The use of d-truncated Gröbner bases in cryptanalysis of symmetric ciphers

Jens-Are Amundsen

Masteroppgave
Master i informasjonssikkerhet
30 ECTS
Avdeling for informatikk og medieteknikk
Høgskolen i Gjøvik, 2010

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

The use of d-truncated Gröbner bases in cryptanalysis of symmetric ciphers

Jens-Are Amundsen

2010/05/30

Abstract

Solving systems of multivariate polynomial equations is hard, even $\mathcal{NP}$-hard in the general case. The method of Göbner bases can be used to solve such systems, and thus has a running time complexity at least that of solving systems of polynomial equations. The running time complexity is often expressed as exponential in D, where D is the largest degree of a polynomial during computation. Even though the method of Gröbner bases belongs to the complexity class PSPACE for zero-dimensional ideals, it still implies a need for huge amounts of computer memory. Computational algebra systems utilizing Gröbner bases algorithms are well known to crash during computation due to a lack of computer memory.

In this thesis we investigate the use of d-truncated Gröbner bases over a Boolean ring, applied to systems of equations induced from symmetric ciphers. A Gröbner basis algorithm based on Buchbergers Homogeneous Algorithm is implemented, and we apply this algorithm on systems of equations induced from the ciphers LILI-128 and KASUMI. We show that we can solve a system of equations, induced from 3 rounds KASUMI, in over 9000 unknowns using 3-truncated Gröbner bases in both reasonable time and reasonable amounts of computer memory. For systems of equations induced from LILI-128, using 4-truncated Gröbner bases, we experienced an exponential increase in the number of output bits needed for finding a solution.

Sammendrag

Å løse systemer av ikke-lineære polynomer i flere variable er et vanskelig problem, og det generelle problemet tilhører klassen av $\mathcal{NP}$-komplette problemer. Gröbner baser kan benyttes for å løse slike ligningssystemer, og har dermed minst like høy tids kompleksitet. Tids kompleksiteten for konstruksjon av Gröbner baser er ofte gitt som eksponentiell i D, der D er graden til det største polynomet under eksekvering. Selv om konstruksjon av Gröbner baser tilhører kompleksitetsklassen PSPACE for 0-dimensjonale ideal, betyr det allikevel at det vil være behov for store mengder dataminne. Systemer for symbolsk beregning av algebraiske problemer som bruker Gröbner base metoder er kjent for å krasje på grunn av minne problemer.

I dette prosjektet undersøker vi bruk av d-begrensede Gröbner baser over boolske ringer, og anvender det på systemer av ikke-lineære polynomligninger som beskriver symmetriske krypto algoritmer. En Gröbner base algoritme, basert på Buchbergers Homogene algoritme, er implementert og vi anvender denne på systemer av ligninger fra krypto algoritmene LILI-128 og KASUMI. Vi viser at det er mulig å løse ligningssystemer fra 3 runder KASUMI med over 9000 variable på overkommelig tid og overkommelig bruk av dataminne. For ligningssystemer fra LILI-128, der 4-begrensede Gröbner baser er benyttet, fant vi en eksponentiell økning i antallet ut bits som behøves for å løse ligningssystemene.

Preface

Being a part time student, it has been hard to balance studies, daytime job and family, but I have enjoyed every second of it.

I would like to extend my sincerest gratitude to the *inventors of strong coffee*. Without Your help, my extracurricular activities would be non-existent and my life would be mundane and dull. I would also like to thank A.J.M. Segers [1] for setting the bar for master's theses, and Gregory V. Bard [2] for actually answering my emails. Last, but not least, I would like to thank my thesis supervisor Prof. Slobodan Petrovic for his patience and help, and especially for giving me this exiting project. I am sad to see it end.

Jens-Are Amundsen, 2010/05/30

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 Topic

Cryptology, from the Greek words kryptos - κρυπτόσ meaning "hidden" and logia - λόγια meaning "speech", in its various forms goes back thousand of years. It is not difficult to imagine early man leaving obfuscated instructions for finding good hunting sites or fishing sites. And it is equally easy to imagine the early cryptanalyst wanting to find these hunting sites by decoding the instructions. The Greeks and Spartans most probably had cryptographic protocols, and the Romans certainly had them. A well known example of the latter is the Caesar cipher, a substitution cipher known to have been used by Julius Caesar himself.

Cryptanalysis, a term coined by a US army cryptographer William F. Friedman in the 1920's, is the study of methods for reading encrypted information without the knowledge of the secret information normally required for doing so. The history of cryptanalysis is more or less dual to the history of cryptology simply because it is very human to want to read other peoples secrets. Certainly the birth of advanced warfare must have triggered the need for passing on encrypted secret information, and thus the need for the opposing party to break the encryption. The earliest known treatise on cryptanalysis, by the Arabian genius Al-kindi, goes back to 9-th century BC. The treatise includes a description of a method, now called *frequency analysis*, used to break most classical ciphers.

In the age of computers, cryptographic methods have become vastly more complex than the old pen-and-paper systems, and cryptanalysis has invented an array of tools and methods to combat these. Most attack methods are statistical in nature. Statistical attacks use large amounts of known - even chosen - plaintext/ciphertext pairs to look for correlations which can reveal the whole or parts of the encryption key. The amount of plaintext and ciphertext used in these attacks are often completely unrealistic. The two most widely used attacks of this type is *Linear cryptanalysis* [3] and *Differential cryptanalysis* [4].

In this thesis we concern ourselves with a different, non-probabilistic type of attack called *Algebraic cryptanalysis*. It is well known that a map $f : K^n \to K^m$, where K is a finite field, is polynomial, i.e. there exists polynomials $p_1, \ldots, p_m \in K[x_1, \ldots, x_n]$ such that

$$f(a_1, \ldots, a_n) = (p_1(a_1, \ldots, a_n), \ldots, p_m(a_1, \ldots, a_n)) \text{ , for all } a_1, \ldots, a_n \in K.$$

Since most modern ciphers can be expressed as such a map, we can find multivariate polynomial equations linking plaintext to ciphertext for these ciphers. If the variables $x_1, \ldots, x_n$ are a representation of the unknown encryption key, as the case usually is, finding a simultaneous solution to these equations is synonymous with finding the encryption key itself. So in simple terms, algebraic cryptanalysis is the study of methods for representing cipher algorithms as a system of algebraic equations, and finding a solution for these. This sounds simple, but algebraic cryptanalysis is certainly not a panacea since the general problem of solving multivariate polynomial equations is $\mathcal{NP}$-complete (see e.g. [5]).

Algebraic cryptanalysis is a relatively new field, but the idea is not new. Already in 1949, Claude E. Shannon - the father of *information theory* - remarked in a landmark paper [6] "*that solving a certain system requires at least as much work as solving a system of simultaneous equations in a large number of unknowns, of a complex type*". The reason for the late blooming of this field is most probably due to the intrinsic need for efficient computer resources. This has changed dramatically in the last decades, and algebraic cryptanalysis is now an established field and an active area of research. The big success stories are far apart. But there have been reported complete breaks, for instance the stream cipher Hitaq2 (see [7]) and the block cipher KeeLoq [8]. Algebraic cryptanalysis has also discovered a range of new weaknesses in ciphers which thus adds to the list of secure design rules to be followed when constructing cipher algorithms.

The resent developments in algebraic cryptanalysis seems to be an integration of earlier types of attacks with algebraic attacks. This includes combinations of algebraic and *slide attack* [8], differential cryptanalysis [9] and side-channel attacks [10]. Since the underlying problem, i.e. solving equations systems, is $\mathcal{NP}$-complete and thus have no general efficient solution method, combining attack vectors feels like the natural way for algebraic cryptanalysis to evolve.

The notion and theory of Gröbner bases was developed by Bruno Buchberger in his PhD thesis from 1965 [11], and for a light introduction to the topic we recommend [12] and [13]. Gröbner bases has many applications in ideal theory, e.g. *the ideal membership problem*, but in algebraic cryptanalysis this technique is used because, when properly implemented [14] [15], it is one of the most efficient ways of finding simultaneous solutions to polynomial equations.

In this thesis we study and implement attacks on both the stream cipher LILI-128 [16] and the block cipher KASUMI [17] using d-truncated Gröbner bases over Boolean rings. The reason for attacking two classes of cipher algorithms is to highlight the differences in approach. Material regarding *d-truncated Gröbner bases*, or bases restricted by polynomial degree, is hard to find in the scientific literature. Most Gröbner bases implementations will increase polynomial degree until a solution is found - but this also means that shortage of computer memory will be a big problem. Our interpretation of using d-truncated bases is to restrict computations of degree to an upper limit, and use other techniques than increasing the degree to find a solution.

***Keywords**: Algebraic cryptanalysis, multivariate polynomial equation over $\mathbb{F}_2$, relinearization, XL, XSL, MutantXL, SAT, KASUMI, LILI-128, d-truncated Gröbner bases, Buchbergers Homogeneous algorithm*

1.2 Problem Description

The construction of Gröbner bases has high complexity, and a seemingly benign looking system of polynomials in three or four variables of degree three or four may fail to terminate in a reasonable time [12]. It can be shown (see [18]) that "*most*" *ideals generated by s polynomials in n variables of degree bounded by d are such that their Gröbner bases have degree bounded by* $(n+1)d-n$. Polynomials of this kind will have a huge number of monomials. Since the worst case complexity of the running time for a Gröbner run is often expressed in terms of the largest degree D of a polynomial during the computation, i.e. $O(2^D)$, we see that this implies a running time of complexity $O(2^{nd})$ at worst. To paraphrase Gregory V. Bard ([2]): *submit your problem*

to several Gröbner basis implementations, and wait for the program to either crash due to a lack of memory, or output a result. This motivates investigations into the construction of d-truncated Gröbner bases and possible benefits in form of reduced memory consumption and/or reduced running time.

1.3 Justification, Motivation and Benefits

The construction of Gröbner bases for systems of polynomial equations, is a fundamental tool for many complex problems. Finding new- or improving old algorithms for efficiently constructing Gröbner bases are of great interest to many scientific disciplines. Regarding the field of algebraic cryptanalysis, knowledge in the use and construction of low-degree Gröbner bases may be highly beneficial. Direct benefits to the field may be the construction of stronger cryptographic algorithms which are more resistant to this type of attack.

1.4 Research Questions

The problem description thus leads to the following research questions:

1. Can we build a Gröbner basis algorithm that is less memory intensive using d-truncated Gröbner bases over a Boolean ring?
2. How will it perform on real life systems of equations?
3. What compromises must be made for achieving this?

1.5 Method

Due to the nature of the research problems, a mixed research approach was chosen. This means combining both quantitative and and qualitative methods. Existing literature has been studied, and we have focused on trying to solve systems of equations induced from two cipher algorithms. Since we where interested in overseeing all aspects of computation - speed, memory consumption and utilization of the processor - and in seeing how a potential d-truncated Gröbner basis evolves during computation, it was decided to implement a Gröbner bases algorithm. Since we where willing to trade speed over control, agility and visibility over the inner workings, all implementations where written in the fast-prototyping language *python* [1]. Due to the time consuming nature of the problem, the quantitative part is limited to simple statistics.

1.6 Outline of Chapters

This thesis is organized as follows: in Chapter 2 we give a relatively short run-through of mathematical topics leading up to Gröbner bases and Buchberger's algorithms. We have also included a section on $\mathcal{NP}$-completeness of solving general systems of polynomial equations. In Chapter 3 we give an overview of well known methods in algebraic cryptanalysis. Chapter 4 deals with technical and practical aspects of our Gröbner basis implementation. In Chapter 5 we introduce the ciphers LILI-128 and KASUMI. We introduce some trivia regarding polynomials and systems of polynomial equations. Here we also explain how we generated polynomial systems for the two ciphers. In Chapter 6 we try to give an account of our experiences during these attacks, and

[1] http://www.python.org

finally, in Chapter 7 we answer the research questions.

2 Mathematical Preliminaries

To fully understand what a Gröbner basis is, we need to have a grasp of the underlying theory. In this chapter we introduce the basic theory of multivariate polynomials and polynomial ideals. This will lead us to the famous *Hilbert's basis theorem* which is fundamental for the theory of Gröbner bases. We touch upon *Buchberger's algorithms* which is a basic building block in any Gröbner basis algorithm. We also state the *Shape Lemma* which shows why we use Gröbner bases for solving systems of multivariate polynomial equations, and we end this chapter with showing the $\mathcal{NP}$-completeness of solving multivariate polynomial equations.

2.1 Abstract Algebra Essentials

Abstract algebra deals with algebraic structures such as *groups*, *rings* and *fields*. We start off by defining these structures.

Definition 2.1 (Monoid). *A monoid* $(S, \times)$ *is a set S, with an associative binary operation* $\times : S \times S \to S$. *There exist an identity element* 1_S, *such that* $1_S \times s = s, \forall s \in S$. *A monoid is called abelian, or commutative, if for all* $s, s' \in S$ *we have that* $s \times s' = s' \times s$.

Definition 2.2 (Group). *A group* $(G, \times)$ *is a monoid in which every element is invertible, i.e* $\forall g \in G, \exists g' \in G$ *such that* $g \times g' = g' \times g = 1_G$.

Definition 2.3 (Ring). *A ring* $(R, +, \cdot)$ *is a set R with two associative operations* $+ : R \times R \to R$, $\cdot : R \times R \to R$. $(R, +)$ *is an abelian group with identity element* 0_R *and* $(R \setminus \{0_R\}, \cdot)$ *is an abelian monoid with identity element* 1_R. *The operation* $\cdot$ *must distribute over the operation* $+$.

Definition 2.4 (Field). *A field* $(K, +, \cdot)$ *is a ring such that* $(K \setminus \{0_K\}, \cdot)$ *is a group.*

Apart from rings and fields, the basic element we work mostly with is the polynomial. First we mention that a mathematical *variable* has two meanings depending on context: a variable may be able to "vary" as in a function context, or it is an "unknown quantity" as in an equation context. In the following, we mean "unknown quantity" when we refer to variables.

Definition 2.5 (Monomials, terms and total degree). *A monomial* m *in the variables* $x_1, x_2, \ldots, x_n$ *is a product of the form*

$$m = x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdot \ldots \cdot x_n^{\alpha_n},\ \alpha_i \in \mathbb{N}$$

The number $|\alpha| = \alpha_1 + \alpha_2 + \ldots + \alpha_n$ *is called the monomial* total degree*, or simply degree, and is denoted* $\deg(m)$. *A monomial with coefficient in a field* K *is called a* term *and has a short notation* $c_\alpha X^\alpha,\ c_\alpha \in K$.

Definition 2.6. *A polynomial* p *in the variables* $x_1, x_2, \ldots, x_n$*, with coefficients in a field* K *is a*

finite linear combination of terms

$$p = \sum_{\alpha} c_\alpha x^\alpha,\ c_\alpha \in K$$

The set of all monomials of p*, called the* support of p*, is denoted* $T(p)$*. The* total degree *of p, denoted deg(p), is the number* $\max(\deg(m),\ \forall\ m \in T(p))$*. The set of all polynomials in the variables* $x_1, x_2, \ldots, x_n$*, with coefficients in the field* K *forms a ring, i.e.* a polynomial ring *and is denoted* $K[x_1, x_2, \ldots, x_n]$

A polynomial $p \in K[x_1, x_2, \ldots, x_n]$ can also be seen as a map $p : K^n \to K$ if we allow the variables to take values in K. Assigning values $(a_1, a_2, \ldots, a_n)$ to the variables $x_1, x_2, \ldots, x_n$ in p is called *evaluating* p *at* $(a_1, a_2, \ldots, a_n)$.

A tuple $(a_1, a_2, \ldots, a_n) \in K^n$ which evaluates $p(x_1, x_2, \ldots, x_n)$ to zero is called a solution to the *polynomial equation*

$$p(x_1, x_2, \ldots, x_n) = 0, x_1, x_2, \ldots, x_n \in K^n.$$

Assume we are given an equation system of multivariate polynomials over $K[x_1, x_2, \ldots, x_n]$.

$$\begin{aligned} p_1(x_1, x_2, \ldots, x_n) &= 0 \\ p_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\cdots \\ p_s(x_1, x_2, \ldots, x_n) &= 0 \end{aligned}$$

If solutions exist, a solution that evaluates all polynomials to zero is called a *simultaneous solution*. The collection of all simultaneous solutions for a given system of polynomial equations is another algebraic structure called a *variety*.

Definition 2.7 (Affine variety). *Let* K *be a field, and let* $p_1, p_2, \ldots, p_m$ *be polynomials in* $K[x_1, x_2, \ldots, x_n]$*. Further, let* K^n *be the* affine space $(a_1, a_2, \ldots, a_n) : a_i \in K$*. The* affine variety *defined by* $p_1, p_2, \ldots, p_m$ *is the set*

$$\mathbf{V}(p_1, p_2, \ldots, p_m) = \{(a_1, a_2, \ldots, a_n) \in K^n : p_i(a_1, a_2, \ldots, a_n) = 0 \text{ for all } 1 \leq i \leq m\}.$$

So finding simultaneous solutions to a system of polynomial equations $p_1 = 0, p_2 = 0, \ldots, p_s = 0$ is synonymous to finding the affine variety defined by $p_1, p_2, \ldots, p_s$.

Another fundamental algebraic structure is the *ideal*:

Definition 2.8 (Ideal). *Let* $\mathbf{I}$ *be a subset of a ring* $(R, +, \cdot)$*.* $\mathbf{I}$ *is an* ideal *if it satisfies the following:*

(i) $0 \in \mathbf{I}$

(ii) If $p, q \in \mathbf{I}$ *then* $p + q \in \mathbf{I}$

(iii) If $p \in \mathbf{I}$ *and* $h \in (R, +, \cdot)$*, then* $hp \in \mathbf{I}$.

From a collection of polynomials $p_1, p_2, \ldots, p_n$ we can construct new polynomials. We can multiply with terms, add them together and so on. Let's consider all possible manipulations of polynomials from a collection:

Definition 2.9. *Let* $p_1, p_2, \ldots, p_s \in K[x_1, \ldots, x_n]$. *We denote the set of all linear combinations of* $p_1, \ldots, p_s$ *with coefficients in* $K[x_1, \ldots, x_n]$ *as:*

$$\langle p_1, p_2, \ldots, p_s \rangle = \left\{ \sum_i^s h_i p_i \ : \ h_1, \ldots, h_s \in K[x_1, \ldots x_n] \right\}$$

We say that $\langle p_1, p_2, \ldots, p_s \rangle$ *is* spanned *by* $p_1, \ldots, p_s$ *or that* $p_1, \ldots, p_s$ *is a basis of* $\langle p_1, p_2, \ldots, p_s \rangle$.

The *space* consisting of all possible polynomials constructible from a given collection is of course an ideal.

Lemma 2.1.1. *Let* $p_1, p_2, \ldots, p_s \in K[x_1, \ldots, x_n]$. *Then* $\langle p_1, p_2, \ldots, p_s \rangle$ *is an* ideal $\subset K[x_1, \ldots, x_n]$

Proof. We have that $0 \in \langle p_1, p_2, \ldots, p_s \rangle$ since $0 = \sum_i^s 0 \cdot f_i$.
Now, let $h, v_i, w_i \in K[x, \ldots, x_n]$ for all $1 \leq i \leq s$, and suppose that $p = \sum_{i=1}^s v_i p_i$ and $q = \sum_{i=1}^s w_i p_i$, so $p, q \in \langle p_1, p_2, \ldots, p_s \rangle$. Then we have that

$$p + q = \sum_{i=1}^s (v_i + w_i)\, p_i \in \langle p_1, p_2, \ldots, p_s \rangle$$

and

$$hp = \sum_{i=1}^s h p_i \in \langle p_1, p_2, \ldots, p_s \rangle$$

Which proves that $\langle p_1, p_2, \ldots, p_s \rangle$ is an ideal of $K[x_1, \ldots, x_n]$ □

So starting from a system of polynomial equations

$$\begin{aligned} p_1(x_1, \ldots, x_n) &= 0 \\ p_2(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ p_s(x_1, \ldots, x_n) &= 0 \end{aligned}$$

we can consider the left hand side of these as elements of the ideal $\langle p_1, p_2, \ldots, p_s \rangle$, and we say that $p_1, p_2, \ldots, p_s$ is a basis of, or generates the ideal $\langle p_1, p_2, \ldots, p_s \rangle$. From the basis, new equations can be created

$$h_1 p_1 + h_2 p_2 + \ldots + h_s p_s = 0, h_i \in K[x_1, \ldots, x_n]$$

by performing the appropriate multiplications and additions. The left hand side is still a member of the ideal $\langle p_1, p_2, \ldots, p_s \rangle$.

Assuming there exist simultaneous solutions to the original system of polynomial equations, we have a corresponding *affine variety*, i.e. $\mathbf{V}(p_1, p_2, \ldots, p_s)$. If the set of simultaneous solutions is finite, e.g. the affine variety is a finite set, the corresponding ideal is said to be zero-dimensional.

It is easy to see that if $p_1, \ldots, p_s$ is a basis to an ideal I, so is $p_1 + p_s, p_2 + p_s, \ldots, p_{s-1} + p_s, p_s$. So an ideal can have several bases, but is the variety the same?

Proposition 2.1.2. *Let* $p_1, p_2, \ldots, p_s$ *and* $q_1, q_2, \ldots, q_t$ *be bases of the same ideal* $\mathbf{I} \in K[x_1, x_2, \ldots, x_n]$*, i.e. we have that* $\langle p_1, p_2, \ldots, p_s \rangle = \langle q_1, q_2, \ldots, q_t \rangle$.
Then $\mathbf{V}(p_1, p_2, \ldots, p_s) = \mathbf{V}(q_1, q_2, \ldots, q_t)$

Proof. Let $\mathbf{V}_p$ be the affine variety defined by $p_1, \ldots p_s$, and $\mathbf{V}_q$ be the affine variety defined by $q_1, \ldots, q_t$. Since $\langle p_1, p_2, \ldots, p_s \rangle = \langle q_1, q_2, \ldots, q_t \rangle$ we have that $q_j \in \langle p_1, p_2, \ldots, p_s \rangle$ for $0 \leq j \leq t$. Thus every $q_j, 0 \leq j \leq t$ can be written as:

$$q_j = \sum_{i=1}^{s} h_i^j p_i, \text{ where } h_i^j \in K[x_1, x_2, \ldots, x_n] \text{ for } 0 \leq i \leq s,\ 0 \leq j \leq t$$

We see that $q_j, 0 \leq j \leq t$ vanishes for every point in $\mathbf{V}_p$ so $\mathbf{V}_p \subseteq \mathbf{V}_q$.
We also have that $p_i \in \langle q_1, q_2, \ldots, q_t \rangle$ for $0 \leq i \leq s$ and p_i can be written as

$$p_i = \sum_{j=1}^{t} f_j^i q_j, \text{ where } f_j^i \in K[x_1, x_2, \ldots, x_n] \text{ for } 0 \leq i \leq s,\ 0 \leq j \leq t$$

We see that $p_i, 0 \leq i \leq s$ vanishes for every point in $\mathbf{V}_q$ so $\mathbf{V}_q \subseteq \mathbf{V}_p$.
Since $\mathbf{V}_p \subseteq \mathbf{V}_q$ and $\mathbf{V}_q \subseteq \mathbf{V}_p$, we must have that $\mathbf{V}_q = \mathbf{V}_p$ □

We see that there is potential to transform our initial, maybe cumbersome, system of polynomial equations into another system with nicer solvability properties. Put another way, by changing the basis for the initial ideal, we have the potential to determine the corresponding variety in an easier way.

Given a variety $\mathbf{V}$ defined by $p_1, p_2, \ldots, p_s$, we know that $p_i, 0 \leq i \leq s$ vanishes on $\mathbf{V}$. But what can we say about *all* polynomials that vanish on a given affine variety?

Lemma 2.1.3. *Let* $\mathbf{V} \in K^n$ *be an affine variety. The set of all polynomials* $p \in K[x_1, x_2, \ldots, x_n]$ *that vanish on* $\mathbf{V}$ *generates an ideal.*

Proof. Let the set $\mathbf{S} \subset K[x_1, x_2, \ldots, x_n]$ contain all polynomials in $K[x_1, x_2, \ldots, x_n]$ that vanish on $\mathbf{V}$. Since the zero polynomial vanishes everywhere, we have that $0 \in \mathbf{S}$
Assume that $p, q \in \mathbf{S}$. Then obviously $p + q$ vanishes on $\mathbf{V}$, and for every non-zero $h \in K[x_1, x_2, \ldots, x_n]$ we have that $h \cdot p = h \cdot 0 = 0$. It follows that $\mathbf{S}$ is an ideal. □

Later we prove *Hilbert's basis theorem* that says that every ideal has a finite basis, but first we must take a closer look at multivariate polynomials and monomial ideals.

2.2 Monomial ordering and multivariate division

In the world of univariate polynomials, i.e. polynomials in one variable, things are fairly intuitive. Division algorithms must take into account that e.g. $X^4 > X^2$. And in linear equation systems in several variables, we order variables in some natural way, say $x > y > z > \ldots$. All of this is easy to accept. In the world of non-linear multivariate polynomials things are a little bit different. There is no natural, or obvious way of ordering multivariate monomials. For instance, which is the *larger* of x^5yz^3 and xz^8? For a division algorithm to work with multivariate polynomials we need a consistent way of ordering multivariate monomials.

First we choose a variable ordering, say $x_i > x_j$ if $i > j, i, j \in \mathbf{N}_0$, and all variables in a monomial follow this ordering.

Definition 2.10. *A* monomial ordering *on* $K[x_1, x_2, \ldots, x_n]$ *is any relation* ">" *on the set of monomials* $X^\alpha, \alpha \in \mathbf{N}_0^n$ *satisfying:*

(i) ">" *is a total ordering on* $\mathbf{N}_0^n$

(ii) *If* $\alpha > \beta$ *and* $\gamma \in \mathbf{N}_0^n$*, then* $\alpha + \gamma > \beta + \gamma$

(iii) ">" *is a well-ordering on* $\mathbf{N}_0^n$

There are of course several ways we can do this, and some well known examples are given below:

Definition 2.11 (Lexicographic Order or Lex Order). *Let* $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ *and* $\beta = (\beta_1, \beta_2, \ldots, \beta_n) \in \mathbf{N}_0^n$*. We say* $x^\alpha >_{lex} x^\beta$*, if the leftmost nonzero entry in the vector difference* $\alpha - \beta \in \mathbf{N}_0^n$ *is positive.*

Definition 2.12 (Graded Lex Order). *Let* $\alpha, \beta \in \mathbf{N}_0^n$*. We say* $x^\alpha >_{grlex} x^\beta$*, if*

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

Definition 2.13 (Graded Reverse Lex Order). *Let* $\alpha, \beta \in \mathbf{N}_0^n$*. We say* $x^\alpha >_{grevlex} x^\beta$*, if*

$$|\alpha| = \sum_{i=1}^{n} \alpha_i > |\beta| = \sum_{i=1}^{n} \beta_i, \text{ or}$$
$$|\alpha| = |\beta| \text{ and the rightmost nonzero entry of } \alpha - \beta \in \mathbf{N}_0^n \text{ is negative}$$

Computationally, *Graded Reverse Lex Order* has a reputation of producing a smaller Gröbner basis compared with the other orderings. To illustrate the different monomial orderings:

Example 2.1. *Consider the monomials* $X^3Y^4Z^5,\ X^5Z^7,\ X^6 \in K[X, Y, Z]$.

