[7] + x[5] + x[6]^2 +
        x[6]*x[8] + x[7]*x[8] + x[7]*x[9] + x[7] + x[8]^2 + x[8]*x[9] + x[9]^2 +
        x[9] + 1,
    x[1]*x[2] + x[1]*x[6] + x[1]*x[8] + x[2]^2 + x[2]*x[3] + x[2]*x[6] +
        x[2]*x[8] + x[2] + x[3]^2 + x[3]*x[5] + x[3]*x[7] + x[3]*x[8] +
        x[3]*x[9] + x[3] + x[4]^2 + x[4]*x[5] + x[4]*x[8] + x[5]^2 + x[5]*x[8] +
        x[5] + x[6]^2 + x[6]*x[7] + x[6]*x[8] + x[6]*x[9] + x[6] + x[7]^2 +
        x[7]*x[9] + x[8]^2 + x[8]*x[9] + x[9]^2,
    x[1]^2 + x[1],
    x[2]^2 + x[2],
    x[3]^2 + x[3],
    x[4]^2 + x[4],
    x[5]^2 + x[5],
    x[6]^2 + x[6],
    x[7]^2 + x[7],
    x[8]^2 + x[8],
    x[9]^2 + x[9]
];
```

Bibliography

[Arm02] F. Armknecht, *A Linearization Attack on the Bluetooth Key Stream Generator*, IACR eprint server, *http://www.iacr.org*, December 2002.

[AT96] G. Attardi and C. Traverso, *Strategy-Accurate Parallel Buchberger Algorithms*, Journal of Symbolic Computation **21** (1996), 411–425.

[BCP97] W. Bosma, J. Cannon, and C. Playoust, *The Magma Algebra System I: The User Language*, Journal of Symbolic Computation **24** (1997), 235–265.

[BMMT94] E. Becker, M.G. Marinari, T. Mora, and C. Traverso, *The shape of the Shape Lemma*, Proceedings of the International Conference on Symbolic and Algebraic Computation, ACM Press, 1994, pp. 129–133.

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

[Buc85] ———, *Multidimensional Systems Theory*, ch. Gröner bases: an algorithmic method in polynomial ideal theory, pp. 184–232, D. Reidel Publishing Company, 1985.

[BW93] T. Becker and V. Weispfenning, *Gröbner Bases, a Computational Approach to Commutative Algebra*, Graduate Texts in Mathematics, Springer, 1993.

[CKM97] S. Collart, M. Kalkbrener, and D. Mall, *Converting Bases with the Gröbner Walk*, Journal of Symbolic Computation **24** (1997), no. 3, 465–469.

[CKPS00] N. Courtois, A. Klimov, J. Patarin, and A. Shamir, *Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations*, Advances in Cryptology - Eurocrypt 2000 **1807** (2000), 392–407.

[CKR04] M. Caboara, M. Kreuzer, and L. Robbiano, *Efficiently Computing Minimal Sets of Critical Pairs*, Preprint submitted to Journal of Symbolic Computation (2004).

[CLO96] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties and Algorithms, Second Edition*, Springer-Verlag, 1996.

[CLO98] ———, *Using Algebraic Geometry*, Springer-Verlag, 1998.

[Coo71] S.A. Cook, *The Complexity of Theorem Proving Procedures*, Conference Record of Third Annual ACM Symposium on Theory of Computation (1971), no. 3-5, 151–158.

[Cou03] N. Courtois, *Higher Order Correlation Attacks, XL algorithm and Cryptanalysis of Toyocrypt*, Proceedings of the International Conference on Information Security and Cryptography, Lecture Notes in Computer Science, no. 2587, Springer, 2003, pp. 182–199.

[CP02] N. Courtois and T. Pieprzyk, *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations*, IACR eprint server, *http://www.iacr.org*, March 2002.

[CW90] D. Coppersmit and S. Winograd, *Matrix Multiplication via Arithmetic Programming*, Journal of Symbolic Computation **9** (1990), 251–280.

[Die04] C. Diem, *An analysis of the XL-algorithm*, Private communication, April 2004.

[DR99] J. Daemen and V. Rijmen, *AES Proposal: Rijndael (Version 2)*, NIST AES website, *http://csrc.nist.gov/encryption/aes*, 1999.

[FA04] J.-C. Faugère and G. Ars, *Comparison of XL and Gröbner basis algorithms over Finite Fields*, Tech. Report 5251, INRIA, July 2004.

[Fau99] J.-C. Faugère, *A New Efficient Algorithm for Computing Gröbner Bases (F4)*, Journal of Pure and Applied Algebra **139** (1999), 61–88.