- $X^6 >_{lex} X^5Z^7 >_{lex} X^3Y^4Z^5$ *since the leftmost nonzero entries in* $(6-5, 0-0, 0-7)$ *and* $(5-3, 0-4, 7-5)$ *are positive.*
- $X^5Z^7 >_{grlex} X^3Y^4Z^5 >_{grlex} X^6$ *since* $|5+7| = |3+4+5| > |6|$ *and* $X^5Z^7 >_{lex} X^3Y^4Z^5$.

- $X^3Y^4Z^5 >_{grevlex} X^5Z^7 >_{grevlex} X^6$ *since* $|5+7| = |3+4+5| > |6|$ *and the rightmost nonzero entry in* $(3-5, 4-0, 5-7)$ *is negative*

We need the following common definitions regarding term-ordering in multivariate polynomials

Definition 2.14. *Let* $p = \sum_{\alpha}^{n} a_\alpha x^\alpha$ *be a polynomial in* $K[x_1, x_2, \ldots, x_n]$*, and let* $>$ *be a* monomial ordering*. We say that:*

(i) The ***multidegree*** *of* p *is* $\mathrm{multideg}(p) = \max_>(\alpha \in \mathbf{N}_0^n : a_\alpha \neq 0)$

(ii) The ***totaldegree*** *of* p *is* $\mathrm{totaldeg}(p) = |\mathrm{multideg}(p)|$

(iii) The ***leading coefficient*** *of* p *is* $\mathrm{LC}(p) = a_{\mathrm{multideg}(p)} \in K$

(iv) The ***leading monomial*** *of* p *is* $\mathrm{LM}(p) = x^{\mathrm{multideg}(p)}$

(v) The ***leading term*** *of* p *is* $\mathrm{LT}(p) = \mathrm{LC}(p) \cdot \mathrm{LM}(p)$

We can now state the general form of the multivariate division algorithm.

Algorithm 2.1 General multivariate division algorithm

```
Input: f, p_1, p_2, ..., p_s
Output: a_1, a_2, ..., a_s, r
a_1 := a_2 := ... := a_s := r := 0
p := f
while p ≠ 0 do
    i := 1
    divisionoccured := False
    while i < s AND divisionoccured = False do
        if LT(p_i) divides LT(p) then
            a_i := a_i + LT(p)/LT(p_i)
            p := p − (LT(p)/LT(p_i)) · p_i
            divisionoccured := True
        else
            i := i + 1
        end if
    end while
    if divisionoccured = False then
        r := r + LT(p)
        p := p − LT(p)
    end if
end while
return a_1, a_2, ..., a_s, r
```

In [19] the correctness of the general multivariate division algorithm is proven, and we just state the theorem here.

Theorem 2.2.1 (**Multivariate Division Algorithm in** $K[x_1, x_2, \ldots, x_n]$). *Given a monomial order* $>$ *on* $\mathbf{N}_0^n$*, and let* $S = (p_1, p_2, \ldots, p_s)$ *be an* $\mathrm{ordered}_>$ *s-tuple of polynomials in* $K[x_1, x_2, \ldots, x_n]$

Then every $p \in K[x_1, x_2, \ldots, x_n]$ *can be expressed as*

$$p = a_1 p_1 + a_2 p_2 + \ldots + a_s p_s + r,$$

where $a_i, r \in K[x_1, x_2, \ldots, x_n]$*, and either* $r = 0$ *or* r *is a linear combination, with coefficients in* K*, of monomials, none of which is divisible by any of* $LT(p_1), \ldots, LT(p_s)$*. We call* r *a* ***remainder*** *of* p *on division by* S*. Furthermore, if* $a_i p_i \neq 0$*, then we have*

$$\mathrm{multideg}(p) \geq \mathrm{multideg}(a_i p_i),\ 1 \leq i \leq s.$$

Proof. See [19] □

But there are important things to notice here. There is nothing that states that the a_i's and r are unique. This is best illustrated by an example:

Example 2.2. *We want to find the remainder of* $p = X^2Y + XY^2 + Y^2 - 3$ *after division by* $S = (p_1, p_2)$ *where* $p_1 = XY - 1$ *and* $p_2 = Y^2 - 1$*), and we use graded lex order.*
On run (1) we see that $LT(p_1) = XY$ *divides* $LT(p) = X^2Y$*, and* $\frac{LT(p)}{LT(p_1)} = X$ *so*

$$\begin{aligned} a_1 &:= X \\ p &:= XY^2 + Y^2 + X - 3 \end{aligned}$$

On run (2) we see that $LT(p_1) = XY$ *divides* $LT(p) = XY^2$*, and* $\frac{LT(p)}{LT(p_1)} = Y$ *so*

$$\begin{aligned} a_1 &:= X + Y \\ p &:= Y^2 + X + Y - 3 \end{aligned}$$

Now we see that $LT(p_1) = XY$ *does not divide* $LT(p) = Y^2$*, so we try the next polynomial in S and* $LT(p_2) = Y^2$ *divides* $LT(p) = Y^2$*, and* $\frac{LT(p)}{LT(p_1)} = 1$ *so*

$$\begin{aligned} a_2 &:= 1 \\ p &:= X + Y - 2 \end{aligned}$$

None of the remaining terms in p *is divisible by any of the leading terms in S, which gives*

$$\begin{aligned} a_1 &:= X + Y \\ a_2 &:= 1 \\ r &:= X + Y - 2 \end{aligned}$$

and we are finished.

But here is the important point: if we go back to run (2) we see that $LT(p) = XY^2$ *is also divisible by* $LT(p_2) = Y^2$*. If we perform the reduction with* p_2 *from run (2) we get*

$$\begin{aligned} a_1 &:= X \\ a_2 &:= X + 1 \\ r &:= 2X - 2 \end{aligned}$$

We end up with two different remainders depending on the polynomials used when reducing. Both are valid of course, but if this where operations involved in solving the equation system $p = 0, p_1 = 0, p_2 = 0$ *then* $r = X + Y - 2$ *is just another multivariate polynomial - albeit linear, but* $r = 2X - 2$ *is a univariate polynomial and thus a solution for the variable* X*, i.e.* $X = 1$.

We see that the result, or remainder, is not unique and the order in which we choose polynomials from the s-tuple matters. We also see that monomials and leading terms play a very important role in polynomial division and we must investigate this further.

2.3 Hilbert Basis Theorem

In this section we show that every polynomial ideal has a finite basis. First we establish important results concerning monomial ideals.

Definition 2.15. *An ideal* $I \subset\in K[x_1, x_2, \ldots, x_n]$ *is a* ***monomial ideal*** *if there is a subset* $A \subset \mathbf{N}_0^n$ *(possibly infinite) such that* $I = \langle x^\alpha : \alpha \in A \rangle$ *consists of all polynomials of the form*

$$p = \sum_{\alpha \in A} h_\alpha x^\alpha, \ h_\alpha \in K[x_1, x_2, \ldots, x_n].$$

We denote this as $I = \langle x^\alpha : \alpha \in A \rangle$

That is, a *monomial ideal* is an ideal where the basis consists of, possibly infinite, monomials. A monomial ideal is of course also a polynomial ideal.

Lemma 2.3.1. *Let* $I = \langle x^\alpha : \alpha \in A \rangle$ *be a monomial ideal and* $p \in K[x_1, x_2, \ldots, x_n]$ *be a polynomial. Then the following are true:*

(i) *A monomial* x^β *lies in* I *if and only if* x^β *is divisible by* x^α *for some* $\alpha \in A$.

(ii) $p \in I$ *if and only if* p *is a K-linear combination of monomials in* I.

(iii) *Two monomial ideals are the same if and only if they contain the same monomials.*

Proof. For the part (i), if x^β is a multiple of some $x^\alpha \in I$ where $\alpha \in A$ then $x^\beta \in I$ by the definition of ideal. Conversely, if $x^\beta \in I$ then

$$x^\beta = \sum_{i=1}^{s} h_i x^{\alpha_i} \text{ where } h_i \in K[x_1, \ldots, x_n] \text{ and } \alpha_i \in A.$$

Now, expand each h_i as a K-linear combination of monomials. Considering each variable and variable-degree $x_i^{\beta_i}$ in x^β, to maintain equality each term on the right must have the same variable-degree. Terms differing in variable-degree must cancel out. So x^β must have the form

$$x^\beta = K_{s_1} \cdot x^{h_{s_1} + \alpha_1} + K_{s_2} \cdot x^{h_{s_2} + \alpha_2} + \ldots + K_{s_T} \cdot x^{h_{s_T} + \alpha_s}$$

where each term's multidegree must satisfy $h_{s_j} + \alpha_j = \beta$. This proves that x^β is divisible by at least some $x^{\alpha_j} \in I$.
For the part (ii) we see that if each term of p is divisible by some $x^\alpha \in I$, then p can be written as $p = \sum_{i=1}^{s} h_i x^{\alpha_i}$, and by the definition of ideal, $p \in I$. On the other hand, if $p \in I$ then

$p = \sum_{i=1}^{s} h_i x^{\alpha_i}$. Expanding each h_i as a K-linear combination of monomials in $K[x_1, \ldots, x_n]$, turns p into a K-linear combination of monomials where each monomial is divisible by some monomial $\in I$. This proves part (ii).
Part (iii) is a consequence of part (ii). Let I, J be monomial ideals and let $x^\beta \in I$ be a monomial. If $x^\beta \notin J$ then there exists an $x^\alpha \in I$ with $\alpha \in A$, which is missing in J, thus $I \neq J$. If I, J contain the same monomials then every polynomial $p \in I$ is also an element of J, thus $I = J$. This proves (iii). □

The next important, and well known theorem states that every monomial ideal is finitely generated, i.e. it has a finite monomial basis, even though it might consist of countably infinite monomials. We omit the proof here.

Theorem 2.3.2 (Dickson's Lemma). *Let* $I = \langle x^\alpha : \alpha \in A \langle$ *be a monomial ideal. Then* I *can be written in the form* $I = \left\langle x^{\alpha(1)}, \ldots, x^{\alpha(s)} : \alpha(i) \in A \text{ for } 1 \leq i \leq s \right\rangle$. *In particular,* I *has a finite basis.*

Proof. See [19] □

A little shorthand: if $I \subset K[x_1, x_2, \ldots, x_n]$ is an ideal different from 0, then we denote the set of leading terms from elements in I as

$$LT(I) = \{ax^\alpha : \text{ there exists } p \in I \text{ with } LT(p) = ax^\alpha\}$$

The ideal generated from the elements of $LT(I)$ is denoted $\langle LT(I)\rangle$. But note that if we are given a finite generating set, thus $I = \langle p_1, p_2, \ldots, p_s\rangle$, the two monomial ideals $\langle LT(I)\rangle$ and $\langle LT(p_1), LT(p_2), \ldots, LT(p_s)\rangle$ is not necessarily the same ideal.

We need the following result for monomial ideals, saying that for any polynomial ideal I there exist polynomials belonging to the ideal whose leading terms generate $LT(I)$.

Lemma 2.3.3. *Let* $I \subset K[x_1, x_2, \ldots, x_n]$ *be an ideal.*

(i) $\langle LT(I)\rangle$ *is a monomial ideal.*

(ii) There are $g_1, g_2, \ldots, g_t \in I$ *such that* $\langle LT(I)\rangle = \langle LT(g_1), LT(g_2), \ldots, LT(g_t)\rangle$

Proof. For the part (i). Let g be any non-zero element of I. Then $\langle LM(g) : g \in I\rangle$ is a monomial ideal. Since $LM(g)$ and $LT(g)$ differ by a nonzero constant, the ideal $\langle LT(g) : g \in I\rangle$ equals $\langle LM(g) : g \in I\rangle$. Thus $\langle LT(g) : g \in I\rangle$ is a monomial ideal.
For the part (ii). Dickson's Lemma states that $\langle LT(I)\rangle = \langle LM(g_1), \ldots, LM(g_t)\rangle$ for finitely many $g_1, \ldots, g_t \in I$. Since $LM(g_j)$ and $LT(g_j)$ differ by a nonzero constant it follows that $\langle LT(I)\rangle = \langle LT(g_1), \ldots, LT(g_t)\rangle$. This completes the proof. □

We now have enough to establish the famous Hilbert Basis Theorem.

Theorem 2.3.4 (Hilbert Basis Theorem). *Every ideal* $I \subset K[x_1, x_2, \ldots, x_n]$ *has a finite generating set, i.e. for every ideal* I*, there are* $g_1, g_2, \ldots, g_t \in I$ *such that* $I = \langle g_1, g_2, \ldots, g_t\rangle$

Proof. If $I = \{0\}$ we can take the finite generating set to be $\{0\}$. Now, let I contain some nonzero elements. According to Lemma 2.3.3 there are $g_1, \ldots, g_t \in I$ such that $\langle LT(I)\rangle = \langle LM(g_1), \ldots, LM(g_t)\rangle$.

Consider the polynomial ideal $\langle g_1, \ldots, g_t \rangle$. It is clear that $\langle g_1, \ldots, g_t \rangle \subset I$ since each $g_i \in I$. Let $p \in I$ be arbitrary. Choosing an admissible monomial ordering, and dividing p by $\langle g_1, \ldots, g_t \rangle$ we get, according to Theorem 2.2.1, an expression of the form

$$p = a_1 g_1 + a_2 g_2 + \ldots + a_t g_t + r$$

where no term of r is divisible by any $LT(g_i)$. Since $p \in I$ we have that $r \in I$ because

$$r = p - a_1 g_1 + a_2 g_2 + \ldots + a_t g_t.$$

If $r \neq 0$, then $LT(r) \in \langle LT(I) \rangle = \langle LM(g_1), \ldots, LM(g_t) \rangle$. But as shown earlier, this means that $LT(r)$ is divisible by some $LT(g_i)$. This contradiction shows that $r = 0$. Thus

$$p = a_1 g_1 + a_2 g_2 + \ldots + a_t g_t$$

Since the choice of p was arbitrary, this shows that $I \subset \langle g_1, \ldots, g_t \rangle$. Since $\langle g_1, \ldots, g_t \rangle \subset I$ and $I \subset \langle g_1, \ldots, g_t \rangle$ we must have that $I = \langle g_1, \ldots, g_t \rangle$. This proves that there exists a finite basis for any polynomial ideal I.

□

Knowing that any polynomial ideal has a finite basis is neat in itself. But we can also use this theorem to establish other important results.

The following theorem is a consequence of Hilbert Basis Theorem, and it shows what happens to an initial ideal when we keep adding polynomials to the base.

First, an *ascending chain* of ideals in $K[x_1, \ldots, x_n]$ is a nested increasing sequence of ideals and can be extended by adding generators, or basis elements.

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

The important point is that the sequence stabilizes after a finite number of steps.

Theorem 2.3.5 (**Ascending Chain Condition**). *Let*

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

be an ascending chain of ideals in $K[x_1, x_2, \ldots, x_n]$. *Then there exists an* $N \geq 1$ *such that*

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

Proof. Consider the set $I = \bigcup_{i=1}^{\infty} I_i$, where $I_1 \subset I_2 \subset \ldots$ is an ascending chain of ideals in $K[x_1, \ldots, x_n]$. We must show that I is also an ideal. Now $0 \in I$ since $0 \in I_i$, $\forall\, i$. Let $p, q \in I$, and assume $p \in I_i$ and $q \in I_j$, $j > i$. The ideals form an ascending chain, and we must have that both p and q are elements in I_j. Also, since I_j is an ideal then $p + q \in I_j$, and thus $p + q \in I$. If $p \in I_i$ and $h \in K[x_1, \ldots, x_n]$ then $h \cdot p \in I_j$ and is thus also an element of I. This shows that I is an ideal.

According to the Hilbert Basis Theorem 2.3.4, the ideal I must have a finite basis $I = \langle g_1, g_2, \ldots, g_s \rangle$. But each of the basis elements is contained in some I_i and all subsequent ideals in the chain. Let I_N be the first ideal in the chain, which contains all of $(g_1, \ldots g_s)$. Then

$$I = \langle g_1, g_2, \ldots, g_s \rangle \subset I_N \subset I_{N+1} \subset \ldots \subset I$$

We see that the ascending chain stabilizes with I_N, and all subsequent ideals are equal. □

Assume we have a large, or infinite system of polynomial equations, where each polynomial $\in K[x_1, \ldots, x_n]$. Also assume we have an algorithm that uses the division algorithm, and we keep feeding the algorithm polynomials from the initial system of polynomial equations. The *Ascending Chain Condition* says we will reach a final ideal after a finite number of steps.

The next result is also a consequence of Hilbert Basis Theorem, and tells us that the affine variety of an ideal is the same as the affine variety of the basis.

Proposition 2.3.6. $\mathbf{V}(I)$ *is an* affine variety. *In particular, if* $I = \langle p_1, p_2, \ldots, p_s \rangle$ *then* $\mathbf{V}(I) = \mathbf{V}(p_1, p_2, \ldots, p_s)$.

Proof. By the Hilbert Basis Theorem,$I = \langle p_1, \ldots, p_s \rangle$ for some finite generating set. Since $p_i \in I$ and if $p(a) = 0,\ a \in K^n$ for all $p \in I$ then $p_i(a) = 0$, thus $V(I) \subset V(f_1, \ldots, f_s)$. On the other hand, let $a \in V(p_1, \ldots, p_s)$, and let $p \in I$, where $I = \langle p_1, \ldots, p_s \rangle$. Then we can write

$$p = \sum_{i=1}^{s} h_i p_i, \text{ for some } h_i \in K[x_1, \ldots, x_n]$$

But then $p(a) = 0$ since all $p_i(a) = 0$. Thus $V(p_1, \ldots, p_s) \subset V(I)$ Since we have that $V(I) \subset V(p_1, \ldots, p_s)$ and $V(p_1, \ldots, p_s) \subset V(I)$ we must have that $V(p_1, \ldots, p_s) = V(I)$. □

We see that we have a nice correspondence between varieties and ideals in polynomial rings. The *Hilbert Basis Theorem* gives us the existence of a finite basis for any polynomial ideal, but the basis are in no way unique. Given a set of polynomial equations and our goal being to find a simultaneous solution for the set, we now know that the initial set of polynomial equations generates an ideal. The ideal has many generating sets or bases, and the variety of the ideal is the same as the variety of each such basis. This implies that if we can find some basis for the ideal which makes it easier to determine the affine variety, we have found a solution or solutions to the initial set of polynomial equations.

Now, let's take a look at a special kind of bases, namely Gröbner bases.

2.4 Gröbner bases

A Gröbner basis is a particular kind of generating set for an ideal in a polynomial ring. The theory was developed by Bruno Buchberger in his doctoral thesis [11] in 1965, and he named this special kind of basis after his thesis advisor Wolfgang Gröbner. In his doctoral thesis, Buchberger developed an algorithm for the construction of this type of basis, now called *Buchberger's algorithm*, and proved the correctness and the termination of the algorithm. He applied the theory on the *Ideal Membership Problem*, i.e. how to decide if a polynomial p in a polynomial ring $K[x_1, \ldots, x_n]$ is a member of an ideal $I \subset K[x_1, \ldots, x_n]$. The theory of Gröbner bases is a major tool for solving a great variety of problems in computational algebra.

So what is a Gröbner basis? A Gröbner basis for an ideal I is the generating set for the monomial ideal $\langle LT(I) \rangle$, and the formal definition is

Definition 2.16 (Gröbner basis). *Let a monomial ordering be given.*
A finite subset $G = \{g_1, g_2, \ldots, g_s\}$ *of an ideal* I *is called a* ***Gröbner basis*** *if*

$$\langle LT(I) \rangle = \langle LT(g_1), LT(g_2), \ldots, LT(g_s) \rangle .$$

More clearly, a set $\{g_1, \ldots, g_s\} \in I$ is a Gröbner basis of I if and only if for any $p \in I$, $LT(p)$ is divisible by one of the $LT(g_i)$.

Corollary 2.4.1. *Let a monomial ordering be given. Then every ideal $I \subset K[x_1, x_2, \ldots, x_n]$ other than $\{0\}$ has a Gröbner basis. Furthermore, any Gröbner basis of I is a basis of I.*

Proof. This follows from the Hilbert Basis Theorem. The basis constructed in 2.3.4 is a Gröbner basis by definition. It is also shown in 2.3.4 that the basis generates I. □

We denote by $\overline{p}^S$ the remainder on division of p by *the ordered* s-tuple $S = (p_1, p_2, \ldots, p_s)$. As shown earlier, a reordering of the s-tuple S may produce a different remainder. Gröbner bases have a well of favorable properties, and one important property is that it eliminates this undesirable effect. This is shown in the following proposition:

Proposition 2.4.2. *Let $G = \{g_1, g_2, \ldots, g_s\}$ be a Gröbner basis for an ideal $I \subset K[x_1, x_2, \ldots, x_n]$ and $p \in K[x_1, x_2, \ldots, x_n]$. Let $r = \overline{p}^G \in K[x_1, x_2, \ldots, x_n]$ be the remainder on division of p by G.*

(i) No term of r is divisible by any of $LT(g_1), \ldots, LT(g_t)$

(ii) There exists $g \in I$ such that $p = g + r$

(iii) The remainder r of p on division by G is unique regardless of the order of the elements of G.

(iv) $p \in I$ if and only if remainder r on division of p by G is zero.

Proof. (i) This is proven in 2.2.1.

(ii) Also proven in in 2.2.1 by setting $g = a_1p_1 + a_2p_2 + \ldots + a_sp_s \in I$.

(iii) To prove uniqueness, suppose that $p = g + r = \overline{g} + \overline{r}$ satisfy (i) and (ii). Assume $r \neq \overline{r}$, then $g - \overline{g} = r - \overline{r} \in I$. Then $LT(r - \overline{r}) \in LT(I)$ is divisible by some $LT(g_i)$ by the definition of Gröbner basis. But this contradicts (i). Thus $r - \overline{r}$ must be zero and uniqueness is proved.

(iv) If the remainder r is zero, then the question of membership of p in I is answered by definition. Conversely, given $p \in I$, then $p = p + 0$ satisfies (i),(ii) and (iii) so 0 is the unique remainder of p on division by G.

□

Before we state Buchberger's algorithm, we need an element called the S-polynomial. The S-polynomial is constructed such that it, in a sense, cancel leading terms.

Definition 2.17 (S-polynomial). *Let $p, q \in K[x_1, x_2, \ldots, x_n]$ be non-zero polynomials.*
*(i) If $\mathrm{multideg}(p) = \alpha$ and $\mathrm{multideg}(q) = \beta$, then let $\gamma = (\gamma_1, \ldots, \gamma_n)$, where $\gamma_i = \max(\alpha_i, \beta_i)$. We call x^γ the **least common multiple** of $LM(p)$ and $LM(q)$.*
*(ii) The **S-polynomial** of p and q is the combination*

$$S(p, q) = \frac{x^\gamma}{LC(p)^{-1} \cdot LM(p)} \cdot p \; - \; \frac{x^\gamma}{LC(q)^{-1} \cdot LM(q)} \cdot q$$

(p, q) is called a critical pair.

Then we are ready to state Buchberger's Algorithm in it's most simplistic form.

Theorem 2.4.3 (**Buchberger's Algorithm**). *Let* $I = \langle p_1, p_2, \ldots, p_s \rangle$ *be a polynomial ideal. Then a Gröbner basis for* I *can be constructed in a finite number of steps by the following algorithm:*

Input: $P = (p_1, p_2, \ldots, p_s)$
Output: a Gröbner basis $G = (g_1, g_2, \ldots, g_t)$ for I, with $P \subset G$
$G := F$
repeat
  G':=G
  for each pair $\{p, q\}, p \neq q$ in G' **do**
    $S := \overline{S(p, q)}^{G'}$
    if $S \neq 0$ **then**
      $G := G \cup \{S\}$
    end if
  end for
until $G = G'$
return G

Proof. See [19] □

As seen, we start with a set of polynomials P. For each polynomial $p, q \in P$ we form the corresponding S-polynomial. Note that if two polynomials have identical leading terms then an ordinary subtraction is performed. The non-zero remainders are added to the set of polynomials. We keep on creating S-polynomials from the set of polynomials until all remainders from division by the set of polynomials are zero. The algorithm then terminates. It is quite natural to terminate the algorithm at this point, because we have depleted all options for creating new leading terms from the current set of polynomials not divisible by existing leading terms. Buchberger's Criterion tells us that the polynomials in the current set is a Gröbner basis for the ideal generated by the initial set of polynomials. The proof is omitted here.

Theorem 2.4.4 (**Buchberger's Criterion**). *Let* I *be a polynomial ideal. Then a basis* $G = (g_1, g_2, \ldots, g_t)$ *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 *(in some order) is zero.*

Proof. See [19] □

It is easy to see that we may be dealing with a large number of polynomials, since every polynomial in the current set is combined with all the others to create S-polynomials. So it is wise to use every possible technique to reduce the number of polynomials we have to work on. The following lemma tells us that we can eliminate unnecessary generators. Also note that a Gröbner basis for an ideal is not unique. The result is dependent on the monomial ordering, as well as the order in which we process the initial polynomials.

Lemma 2.4.5. *Let* G *be a Gröbner basis for a polynomial ideal* I*. Let* $g \in G$ *be a polynomial such that* $LT(g) \in \langle LT(G \setminus \{g\}) \rangle$*. Then* $G \setminus p$ *is also a Gröbner basis for* I*.*

Proof. By definition, we know that $\langle LT(G)\rangle = \langle LT(I)\rangle$. Then if for some $p \in G$ we have that $LT(p) \in \langle LT(G \setminus \{g\})\rangle$, we still have that $\langle LT(G \setminus \{g\})\rangle = \langle LT(I)\rangle$. By definition it follows that $G \setminus p$ is also a Gröbner basis for I. □

Gröbner bases come in different flavors. By eliminating unnecessary generators and adjusting all leading coefficients to 1 we have what is called a minimal Gröbner basis.

Definition 2.18 (**Minimal Gröbner basis**). *A **minimal Gröbner basis** for a polynomial ideal I is a Gröbner basis G for I such that:*
(i) $LC(g) = 1$ *for all* $g \in G$
(ii) For all $g \in G$, $LT(g) \notin \langle LT(G - g)\rangle$.

An ideal may have many minimal Gröbner bases. But if we increase our constraints on the minimal Gröbner basis, we get a *reduced Gröbner basis*.

Definition 2.19 (**Reduced Gröbner basis**). *A **reduced Gröbner basis** for a polynomial ideal I is a Gröbner basis G for I such that:*
(i) $LC(g) = 1$ *for all* $g \in G$
(ii) For all $g \in G$, *no monomial of* g *lies in* $\langle LT(G \setminus g)\rangle$.

For a given a monomial ordering the reduced Gröbner basis for an ideal is unique. The proof for this is omitted.

Definition 2.20. *Let* $I \subset K[x_1, x_2, \ldots, x_n]$ *be an ideal. We denote by* $\mathbf{V}(I)$ *the set*

$$\mathbf{V}(I) = \{(a_1, a_2, \ldots, a_n) \in K^n : p(a_1, a_2, \ldots, a_n) = 0 \text{ for all } p \in I\}$$

But since we deal with algebraic cryptanalysis, our main goal is to solve systems of multivariate polynomial equations. How can constructing Gröbner bases help? *The Shape Lemma* gives an answer to this question, but we need to introduce some terminology and do some work before we state the lemma.

Let L be a field. If $K \subset L$, and K is a field in itself, we say that L is an *extension field* of K, and we call L/K a *field extension*. If every element of L is a root of some non-zero polynomial with coefficients in K, we call L/K an *algebraic field extension*. A field is said to be *algebraically closed* if every polynomial with coefficients in the field also has a root in the field. An *algebraic field extension* of K which is *algebraically closed* is called the *algebraic closure* of K, and is denoted $\overline{K}$. A good example of the above are the two fields R - the field of real numbers - and C - the field of complex numbers -, where C is the algebraic closure of R.

In algebraic cryptanalysis, we want our polynomial ideals to have finitely many solutions of course. Such ideals are called *zero-dimensional*, and for completeness we state the *Finiteness Criterion* for zero-dimensional field but omit the proof.

Proposition 2.4.6 (**Finiteness Criterion**). *Let* $T(x_1, x_2, \ldots, x_n)$ *be the set of monomials on* $\overline{K}[x_1, x_2, \ldots, x_n]$ *and let* $>$ *be a monomial ordering on this set. Further, let* $I = \langle p_1, p_2, \ldots, p_s\rangle$ *be an ideal. The following are equivalent*

(i) The system $p_1, p_2, \ldots, p_s$ *has finitely many solutions*

(ii) *For* $i = 1, \ldots, n$ *we have that* $I \cap \overline{K}[x_i] \neq \emptyset$

(iii) *The* K*-vector space* $K[x_1, x_2, \ldots, x_n]/I$ *is finite.*

(iv) *The set* $T(x_1, \ldots, x_n) \backslash LT_>(I)$ *is finite.*

Proof. See [20] □

We also need the following definitions:

Definition 2.21. *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 in* K *has a* p^{th}*-root in* K.

Definition 2.22. *Let* $I \subset K[x_1, x_2, \ldots, x_n]$ *be an ideal. The radical of* I*, denoted* $\sqrt{I}$ *is the set*

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

If $I = \sqrt{I}$ *then* I *is called a radical ideal.*

Definition 2.23. *Suppose that* I *is a zero-dimensional ideal in* $K[x_1, x_2, \ldots, x_n]$*, and let* $i = 1, \ldots, n$*. We say that* I *is* ***in normal*** x_i***-position*** *if any two zeros* $(a_1, \ldots, a_n), (b_1, \ldots, b_n) \in \overline{K}^n$ *of* I *satisfy* $a_i \neq b_i$.

What we want with this is to reduce the problem of solving a system of polynomial equations $p_1 = p_2 = \ldots, p_n = 0$ to the case when $I = \langle p_1, p_2, \ldots, p_n \rangle$ is a zero-dimensional radical ideal in normal x_n position over a perfect field K. The reason is that the Shape Lemma ensures that we can find univariate polynomials in the base. Seidenberg's Lemma provides one key for this:

Proposition 2.4.7 (Seidenberg's Lemma). *Let* K *be a field, and let* $I \subset K[x_1, x_2, \ldots, x_n]$ *be a zero-dimensional ideal. Suppose that, for* $i = 1, \ldots, n$*, there exists a non-zero polynomial* $g_i \in I \cap K[x_i]$*, such that the greatest common divisor of* g_i *and its derivative equals 1. Then* I *is a* radical ideal.

Proof. See [20] □

The next for us to do is to add field equations to the initial system of polynomial equations. These are relatively prime to their derivatives. Due to Seidenberg's Lemma, the ideal is a radical ideal and the Finiteness Criterion ensures that the ideal is zero-dimensional. Since we have complete factorization of the field equations over K, the variety of simultaneous solutions to the extended system of polynomial equations does not contain points in $\overline{K} \backslash K$. The last thing we need is the notion of an *elimination ideal*:

Definition 2.24. *Let* $I \subset K[x_1, x_2, \ldots, x_n]$ *be an ideal. The l-th elimination ideal* I_l*, is the ideal of* $K[x_{l+1}, x_{l+2}, \ldots, x_n]$ *defined by*

$$I_l = I \cap K[x_{l+1}, x_{l+2}, \ldots, x_n]$$

Now we can state the Shape Lemma.

Theorem 2.4.8 (**The Shape Lemma**). *Let* K *be a perfect field, and let* $I \subset K[x_1, x_2, \ldots, x_n]$ *be a zero-dimensional radical ideal in normal* x_n *position. Let* $g_n \in K[x_n]$ *be the* monic *generator of the elimination ideal* $I \cap K[x_n]$*, and let* d *be the degree of* g_n*.*
(i) The reduced Gröbner basis of I *with respect to the lexicographic ordering 2.11 is of the form*

$$\{x_1 - g_1, \ldots, x_{n-1} - g_{n-1}, g_n\} \text{ where } g_1, \ldots, g_n \in K[x_n]$$

(ii) The polynomial g_n *has* d *distinct zeros* $a_1, \ldots, a_d \in \overline{K}^n$*, and the set of zeros of* I *is*

$$Z(I) = \{(g_1(a_i), g_2(a_i), \ldots, g_{n-1}(a_i), a_i) | i = 1, \ldots, d\}$$

Proof. See [20] □

As we can see in (i) in the Shape Lemma, the reduced Gröbner basis of such a restricted ideal has a very special shape, thereof the name. It also contains a univariate element g_n which we can solve for instance with numerical methods to find values for x_n. And from (ii) we see how the rest of the solution looks like.

2.4.1 Truncated Gröbner bases

In this section we look at the homogeneous case. The *Homogeneous Buchberger Algorithm* is an important framework for many advanced algorithms, since it enables computation of bases up to a certain total degree. For instance, if we somehow happen to know beforehand the maximum degree of elements in a homogeneous Gröbner basis, we could save computational time. But this is rarely the case. The importance of controlling the degree of polynomials during a Gröbner computation stems from fact that the running time complexity is often expressed as $O(2^D)$, where D is the largest degree of a polynomial during a computation. In [18] the following theorem is stated:

Theorem 2.4.9. *"Most" of the ideals generated by s polynomials in n variables of degree bounded by d are such that their Gröbner bases have degree bounded by* $(n+1)d - n$.

This regards polynomials with random coefficients and "most" means *all* except a set of measure zero. The implication is that in worst case, the running time of a Gröbner computation has complexity $O(2^{nd})$. In short: degree matters.

Definition 2.25. *A polynomial* $p \in K[x_1, \ldots, x_n]$ *is called* homogeneous of total degree d *if every term appearing in* p *has total degree d. For general* $p \in K[x_1, \ldots, x_n]$*, the homogeneous component of degree d of* p *is the set of terms having total degree d.*

As an example, $X^2YZ^2 + XY^4 + Z^5$ is a homogeneous polynomial of total degree 5. If we start out with a non-homogeneous polynomial, say $X^2YZ^2 + XY^2 + Z$, we can *homogenize* the polynomial by introducing additional variables, say W, to create a homogeneous one, i.e. $X^2YZ^2 + XY^2W^2 + ZW^4$. We can of course reconstruct our initial polynomial by setting $W = 1$.

Theorem 2.4.10 (Homogeneous Buchberger Algorithm). *Let* $\{p_1, ..., p_m\}$ *be a set of homogeneous polynomials generating the ideal* I*. Then a homogeneous Gröbner Basis for* I *can be constructed in a finite number of steps by the Homogeneous Buchberger Algorithm 2.2.*

Proof. See [21] □

Algorithm 2.2 Homogeneous Buchberger Algorithm

Input: $P = (p_1, p_2, \ldots, p_s)$
Output: Gröbner basis $G = (g_1, g_2, \ldots, g_t)$ of I, the elements of which satisfy

$$totaldeg(g_1) \leq \ldots \leq totaldeg(g_t)$$

$B := \{\}$
$G = ()$
$s := 0$
repeat
  $d_1 := min\{(totladeg(p) : \forall p \in P\}$
  $d_2 := min\{(totladeg(LCM(LT(p_i), LT(p_j)) : (i,j) \in B\}$
  $d := min\{d_1, d_2\}$
  $B_d := \{(i,j) \in B : (totladeg(LCM(LT(p_i), LT(p_j))) = d\}$
  $B := B \backslash B_d$
  $P_d := \{p \in P : totaldeg(p) = d\}$
  $P := P \backslash P_d$
  repeat
    MARK
    if $B_d = \{\}$ **then**
      choose $p \in P_d$ and delete it from P_d
    else
      choose a pair $(i,j) \in B_d$ and delete it from B_d
      $p = S(g_i, g_j)$
    end if
    if $\cap p^G = 0$ **then**
      GOTO:MARK
    else
      $s = s + 1$
      $g_s = \cap p^G$, add g_s to G
      Add pairs $(1,s), (2,s), \ldots, (s-1,s)$ to B
    end if
  until $B_d = \{\}$ AND $P_d = \{\}$
until $B = \{\}$ AND $P = \{\}$
return G

We see that for S-polynomial computation, the algorithm picks polynomials from the set with the smallest total degree. This gives us the opportunity to interrupt the algorithm when a certain total degree is finished, which Kreuzer and Robbiano ([21]) call *Calculus Interruptus*. What we are left with then is called a *truncated Gröbner Basis*.

Definition 2.26. *Let* $G = g_1, \ldots, g_n$ *be a result from the Homogeneous Buchberger Algorithm for an ideal* I *generated by a set of homogeneous polynomials. Elements from* G *with total degree less than* d *form a* d-truncated Gröbner Basis *for the ideal* I*, and will be denoted* $G_{\leq d}$.

If we start out with an ideal generated by a set of non-homogeneous polynomials, the Homogeneous Buchberger Algorithm will produce a correct Gröbner Basis. In that case, a d-truncated Gröbner Basis $G_{\leq d}$ is characterized by the following: $\forall(f, g) \in G_{\leq d}$ such that the S-polynomial $S(f, g)$ has total degree $\leq d$, the S-polynomial reduces to zero by $G_{\leq d}$. I.e. if we have a d-truncated Gröbner Basis, we know that if a polynomial reduces to degree d, then the polynomial will eventually be reduced to zero and we can skip the rest of the computation.

2.5 On the complexity of solving polynomial equations

Solving multivariate polynomial equations is hard. Even the simplest case of non-linear multivariate polynomial equations, i.e. multivariate quadratic, is hard in the sense that it does not exist, and probably will never exist, an efficient algorithm that solves all systems of multivariate quadratic polynomial systems. This sounds harsh, but there is not much hope to find such an algorithm.

In complexity theory, the class $\mathcal{P}$ is the set of problems that can be solved efficiently, or are tractable in practice. Efficient, or tractable, here means that the problems can be solved in polynomial time. But there are problems beyond this.

Definition 2.27 ($\mathcal{NP}$)**.** *$\mathcal{NP}$ is the set of all decision problems, i.e. having "yes/no" answers, that if the answer to the decision problem is "yes" there exists evidence, or a* certificate*, such that the "yes" answer can be verified in polynomial time.*

$\mathcal{NP}$ thus contains all problems that can be verified deterministically in polynomial time, given the evidence. Formally, this means that $\mathcal{NP}$ contains all decision problems solvable by a non-deterministic Turing machine.

Definition 2.28 ($\mathcal{NP}$-complete)**.** *A decision problem P is in $\mathcal{NP}$-complete, if every decision problem in $\mathcal{NP}$ can be reduced, or transformed to P in polynomial time.*

This means that $\mathcal{NP}$-complete contains all the really difficult problems where no efficient algorithm is known. It also means that if you can solve one such problem, call it P, in polynomial time you can solve every $\mathcal{NP}$-complete problem in polynomial time by reducing them to P. There is a famous conjecture, the $\mathcal{P} \neq \mathcal{NP}$ conjecture meaning that there exist problems where efficient algorithms will never be found.

The problem of solving the general (with emphasis on general), system of multivariate quadratic polynomial equations is called the $\mathcal{MQ}$ problem. The sad part is that $\mathcal{MQ}$ is $\mathcal{NP}$-complete. To show this [5], we must first show that $\mathcal{MQ}$ is solvable in Non-deterministic Polynomial-time, or in $\mathcal{NP}$-time. Let a system of multivariate quadratic polynomial equations over $\mathbb{F}_2$ be given

$$q_1(x_1, x_2, \ldots, x_n) = q_2(x_1, x_2, \ldots, x_n) = \ldots = q_s(x_1, x_2, \ldots, x_n) = 0$$

The following $\mathcal{NP}$-time algorithm solves the given system of equations:

1. Guess an assignment A for $(x_1, x_2, \ldots, x_n)$.
2. Check that all s equations are satisfied by A.
3. Output *true* if step 2. is true, else go to step 1.

Now, each equation has at most

$$1 + \binom{n}{1} + \binom{n}{2} = 1 + \frac{n(n+3)}{2}$$

terms. There are s equations, so step (2) requires at most polynomial time. If step(2) is successful, the algorithm terminates. We can not determine when step 2. will be true. So $\mathcal{MQ}$ is solvable in $\mathcal{NP}$-time.

Next we show that we can reduce an $\mathcal{NP}$-complete problem to $\mathcal{MQ}$. *SAT*, or satisfiability, is the problem of determining if a given Boolean formula can be evaluated to TRUE by assigning values to the variables in the formula. The formalism of complexity and the proof of the $\mathcal{NP}$-completeness of SAT was done by Cook (see [22]) in 1971. SAT can be reduced to 3-SAT in polynomial time thus 3-SAT is also in $\mathcal{NP}$-complete.

Definition 2.29 (3-SAT). *Let* $B = (b_1, b_2, \ldots, b_n)$ *be a set of Boolean variables.*
Let $L = (b_1, \overline{b_1}, \ldots, b_n, \overline{b_n})$ *be the corresponding literals, and let* $c_i \in (L \cup L^2 \cup L^3)$ *be clauses of at most 3 literals, and let* $C = (c_1, c_2, \ldots, c_t)$ *be a set of these clauses. The corresponding 3-SAT problem is to determine if there exists an assignment* $A \in \{0,1\}^n$ *for* B*, such that all* c_i *are true and hence* C *is satisfied.*

Theorem 2.5.1. *$\mathcal{MQ}$-$\mathbb{F}_2$ is $\mathcal{NP}$-complete.*

Proof. We need to transform our 3-SAT syntax to a $\mathbb{F}_2$ formulation, where $l_i, l_j, l_k \in L$, $b_i \in B$, $\vee$ is logical **OR** and $+$ is addition modulo 2. Let $X = (x_1, x_2, \ldots x_n)$ be variables over $\mathbb{F}_2$.

(i) Replace $(l_i \vee l_j \vee l_k)$ with $(l_i + l_j + l_k + l_i \cdot l_k + l_i \cdot l_j + l_j \cdot l_k + l_i \cdot l_j \cdot l_k)$ in all clauses

(ii) Replace $(l_i \vee l_j)$ with $(l_i + l_j + l_i \cdot l_j)$ in all clauses

(iii) For each boolean variable b_i, set $b_i = x_i$ and $\overline{b_i} = x_i + 1$.

(iv) For each transformed clause c_i', construct an equation $e_i = 1$.

Now, each equation e_i has at most degree 3. Introduce $\binom{n}{2}$ new variables $y_{i,j}, 1 \leq i < j \leq n$. Introduce $\binom{n}{2}$ new equations $\overline{e_{i,j}} = y_{i,j} + x_i \cdot x_j = 0, 1 \leq i < j \leq n$. Substitute $y_{i,j}$ for every instance of $x_i \cdot x_j$ in all equations e_i. Now since every $x_i \cdot x_j \cdot x_k$ has the form $y_{i,j} \cdot x_k$ (or e.g. $x_i \cdot y_{i,j}$ depending on the order of substitution), all equations have at most degree 2. We now have $m + \binom{n}{2}$ equations in $n + \binom{n}{2} = \binom{n+1}{2}$ variables, and if there is a simultaneous solution for this new set of equations, we have a solution for the original 3-SAT problem. Since each step involved only requires polynomial time and space, we have reduced the 3-SAT problem to an $\mathcal{MQ}$-$\mathbb{F}_2$ problem in polynomial time, i.e. 3-SAT $\leq_{poly}$ $\mathcal{MQ}$-$\mathbb{F}_2$ which implies that $\mathcal{MQ}$-$\mathbb{F}_2$ is $\mathcal{NP}$-complete. □

At first glance, this seems to be the death of trying to solve large systems of polynomial equations in general, and algebraic cryptanalysis in particular. But the formalism above regards *the general* case, i.e. all cases. This is equivalent to the unsolvability of the quintic by radicals; there is no general formula for the general quintic over the rationals in terms of radicals. But we know there exist quintics that are easily solvable this way. In the same way, there are systems of polynomial equations that can be solved in polynomial time.

For instance, there are hugely overdefined systems of multivariate quadratic polynomial equations that can always be solved in polynomial time: and that is the case when we can substitute all quadratic monomials with new variables in such a way that the new system has full rank. This system can be solved with Gaussian elimination which has complexity $O(N^3)$, where N is the total number of variables.

In this thesis we show that we encounter both sides of this. LILI-128 produces seemingly random polynomials, and shows exponential complexity. With only 40 unknowns, this becomes very hard to solve in a reasonable time. KASUMI on the other hand, produces a highly structured equation system and we can find solutions for the unknown key bits in a system of equations of near 10000 variables.

In general, estimating complexity for solving overdefined systems of polynomial equations over $\mathbb{F}_2$ is difficult. There are upper bounds when we approximate the problem to systems of "random" equations. The worst case is polynomial in 2^n , i.e. exponential in n and we refer to [23] for a discussion of this.

3 Methods of Algebraic Cryptanalysis

In this chapter we do a short survey of cryptanalytic techniques and variants. The list is not meant to be complete, but shows different and quite general ways of attacking systems of multivariate polynomial equations.

3.1 Linearization methods

Cryptanalytic linearization does not necessarily mean *linearization* in the strong sense. One may use linearization methods just to lower the polynomial degree.

3.1.1 Plain linearization

A method for solving non-linear multivariate polynomial equations is linearization. In it's simplest form we just introduce new variables for all non-linear terms, and solve it by Gauss elimination. This means we need an overdefined system of equations. Consider a system of polynomial equations in n variables over $\mathbb{F}_2$ of degree d. As a worst case, we have $T = \sum_{i=2}^{d} \binom{n}{i}$ monomials of degree ≥ 2, and will need to introduce T new variables. As an upper bound we will need $T + n$ linearly independent equations to be able to solve it using Gauss-elimination, i.e. hugely overdefined.

Example 3.1 (**Simple linearization**). *Consider the problem of finding a solution to the following system of polynomial equations of degree 2 over $\mathcal{F}_{\in}$.*

$$\begin{aligned} xy + z + 1 &= 0 \\ xy + xz + y &= 0 \\ yz + x + 1 &= 0 \end{aligned}$$

Simple linearization will not work, because we have three different monomials of degree two. By introducing three new variables we will end up with an equation system not of full rank. We need three more linearly independent equations. So assume we can get our hands on such.

$$\begin{aligned} xy + z + 1 &= 0 \\ xy + xz + y &= 0 \\ yz + x + 1 &= 0 \\ xz + x + y + z &= 0 \\ xy + yz + x + y + 1 &= 0 \\ xz + yz + y + 1 &= 0 \end{aligned}$$

Now, set

$$\begin{aligned} t &= xy \\ u &= xz \\ v &= yz \end{aligned}$$

and we end up with the following system

$$\begin{array}{cccccccccccccc}
t & & & & & & & & & + & z & + & 1 & = 0 \\
t & + & u & & & & + & y & & & & & & = 0 \\
 & & & v & + & x & & & & & & + & 1 & = 0 \\
 & & u & & & + & x & + & y & + & z & & & = 0 \\
t & & & + & v & + & x & + & y & & & + & 1 & = 0 \\
 & & u & + & v & & & + & y & & & + & 1 & = 0
\end{array}$$

Solving this system in the usual way returns the consistent solution

$$t = 1, u = 0, v = 0, x = 1, y = 1, z = 0$$

3.1.2 Relinearization

Relinearization was first presented in [24] and applied to cryptanalysis of HFE public key crypto system. Relinearization is a technique for lowering the amount of equations needed. Basically the trick is to note that xy and xz are linearly independent but they are still algebraically related by the variable x, and we use this to generate new equations.

Example 3.2 (Relinearization). *Consider the same problem as above, of finding a solution to the following system of polynomial equations of degree two over $\mathcal{F}_2$*

$$\begin{aligned}
xy + z + 1 &= 0 \\
xy + xz + y &= 0 \\
yz + x + 1 &= 0
\end{aligned}$$

Now, set

$$\begin{aligned}
t &= xy \\
u &= xz \\
v &= yz
\end{aligned}$$

and simplify. We end up with the following system, where t, u and v are parametrized by x, y and z.

$$\begin{aligned}
t &= z + 1 \\
u &= y + z + 1 \\
v &= x + 1
\end{aligned}$$

But in the definition of t, u and v above we see that

$$\begin{aligned}
tu &= tv \\
tv &= uv \\
zt &= yu \\
zt &= xv
\end{aligned}$$

This gives us four more equations in x, y, z, t, u and v. We have now seven equations in six variables, but they are not algebraically independent. Substituting the parametric solution into these equations gives

$$0 = tu + tv = (z+1)(y+z+1) + (z+1)(x+1) = u + v + x + y$$
$$0 = tv + uv = (z+1)(x+1) + (x+1)(y+z+1) = t + y$$
$$0 = zt + yu = z(z+1) + y(y+z+1) = v$$
$$0 = zt + xv = 0 \text{ ,algebraically dependent}$$

We now have six linear equations in six variables.

$$\begin{aligned} t + z + 1 &= 0 \\ u + y + z + 1 &= 0 \\ v + x + 1 &= 0 \\ u + v + x + y &= 0 \\ t + y &= 0 \\ v &= 0 \end{aligned}$$

And solving this system in the usual way returns the consistent solution

$$t = 1, u = 0, v = 0, x = 1, y = 1, z = 0$$

3.1.3 XL

The XL, or eXtended Linearization (see [25]), is a method based on Relinearization for solving systems of quadratic equations. In short: consider a system of quadratic equations

$$p_1(x_1, \ldots, x_n) = p_2(x_1, \ldots, x_n) = \ldots, p_m(x_1, \ldots, x_n) = 0$$

We multiply each of these equations with all monomials m_i up to a prescribed degree D-2, and try to solve the resulting system of equations of type

$$m_i \cdot p_j(x_1, \ldots, x_n) = 0$$

by linearization. Hopefully this new system returns univariate polynomials, which can then be eliminated from the original system, and the process is repeated.

3.1.4 XSL

The XSL algorithm (see [26], [27]), is a linearization algorithm custom made to take advantage of structure and sparsity of equations from block ciphers of a particular kind. To be effective, the system of equations must have a special form, such as equations from S-boxes with overdefined system of equations, equations from repeated linear diffusion layers and so on.

Both XL and XSL are termed *redundant versions* of Gröbner algorithms (see [28], [29]), but they have the advantage over computer algebra systems that they do not need "true polynomials" when implementing the algorithms.

3.1.5 MutantXL

MutantXL [30] is an algorithm that takes advantage of low degree polynomials in the system. Let

$$p_1(x_1,\ldots,x_n) = p_2(x_1,\ldots,x_n) = \ldots, p_m(x_1,\ldots,x_n) = 0$$

generate the ideal $I \in K[x_1,\ldots,x_n]$. A polynomial $g \in I$ can then be represented as

$$g = h_1p_1 + h_2p_2 + \ldots + h_mp_m$$

The *level* of this representation is defined as $max\{deg(h_ip_i) :$ all terms h_ip_i in $g\}$ The *level of g* is defined to be the minimum level of all representations of g. The polynomial g is called a *mutant* with respect to $(p_1,\ldots,p_m)$ if the degree of g is less than the level of g. We see that g can not be represented as a linear combination of m_ip_i where $m_i \in K[x_1,\ldots,x_n]$ is a monomial and $deg(m_ip_i) \leq deg(g)$ as is done in the XL algorithm. The MutantXL algorithm is defined as follows:

Definition 3.1 (MutantXL algorithm). *Let* $F = (p_1, p_2, \ldots, p_m)$

1. *Interreduce F. Set* $d = e = min\{deg(p_i) : p_i \in F\}$*, and set G=F.*
2. *Linearize G and reduce to row echelon form.*
3. *If univariate polynomials are found, solve and eliminate solved variables in F. If this solves the entire system we are finished, else go to step* 1.
4. *Form the set of mutants with respect to F:* $M = \{deg(\tilde{p_i}) < e :$ *for all* $\tilde{p_i} \in G\}$
5. *If* $M \neq \emptyset$*, for each* $g_i \in M$ *create the set* $M_{g_i} = \{m_jg_i :$ *all monomials* m_j *where* $deg(m_j) = d - deg(g_i)\}$*. For each* $g_i \in M$*, replace* g_i *in* G *with the elements in* M_{g_i}*. Set* $e = min\{deg(g_i) : g_i \in M\} + 1$ *and go to step 2.*
6. *For each* $g_i \in M$ *create the set* $M_{g_i} = \{x_jg_i : 1 \leq j \leq n\}$*. For each* $g_i \in M$*, replace* g_i *in* G *with the elements in* M_{g_i}*. Set* $e = d = d + 1$ *and go to step 2.*

Some versions of MutantXL algorithms seems to be comparable in speed to advanced Gröbner basis algorithms and is thus an important addition. An application of mutant strategies, or similar strategies, may possibly improve on Gröbner basis computations.