[Fau02] ———, *A New Efficient Algorithm for Computing Gröbner Basis without Reduction to Zero (F5)*, Proceedings of the 2002 international symposium on Symbolic and algebraic computation, ACM Special Interest Group on Symbolic and Algebraic Manipulation, 2002.

[FGLM93] J.-C. Faugère, P. Gianni, D. Lazard, and T. Mora, *Efficient Computation of Zero-dimensional Gröbner Bases by Change of Ordering*, Journal of Symbolic Computation **16** (1993), 329–344.

[FJ03] J.-C. Faugère and A. Joux, *Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases*, Advances in Cryptology - Crypto 2003 **2729** (2003), 44–60.

[Fro85] R. Froberg, *An inequality for Hilbert series of graded algebras*, Math. Scand. **56** (1985), 117–144.

[GG99] J.v.z. Gathen and J. Gerhard, *Modern Computer Algebra*, Cambridge University Press, 1999.

[GJ79] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, New York, 1979.

[GM88] R. Gebauer and H.M. Möller, *On an Installation of Buchberger's Algorithm*, Journal of Symbolic Computation **6** (1988), 275–286.

[HPS93] J. Håstad, S. Philips, and S. Safra, *A Well-Characterized Approximation Problem*, Information Processing Letters (1993), no. 47:6, 301–305.

[Kob97] N. Koblitz, *Algebraic Aspects of Cryptography*, Algorithms and Computation in Mathematics, vol. 3, Springer, 1997.

[KR00] M. Kreuzer and L. Robbiano, *Computational Commutative Algebra 1*, Springer, 2000.

[KR04] ———, *Computational Commutative Algebra 2*, To Appear, 2004, draft May.

[KS99] A. Kipnis and A. Shamir, *Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization*, Advances in Cryptology - Crypto 1999 **1666** (1999), 19–30.

[Laz83] D. Lazard, *Gröbner Bases, Gaussian Elimination and Resolution of Systems of Algebraic Equations*, Proc. Eurocal '83 **162** (1983), 146–157.

[LRK79] J.K. Lenstra and A.H.G. Rinnooy Kan, *Computational Complexity of Discrete Optimization Problems*, Annals of Discrete Mathematics, vol. 4, North-Holland Publishing Company, 1979.

[MFS03] Bardet. M., J.-C. Faugère, and B. Salvy, *Complexity of Gröbner basis computation for Semi-regular Overdetermined sequences over $\mathbb{F}_2$ with solutions in $\mathbb{F}_2$*, Tech. Report 5049, INRIA, 2003.

[MI88] T. Matsumoto and H. Imai, *Public Quadratic Polynomial-Tuples for Efficient Signature-Verification and Message-Encryption*, Advances in Cryptology - Eurocrypt '88 **330** (1988), 419–453.

[Mis93] B. Mishra, *Algorithmic Algebra*, Texts and Monographs in Computer Science, Springer-Verlag, 1993.

[Moh01] T. Moh, *On the Method of "XL" and its Inefficiency to TTM*, IACR eprint server, *http://www.iacr.org*, June 2001.

[MR02a] S. Murphy and M.J.B. Robshaw, *Comments on the Security of the AES and the XSL Technique*, *http://www.isg.rhul.ac.uk/ sean/Xslbes8.ps*, September 2002.

[MR02b] ———, *Essential Algebraic Structure within the AES*, Advances in Cryptology - Crypto 2002 **2442** (2002), 1–16.

[Nat01] National Institute of Standards and Technology, *Advanced Encryption Standard*, FIPS 197, 26 November 2001.

[Pat96a] J. Patarin, *HFE first challenge*, *http://www.minrank.org/challenge1.txt*, 1996.

[Pat96b] ———, *Hidden Field Equations (HFE) and Isomorphism of Polynomials (IP): Two New Families of Asymmetric Algorithms*, Advances in Cryptology - Eurocrypt '96 **1070** (1996), 33–48.

[Sha49] C.E. Shannon, *Communication Theory of Secrecy Systems*, Bell System Technical Journal (1949), no. 28.

[SKI04] M. Sugita, M. Kawazoe, and H. Imai, *Relation between XL algorithm and Gröbner Bases Algorithms*, IACR eprint server, *http://www.iacr.org*, May 2004.

[Ste04] A. Steel, *Allan Steel's Gröbner Basis timings page*, *http://magma.maths.usyd.edu.au/users/allan/gb/*, May 2004.

[Tra97] C. Traverso, *Hilbert Functions and the Buchberger Algorithm*, Journal of Symbolic Computation **22** (1997), 355–376.

[Val96] G. Valla, *Six Lectures on Commutative Algebra*, Progress in Mathematics, vol. 166, ch. Problems and Results on Hilbert Polynomials of Graded Algebras, Birkhäuser Verlag, Basel, 1996.

Index