3.2 SAT solving

We have previously mentioned the *Satisfiability* problem. This is the problem of deciding whether a given Boolean formula can be evaluated to TRUE. Problem such as graph coloring or scheduling problems can easily be encoded as a satisfiability problem. In complexity theory, the *Boolean Satisfiability Problem*, or SAT, is a decision problem where the boolean expression to be evaluated is in Conjunctive Normal Form (CNF), i.e. a Boolean expression using only logical AND, OR, NOT, variables and parentheses.

$$\text{Is } \phi = (x_i \vee x_j \vee \ldots \vee x_k) \wedge \ldots \wedge (x_v \vee x_w \vee \ldots \vee x_z) \text{ SAT?}$$

SAT is known to have exponential running time. *SAT solvers* comes in different flavors, e.g. conflict-driven or look-ahead, but basically they guess values and look for inconsistencies in the

systems of clauses. If inconsistencies are found, they do a *backtrack*, i.e. take a few steps back to resolve the inconsistency, then continue on another path. An interesting "Gluing and Agreeing" approach is found in [31]. This works roughly as follows: Given a system of equations

$$\begin{aligned} f_1(X) &= 0 \\ f_2(X) &= 0 \\ &\vdots \\ f_l(X) &= 0 \end{aligned}$$

we encode this as sets, or symbols

$$S_i(X_i, V_i)$$

where X_i are the variables contained in the expression f_i and V_i is the satisfying vector, i.e. values for which f_i is SAT.

Example 3.3. *Let an equation be*

$$f_i(x_1, x_2, x_3) = x_1 \cdot x_2 + x_3$$

The corresponding symbol would then be

S_i	x_1	x_2	x_3
a_1	*0*	*0*	*0*
a_2	*0*	*1*	*0*
a_3	*1*	*0*	*0*
a_4	*1*	*1*	*1*

Agreeing is for two or more symbols to agree on their satisfying vectors, deleting those vectors which lead to contradictions.

Example 3.4 (**Agreeing**)**.** *Given two symbols* S_i *and* S_j

S_i	x_1	x_2	x_3
a_1	0	0	0
a_2	0	1	0
a_3	1	0	0
a_4	1	1	1

S_j	x_3	x_4	x_5
b_1	1	0	0
b_2	1	0	1
b_3	1	1	1

All vectors leading to contradictions are deleted, resulting in

S_i	x_1	x_2	x_3
a_4	1	1	1

S_j	x_3	x_4	x_5
b_1	1	0	0
b_2	1	0	1
b_3	1	1	1

Gluing means to create new symbols from earlier symbols, i.e. $S_3 = S_1 \circ S_2$. So the Gluing-Agreeing algorithm is the following: glue intermediate symbols with other symbol and agree on the intermediate equation system.

Example 3.5 (**Gluing**). *Given two symbols S_i and S_j as above, the new symbol $S_i \circ S_j$ will be*

$S_i \circ S_j$	x_1	x_2	x_3	x_4	x_5
c_1	*1*	*1*	*1*	*0*	*0*
c_2	*1*	*1*	*1*	*0*	*1*
c_3	*1*	*1*	*1*	*1*	*1*

In [31] another step in combination with the guessing strategy, called *Agreeing2* reduces the steps in the previous Agreeing step in that it marks groups of *common projections*. Storing the assignment vectors as tuples, it is more efficient when looking for contradictions.

Example 3.6 (**Agreeing**). *Given two symbols S_i and S_j*

S_i	x_1	x_2	x_3
a_1	*0*	*0*	*0*
a_2	*0*	*1*	*0*
a_3	*1*	*0*	*0*
a_4	*1*	*1*	*1*

S_j	x_3	x_4	x_5
b_1	*0*	*0*	*0*
b_2	*1*	*0*	*0*
b_3	*1*	*0*	*1*
b_4	*1*	*1*	*1*

The corresponding assignment vectors are stored as tuples

$$S_i : (a_1, a_2, a_3; b_1)$$
$$S_j : (a_4; b_2, b_3, b_4)$$

Now, marking a_1, a_2 and a_4 also marks b_2, b_3 and b_4.
We see that information is propagated through tuples, and this leaves us with

S_i	x_1	x_2	x_3
a_3	1	0	0

S_j	x_3	x_4	x_5
b_1	0	0	0

3.3 Fast Algebraic Attack

Fast algebraic attack (see [32], [33]) is a technique used on certain streamciphers. Let K be the initial state of the Linear Feedback Shift Registers (LFSRs) and let $K_t = L^t(K)$ be the state after t regularly clocked shifts, and let z_t be the corresponding output. We need two conditions: 1:we can build a system of equations from one function F.

$$\begin{aligned} F(K_{t_1}, K_{t_1+1}, \ldots, K_{t_1+r}, z_{t_1}, z_{t_1+1}, \ldots, z_{t_1+r}) &= 0 \\ F(K_{t_2}, K_{t_2+1}, \ldots, K_{t_2+r}, z_{t_2}, z_{t_2+1}, \ldots, z_{t_2+r}) &= 0 \\ &\vdots \\ F(K_{t_N}, K_{t_N+1}, \ldots, K_{t_N+r}, z_{t_N}, z_{t_N+1}, \ldots, z_{t_N+r}) &= 0 \end{aligned}$$

and 2: F can be split

$$\underbrace{F(X,z)}_{deg(F)=d} = \underbrace{G(X)}_{deg(G)=d} + \underbrace{H(X,z)}_{deg(H)<d}$$

where G is independent of z. This means we can write the above as

$$\begin{aligned}
F_{t_1}(X, z_{t_1}, \ldots, z_{t_1+r}) &= G_{t_1}(X) + H_{t_1}(X, z_{t_1}, \ldots, z_{t_1+r}) = 0 \\
F_{t_2}(X, z_{t_2}, \ldots, z_{t_2+r}) &= G_{t_2}(X) + H_{t_2}(X, z_{t_2}, \ldots, z_{t_2+r}) = 0 \\
&\vdots \\
F_{t_N}(X, z_{t_N}, \ldots, z_{t_N+r}) &= G_{t_N}(X) + H_{t_N}(X, z_{t_N}, \ldots, z_{t_N+r}) = 0
\end{aligned}$$

The following proposition is needed

Proposition 3.3.1. *There exists an integer* $T \geq 1$ *and coefficients* $\alpha_0, \ldots, \alpha_T \in \mathbb{F}_2$*, such that*

$$\sum_{i=0}^{T} \alpha_i \cdot G_{t+i}(X) \equiv 0, \ \forall t \geq 0$$

Proof. $G_j(X)$ are Boolean multivariate polynomials of degree d in n variables. The number of possible different monomials of degree $\leq d$ in n variables have an upper bound of $\overline{T} = \sum_{i=0}^{d} \binom{n}{d}$. Then if $G_i \neq G_j$ when $i \neq j$ the polynomials $G_0, G_1, \ldots, G_{\overline{T}}$ are linearly dependent, and there exist numbers $\alpha_0, \ldots \alpha_T \in \mathbb{F}_2$ such that

$$\sum_{i=0}^{T} \alpha_i \cdot G_i(X) \equiv 0$$

where $T \leq \overline{T}$. Let t be arbitrary. Substituting $X_t = L^t(X)$ for X in the above we get

$$\sum_{i=0}^{T} \alpha_i \cdot G_i(X_t) = \sum_{i=0}^{T} \alpha_i \cdot G_i(L^t(X)) = \sum_{i=0}^{T} \alpha_i \cdot G_{t+i}(X)$$

Since t was arbitrary it holds for all t.

□

Having found such a sequence of α_i's, we can write our system of polynomials as

$$\sum_{i=0}^{T} \alpha_i \cdot F_{t+i}(X, z) = \sum_{i=0}^{T} \alpha_i \cdot G_{t+i}(X) + \sum_{i=0}^{T} \alpha_i \cdot H_{t+i}(X, z) = 0$$

Since this holds for all t, we now have a new set of equations

$$\begin{aligned}
\sum_{i=0}^{T} \alpha_i \cdot H_{t_1+i}(X, z) &= 0 \\
\sum_{i=0}^{T} \alpha_i \cdot H_{t_2+i}(X, z) &= 0 \\
&\vdots \\
\sum_{i=0}^{T} \alpha_i \cdot H_{t_N+i}(X, z) &= 0
\end{aligned}$$

with lower degree than the original.

3.4 Gröbner bases techniques

Since the birth of Göbner bases in 1965, there have been many proposals for improvements of the Buchberger algorithms. Some of these are implemented in well known computational algebra systems as Magma, Singular, Maple etc. The most efficient implementations are known as F4 and F5 (see [14] [15]), both due to Jean-Charles Faugère. F4 and F5 where both used to crack the HFE challenge [34]. A specialization to Gröbner bases over $\mathbb{F}_2$ is implemented in Polybori [1] [35] and has shown improvements in certain types of problems. SAGE [2] is an open-source mathematics system with computational algebra functionality and comes with the Polybori framework. SAGE does not seem to have implemented the newest achievements regarding Gröbner bases construction, but open-source means that researchers, and all other interested, have full access to source code which can be a great asset.

[1] http://polybori.sourceforge.net/
[2] http://www.sagemath.org

4 Implementation

To get a deeper insight into algebraic cryptanalysis with Gröbner bases and computational algebra, it was decided to start the implementation from scratch. *Python* [1] was chosen as the programming language because of its usefulness in rapid prototyping. Since the literature concerning implementation issues was sparse , we adopted an *evolutionary prototyping* approach to the work at hand, i.e. we would rapidly implement a model, refine it until we reached the model's limitation, then change the model and so on. Developing a computational model for multivariate polynomials suitable for our needs was a difficult task. Most mathematical libraries for programming languages come with polynomial models, but few are designed for this particular use. We needed a polynomial implementation that supports computation in a Boolean ring, which has a small memory footprint, must support fast addition and multiplication and is easily written to, and read from, files. We ended up with a computational model that works surprisingly well. It also takes care of false solutions, i.e. solutions lying in the closure $\overline{\mathbb{F}_2}$, since reduction by field equations is implicit. To speed up the computations, *psyco*[2] was used. Psyco is a kind of JIT (just-in-time) compiler for python, and the improvements in speed was considerable. We also used the python module PP (or Parallel Python) [3] for efficient use of processors with multiple cores. This also required rewriting parts of the code so it could be run in parallel.

4.1 Polynomial model

We went through a large array of polynomial models, but we do not elaborate on this since they all performed poorly when considering memory footprint, computational speed or both. They were much too slow on even the simpler problems. A polynomial is an array of monomials, so the most important mission is to develop a computationally efficient monomial model. A monomial contains quite a lot of information: first we have the variables, which are present in the monomial or not, and all variables have degrees. The monomial must also support a monomial ordering rule. Putting all of this together we end up with quite a lot of numbers, flags etc., to handle. A multivariate polynomial in thousands of variables may have tens of thousands of monomials, and it is easy to see that shortage of memory will be a major problem. The other important thing is computational speed. Two operations that will be happening frequently are monomial comparison and monomial multiplication. If our monomial model consists of arrays of numbers representing variables and degrees, we see that there is a lot to consider when doing these two operations. Besides memory footprint and computational speed, the third cumbersome part was adding field equations. Field equations must be added to the system of polynomial equations to avoid false solutions. Having a large number of variables, means we must add the same amount of field equations to the system thus slowing down polynomial division. For a long time this

[1] http://www.python.org
[2] http://psyco.sourceforge.net
[3] http://www.parallelpython.com

problem would not go away. After many unsuccessful attempts, one polynomial model clearly stood out both in speed and memory footprint. The model is very simple and solves (relatively speaking) our three computational problems. If we model a monomial as a binary number, where a binary **1** in place n represents the variable X_n, and a binary **0** represents a missing variable we have captured almost all the monomial information in a Boolean ring - after reduction by field equations (remember that $X^s = X$, for $s > 0$ in a Boolean ring). This is probably a very old idea, but literature is sparse on this issue. Modeling monomials as binary numbers, we can perform monomial multiplication as bitwise operations, and this is what the processor is good at. The memory imprint will be minimal since we can model any 32 variable monomial with just a 32-bit integer. Python comes with multiprecission integers, so monomials with any amount of variables are easily constructed. Doing it this way, Lex order (see 2.11) is now just an integer comparison, and monomial total degree is the number of binary **1**s in the integer representation of the monomial. Counting **1**s can be performed quite effectively by an efficient little algorithm found in [36], and it is reproduced below for aesthetic reasons.

```
unsigned int v; // count the number of bits set in v
unsigned int c; // c accumulates the total bits set in v
for (c = 0; v; c++)
{
  v &= v - 1; // clear the least significant bit set
}
```

Listing 4.1: Counting bits set the Brian Kernighan's[36] way

In the following examples, we will be working in $\mathbb{F}_2[x_1, x_2, x_3, x_4, x_5]$, i.e. a polynomial ring with only 5 variables.

Example 4.1 (**Monomial as binary numbers**). *Since our ring contain only five variables, we can think of all monomial integers as 5-bit integers. Then a monomial, for instance* $x_1x_2x_5$*, can be written as*

$$11001 = 25,$$

i.e. a ***1*** *in places* 1,2 *and* 5 *from left to right, zero in all other places. It has degree* 3 *since there are 3* ***1****s in the binary representation of* 25*. The monomial* 1*, is represented by a zero. It is easy to see that this scheme follows simple Lexicographic Ordering, since*

$$x_1 = 10000_2 > x_2 = 01000_2 > \ldots > x_5 = 00001_2 > 1 = 00000_2.$$

If we also know the number of "ones" in the monomial representation, we have Graded Lexicographic Ordering.

Example 4.2 (**Monomial multiplication**). *Monomial multiplication is bitwise* ***OR****. Let* $X_2X_3X_5$ *be a monomial. We know that*

$$X_2 \cdot X_2X_3X_5 = X_2X_3X_5 \text{ because of the field equation } X_2^2 + X_2 = 0.$$

Since

$$X_2 \rightarrow 01000_2 = 8_{10} \text{ and}$$
$$X_2X_3X_5 \rightarrow 01101_2 = 13_{10}$$

we have that

$$\begin{aligned} & 01000_2 \\ \textbf{OR}\ & 01101_2 \\ = {} & 01101_2 \rightarrow X_2X_3X_5. \end{aligned}$$

In the same way we have that

$$X_1 \cdot X_2X_3X_5 = X_1X_2X_3X_5,$$

and in the binary case

$$\begin{aligned} & 10000_2 \rightarrow X_1 \\ \textbf{OR}\ & 01101_2 \rightarrow X_2X_3X_5 \\ = {} & 11101_2 \rightarrow X_1X_2X_3X_5 \end{aligned}$$

Example 4.3 (Monomial division). *Let m_1, m_2 be non-zero monomials modeled as binary numbers. Then m_2 divides m_1 if and only if*

$$(m_1 \textbf{ AND } m_2) \textbf{ XOR } m_1 = 0 \text{ , or equivalently}$$
$$m_1 \textbf{ AND } m_2 = m_2.$$

*To see this, let e.g. $m_1 = x_2x_3x_5$ and $m_2 = x_2x_5$. We see that m_2 divides m_1 because m_2 does not contain other variables that are contained in m_1. In the binary case we see that the number $(m_1$ AND $m_2)$ has **1**s only in places that are common to m_1 and m_2. So if this number equals m_2 then no other "ones" have been lost in the bitwise AND. If we know that m_2 divides m_1, then $(m_1$ XOR $m_2)$ is the result from division, since $(m_1$ XOR $m_2)$ are the **1**s left in m_1 after removing the corresponding **1**s in m_2.*

We do not want to count one-bits too often, so we group together all monomials with the same one-count, i.e. degree. To help keep track of monomial degrees, we modeled the polynomial as a *dictionary*, more commonly referred to as a *hash-map* or a key-value mapping in other programming languages. The key is an integer representing degree and the value is a list of ordered integers representing the monomials with this degree. So an n-key ($n > 0$) points to a list of monomials with degree n, and the zero-key is used for telling the polynomial multidegree, i.e. pointing to the key where we can find the leading term (LT). An example of representing an n-degree polynomial is given below.

Binary_Polynomial={	[0]: n	// monomial MAX_DEGREE
	[1]: $[X_1, X_2, \ldots]$	// LINEAR MONOMIALS
	[2]: $[X_1X_2, X_1X_3, \ldots]$	// QUADRATIC MONOMIALS
	.	.
	.	.
	.	.
	[n]: [...]	// DEGREE n MONOMIALS }

Example 4.4. *Let* $p = X_1X_3X_5 + X_2X_4X_5 + X_1X_2 + X_2X_4 + X_4X_5 + X_2 + X_3 + X_5 + 1$ *be a polynomial. As a first step, this will be represented as:*

Binary_Polynomial={	[0]: 3	*// monomial MAX_DEGREE*
	[1]: $[X_2, X_3, X_5, 1]$	*// LINEAR MONOMIALS*
	[2]: $[X_1X_2, X_2X_4, X_4X_5]$	*// QUADRATIC MONOMIALS*
	[3]: $[X_1X_3X_5, X_2X_4X_5]$	*// DEGREE n MONOMIALS }*

and as a data structure with integers, it will be represented as:

Binary_Polynomial={	[0]: 3	*// monomial MAX_DEGREE*
	[1]: [8, 4, 1, 0]	*// LINEAR MONOMIALS*
	[2]: [24, 10, 3]	*// QUADRATIC MONOMIALS*
	[3]: [21, 11]	*// DEGREE n MONOMIALS }*

We see that the leading term is the first entry in the list which key is pointed to by the zero-key value.

Each list of monomials in the dictionary uses binary search methods for finding, adding or deleting monomials in the list.

4.2 Main algorithm

Solving polynomial equations is a balancing act, and since processing power and memory are scarce resources it is important to do the right things at the right time, and just enough of them. Both our algebraic attacks are known-plaintext attacks, i.e. we know the input and the output - only the key is unknown. Due to hardware constraints we assume to know part of the key and treat the unknown key-bits as binary variables. The algorithm halts if it has found all the unknown key-bits, so we need to tell the algorithm which variables we seek solutions for because intermediate variables may be introduced to lower the polynomial degrees. In the algorithm, KB is the set of variables we are seeking a solution for. F is the current set of polynomials to process. G is our current Gröbner base, G_{LIN} is all linear polynomials in the current base and G_{SOL} is all univariate polynomials, i.e. solutions.

Algorithm 4.1 Main algorithm

1: $KB = \{x_m, x_{m+1}, \ldots, x_n\}$, unknown key-bits
2: $F = \emptyset$, set of work-polynomials
3: $G = \emptyset$, current base
4: $G_{LIN} = \emptyset$, current linear base
5: $G_{SOL} = \emptyset$, solved variables
6: $B = \emptyset$, critical pairs
7: $P_{source} \neq \emptyset$, initial polynomial source
8: **repeat**
9: $F = \{f_1, f_2, \ldots, f_l\} =$ **read_polynomials**(P_{source})
10: $F = \{\overline{f_1}, \overline{f_2}, \ldots, \overline{f_t}\} =$ **pre_elimination**(G_{LIN}, G_{SOL}, F)
11: **repeat**
12: $\{\overline{g_1}, \overline{g_2}, \ldots, \overline{g_s}\} =$ **reduce_mod_G**(F, G)
13: **for all** $\overline{g_i}$ **do**
14: **if** $LT(\overline{g_i})$ divides $LT(g_i)$ for some $g_i \in G$ **then**
15: Remove g_i from G
16: Add g_i to F
17: **end if**
18: $G = \{\overline{g_i}\} \cup G$
19: Find critical pairs involving $\overline{g_i}$, add to B
20: **if** $deg(\overline{g_i}) = 1$ **then**
21: $G_{LIN} = \{\overline{g_i}\} \cup G_{LIN}$
22: **Gauss_Jordan_eliminate**(G_{LIN})
23: $G_{SOL} = \{p : p = x_i$ or $p = x_i + 1$, $\forall p \in G_{LIN}\}$
24: **pre_elimination**$(G_{LIN}, G_{SOL}, G \cap G_{LIN})$
25: **end if**
26: **end for**
27: **until** $F = \emptyset$ or $KB \subseteq G_{SOL}$
28: **until** Run out of polynomials or $KB \subseteq G_{SOL}$
29: **return** G_{SOL}

The algorithm starts with the sub-function **read_polynomials**(). It feeds the algorithm with polynomials from some source. These sources may take several forms, e.g. they compute polynomials on the fly taking into account the results so far, they compute S-polynomials from the current Gröbner base based on the d-truncation limit, or they simply reads from a file of pre-computed polynomials. It is important to feed the algorithm in a balanced way, such that we can take near realtime advantage of current states.

The **pre_elimination**() sub-function eliminates variables in polynomials received from **read_polynomials**(). In the case where all input-polynomials are pre-computed and read from file, we get polynomials in variables we may already have a simple linear expression for. Instead of eliminating these variables in the division algorithm, we handle the following three cases more efficiently in **pre_elimination**(), namely variables with solution ONE ($X_s + 1 = 0$), variables with solution ZERO ($X_s = 0$) and linear polynomials in two variable, i.e. $X_s + X_k = 0$. We construct three binary numbers: ZERO_SUPER_MONOMIAL, ONE_SUPER_MONOMIAL and TWO_SUPER_MONOMIAL with a supporting substitution dictionary(or hash-map). If the algorithm reduces a polynomial to one of these three cases, say a polynomial is reduced to $X_s+1 = 0$, we multiply, or bitwise-or, ONE_SUPER_MONOMIAL with the binary expression for X_s. In this way, ZERO_SUPER_MONOMIAL consist of all variables with solution **0**, ONE_SUPER_MONOMIAL consists of all variables with solution **1** and TWO_SUPER_MONOMIAL consists of all variables which can be substituted by another variable. The TWO_SUPER_MONOMIAL is backed by a dictionary (or hash-map) which tells which variable is to be substituted by what. During the **pre_elimination**() of an input-polynomial, each monomial m_i is checked and handled according to the following rules:

1. If m_i bitwise-AND ZERO_SUPER_MONOMIAL > 0, then the monomial contains a variable with solution ZERO, and the monomial can thus be removed from the polynomial.
2. If m_i bitwise-AND ONE_SUPER_MONOMIAL > 0, then the monomial contains at least one variable with solution ONE. The variable(s) can thus be removed from the monomial by doing $m_i = (m_i$ bitwise-AND ONE_SUPER_MONOMIAL$)$ bitwise-XOR m_i. We need to count ONE-bits since there may be more than one variable in the m_i that has solution ONE.
3. If MON bitwise-AND TWO_SUPER_MONOMIAL > 0, then the monomial contains a variable, which is to be substituted by another. If variable X_s is to be substituted by X_t, found in the substitution dictionary then we do MON = MON bitwise-XOR X_s bitwise-OR X_t. The monomial degree is unchanged.

The **reduce_mod_G**() sub-function is the division algorithm which we take a closer look at below. It reduces a set of polynomials using multivariate polynomial division. Since this is a resource intensive operation, it is done in parallel whenever we can save computation time. When performing parallel division we have to take into consideration that results may not be orthogonal, i.e. results from different instances of the division algorithm they may be pairwise divisible.

The **Gauss_Jordan_eliminate** sub-function is just a plain Gauss-Jordan elimination, which tries to reduce the linear polynomials to reduced row echelon form.

The **eliminate_base** sub-function reduces the current Gröbner basis. If an input-polynomial,

say P, reduces to a degree less than the current maximum degree, we loop through the current Gröbner basis to see if P will reduce any base-polynomials, i.e. if any leading term in the current Gröbner basis is divisible by the leading term of P. If such a leading term is found, the corresponding polynomial is removed from the current Gröbner basis and treated as an input polynomial.

The algorithm halts when we have found solutions for all variables in KB or we run out of polynomials from the source/feed.

4.3 Division algorithm

We have two variants of the division algorithm. The common multivariate division algorithm (see 2.1) for binary rings takes input (p, P), where p is a polynomial to be reduced, and P is a set of polynomials. It loops through all monomials in p to see if any monomial m is divisible by any leading terms of polynomials in P. If a monomial m is divisible by a leading term $LT(g)$ of a polynomial g in P, it multiplies g with a suitable monomial h such that $LT(g) * h$ equals m, and then adds $h * g$ to p. The drawback with this approach is that it potentially ruins the sparsity and structure of the current Gröbner basis because large and dense polynomials may be repeatedly added to all other polynomials. Our other version of division algorithm 4.2 returns when it encounters a monomial not divisible by any leading terms. Cryptanalysis of ciphers, which produce a sparse and structured set of polynomials, like KASUMI, run much faster with this type of division algorithm. Any Gröbner basis algorithm will spend most of it's time executing a division algorithm and it is thus the most important sub-function to optimize. A lot of effort was invested in optimizing the function as much as possible. The function is very CPU time consuming needy and is run in parallel (wherever practical), taking advantage of as much computing resources as possible.

We avoid doing costly polynomial operations, so we just keep track of monomials by a list of indices and assemble the resulting polynomial from these indices. The following example explains the inner workings:

Example 4.5. *Let* $p = x_1x_2 + x_1x_3 + x_1x_4 + x_1 + x_4 \in \mathbb{F}_2[x_1, x_2, x_3, x_4]$ *be a polynomial to be reduced and let*

$g_1: \quad x_2x_4 + x_3$

$g_2: \quad x_1 + x_2 + x_4$

$g_3: \quad x_3 + x_4$

be the current base. The bar *shows which monomial the polynomial index is pointing to, and we mark with* bold *the leading term for the polynomial last added to the list. We start with the polynomial* p*, and move the index from* x_1x_2 *to* x_1x_3*. Current monomial to be eliminated is thus* x_1x_2*, and we set* $Cm = x_1x_2$*. The base contains no polynomial with* x_1x_2 *as leading term, but* x_1x_2 *is divisible by the leading term of* g_2*. We multiply* g_2 *with* x_2*, move indexes, and add it to the list.*

$p:$	x_1x_2	$+$	$\overline{x_1x_3}$	$+$	x_1x_4				$+$	x_1				$+$	x_4			
$x_2 \cdot g_2:$	$\mathbf{x_1x_2}$					$+$	$\overline{x_2x_4}$				$+$	x_2						

Comparing the monomials for all the current indices, we see that x_1x_3 *is the largest. We do not*

Algorithm 4.2 Truncated Division Algorithm

Input: $p \in K[x_1, x_2, \ldots, x_n]$
Input: $G = (g_1, g_2, \ldots, g_t)$ *# current polynomial base*
Output: $p \bmod G$
$Cm := LT(p)$ *# current monomial*
$P_{ind} := \{(p, 0)\}$ *# set of pairs of polynomial and polynomial index*
$halt = FALSE$
repeat
    repeat
        $\#p[i]$ *is the i'th monomial according to monomial ordering*
        $Cm = max_{>}(p[i] : \forall (p, i) \in P_{ind})$
        $l = |\{p[i] : \forall (p, i) \in P_{ind}$ *where* $p[i] = Cm\}|$
        for all $(p, i) \in P_{ind}$ where $p[i] = Cm$ **do**
            if $p[i + 1]$ valid **then**
                $i = i + 1$ *# increase index*
            else
                Remove (p, i) from P_{ind}
            end if
        end for
    until l is odd, or $P_{ind} = \{\}$ *# if* l *even - they cancel*
    if $Cm = LT(g_i)$ for some $g_i \in (LT(g_1), LT(g_2), \ldots, LT(g_t))$ **then**
        Add $(g_i, 1)$ to P_{ind}
    else if $LT(g_i)$ divides Cm for some $g_i \in G$ **then**
        Find $h \in K[x_1, x_2, \ldots, x_n]$ such that $h \cdot LT(g_i) = Cm$
        Add $(h \cdot g_i, 1)$ to P_{ind}
    else
        $halt = TRUE$ *# Found irreducible monomial*
    end if
until $halt = TRUE$ or $P_{ind} = \emptyset$
$p_{new} = 0$ *# empty polynomial*
If needed, adjust indices so we do not lose monomials
for all $(p, i) \in P_{ind}$ **do**
    while $p[i]$ valid **do**
        $p_{new} = p_{new} + p[i]$
    end while
end for
return p_{new}

have a leading term in the base-polynomials with this value, but we can make one. We see that x_1x_3 *is divisible by the leading term of* g_3*. So by multiplying* g_3 *with* x_1 *we get a new polynomial with leading term* x_1x_3*, and we add it to the list and move the indices for* p *and* $x_1 \cdot g_3$*.*

$p:$	x_1x_2	$+$	x_1x_3	$+$	$\overline{x_1x_4}$					$+$	x_1					$+$	x_4		
$x_2 \cdot g_2:$	x_1x_2					$+$	$\overline{x_2x_4}$					$+$	x_2						
$x_1 \cdot g_3:$			$\mathbf{x_1x_3}$	$+$	$\overline{x_1x_4}$														

Comparing the monomials for all the current indices, we see that x_1x_4 *is the largest but there are an even number of polynomials with* x_1x_4 *as leading term, and an even number equals zero in* $\mathbb{F}_2$*. We thus move the indices for these polynomials.* $x_1 \cdot g_3$ *will now have an invalid index, and we can remove it form the list.*

$p:$	x_1x_2	$+$	x_1x_3	$+$	x_1x_4					$+$	$\overline{x_1}$					$+$	x_4		
$x_2 \cdot g_2:$	x_1x_2					$+$	$\overline{x_2x_4}$					$+$	x_2						

Comparing the monomials for all the current indices, we see that x_2x_4 *is the largest, so we add* g_1 *to the list and move indices.*

$p:$	x_1x_2	$+$	x_1x_3	$+$	x_1x_4					$+$	$\overline{x_1}$						$+$	x_4	
$x_2 \cdot g_2:$	x_1x_2					$+$	x_2x_4					$+$	$\overline{x_2}$						
$g_1:$						$+$	$\mathbf{x_2x_4}$							$+$	$\overline{x_3}$				

We see that there are an even number of polynomials, which have an index equal to the largest current monomial. We move the indices for these polynomials, and get x_1 *as the largest. We now add* g_2 *to the list.*

$p:$	x_1x_2	$+$	x_1x_3	$+$	x_1x_4					$+$	x_1						$+$	$\overline{x_4}$	
$x_2 \cdot g_2:$	x_1x_2					$+$	x_2x_4					$+$	$\overline{x_2}$						
$g_1:$						$+$	x_2x_4							$+$	$\overline{x_3}$				
$g_2:$										$\mathbf{x_1}$			$\overline{x_2}$				$+$	x_4	

Again we see that there are an even number of polynomials, which have an index equal to the largest current monomial, namely x_2*. Moving the indexes we see that* $x_2 \cdot g_2$ *now has an invalid index and can be removed. Then* x_3 *is the current largest monomial, and we add* g_3 *to the list.*

$p:$	x_1x_2	$+$	x_1x_3	$+$	x_1x_4					$+$	x_1						$+$	$\overline{x_4}$	
$g_1:$						$+$	x_2x_4							$+$	x_3				
$g_2:$										x_1		x_2					$+$	$\overline{x_4}$	
$g_3:$															$\mathbf{x_3}$	$+$	$\overline{x_4}$		

We have now reached a halting condition, i.e. we do not have a base-polynomial with x_4 *as leading term (or that divides* x_4*). If we assemble a new polynomial by taking all monomials from the current polynomial indices of all polynomials, we get the polynomial* $p_{red} = x_4$*. This means we have found a solution,* $x_4 = 0$*. Next step will then be to eliminate* x_4 *from the base.*

4.4 On parallel execution and caching

Python is not well suited for utilizing multiple processors or cores. The GIL (global interpreter lock) prevents a python process to utilize several CPU's or cores, thus threading will not increase computational speed. The solution to parallelism in python is to start multiple processes, each running in different CPU's or cores. As mentioned earlier, PP is a free add-on to python that helps single out functions that we want to run in parallel on separate cores or CPU's. The computer we used for computations had a single *Intel Core2 Quad* processor. Computationally we can think of

this as 4 separate processors, and PP can start a python process on each of those. So naíve logic tells us we should get our work done 4 times faster, but things are not that simple unfortunately. To parallelize the division algorithm by running it on separate processes, we have to copy all the data each process needs. The division algorithm needs the total current base to perform correctly, and the current base can be quite large seen from a memory perspective. So we obviously have to consider an ad-hoc cost-benefit approach to this. When there is too much data to copy, parallel computing the PP way will slow things down.
When we have enough computer memory it can be useful to store all reusable results, e.g. the results from polynomial multiplications. Having a cache of polynomials - all created from intermediate polynomial multiplications during a division job - we can do a look-up job in the cache the same way as we would do a look-up in the current base - instead of doing the same multiplication over and over. It obviously save computational time - but it hogs up much memory. When caching is used it renders parallelism almost useless, there is just too much data to be copied to each process. A solution could be to use inter-process communication or other shared-memory techniques, but we have not investigated this to the fullest as it was not our goal to create the optimal implementation but rather to investigate aspects of it.
As an empirical rule, because of lack of shared-memory between processes, we decided to either use parallelism or caching - not both.

4.5 Testing the software

To test this kind of software we need systems of equations with a given solution, which are hard to come by in general. Nicolas T. Courtois published [4] (public domain) a program that generates systems of equations for CTC2 which we used extensively. CTC2 [37] [38] is a tweakable toycipher made for trying out algebraic attack techniques, and it has an optional number of rounds, optional number of S-boxes, optional number of plaintext/ciphertext pairs etc., thus perfect for generating test data.

[4] http://www.cryptosystem.net/aes/toyciphers.html

5 Cryptanalysis

For this project we chose two well known ciphers, a representative of a stream cipher and a representative of a block cipher. We chose these ciphers since there are differences in how one organizes algebraic attack for these two cipher categories. The stream cipher, LILI-128, is well known in the area of cryptanalysis since it was shown to have an exploitable weakness, and we use this weakness to lower the polynomial degree in our attack. The block-cipher, KASUMI, is very much in use today and we are not aware of any published algebraic weaknesses.

5.1 The ciphers

5.1.1 KASUMI

KASUMI is a blockcipher used in the security architecture of 3GPP systems [17]. Both the confidentiality function (f8) and integrity function (f9) in UMTS are based on KASUMI. In GSM, KASUMI is used in the **A5/3** key stream generator, and in GPRS it is used in the **GEA3** key stream generator. KASUMI is a *Feistel network*, it operates on 64 bit input to produce 64 bit output under a 128 bit key. We take a closer on the inner functioning when we generate polynomial equations for KASUMI (see 5.3.2).

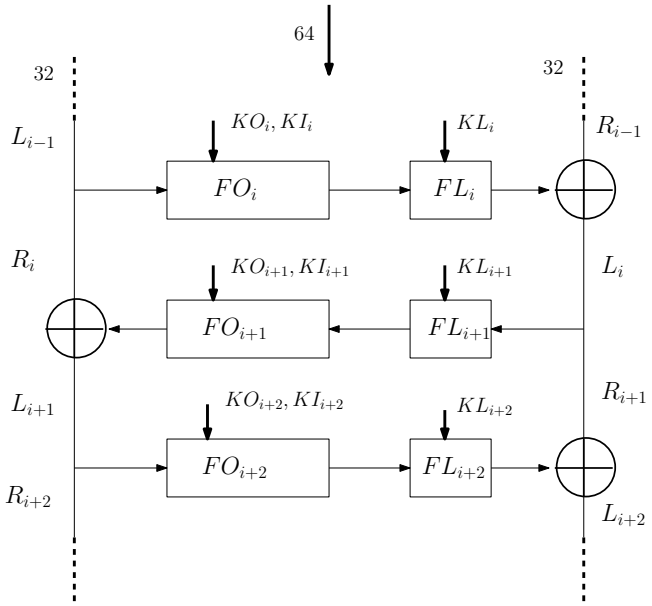
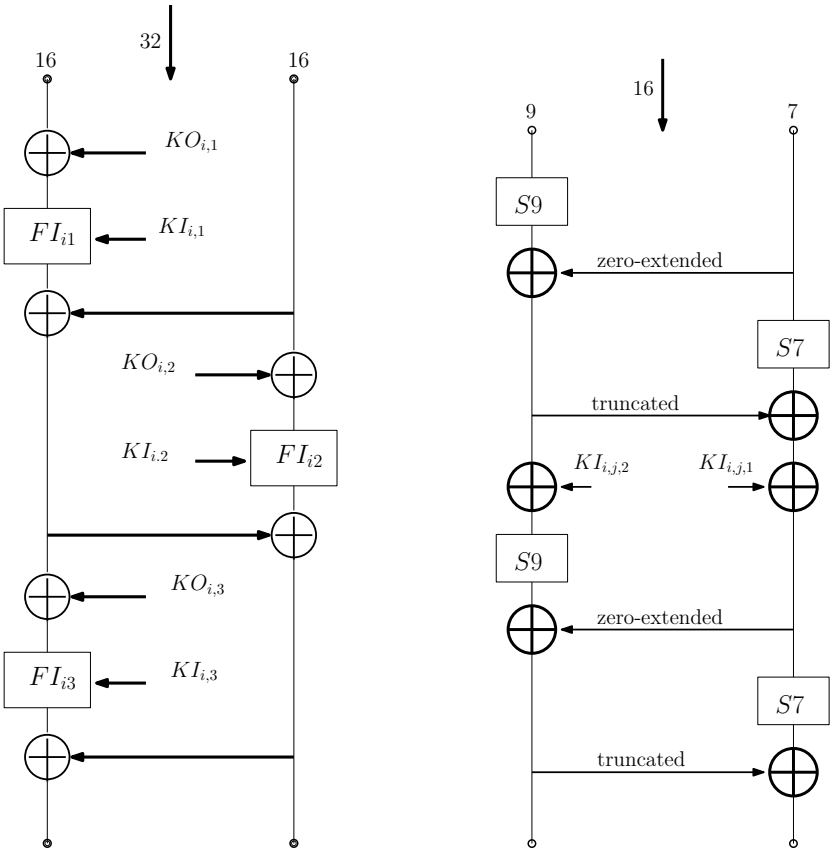


Figure 1: KASUMI.

In Figure 1 we see three rounds of KASUMI. Input to each round i is 64 bit, which is split in



(a) FO function (b) FI function

Figure 2: FO and FI functions.

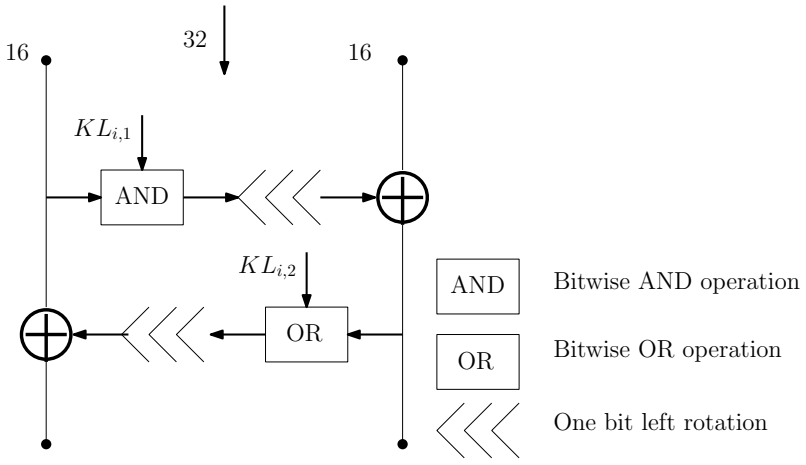


Figure 3: FL function.

half such that L_{i-1} and R_{i-1} is 32 bit. From the start, a 64 bit plaintext is split in half:

$$Plaintext = L_0 \parallel R_0$$

Then for each round $i, 1 \leq i \leq 8$;

$$\begin{aligned} L_i &= R_i \otimes f_i(L_{i-1}, RK_i) \\ R_i &= L_i \end{aligned}$$

where KO_i, KL_i and KI_i are the i'th round keys. See Appendix A for the construction of round keys. The function f_i for rounds $1 \leq i \leq 8$ is composed of subfunctions FL, FO and FI (see Appendix A.2) as follows:

$$f_i(I, RK_i) = \begin{cases} FL(\ FO(I, KO_i, KI_i), KL_i) & \text{if } n \text{ is even} \\ FO(\ FL(I, KL_i), KO_i, KI_i) & \text{if } n \text{ is odd} \end{cases}$$

The non-linear parts of KASUMI are the S-boxes and the FL function.

5.1.2 LILI-128

LILI-128 [16] keystream generator is an LFSR based synchronous stream cipher with a very simple structure, and uses a 128 bit key. Two binary LFSRs (linear feedback shift register) and two functions are used to generate a pseudo random binary keystream sequence. It consists of two subsystems, the clock control subsystem and the data generation subsystem. The data generation subsystem uses an integer sequence from the clock control subsystem to control the clocking. LILI-128 was broken completely in [39].

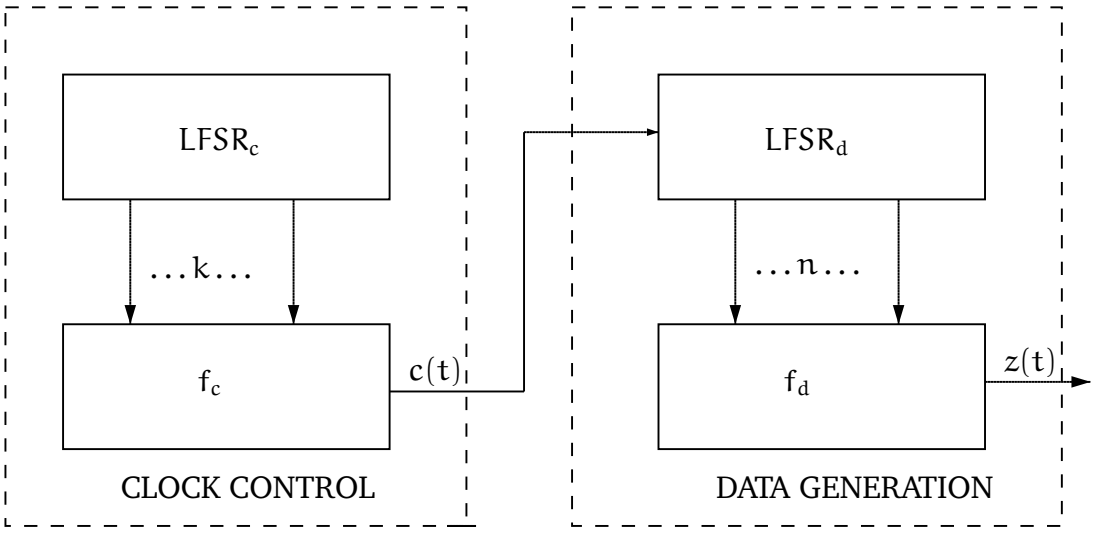


Figure 4: LILI-128

The clock control subsystem consist of a 39 bit register, initialized by the first 39 bits of the key and is regularly clocked. The feedback polynomial of $LFSR_c$ is the primitive polynomial

$$x^{39} + x^{35} + x^{33} + x^{31} + x^{17} + x^{15} + x^{14} + x^2 + 1$$

which gives a maximum-length sequence of period $P_c = 2^{39} - 1$. To produce the non-zero integer sequence $c(t) \in \{1, 2, 3, 4\}$ it uses the function

$$f_c(x_{12}, x_{20}) = 2 \cdot x_{12} + x_{20} + 1$$

which operates on $k = 2$ stages.

The data generation subsystem consists of an 89 bit register, initialized with the last 89 bits of the 128 bit key. The feedback polynomial of $LFSR_d$ is the primitive polynomial

$$x^{89} + x^{83} + x^{80} + x^{55} + x^{53} + x^{42} + x^{39} + x + 1$$

which gives a maximum-length sequence of period $P_d = 2^{89} - 1$. The non-linear Boolean output function f_d takes $n = 10$ stages as input (see Appendix B). Written as a polynomial, f_d is a degree 6 polynomial in ten variables (see Appendix B). If the data generation subsystem was regularly clocked, it would be a non-linear filter generator. With the irregular clocking of $LFSR_d$, the output $z(t)$ can be viewed as a decimated sequence from a regularly clocked non-linear filter generator.

5.2 Miscellaneous

As explained earlier, we implemented monomials and polynomials in such a way that reduction by field equations is implicit. Even though $X \cdot Y^2 \cdot Z^3$ and $X \cdot Y \cdot Z$ are distinct monomials they will evaluate to the same on all possible inputs in $\mathbb{F}_2$. Thus to be formally correct, we are working in $\mathbb{F}_2[x_1, x_2, \ldots, x_n]/(x_1^2 + x_1, x_2^2 + x_2, \ldots, x_n^2 + x_n)$. Automatic reduction by field equations will simplify things a lot.

It will reduce memory impact since our polynomial ideal will be smaller by the fact that the field equations $x_i^2 + x_i = 0 \forall x_i \in \mathbb{F}_2[x_1, x_2, \ldots, x_n]$ are already elements in the ideal. We can see this by considering two monomials, for instance $X_1^a \cdot X_2 \cdot X_3^b$ and $X_1^{a+1} \cdot X_2^2 \cdot X_3^b$. These two classes of monomials are not divisible and may potentially be the leading terms of base polynomials. When reduction by field equations is implicit, these leading terms are the same and only one instance will exist in the base.

On the other hand we cannot take advantage of algorithmic and algebraic tricks concerning homogeneous polynomials since variable degree is no longer an issue.

From a practical point of view, we do not need to consider any existence of solution issues. The simple fact is that all polynomials come from a cipher-algorithm, and every output bit is a binary combination of plain-text bits, key-bits and constants so we can safely assume uniqueness, i.e. given plaintext and key produce unique ciphertext. The primary question is how to solve these equations using Gröbner Bases techniques in the most efficient way.

Let us start with some basic facts concerning binary polynomials.

Definition 5.1. *(i) The maximum number of different monomials of degree* d *in* n *variables is :*

$$M_{max}^d = \binom{n}{d}.$$

(ii) The maximum number of terms in a polynomial of total degree d *in* n *is:*

$$T^d_{max} = \sum_{i=0}^{d} M^i_{max} = \sum_{i=0}^{d} \binom{n}{d}$$

Polynomial division algorithm deals with monomials one by one, so the number of monomials per polynomial is obviously important. From the above we can estimate that:

(1) Keeping the number of variables constant, and increasing the degree d by one will increase the maximum number of monomials by a factor of $\approx \frac{n}{d+1}$

(2) Keeping the degree constant, and increasing the number of variables n by one will increase the maximum number of monomials by a factor of ≈ 1

So the polynomial total degree is computationally much more important than the number of variables. Another thing to worry about is *reduction to zero*, i.e. polynomials that are reduced to nothing. This consumes a lot of resources and gives us no new information about the solution. So *reduction to zero* must be avoided if possible. The art of solving polynomial systems is an active research field and there are much more to solving polynomial equations. A very good book on these issues is [2].

Let us investigate the dynamics of our algorithm. Since we are constructing a d-truncated basis, we are not interested in anything with a degree $> d$. Thus we restrict S-polynomials to a maximum degree also. If we feed the algorithm polynomials from a list of precomputed polynomials, it is useful to order the precomputed list by degree, then by leading terms. Seen from the algorithm's perspective: the only visible part of a polynomial is the leading term, thus a polynomial is defined by its leading term only. So internally we only have to keep track of all leading terms of the current base. We do this by having d ordered lists of leading terms, one for each degree. The list over leading terms of degree w , or w-list, needs room for $\binom{n}{w}$ monomials.

So assume an input-polynomial p of degree d, i.e. the maximum degree. Then we just check the d-list whether the place for $LT(p)$ is vacant. If it is, we occupy it. If it isn't, we know that we can reduce p since a polynomial with the same leading term exists in the current base. Reducing p gives us a new polynomial $\tilde{p}$ with a smaller, in the monomial ordering sense, leading term $LT(\tilde{p})$, and we check if this leading term exists in the list, and so on. When we know that $LT(\tilde{p})$ is not in the d-list, we must check if we have leading terms in a list for degrees $< d$ which divides $LT(\tilde{p})$. If this is the case, we can further reduce $LT(\tilde{p})$. Now assume that the degree of $m = LT(p)$ initially was $w < d$, or that p was reduced until the degree of $m = LT(\tilde{p})$ is $w < d$. Since we require that all leading terms are mutually prime, i.e. no leading term is divisible by another leading term, we must check if m divides any leading term in the current base with degree $> w$. If m of degree w divides the leading term $LT(q)$ for some polynomial q in the current base of degree $t > w$, we can reduce q. This means that the place for $LT(q)$ in the t-list is permanently vacant, and by the same argument all places for leading terms divisible by m are rendered permanently vacant.

Proposition 5.2.1. *Let* $l < d$ *where* $l, d \in \mathbf{N}$*, and let* M^d_n *be monomials in* $\mathbb{F}_2[x_1, x_2, \ldots, x_n]$ *of degree* d*. Let* G *be a base of a polynomial ideal* $I \subseteq \mathbb{F}_2[x_1, x_2, \ldots, x_n]$*. A polynomial with a leading term of degree* $l < d$ *will render* $\binom{n-l}{d-l}$ *permanently vacant places in the d-list.*

Proof. Given a leading term m of degree $l < d$. Since there are l variables in a leading term of degree l, we can multiply m with $d - l$ variables, not in m, to create $\tilde{m}$ of degree d. Thus $d - l$ variables can be chosen out of $n - l$ variables. The number of different combinations of $d - l$ variables out of $n - l$ variables is $\binom{n-l}{d-l}$. Each of these monomials can be a leading term and they are all divisible by m. □

Corollary 5.2.2. *Let $l, d \in \mathbf{N}$ and $p \in \mathbb{F}_2[x_1, x_2, \ldots, x_n]$ be a polynomial of degree $l < d$, and let L_d be the set of all t-lists, where $l < t \leq d$. p renders a total number of*

$$\sum_{i=l}^{d} \binom{n-l}{i-l}$$

permanently vacant places in the lists in L_d.

Proof. This follows from 5.2.1 when we consider all t-lists, $l < t \leq d$. □

An interesting question to ask is: how many base-polynomials of a certain degree do we need to reduce all input polynomials of larger degree? It is easy to construct a 2-list that will reduce all polynomials of degree > 2, to polynomials of degree ≤ 2.

Example 5.1 (Full degree 2 list). *Assume we are given a set of N different variables. Split the set in half. If N is even we have two sets containing $\frac{N}{2}$ variables, and if N is odd we have two set containing $\frac{N+1}{2}$ and $\frac{N-1}{2}$ variables. To form a monomial in 3 variables or more, i.e. degree 3 or more, we need to pick at least two variables from the same set. We now fill the 2-list with all possible monomials of degree 2 constructed from the variables in the same set. Since any degree ≥ 3 monomial M must be constructed by picking at least two variables from the same set, we know that there exists a monomial in our 2-list that divides M. For N even, this gives*

$$2 \cdot \binom{\frac{N}{2}}{2} = \frac{N(N-2)}{4} \text{ monomials}$$

in the 2-list. If N is odd, this gives

$$\binom{\frac{N+1}{2}}{2} + \binom{\frac{N-1}{2}}{2} = \frac{(N-1)^2}{4} \text{ monomials}$$

in the 2-list. Since the total number of possible monomials of degree 2 is $\binom{N}{2}$, and

$$\frac{\frac{N(N-2)}{4}}{\binom{N}{2}} = \frac{1}{2} - \frac{1}{2(N-1)}$$
$$\frac{\frac{(N-1)^2}{4}}{\binom{N}{2}} = \frac{1}{2} - \frac{1}{2N}$$

we see that we need under 50 % of the total number of possible degree 2 monomials to assure that our base stays at degree 2. Further, assuming we already have $N_L < N$ linear polynomials in our base, we can substitute $(N - N_L)$ for N in the above calculations.

Intuition, and the Shape Lemma tells us that given enough polynomials we will eventually fill up every non-permanently vacant place in the leading term lists, which eventually will reduce every new polynomial to a linear polynomial.

A little about S-polynomials. Creating S-polynomials is a clever way of "lifting out" new leading terms hidden behind old ones. But there are three undesirable consequences with creating S-polynomials: S-polynomials have increased total degree, they are very often reduced to zero and they have the uncanny ability to result in dense polynomials with a large number of monomials. These dense polynomials soon "infect" part of the current base by being involved in reductions, and thus ruins any structure and/or sparsity the current base possibly had. Given enough input-polynomials, S-polynomials should be avoided. It is important to recognize that constructing S-polynomials from a linear polynomial will always reduce to zero. Since this is not entirely trivial we will present it as a lemma.

Lemma 5.2.3. *S-polynomials constructed from a linear polynomial will reduce to zero.*

Proof. We construct an S-polynomial of degree d from polynomials P_1 and P_2 where we are using monomial ordering $>_{GRLEX}$ and

$$P_1 = \sum_{i=0}^{S} X_i \text{ ,non-zero variables } X_i$$
$$P_2 = \sum_{i=0}^{T} M_i \text{ ,of degree } d-1 \text{ where the } M_i\text{'s are non-zero monomials}$$

where M_0 not divisible by X_0. Then the S-polynomial will have the form

$$S_{pol} = X_0 \sum_{i=1}^{S} M_i + M_0 \sum_{i=1}^{T} X_i \text{ ,of degree } d$$

Assume that $X_0 \cdot M_i >_{GRLEX} X_1 \cdot M_0$ for $0 < i < k$. Then we can reduce repeatedly by $M_i \cdot P_1$, $0 < i < k$ since the leading term contains X_0. Then

$$S_{pol}^{red} = X_0 \sum_{i=k}^{S} M_i + (M_0 + \ldots + M_{k-1}) \sum_{i=1}^{T} X_i$$

Now we must have that $M_0 \cdot X_1 >_{GRLEX} X_0 \cdot M_k$. Then $M_0 \cdot X_i >_{GRLEX} X_0 \cdot M_k$ for $0 < i < T-1$. Since the leading term of the S-polynomial contains M_0 we can reduce repeatedly by $X_i \cdot P_2$ for $0 < i < T-1$. We now get that

$$S_{pol}^{red} = (X_0 + \ldots + X_{T-1}) \sum_{i=k}^{S} M_i + (M_0 + \ldots + M_{k-1}) \cdot X_T$$

If $M_0 \cdot X_T >_{GRLEX} X_0 \cdot M_k$ (we may have that $X_T \equiv 1$) then we can reduce further by $X_T \cdot P_2$ and we end up with

$$S_{pol}^{red} = (X_0 + \ldots + X_T) \sum_{i=k}^{S} M_i = P_1 \cdot \sum_{i=k}^{S} M_i$$

We see that the leading term contains X_0, and it will reduce to zero by repeatedly reducing by $M_i \cdot P_1$ for $k \leq i \leq S$.

If $X_0 \cdot M_k >_{GRLEX} M_0 \cdot X_T$ we must have that $X_T \equiv 1$ and

$$S_{pol}^{red} = (X_0 + \ldots + X_{T-1}) \sum_{i=k}^{S} M_i + (M_0 + \ldots + M_{k-1})$$

Assume that $X_0 \cdot M_i >_{GRLEX} M_0$ for $k \leq i < l$. Then we can reduce repeatedly by $M_i \cdot P_1$ for $k \leq i < l$ and we get

$$S_{pol}^{red} = (X_0 + \ldots + X_{T-1}) \sum_{i=l}^{S} M_i + (M_0 + \ldots + M_{l-1})$$

Now M_0 is the leading term and we can reduce by P_2 and get

$$S_{pol}^{red} = (X_0 + \ldots + X_{T-1} + 1) \sum_{i=l}^{S} M_i = P_1 \cdot \sum_{i=l}^{S} M_i$$

As before, this clearly results in reduction to zero by repeatedly reducing using $M_i \cdot P_1$ for $l \leq i \leq S$, and this proves the lemma. □

5.3 Generating polynomials

How to generate a system of equations which reflects all aspects of the cipher? In our case, this means taking the original cipher algorithm, which operates on bits and sequences of bits, and rewrite it such that it operates on polynomials instead. Thus every bit, during every stage in the cipher algorithm is considered a polynomial. And the only unknown bits are the key bits, and those will be the variables. We assume to know, or to have guessed correctly, part of the key.

5.3.1 LILI-128

We only consider the data-generation subsystem of LILI-128, thus we assume to know at least the first 39 bits of the key. For a strategy for guessing the clocking sequence in LILI-128 see [40].

For the data-generation subsystem we have irregular clocking from the clocking-subsystem and an 89-bit register with the feedback polynomial:

$$x^{89} + x^{83} + x^{80} + x^{55} + x^{53} + x^{42} + x^{39} + x + 1$$

This means that if the integer value from the clocking-subsystem is t then values in the stages move t places, i.e. the value in register x moves to register $x - t \bmod 89$. The last register, stage 89, is controlled by the feedback polynomial, or as a recurrence equation

$$u[89+t] = u[88+t] \oplus u[50+t] \oplus u[47+t] \oplus u[36+t] \oplus u[34+t] \oplus u[9+t] \oplus u[6+t] \oplus u[t]$$

After all the bits in the registers have shifted, we take the values in the stages 0,1,3,7,12,20,30,44,65 and 80 and create a 1024 bit number from them, and pass it through the output function (see Appendix 12). A Boolean function can be transformed to an *algebraic normal form* - or polynomial - via it's truth table and the Möbius transform, and the polynomial in 10 variables corresponding to the output function f_d is seen in Appendix 13.

Considering the initial state of the registers as unknown variables, we see that stage 89 is a linear combination of 8 earlier stages. So the stages can be modeled as linear polynomials. The clocking will be an integer in the set $(1,2,3,4)$, and the shift procedure is done just as many times.

The function f_c in the clocking subsystem takes the value from two stages:

$$f_c(x_{12}, x_{20}) = 2 \cdot x_{12} + x_{20} + 1$$

The different outcomes, i.e. 1,2,3 or 4, from f_c are determined by quadratic equations in x_{12} and x_{20}. If we were to consider the first 39 bits as unknowns we would have to multiply the stage polynomials with these quadratic equations for each shift.

If we look at f_c we see that to shift only once, then $(x_{12}, x_{20}) = (0,0)$. A polynomial modeling this is $f_{00} = x_{12} \cdot x_{20} + x_{12} + x_{20} + 1$, which is 1 when $(x_{12}, x_{20}) = (0,0)$ and zero otherwise. For the stages to shift twice, the current $(x_{12}, x_{20}) = (0,1)$, and a polynomial to model this is $f_{01} = x_{12} \cdot x_{20} + x_{20}$, and so on. So at time t, the value of stage x (< 89) is

$$u[x] = f_{00} \cdot u[x+1] + f_{01} \cdot u[x+2] + f_{10} \cdot u[x+3] + f_{11} \cdot u[x+4]$$

We see that for each clocking the stages are multiplied by a quadratic polynomial and will soon grow beyond any hope of dealing with them computationally. So this is clearly not the way to go. We refer again to [40] for an idea how to find the clocking bits of the key. We will from here on assume to know the clocking key bits.

If we know the clocking bits of the key, it is easy to model the internal state of the data-generation subsystem. Each stage is a linear polynomial, and is initialized with the last 89 bits of the key. If the last $n < 89$ bits of the key are unknown then the key will be:

$$(bit_0, bit_1, \ldots, bit_{n-1}, p_1, p_1, \ldots, p_n)$$

where $bit_j \in \{0,1\}$ is known and each linear polynomial p_i is modeled as in Section 4.1. Under these conditions, each shift is done the usual way and the feedback stage is just addition of linear polynomials.

The output polynomial (see Appendix 13) is a degree 6 polynomial in 10 variables. As we have seen earlier, a degree 6 polynomial gives us in worst case a polynomial with $\sum_{i=0}^{6} \binom{n}{i}$ monomials. With e.g. 40 unknown bits this will take $\approx$ 30 Mb of storage per polynomial - and we are going to need a lot of polynomials.

Thus we need to lower the degree of the polynomials, and there exist some techniques for this.

Definition 5.2 (Annihilator). *Let* $f \in K[x_1, \ldots, x_n]$ *be a polynomial of degree* $d > 1$. *A non-zero polynomial* $g \in K[x_1, \ldots, x_n]$ *is called an Annihilator if*

$$f \cdot g = 0$$

We need low degree annihilators.

Definition 5.3 (Degree-Reductor). *Let* $f \in K[x_1, \ldots, x_n]$ *be a polynomial of degree* $d > 1$. *A non-zero polynomial* $q \in K[x_1, \ldots, x_n]$ *we have called a Degree-Reductor if*

$$f \cdot q = h, \text{ and } \deg(h) < \deg(f)$$

Example 5.2. *Let us look at the polynomial of degree 3*

$$f(x_1, x_2, x_3) = x_1x_2x_3 + x_1x_2 + x_2x_3 + x_1 + x_2$$

Now, $x_1 \text{and} x_3$ *are a degree-reductor since*

$$x_1 \cdot 0(x_1x_2x_3 + x_1x_2 + x_2x_3 + x_1 + x_2) = x_1x_2x_3 + x_1x_2 + x_1x_2x_3 + x_1 + x_1x_2 = x_1$$

and

$$x_3 \cdot (x_1x_2x_3 + x_1x_2 + x_2x_3 + x_1 + x_2) = x_1x_2x_3 + x_1x_2x_3 + x_2x_3 + x_1x_3 + x_2x_3 = x_1x_3$$

And $g = x_1x_3 + x_1$ *is clearly an annihilator.*

How can we take advantage of this?
Since we perform a known-plaintext attack, this is equivalent to a known keystream sequence since plaintext is XOR'ed with the keystream to produce the ciphertext. The output function of LILI-128 produces one bit at each clocking t, i.e.

$$f_d(Y(t)) = f_d(y_0(t), y_1(t), y_3(t), y_7(t), y_{12}(t), y_{20}(t), y_{30}(t), y_{44}(t), y_{65}(t), y_{80}(t)) \in \{0, 1\}$$

where $y_i(t)$ is the value in stage i at clocking t.

If the cipher outputs a **1** at clocking t, i.e. $f_d(Y(t)) = 1$ and we have an annihilator polynomial g, such that $f_d(X) \cdot g(X) = 0 \; \forall X$, then we must have that $g(Y(t)) = 0$ for this to be satisfied. In the same way, if the cipher outputs a **0** at clocking t, i.e. $f_d(Y(t)) = 0$ and we have a degree-reductor polynomial q such that $f_d(X) \cdot q(X) = h(X) \; \forall X$, then we must have that $h(Y(t)) = 0$. If we can find annihilators and degree-reductors, we can work with polynomials with degree < 6 instead, and this is a huge advantage.

How to find annihilators polynomials? There are many ways of attacking this problem (see e.g. [41]), but since LILI-128's combiner polynomial only consists of 10 variables, we can easily brute-force it.

If we try to generate all possible degree 4 polynomials the naíve way, we soon run into trouble. Since we have 10 variables there are $\sum_{i=0}^{4} \binom{10}{i} = 386$ different monomials of degree 4 or less.

This gives us $2^{386} \approx 10^{116}$ different polynomials. The number of elementary particles in the universe is estimated to $\approx 10^{80}$ so we see that this is not the right approach.

What we do is to multiply $f_d(X)$ with a general degree 4 polynomials with 386 binary parameters:

$$P_{gen} = a_0 + a_1 \cdot x_1 + \ldots + a_{1,2} \cdot x_1x_2 + \ldots + a_{6,8,9,10} \cdot x_6x_8x_9x_{10} + a_{7,8,9,10} \cdot x_7x_8x_9x_{10}$$

and we compute all values which can be assigned to the binary parameters to ensure that $f_d(X) * P_{gen}(X) = 0$. This can be done quite fast by generating a large binary table, where columns are labeled with all possible monomials up to degree 10, and rows are labeled with $f_d(X)$ multiplied with all possible monomials up to degree 4 corresponding to a binary parameter. Since each column represents a monomial in the the product $P_{gen} * f_d(X)$, we must have that the sum of all parameters in each column is zero. This gives us one linear equation in 386 parameters for each possible monomial. See below for an example:

	x_1	x_2	...	m_i	m_{i+1}	...	$x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}$
$1 \cdot f_d$	a_0	a_0	...	0	a_0	...	0
$x_1 \cdot f_d$	a_1	0	...	a_1	0	...	0
$x_2 \cdot f_d$	0	a_2	...	0	0	...	0
...	...	...	...	...	...	...	...
$x_{10} \cdot f_d$	0	0	...	a_{10}	a_{10}	...	0
$x_1x_2 \cdot f_d$	0	0	...	$a_{[1,2]}$	0	...	0
$x_1x_3 \cdot f_d$	0	0	...	0	$a_{[1,3]}$	...	0
...	...	...	...	...	...	...	...
$x_6x_8x_9x_{10} \cdot f_d$	0	0	...	$a_{[6,8,9,10]}$	$a_{[6,8,9,10]}$	...	0
$x_7x_8x_9x_{10} \cdot f_d$	0	0	...	0	$a_{[7,8,9,10]}$	...	$a_{[7,8,9,10]}$

Now we sum all parameters in each column

x_1	$a_0 + a_1 = 0$
x_2	$a_0 + a_2 = 0$
...	...
m_i	$a_{10} + a_{[1,2]} + \ldots = 0$
m_{i+1}	$a_{10} + a_{[1,3]} + \ldots = 0$
...	...

This system of equations turns out to be underdefined with 14 free variables due to identical equations. Generating all possible polynomials using values for the free variables gives 16383 annihilator polynomials (14 free variables, thus $2^{14} - 1$ possible choices). All these polynomials are not linearly independent of course. Now, if $g_i, i \in \mathbf{N}$ are annihilators for f_d, so are all combinations $g_s = \sum_i a_i \cdot g_i, a_i \in K[x_1, \ldots, x_n]$. Thus we can process this system of equations with our Gröbner bases implementation. The 4-truncated basis for this system turned out to consist of 16 annihilator polynomials of degree 4 (see Appendix B).

For the degree-reductor polynomials we do a similar thing. The only difference is that we allow the sum of all parameters in each column to be non-zero. As before we get a system with a certain number of free variables, and run the system of equations through our Gröbner bases implementation and pick the polynomials that are reduced to a lesser degree. We found 17 degree-reductor

polynomials (see Appendix B).
Since each cipher-internal bit is represented by a polynomial, the 10 stages that are input to the combiner function are polynomials. The new combiner function is represented by 15 polynomials if the output bit is a **1** and 17 polynomials if the output bit is a **0**. So instead of producing one degree 6 polynomial for each output bit, we now produce 15, or 17, degree 4 polynomials depending on the output bit.

5.3.2 KASUMI

As in LILI-128, we represent the unknown key-bits as linear polynomials in one variable. After the key schedule, the round keys are represented by linear polynomials in the unknown key-bits. The input to the sub-functions for each round are now polynomials representing the partly encrypted plaintext, and linear polynomials representing the round-keys. The only non-linear parts in KASUMI algorithm are the S-boxes and the Boolean AND and OR parts in the FL subfunction.

As presented in Appendix A.3, the S-boxes of KASUMI can be represented as a system of polynomials. The polynomial system for S7 contains degree 3 polynomials and the polynomial system of S9 consists of degree 2 polynomials. The sub-function FL produces quadratic polynomials, and sub-function FI passes polynomials repeatedly through non-linear S-boxes. It is easy to see that the degree of the resulting polynomials soon get very large. To mitigate this we must introduce intermediate variables. For each polynomial $p(x_1, x_2, \ldots, x_s)$, where s is the total number of variables so far, we check the degree before we send it through the S-box. If $deg(p) > 1$ we introduce a new intermediate variable x_{s+1} and generate the equation

$$x_{s+1} + p(x_1, x_2, \ldots, x_s) = 0$$

We see that x_{s+1} "absorb" the value of $p(x_1, x_2, \ldots, x_s)$, and can be seen as an alias for p. The equation is considered as part of the final system of equations.

Then we send x_{s+1} through the S-box instead, forcing linear input to the S-boxes. For each call to subfunction FI we have 4 calls to S-boxes, i.e. 32 bits are input to S-boxes thus potentially producing 32 new variables. FO calls FI three times, i.e. potentially producing 96 new variables per round, FL may produce 32 variables per round, and it all sum to potentially 128 new variables per round.

If all key-bits are unknown, and let n_r be the number of rounds and n_{pc} be the number of plaintext-ciphertext pairs, we will produce

$$n_{pc} \cdot n_r \cdot 128 + 128 \text{ variables}$$

and

$$n_{pc} \cdot n_r \cdot 128 + n_{pc} \cdot 64 \text{ equations}$$

At the end of the last round we get 64 polynomials representing the 64-bit ciphertext $c_i, 1 \leq i \leq 64$. To get valid equations we must add the ciphertext bits to the polynomials, i.e. if $p_i(X)$ represents the polynomial output for ciphertext bit c_i, then $p_i(X) + c_i = 0$ is the equation for the i'th ciphertext bit.
We can also consider decrypting the ciphertext with the unknown key and collect equations. Decryption with all rounds will produce an abundance of variables, which is not desirable. But

we can analyze the internal states of KASUMI when decrypting and compare them with the internal states when encrypting. An internal state is some sequence of bits, somewhere in the cipher during operation. During decryption we stop when we reach a point where intermediate variables must be introduced, since we want more information (or equations) not more variables. If we have N rounds of encryption, then we compare internal states from encryption round i and decryption round $N - i$. If we are at the same place in the same sub-function we have equivalent states. If we take a polynomial from the encryption state and a polynomial from the decryption state, the sum must equal zero since the internal state represents the same bit.

6 Results

As we soon found out, generating S-polynomials is not the way to proceed. For LILI-128, generating S-polynomials up to degree 4 meant that most will reduce to zero - which is a total waste of time. For KASUMI it meant producing enormous polynomials since we have an abundance of intermediate variables, thus destroying sparseness and structure. We can compensate to a certain degree by increasing the number of plaintext/ciphertext pairs.

6.1 LILI-128

As described above, the degree 6 combiner polynomial was "split" into 15 annihilator polynomials for a **1** bit output and 17 degree-reductor polynomials for a **0** bit output. One annihilator polynomial was discarded due to it's size, and all test runs showed that the 16'th polynomial increased the running time. In the beginning of this project, the procedure was to generate a huge amount of polynomials and write them to a file. Then the Gröbner basis algorithm repeatedly read a fixed number of polynomials from the file and processed them. When reviewing the logs we noticed that during the first few seconds the algorithm produced a small number of linear polynomials - and less important, a small number of degree 2 and degree 3 polynomials. This number depended on the key: the more zero bits in the known part of the key, the more linear polynomials are produced. The explanation is that the zero bits cancel out a lot of monomials in the beginning before the registers are properly mixed. After these few seconds, LILI-128 produces only degree 4 polynomials, with a very high probability. Solving systems of equations with degree 4 polynomials is hard, so generating S-polynomials beyond degree 4 is not even an option. So we must limit our polynomial base to degree 4, or a 4-truncated basis. To be sure we are not running out of polynomials during a Gröbner basis algorithm run, a lot of polynomials must be generated, and that takes a lot of time - the more variables involved, the more time it takes to generate a polynomial. To take advantage of the linear polynomials produced during the first seconds, the LILI-128 algorithm was incorporated into the Gröbner basis algorithm such that we could eliminate variables, (or leading terms) directly in the LILI-128 registers. Then we could generate polynomials, without the leading terms from the linear polynomials, on the fly. Generating polynomials was done using parallel computation. This "feedback" technique saved a lot of time, and also secured us from running out of polynomials during a run. In most instances we had a 700% reduction in running time by doing it this way.

In Figure 5 we can see a graphical representation of 813 LILI-128 polynomials in 40 variables (representing unknown key bits) from 60 output bits. There is one dot for each monomial, and one row represents one polynomial with the leading term to the right, and the last generated polynomial on the bottom. Blue dots represent monomials containing unknown key bits and red dots are monomials containing only intermediate variables. Since we are not using intermediate variables here, the only red dots represent a binary **1**. There are five columns: the first is a graph over the total number of monomials in the polynomials, and the next four are the monomial

degrees. As we can see, in the first row there are monomials only in the degree 1 column, and near the bottom we have dense polynomials with over 35000 monomials. So instead of 60 degree 6 polynomials we have 813 polynomials of degree 4 or less. It is impossible to overstate the importance of this, and it is clear that LILI-128 has a real weakness when it comes to the combiner polynomial. Because of this LILI-128 was upgraded to LILI-II which has a combiner polynomial of degree 10 in 12 variables.

Since LILI-128 produces degree 4 polynomials with a huge number of monomials, we used the parallel division algorithm. Because of the pseudo random nature of the polynomials and the amount of degree 4 monomials possible, we are reducing polynomials to a degree less than 4 with very low probability. Thus the algorithm will spend almost all the time in the division algorithm, and it is slowly filling up the list of leading terms, i.e. the 4-list. In Figure 6 we have a representation of the entire current base near the end of a run with 40 unknown keybits. We see that almost all polynomials are of degree 4. There are large vertical gaps in the monomials and we can identify the same gaps in all degree columns. As previously mentioned, this is due to the fact that in the first few seconds of the run we produce a number of linear polynomials where the corresponding leading terms are eliminated from the LILI-128 registers. This leaves permanent vacancies in the lists of leading terms and a large number of possible monomials will not be generated.

In this particular run we had 40 unknown keybits. In the first few seconds, and for this particular key, 18 degree 1 polynomials, 9 degree 2 polynomials and 19 degree 3 polynomials were generated. Since we have 18 variables eliminated from the registers, corresponding to the leading terms of the degree 1 polynomials, we have in effect reduced our problem to 22 unknown variables. Now, the possible number of degree 4 polynomials in 22 variables is $\binom{22}{4} = 7315$. The 9 degree 2 polynomials leave between 85 and 1254 permanent vacancies, and the 19 degree 3 polynomials leave between 171 and 190. The total number of permanent vacancies created by the collective effect of the degree 2 and degree 3 polynomials depends on the leading terms, but we can safely assume that the number of polynomials in the current base, i.e. 5710, is near the highest possible number of degree 4 polynomials we can get. The next thing we will expect is that each new input polynomial is reduced to a polynomial of degree 3 or less. Each of these degree 3 polynomials will produce permanent vacancies in the 4-list, and the corresponding degree 4 base polynomials are removed from the base and are themselves reduced to degree 3 or less. This avalanche effect continues until we have a solution to the system of equations.

6.1.1 Degree reduction

Degree 4 polynomials are hard to solve, so it is tempting to try degree reduction by introducing intermediate variables. A simple way is to try the trick from Example 5.1. We split our set of variables representing the unknown key bits into two sets, and from each set we generate all possible degree 2 monomials. From these degree 2 monomials we assemble polynomial equations by introducing intermediate variables.

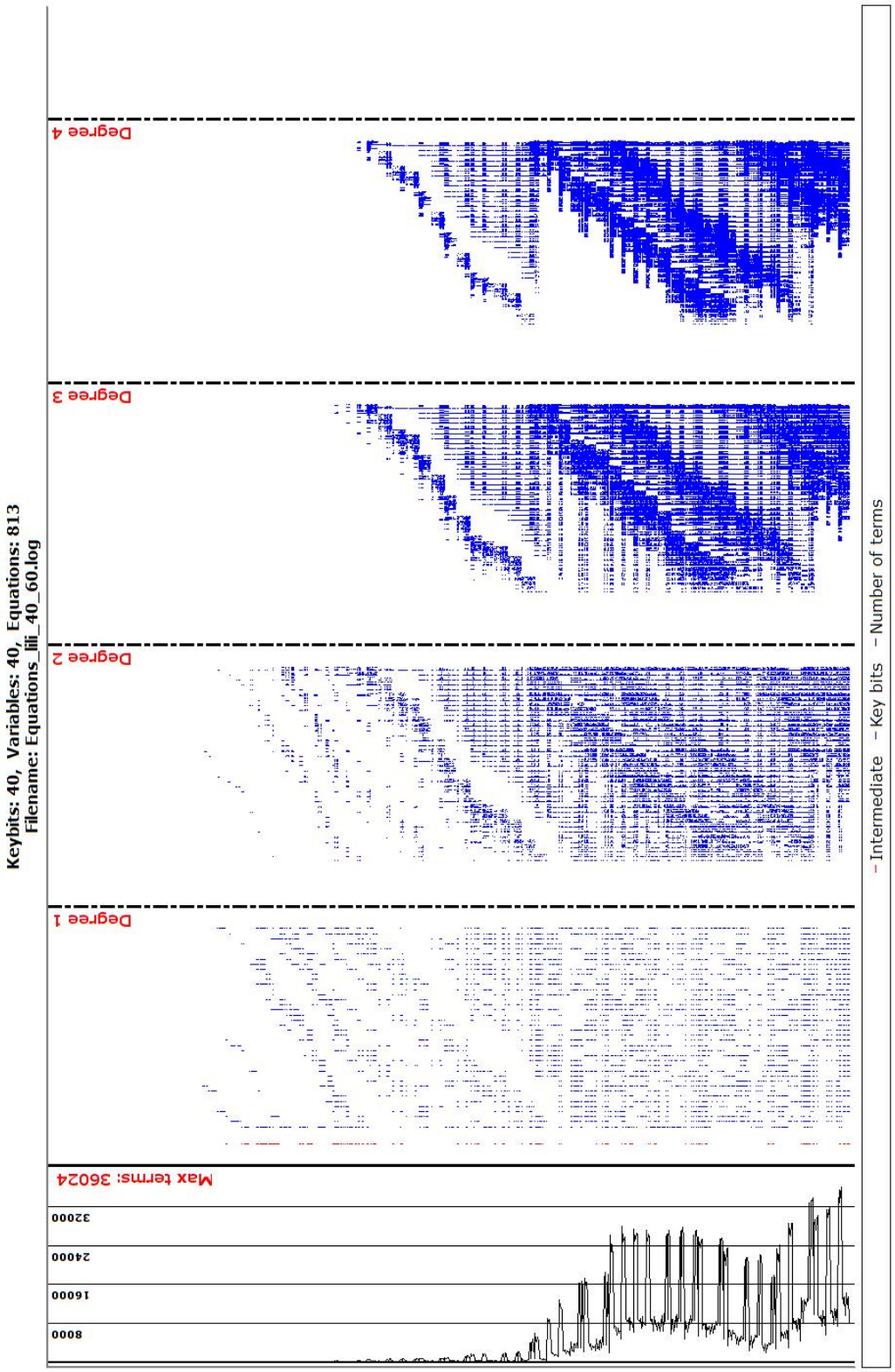


Figure 5: LILI-128 polynomials from 60 output bits.

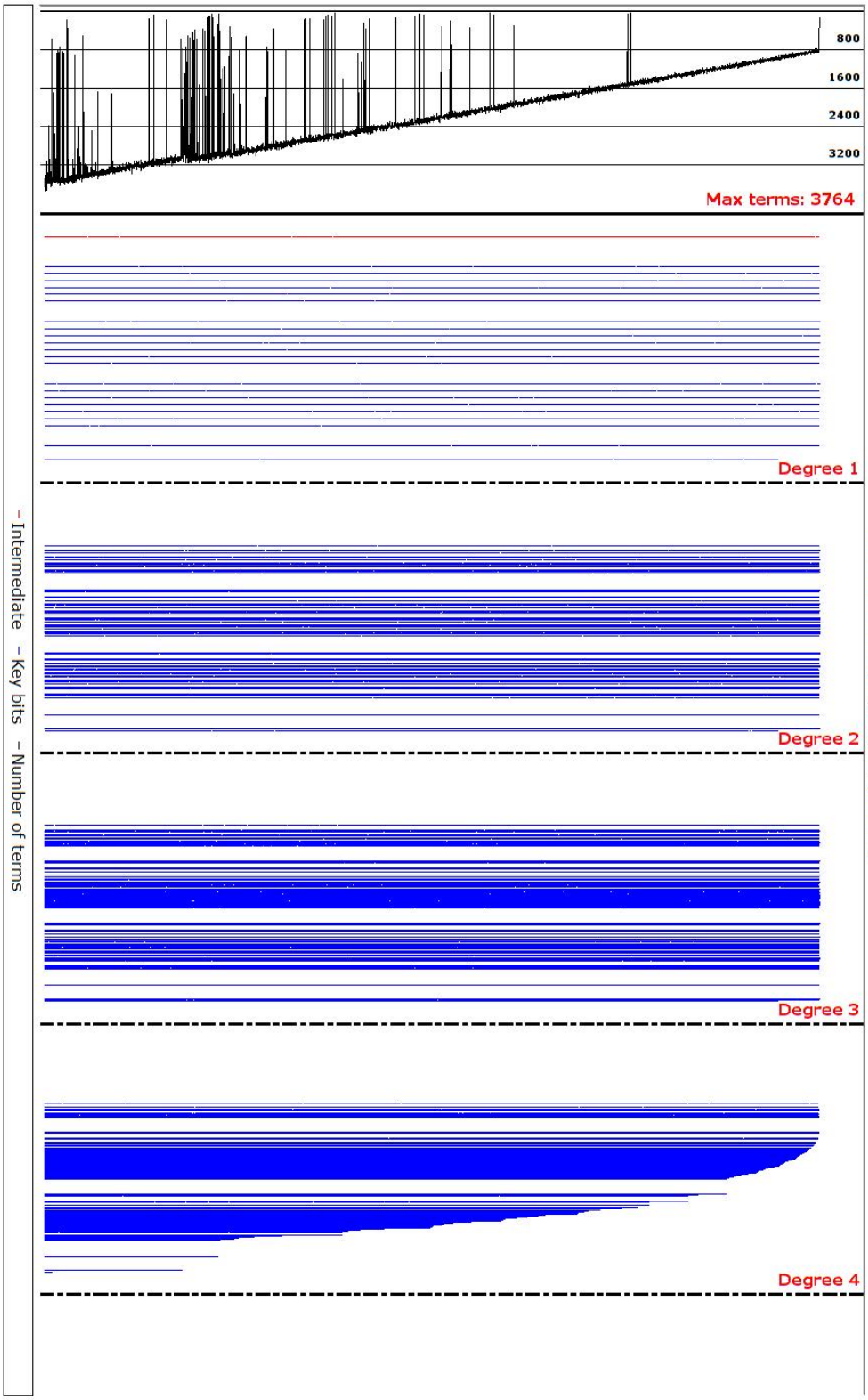


Figure 6: Dump of the current base during a run.

Example 6.1 (**Elimination polynomials**). *Let one set contain* $X_0, X_1, \ldots, X_N$. *The we create* $\binom{N+1}{2}$ *new elimination equations by introducing the same number of intermediate variables as:*

$$\begin{aligned} X_1X_0 + W_{10} &= 0 \\ X_2X_0 + W_{20} &= 0 \\ X_1X_0 + W_{10} &= 0 \\ &\cdots \\ X_mX_n + W_{mn} &= 0 \\ X_{m+1}X_n + W_{(m+1)n} &= 0 \\ &\cdots \\ X_NX_{N-1} + W_{N(N-1)} &= 0 \end{aligned}$$

This way, without introducing too many new variables, each degree 3 monomial is reduced to a degree 2 monomial, and each degree 4 is at least reduced to a degree 3 monomial. In fact we get a 7 to 1 ratio of degree 4 polynomials being reduced to degree 2 rather than three. We add these equations to our current base from the start, such that each new input polynomial to the algorithm with reduced degree. Degree reduction should of course be implemented in the parts of the algorithm that generate polynomials so we do not have to do the same "substitutions" over and over again. But it was done this way just to see the effect on the current base. As we can see in Figure 7 it is not a good idea. We have polynomials with a huge number of monomials, and we know that the division algorithm will have a really hard time. Even the simpler problems, like 32 unknown key bits, are much slower to solve than without degree reduction.

By using relinearization techniques, we get better results (see Figure 8 for a current base with relinearization). As in relinearization (see Section 3.1.2) we find additional dependencies between unknown keybits and intermediate variables, i.e. having $X_AX_B + W_{AB} = 0$ and $X_AX_C + W_{AC} = 0$, we know that $X_CW_{AB} + X_BW_{AC} = 0$ also holds. By running the elimination polynomials from Example 6.1, through the Gröbner algorithm with certain restrictions on the generation of S-polynomials, we can create these dependencies.

6.1.2 Running time results

LILI-128 is very sensitive to the key used for the first output bits. A lot of binary zeros in the key results in many cancellations of monomials, and it thus produces small polynomials for the first few output bits. A key of just zeros will produce zero-polynomials forever. So the real metric for performance is not the number of unknown key bits but the adjusted value, i.e. unknown key bits minus the number of variables (leading terms) in the linear polynomials being created in the first few bits. The use of adjusted values gives us very consistent results, independent on the key.

Figure 7: Dump of the current base with degree reduction.

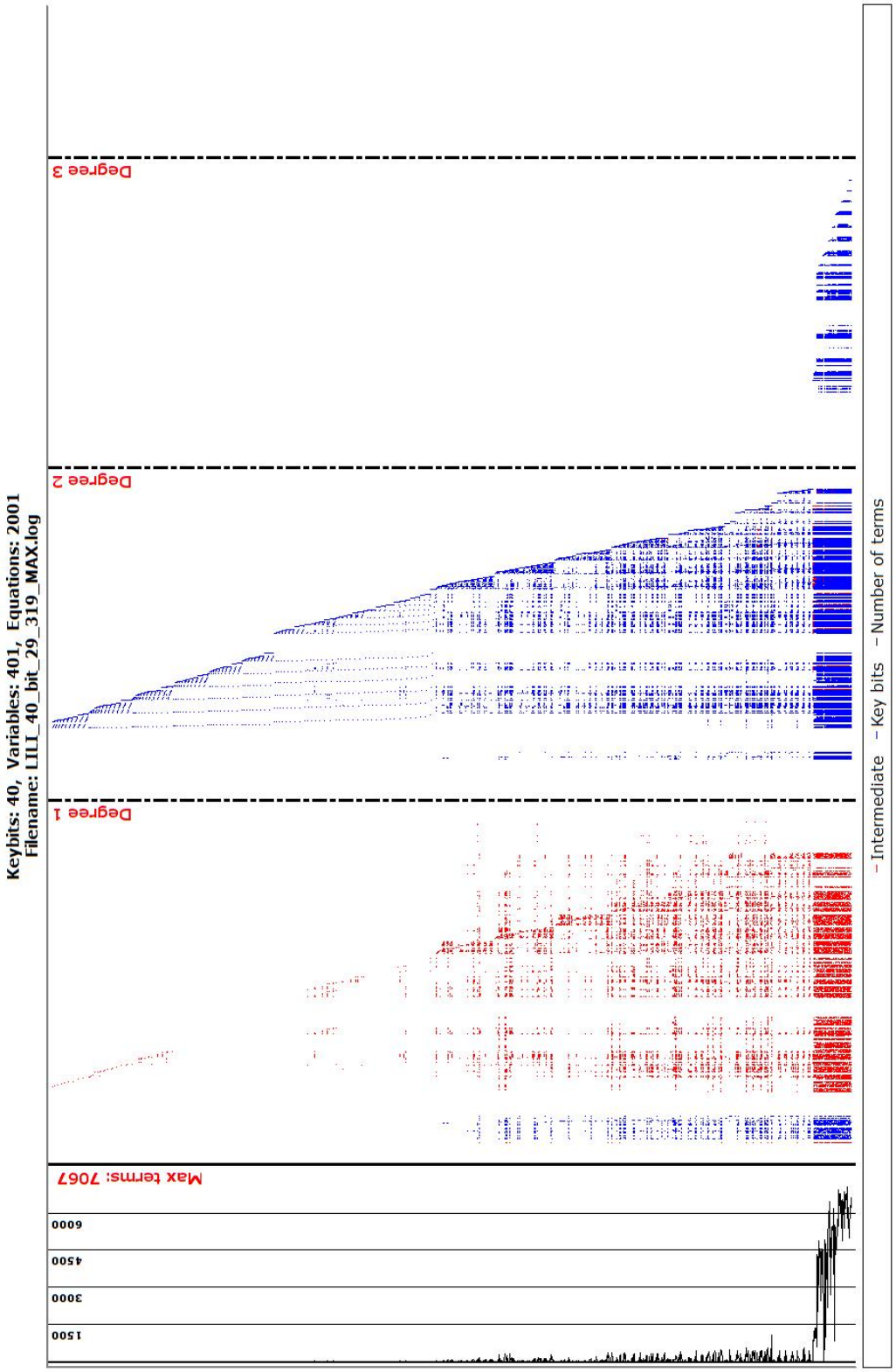


Figure 8: Dump of the current base with relinearized degree reduction.

The running time (see Table 1) is exponential, as theory predicts, and it closely follows the graph of

$$1.38 \cdot 10^{-3} \cdot \mathbf{2^{1.15n}} \text{ , n is adjusted key bits}$$

See Figure 9a for a comparison. The above results are from a computer with a four core processor, and the constants in the expression above are of course considered platform dependent. When run on a computer with a two core processor, the results followed (equally close) the graph of:

$$3.8 \cdot 10^{-3} \cdot \mathbf{2^{1.28n}} \text{ , n is adjusted key bits}$$

Unknown key bits	34	35	36	37	38	39	40
Adjusted key bits	12	14	16	18	19	21	22
Running time (s)	18.8	97.0	565.7	2690.1	5958.0	24618.5	56827.6

Table 1: LILI-128 running time results

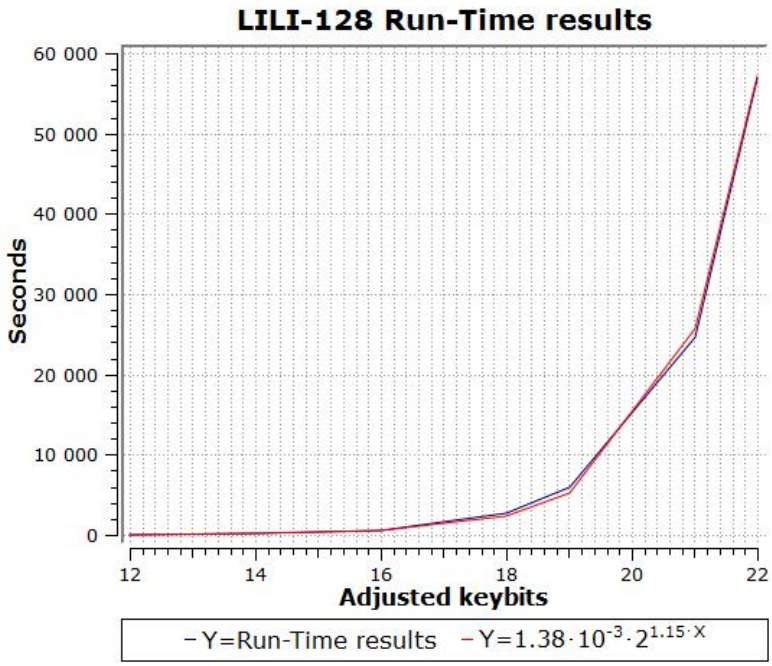
The amount of polynomials needed, or equivalently the amount of output bits, is also exponential. See Table 2, and Figure 9b. The number of output bits needed closely follows the graph of

$$2.5 \cdot \mathbf{2^{0.34n}} \text{ , n is adjusted keybits}$$

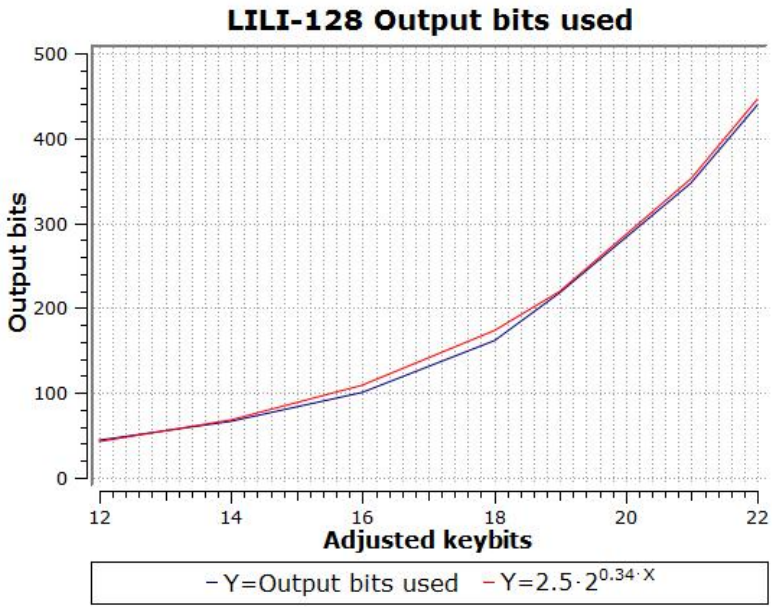
Adjusted keybits	12	14	16	18	19	21	22
Output bits used	45	66	101	162	217	347	438

Table 2: LILI-128 output bits used

Assuming the complexity estimates are good, this means that trying to solve a system of equations for 64 unknown key bits will take 2^{55} seconds, or around 1 billion years using 2^{20} output bits. This is not realistic.



(a) Exponential running time results.



(b) Exponential output bits used.

Figure 9: Exponential results for LILI-128

6.2 KASUMI

The polynomial system for KASUMI was not generated by the "feedback" technique, but polynomials were pre-computed with randomly generated plaintext and written to file. Before being written to file, the polynomials were sorted according to increasing degree and increasing leading terms. The sorting had a big impact on running time. During a run, the algorithm reads a fixed number of polynomials in each "loop" from that file. So for each run we had to decide upon the number of plaintext/ciphertext pairs to use. The total number of variables (unknown key bits and intermediate variables) in the system of equations is directly proportional to the number of plaintext/ciphertext pairs. As opposed to LILI-128, KASUMI does not show the consistency in running time results. The running time results are not directly dependent on the number of unknown key bits or plaintext/ciphertext pairs. In fact, it is quite confusing and no general conclusions could be drawn regarding for instance the optimal amount of plaintext/ciphertext pairs for a given number of rounds and/or unknown key bits (see Table 10).

We had two modes when generating polynomials: with, and without decryption comparisons. With decryption comparisons we compare internal states from encryption with equivalent states from decryption. In such a way, we generate more equations without introducing new intermediate variables. This improved the running time for systems of equations for three rounds only.

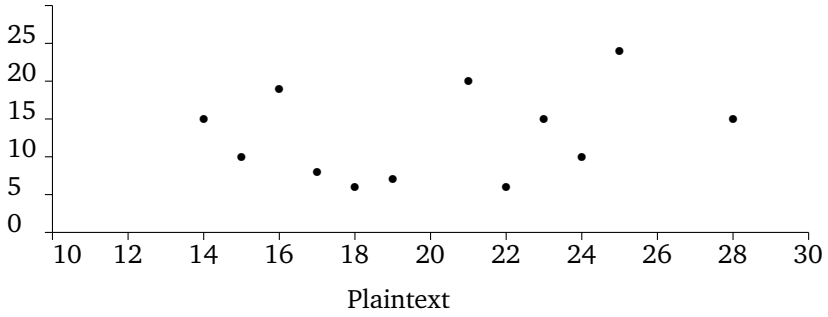


Figure 10: Running time for two round KASUMI, 32 unknown key bits, with varying number of plaintext/ciphertext pairs

The total number of variables depends on the number of rounds and the number of unknown key bits. The goal was to find the spot where the number of plaintext/ciphertext pairs matches the unknown key bits in an optimal way. We did not find a consistent procedure for this.

In Figure 11 we see a typical equations system for KASUMI. The system of equations is highly structured and very sparse.

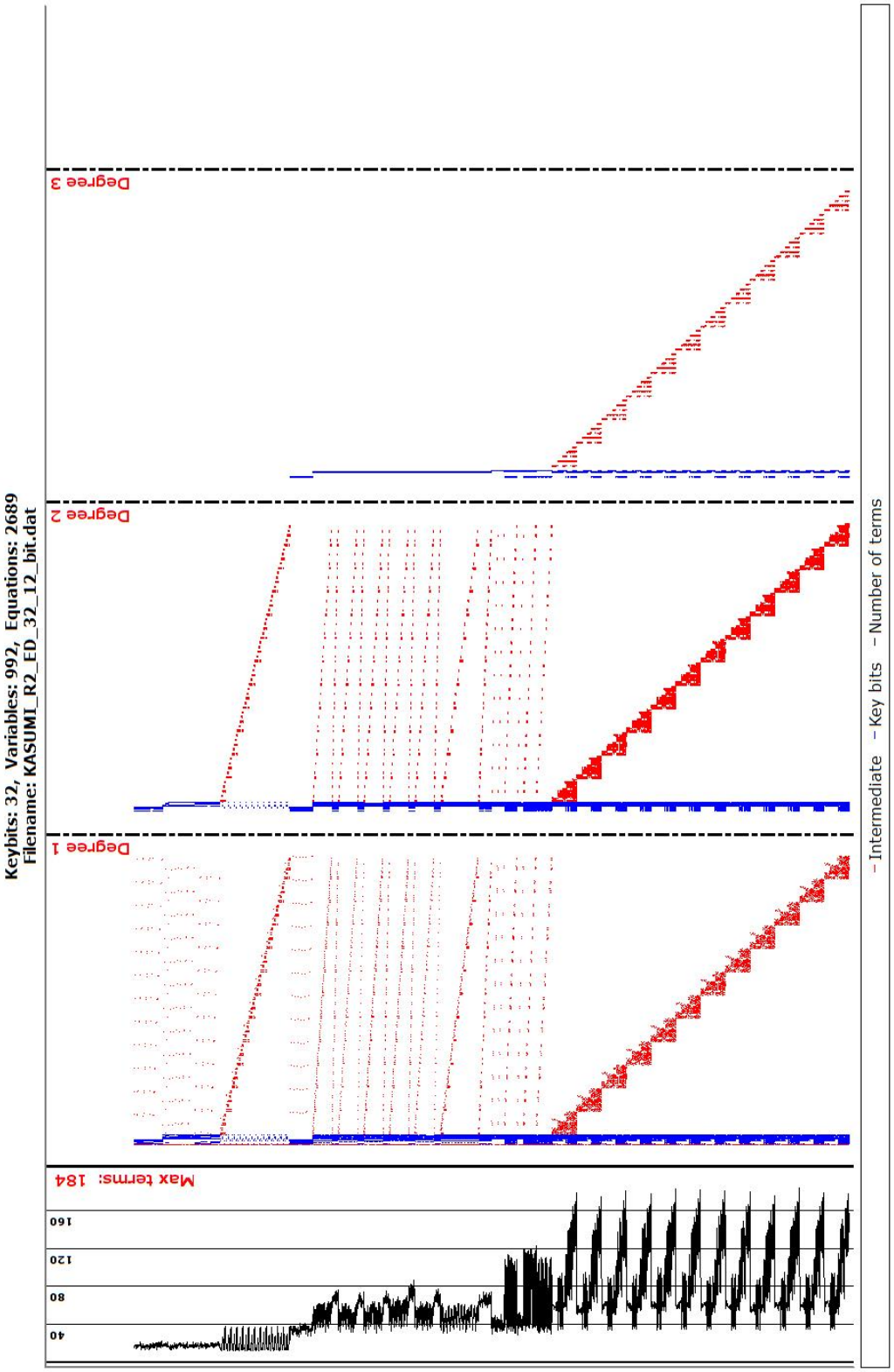


Figure 11: System of equations for two round KASUMI with 32 unknown key bits and 12 plaintext/ciphertext pairs.

6.2.1 One round KASUMI

Since KASUMI only processes 32 bits of input per round we thus have equations for only half the plaintext.

With more unknown keybits we will have more and more keybits in each round-key, so the system of equations changes. We could not solve 88 bits in a reasonable time, but as we see in Figure 12 we do not have to. The figure represents a timeline for the number of key bits found. The dots are time-coordinates for when the algorithm finds the value of an unknown key bit, and the number above the dots are key bits left to find. As we can see, we have 39 key bits left after 750 seconds, and we must wait until 1400 seconds before we have 32 key bits left. But we know from Table 3 that we can solve a system of equations in 39 unknown key bits in 28 seconds, so the logical thing to do is to terminate the 88 bit run when we have 39 key bits left, then generate a system of equations for 39 unknown key bits and solve that. Thus we can solve 88 unknown keybits in 750 + 28 seconds + the time it takes to create a system of equations for 39 unknown keybit, and this is definitely faster than solving the whole 88 bit system of equations. Above 88 unknown key bits we where unsuccessful.

Unknown key bits	Pairs	Variables	Equations used	Time
39	20	679	950	28 seconds
64	40	1984	2200	1 minute 50 seconds
72	50	2472	2750	3 minutes 40 seconds

Table 3: Solved system of equations for one round

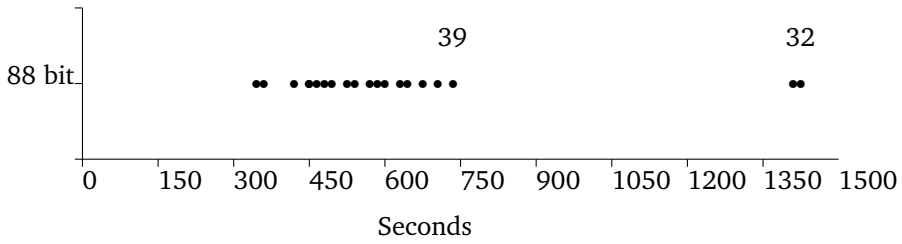


Figure 12: Timeline for one round, 88 unknown key bits

6.2.2 Two round KASUMI

As we can see in Table 4, we have hit the spot for 72 unknown key bits. All unknown key bits are found after just 17 minutes, which is much faster than solving a 64 unknown key bit system of equations. It also uses fewer equations than we have variables, 3600 respective 5832. We have a very similar situation for 88 unknown key bits for two rounds as in one round KASUMI, see the timeline in Figure 13. In the two round case, the 88 unknown key bit run terminated with a memory error, but we can still recover the 88 unknown key bits by generating a system of equations for the 39 key bits not found and solve it.

Unknown key bits	Pairs	Variables	Equations used	Time
32	20	1632	1150	1 minute 34 seconds
48	35	4528	2850	16 minute 4 seconds
64	40	5824	3700	46 minutes 2 seconds
72	40	5832	3600	17 minutes 13 seconds

Table 4: Solved system of equations for two rounds

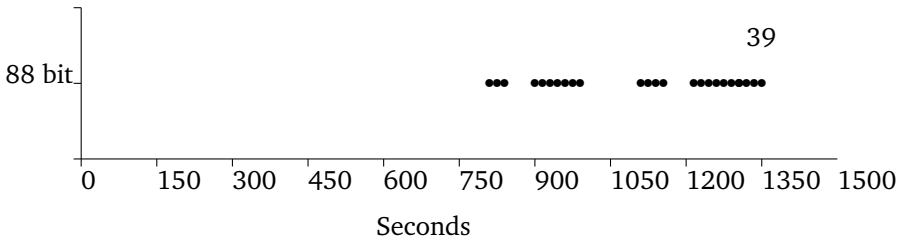


Figure 13: Timeline for two rounds, 88 unknown key bits. This run eventually failed with memory errors.

6.2.3 Three round KASUMI

Now things are getting tougher. The systems of equations are huge; the system of equations for 48 unknown key bits and 40 plaintext/ciphertext pairs have 9008 variables (see Table 5). As we see in the timeline in Figure 14 we have 36 unknown key bits left after 5 hours. After 11 hours there are still 30 key bits left to find. Since we know we can solve an equations system in 40 unknown key bits in under two hours we have just wasted time.

Unknown key bits	Pairs	Variables	Equations used	Time
32	20	3552	2950	5 minutes 38 seconds
40	30	6760	5500	1 hour 19 minutes
48	40	9008	8000	17 hours 21 minutes

Table 5: Solved system of equations for three rounds

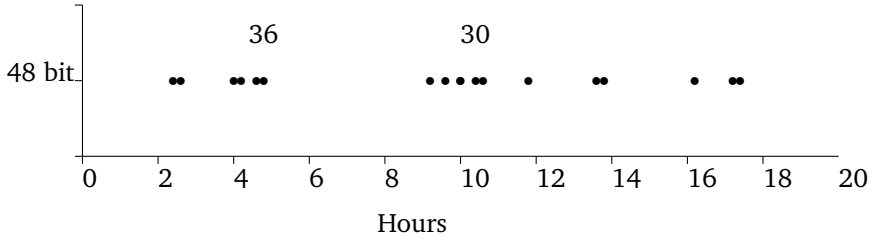


Figure 14: Timeline for three rounds, 48 unknown key bits

6.2.4 Four round KASUMI

Four round KASUMI is a whole new ballgame. We could only solve a system of equations for 3 unknown key bits and 2 plaintext/ciphertext pairs, and that was with generation of S-polynomials. All other variants failed to produce the value of a single unknown key bit. Even increasing to a 4-truncated basis using S-polynomials and letting it crunch for two days did not help.

7 Conclusion

Solving systems of multivariate polynomial equations in an efficient way is important to Algebraic Cryptanalysis. The general problem of finding solutions for systems of polynomial equations belongs to the class of $\mathcal{NP}$-complete problems. Regardless - some large problems, especially sparse and overdefined systems of equations, can be attacked quite efficiently. The method of Gröbner Bases is one of the most efficient ways of attacking such problems, but "*On most problems of even intermediate size, Gröbner Bases oriented methods, [computational algebra systems] like Magma and Singular, crash due to a lack of sufficient memory.*"' [42]. The excessive use of memory when computing Gröbner bases, is mostly due to dense polynomials with high degree. In this thesis we have investigated the use of d-truncated Gröbner bases over a Boolean ring, to mitigate the need for excessive amounts of memory. A Gröbner basis algorithm was developed for this purpose, and the algorithm was applied to systems of equations induced from the symmetric ciphers LILI-128 and KASUMI.

Below, we summarize the main results and try to answer the research questions posed in Section 1.4.

1. *Can we build a Gröbner basis algorithm that is less memory intensive, using d-truncated Gröbner bases over a Boolean ring?*
   In section 4 we presented our implementation of such an algorithm. The algorithm was built from the bottom up, based on Buchberger's Homogeneous Algorithm. Care was taken to minimize memory consumption and a polynomial model based on binary numbers was developed for this purpose.

2. *How will it perform on real life systems of equations?*
   On some levels it performed remarkably well. For systems of equations induced from KASUMI, we where able to solve - in the sense of identifying unknown key bits - a system of equations in over 9000 variables in 17 hours and without memory problems. This particular result was from 3 round KASUMI with 48 unknown keybits and 40 plaintext/ciphertext pairs. As shown in Section 6.2.3, we could have solved this particular system in less time by terminating the algorithm before full time, and used keybits identified so far to generate a new set of equations. The low memory consumption is mostly due to the highly structured and sparse systems of equations. S-polynomials where not computed since we wanted to conserve both structure and sparsity. We did experience computations where the algorithm crashed due to a memory error, but this seemed to be a software issue - probably with python's interpreter or with the JIT compiler (psyco). We where totally unsuccessful above 3 rounds. Even though the systems of equations representing 4 rounds KASUMI are structured and sparse, it seems that the shear number of intermediate variables is too large compared to the number of equations. This implies that the problem cannot be solved by a 3-truncated Gröbner basis.

For seemingly random systems of equations, like those induced from LILI-128, we observed exponential running time - as theory predicts. Even though we used parallel computation for these problems, and as mentioned in Chapter 4 this had memory implications due to the nature of the programming language python, we did not experience problems due to a lack of memory. Large problems did not finish in a reasonable time, and were terminated before memory problems could occur. We dear to propose that the algorithm performed well even in these circumstances.

Performance is a relative notion, and it is difficult to compare results. For instance, MAGMA is a highly optimized system written partly in assembler, while our implementation is done in python. Python is an interpreted language and is thus much slower. But we have a comparison from Gregory V. Bards excellent book [2]. Here he mentions an attack on Keeloq using SINGULAR, a modern computer algebra system. Bard reports that SINGULAR required 70 seconds to solve an equation system induced from 64 rounds Keeloq with 4 plaintext/ciphertext pairs and 10 key bits guessed. Our implementation required 5 seconds. Even though Bard used a less powerful computer, this indicates that our implementation is fairly efficient.

3. *What compromises must be made for achieving this?*
   The compromise for using 4-truncated Gröbner bases on systems of equations induced from LILI-128, is an increase in the number of output bits used that is exponential in the number of unknown key bits.

   The running time results for KASUMI where not consistent enough for identifying areas for nontrivial compromises. In some instances we could solve a large system of equations faster than a smaller one. We found no consistent behavior as a result of varying the number of plaintext/ciphertext pairs.

Bibliography

[1] Segers, A. J. M. Algebraic attacks from a Gröbner basis perspective. Master's thesis, TECHNISCHE UNIVERSITEIT EINDHOVEN, October 2004.

[2] Bard, G. V. 2009. *Algebraic Cyptanalysis*. Springer.

[3] Matsui, M. & Yamagishi, A. 1992. A new method for known plaintext attack of FEAL cipher. In *EUROCRYPT*, 81–91.

[4] Biham, E. & Shamir, A. 1991. Differential cryptanalysis of DES-like cryptosystems. In *CRYPTO '90: Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, 2–21, London, UK. Springer-Verlag.

[5] Wolf, C. 2005. Multivariate quadratic polynomials in public key cryptography. Cryptology ePrint Archive, Report 2005/393. `http://eprint.iacr.org/`.

[6] Shannon, C. E. 1949. Communication Theory of Secrecy Systems. *Bell Systems Technical Journal*, 28, 656–715.

[7] Courtois, N., O'Neil, S., & Quisquater, J.-J. 2009. Practical algebraic attacks on the Hitag2 stream cipher. In *ISC*, 167–176.

[8] Courtois, N. T., Bard, G. V., & Wagner, D. 2008. Algebraic and slide attacks on KeeLoq. *Fast Software Encryption: 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, 97–115.

[9] Renauld, M. & Standaert, F.-X. 5 2009. Combining Algebraic and Side-Channel Cryptanalysis against Block Ciphers. In *30-th Symposium on Information Theory in the Benelux*.

[10] Renauld, M. & Standaert, F.-X. 2009. Algebraic side-channel attacks. Cryptology ePrint Archive, Report 2009/279. `http://eprint.iacr.org/`.

[11] Buchberger, B. *Bruno Buchbergerś PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal*. PhD thesis, Johannes Kepler University of Linz, 1965. English translation by Michael P. Abramson.

[12] Buchberger, B. January 2001. Gröbner bases:a short introduction for systems theorists. *Computer Aided Systems Theory - EUROCAST 2001*, 2178, 1–19.

[13] Buchberger, B. July 2001. Gröbner bases and systems theory. *Multidimensional Systems and Signal Processing*, 12(3-4).

[14] Faugère, J.-C. August 1999. A new effcient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139, 61–88.

[15] Faugère, J.-C. July 2002. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). *International Symposium on Symbolic and Algebraic Computation–ISSAC 2002*, 75–83.

[16] Dawson, E., Clark, A., Golic, J., Millan, W., Penna, L., & Simpson, L. 2000. The LILI-128 keystream generator. NESSIE submission, in the proceedings of the First Open NESSIE Workshop (Leuven, November 2000). `http://www.cryptonessie.org`.

[17] 3rd Generation Partnership Project (3GPP), E. 2007. Specification of the 3GPP confidentiality and integrity algorithms. 3GPP TS 35.202 version 7.0.0 Release 7.

[18] Barkee, B., Can, D. C., Ecks, J., Moriarty, T., & Ree, R. F. 1994. Why you cannot even hope to use Gröbner bases in public key cryptography: an open letter to a scientist who failed and a challenge to those who have not yet failed. *Journal of Symbolic Computations*, 18(6), 497–501.

[19] Cox, D., Little, J., & O'Shea, D. 2007. *Ideals, Varieties, and Algorithms. Third Edition*. Springer New York.

[20] Kreutzer, M. & Robbiano, L. 2000. *Computational Commutative Algebra 1*. Springer-Verlag.

[21] Kreutzer, M. & Robbiano, L. 2005. *Computational Commutative Algebra 2*. Springer-Verlag.

[22] Cook, S. A. 1971. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, 151–158, New York, NY, USA. ACM.

[23] Bardet, M., Faugère, J.-C., & Salvy, B. Complexity of Gröbner basis computation for Semi-regular Overdetermined sequences over $\mathbb{F}_2$ with solutions in $\mathbb{F}_2$. Research Report RR-5049, INRIA, 2003.

[24] Kipnis, A. & Shamir, A. 1999. Cryptanalysis of the HFE public key cryptosystem. In *Proceedings of Crypto'99*. Springer.

[25] Courtois, N., Klimov, E., Patarin, J., & Shamir, A. 2000. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *In Advances in Cryptology, Eurocrypt'2000, LNCS 1807*, 392–407. Springer-Verlag.

[26] Courtois, N. & Pieprzyk, J. 2002. Cryptanalysis of block ciphers with overdefined systems of equations. Cryptology ePrint Archive, Report 2002/044. `http://eprint.iacr.org/`.

[27] Cid, C. & Leurent, G. a. 2005. An analysis of the XSL Algorithm. *Lecture Notes in Computer Science : Advances in Cryptology - ASIACRYPT 2005*, 333–352.

[28] Ars, G. a. a., Faugere, J.-C., Imai, H., Kawazoe, M., & Sugita, M. 2004. Comparison between XL and Groebner basis algorithms. *: Advances in Cryptology - ASIACRYPT 2004*, 338–353.

[29] Sugita, M., Kawazoe, M., & Imai, H. 2006. Relation between the XL Algorithm and Gröbner basis algorithms. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, E89-A(1), 11–18.

[30] Ding, J., Buchmann, J., Mohamed, M. S. E., Mohamed, W. S. A. E., & Weinmann, R.-P. 2008. MutantXL. In *SCC 2008*.

[31] Schilling, T. May 2009. Solving non-linear random sparse equations over finite fields.

[32] Courtois, N. 2003. Fast algebraic attacks on stream ciphers with linear feedback. *Advances in Cryptology - CRYPTO 2003*, 2729/2003, 176–194.

[33] Armknecht, F. *ALGEBRAIC ATTACKS ON CERTAIN STREAM CIPHERS*. PhD thesis, University of Mannheim, 2006.

[34] Faugère, J.-C. & Joux, A. October 2003. Algebraic cryptanalysis of Hidden Field Equation (HFE) cryptosystems using Gröbner bases. *Advances in Cryptology - CRYPTO 2003*, 2729(1-3), 44–60.

[35] Brickenstein, M. & Dreyer, A. November 2007. PolyBoRi: A Gröbner basis framework for Boolean polynomials. *Reports of Fraunhofer ITWM, No. 122*, 15(3-4), 267–278.

[36] Kernighan, B. W. & Ritchie, D. 1988. *C Programming Language 2nd Editon*. Prentice Hall.

[37] Courtois, N. T. 2007. CTC2 and fast algebraic attacks on block ciphers revisited. Cryptology ePrint Archive, Report 2007/152. `http://eprint.iacr.org/`.

[38] Courtois, N. T. 2007. How fast can be algebraic attacks on block ciphers. In *In online proceedings of Dagstuhl Seminar 07021, Symmetric Cryptography*, 07–12.

[39] Huang, X., Huang, W., Liu, X., Wang, C., jing Wang, Z., & Wang, T. 2007. Reconstructing the nonlinear filter function of LILI-128 stream cipher based on complexity. *CoRR*, abs/cs/0702128.

[40] Al-Hinai, S. Z., Dawson, E., Henricksen, M., & Simpson, L. 2007. On the security of the LILI family of stream ciphers against algebraic attacks. *Information Security and Privacy*, 11–28.

[41] Armknecht, F. 2004. On the existence of low-degree equations for algebraic attacks. Cryptology ePrint Archive, Report 2004/185. `http://eprint.iacr.org/`.

[42] Bard, G. V., Courtois, N. T., & Jefferson, C. 2007. Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF(2) via SAT-solvers. `http://eprint.iacr.org/2007/024`.

A KASUMI

A.1 KASUMI key schedule

KASUMI uses a 128 bit key. First the key K is split into eight 16 bit values

$$K = K_1||K_2||\ldots||K_8$$

Then a second set of values is derived from K

$$\overline{K}_i = K_i \otimes C_i$$

where C_i are the constant numbers from Table 7. Now the subroundkeys are created as below 6. Roundkeys are denoted

$$\begin{aligned} RK_i &= (KL_i, KO_i, KI_i) \\ KL_i &= (KL_{i,1}, KL_{i,2}) && 32\text{bit} \\ KO_i &= (KO_{i,1}, KO_{i,2}, KO_{i,3}) && 48\text{bit} \\ KL_i &= (KI_{i,1}, KI_{i,2}, KI_{i,3}) && 48\text{bit} \end{aligned}$$

	1	2	3	4	5	6	7	8
$KL_{i,1}$	$K_1 \lll 1$	$K_2 \lll 1$	$K_3 \lll 1$	$K_4 \lll 1$	$K_5 \lll 1$	$K_6 \lll 1$	$K_7 \lll 1$	$K_8 \lll 1$
$KL_{i,2}$	$\overline{K}_3$	$\overline{K}_4$	$\overline{K}_5$	$\overline{K}_6$	$\overline{K}_7$	$\overline{K}_8$	$\overline{K}_1$	$\overline{K}_2$
$KO_{i,1}$	$K_2 \lll 5$	$K_3 \lll 5$	$K_4 \lll 5$	$K_5 \lll 5$	$K_6 \lll 5$	$K_7 \lll 5$	$K_8 \lll 5$	$K_1 \lll 5$
$KO_{i,2}$	$K_6 \lll 8$	$K_7 \lll 8$	$K_8 \lll 8$	$K_1 \lll 8$	$K_2 \lll 8$	$K_3 \lll 8$	$K_4 \lll 8$	$K_5 \lll 8$
$KO_{i,3}$	$K_2 \lll 13$	$K_3 \lll 13$	$K_4 \lll 13$	$K_5 \lll 13$	$K_6 \lll 13$	$K_7 \lll 13$	$K_8 \lll 13$	$K_1 \lll 13$
$KI_{i,1}$	$\overline{K}_5$	$\overline{K}_6$	$\overline{K}_7$	$\overline{K}_8$	$\overline{K}_1$	$\overline{K}_2$	$\overline{K}_3$	$\overline{K}_4$
$KI_{i,2}$	$\overline{K}_4$	$\overline{K}_5$	$\overline{K}_6$	$\overline{K}_7$	$\overline{K}_8$	$\overline{K}_1$	$\overline{K}_2$	$\overline{K}_3$
$KI_{i,3}$	$\overline{K}_8$	$\overline{K}_1$	$\overline{K}_2$	$\overline{K}_3$	$\overline{K}_4$	$\overline{K}_5$	$\overline{K}_6$	$\overline{K}_7$

Table 6: KASUMI Roundkeys.

C_1	0x0123
C_2	0x4567
C_3	0x89AB
C_4	0xCDEF
C_5	0xFEDC
C_6	0xBA98
C_7	0x7654
C_8	0x3210

Table 7: KASUMI Constants.

A.2 KASUMI Subfunctions

The three subfunctions of KASUMI, called FL, FI and FO, are defined below. We first need to define three "helper" functions

$ROL(X) =$ left circular rotation by one bit

$ZE(X) =$ creates a 9 bit value from a 7 bit by appending zeros to the most significant end

$TR(X) =$ creates a 7 bit value from a 9 bit by discarding 2 bits from the most significant end

Algorithm A.1 KASUMI subfunction FL

Input: $KL_i = KL_{i,1}||KL_{i,2}$, 32 bit key
Input: $I = L||R$, 32 bit value
$\overline{R} = R \otimes ROL(L \text{ AND } KL_{i,1})$
$\overline{L} = L \otimes ROL(\overline{R} \text{ OR } KL_{i,2})$
return $\overline{L}||\overline{R}$, 32 bit

Except for the *OR* part of the function FL, the non-linear parts of KASUMI is offered by the function FI. See A.3 for the S-boxes *S7* and *S9*.

Algorithm A.2 KASUMI subfunction FI

Input: $KI_{i,j}$, 16 bit key split in $K_7||K_9$, 7 and 9 bit each
Input: I, 16 bit value

$$L_1 = R_0 \qquad R_1 = S9[L_0] \otimes ZE(R_0)$$
$$L_2 = R_1 \otimes K_9 \qquad R_2 = S7[L_1] \otimes TR(R_1) \otimes K_7$$
$$L_3 = R_2 \qquad R_3 = S9[L_2] \otimes ZE(R_2)$$
$$L_4 = S7[L_3] \otimes TR(R_3) \qquad R_4 = R_3$$

return $L_4||R_4$, 16 bit

Algorithm A.3 KASUMI subfunction FO

Input: $KO_i = KO_{i,1}||KO_{i,2}||KO_{i,3}$, 48 bit key
Input: $KI_i = KI_{i,1}||KI_{i,2}||KI_{i,3}$, 48 bit key
Input: $I = L_0||R_0$, 32 bit value
for $j = 1$ to 3 **do**
$R_j = FI(L_{j-1} \otimes KO_{i,j}, KI_{i,j})$
$L_j = R_{j-1})$
end for
return $L_3||R_3$, 32 bit

A.3 KASUMI S-boxes

KASUMI has two S-boxes, called *S7* and *S9*. Below is a description of both.

A.3.1 S7

$$y_0 = x_0x_1x_4 + x_1x_5x_6 + x_2x_4x_6 + x_3x_4x_5 + x_4x_5x_6 + x_0x_6 + x_1x_3 + x_1x_6 + x_2x_5 + x_3x_6 + x_4 + x_5 + x_6$$
$$y_1 = x_0x_2x_6 + x_0x_3x_5 + x_1x_2x_5 + x_4x_5x_6 + x_0x_1 + x_0x_4 + x_2x_4 + x_3x_6 + x_5 + x6 + 1$$
$$y_2 = x_0x_1x_6 + x_0x_2x_5 + x_0x_3x_4 + x_1x_2x_4 + x_0x_3 + x_0x_6 + x_1x_5 + x_2x_3 + x_2x_6 + x_4x_6 + x_0 + 1$$
$$y_3 = x_0x_1x_5 + x_0x_1x_2 + x_1x_3x_6 + x_1x_4x_5 + x_2x_3x_5 + x_0x_5 + x_1x_4 + x_2x_6 + x_3x_4 + x_1$$
$$y_4 = x_0x_1x_4 + x_0x_3x_6 + x_0x_4x_5 + x_1x_3x_5 + x_2x_3x_4 + x_0x_2 + x_0x_5 + x_1x_3 + x_1x_6 + x_1x_4 + x_3x_6 + x_5x_6 + x_3 + 1$$
$$y_5 = x_0x_3x_6 + x_0x_2x_4 + x_1x_2x_3 + x_1x_2x_6 + x_2x_5x_6 + x_3x_4x_6 + x_0x_2 + x_0x_3 + x_0x_5 + x_1x_6 + x_2x_5 + x_4x_5 + x_2 + 1$$
$$y_6 = x_0x_1x_3 + x_0x_1x_6 + x_0x_5x_6 + x_1x_4x_6 + x_2x_3x_6 + x_0x_4 + x_1x_2 + x_1x_5 + x_3x_5 + x_6$$

Table 8: KASUMI S-box S7 represented as multivariate polynomials.

54	50	62	56	22	34	94	96	38	6	63	93	2	18	123	33
55	113	39	114	21	67	65	12	47	73	46	27	25	111	124	81
53	9	121	79	52	60	58	48	101	127	40	120	104	70	71	43
20	122	72	61	23	109	13	100	77	1	16	7	82	10	105	98
117	116	76	11	89	106	0	125	118	99	86	69	30	57	126	87
112	51	17	5	95	14	90	84	91	8	35	103	32	97	28	66
102	31	26	45	75	4	85	92	37	74	80	49	68	29	115	44
64	107	108	24	110	83	36	78	42	19	15	41	88	119	59	3

Table 9: KASUMI S-box S7.

A.3.2 S9

$$y0 = x0x2 + x0x7 + x1x7 + x2x5 + x2x7 + x4x8 + x5x6 + x5x8 + x7x8 + x3 + 1$$
$$y1 = x0x1 + x0x4 + x0x5 + x1x4 + x1x7 + x2x3 + x2x7 + x3x5 + x5x8 + x1 + x6 + 1$$
$$y2 = x0x3 + x0x5 + x0x8 + x2x6 + x3x4 + x3x6 + x4x7 + x5x6 + x5x7 + x6x7 + x1 + x8 + 1$$
$$y3 = x0x3 + x0x6 + x0x8 + x1x2 + x1x6 + x1x8 + x2x4 + x4x7 + x7x8 + x0 + x5$$
$$y4 = x0x1 + x0x5 + x0x7 + x1x3 + x1x8 + x2x8 + x3x6 + x3x8 + x6x7 + x4$$
$$y5 = x0x6 + x1x4 + x1x6 + x3x7 + x4x5 + x4x7 + x5x8 + x6x7 + x6x8 + x7x8 + x2 + 1$$
$$y6 = x1x5 + x1x8 + x2x3 + x2x5 + x3x6 + x3x8 + x4x5 + x4x6 + x5x6 + x5x8 + x7x8 + x0 + x7$$
$$y7 = x0x1 + x0x2 + x0x3 + x1x2 + x2x3 + x2x6 + x2x7 + x3x6 + x4x5 + x5x7 + x8 + x3 + 1$$
$$y8 = x0x1 + x1x2 + x1x5 + x1x6 + x2x5 + x2x8 + x3x4 + x4x6 + x3x8 + x2 + x7$$

Table 10: KASUMI S-box S9 represented as multivariate polynomials.

167	239	161	379	391	334	9	338	38	226	48	358	452	385	90	397
183	253	147	331	415	340	51	362	306	500	262	82	216	159	356	177
175	241	489	37	206	17	0	333	44	254	378	58	143	220	81	400
95	3	315	245	54	235	218	405	472	264	172	494	371	290	399	76
165	197	395	121	257	480	423	212	240	28	462	176	406	507	288	223
501	407	249	265	89	186	221	428	164	74	440	196	458	421	350	163
232	158	134	354	13	250	491	142	191	69	193	425	152	227	366	135
344	300	276	242	437	320	113	278	11	243	87	317	36	93	496	27
487	446	482	41	68	156	457	131	326	403	339	20	39	115	442	124
475	384	508	53	112	170	479	151	126	169	73	268	279	321	168	364
363	292	46	499	393	327	324	24	456	267	157	460	488	426	309	229
439	506	208	271	349	401	434	236	16	209	359	52	56	120	199	277
465	416	252	287	246	6	83	305	420	345	153	502	65	61	244	282
173	222	418	67	386	368	261	101	476	291	195	430	49	79	166	330
280	383	373	128	382	408	155	495	367	388	274	107	459	417	62	454
132	225	203	316	234	14	301	91	503	286	424	211	347	307	140	374
35	103	125	427	19	214	453	146	498	314	444	230	256	329	198	285
50	116	78	410	10	205	510	171	231	45	139	467	29	86	505	32
72	26	342	150	313	490	431	238	411	325	149	473	40	119	174	355
185	233	389	71	448	273	372	55	110	178	322	12	469	392	369	190
1	109	375	137	181	88	75	308	260	484	98	272	370	275	412	111
336	318	4	504	492	259	304	77	337	435	21	357	303	332	483	18
47	85	25	497	474	289	100	269	296	478	270	106	31	104	433	84
414	486	394	96	99	154	511	148	413	361	409	255	162	215	302	201
266	351	343	144	441	365	108	298	251	34	182	509	138	210	335	133
311	352	328	141	396	346	123	319	450	281	429	228	443	481	92	404
485	422	248	297	23	213	130	466	22	217	283	70	294	360	419	127
312	377	7	468	194	2	117	295	463	258	224	447	247	187	80	398
284	353	105	390	299	471	470	184	57	200	348	63	204	188	33	451
97	30	310	219	94	160	129	493	64	179	263	102	189	207	114	402
438	477	387	122	192	42	381	5	145	118	180	449	293	323	136	380
43	66	60	455	341	445	202	432	8	237	15	376	436	464	59	461

Table 11: KASUMI S-box S9.

B LILI-128

B.1 LILI-128 output function

0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1
1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0
1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1
1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0
0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0
0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0
0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1
1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0
0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1
0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1
1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0
1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0
0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0
1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1
1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0
1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0
0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0
1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1
1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1
0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0
0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0
1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0

Table 12: LILI-128 boolean output function.

$$
\begin{aligned}
f_d = {} & X_3X_5X_6X_7X_8X_9 + X_4X_5X_6X_7X_8X_9 + X_2X_6X_7X_8X_9 + X_3X_5X_6X_7X_8 + X_3X_5X_6X_8X_9 + X_3X_6X_7X_8X_9 + X_4X_5X_6X_7X_8 + \\
& X_4X_5X_6X_8X_9 + X_0X_7X_8X_9 + X_1X_6X_7X_8 + X_1X_6X_8X_9 + X_2X_6X_7X_9 + X_2X_7X_8X_9 + X_3X_5X_6X_8 + X_3X_6X_7X_8 + \\
& X_3X_6X_8X_9 + X_3X_7X_8X_9 + X_4X_5X_6X_8 + X_4X_6X_7X_9 + X_5X_6X_8X_9 + X_5X_7X_8X_9 + X_1X_8X_9 + X_2X_6X_8 + X_2X_7X_8 + \\
& X_2X_7X_9 + X_2X_8X_9 + X_3X_6X_8 + X_3X_6X_9 + X_3X_7X_9 + X_3X_8X_9 + X_4X_6X_9 + X_4X_8X_9 + X_5X_6X_8 + X_5X_6X_9 + X_5X_7X_8 + \\
& X_0X_7 + X_0X_8 + X_1X_7 + X_2X_8 + X_3X_9 + X_5X_6 + X_5X_9 + X_1 + X_2 + X_3 + X_4
\end{aligned}
$$

Table 13: LILI-128 non-linear filter polynomial.

B.2 LILI-128 degree-reductor and annihilator polynomials

$$
\begin{aligned}
red_1 = {} & Y_7Y_3Y_2Y_1 + Y_7Y_2Y_1Y_0 + Y_6Y_3Y_2Y_0 + Y_6Y_3Y_1Y_0 + Y_6Y_2Y_1Y_0 + Y_5Y_3Y_2Y_0 + Y_5Y_3Y_1Y_0 + Y_4Y_3Y_2Y_1 + \\
& Y_4Y_3Y_1Y_0 + Y_9Y_3Y_2 + Y_9Y_3Y_1 + Y_9Y_2Y_1 + Y_9Y_1Y_0 + Y_8Y_3Y_2 + Y_8Y_2Y_1 + Y_8Y_1Y_0 + Y_7Y_3Y_1 + \\
& Y_7Y_2Y_1 + Y_6Y_1Y_0 + Y_5Y_3Y_0 + Y_5Y_1Y_0 + Y_4Y_3Y_1 + Y_4Y_2Y_1 + Y_9Y_1 + Y_8Y_3 + Y_8Y_1 + Y_7Y_3 + \\
& Y_6Y_3 + Y_6Y_1 + Y_5Y_3 + Y_5Y_1 + Y_4Y_3
\end{aligned}
$$

$$
red_2 = Y_8Y_2Y_1Y_0 + Y_7Y_3Y_2Y_0 + Y_6Y_3Y_2Y_0 + Y_5Y_2Y_1Y_0 + Y_9Y_2Y_0 + Y_6Y_2Y_0 + Y_5Y_2Y_0 + Y_4Y_2Y_0
$$

$$
\begin{aligned}
red_3 = {} & Y_8Y_3Y_1Y_0 + Y_8Y_2Y_1Y_0 + Y_7Y_3Y_2Y_1 + Y_7Y_3Y_1Y_0 + Y_7Y_2Y_1Y_0 + Y_6Y_3Y_2Y_0 + Y_6Y_2Y_1Y_0 + Y_5Y_3Y_1Y_0 + \\
& Y_4Y_3Y_2Y_1 + Y_4Y_3Y_2Y_0 + Y_4Y_3Y_1Y_0 + Y_4Y_2Y_1Y_0 + Y_9Y_1Y_0 + Y_8Y_3Y_2 + Y_8Y_3Y_1 + Y_8Y_2Y_1 + Y_7Y_3Y_2 + \\
& Y_7Y_3Y_1 + Y_7Y_2Y_1 + Y_7Y_1Y_0 + Y_6Y_3Y_2 + Y_6Y_3Y_1 + Y_6Y_3Y_0 + Y_6Y_2Y_1 + Y_6Y_2Y_0 + Y_5Y_3Y_2 + Y_5Y_3Y_1 + \\
& Y_5Y_2Y_1 + Y_4Y_3Y_0 + Y_4Y_2Y_0 + Y_8Y_3 + Y_8Y_2 + Y_8Y_1 + Y_7Y_3 + Y_7Y_2 + Y_7Y_1 + Y_6Y_3 + \\
& Y_6Y_2 + Y_6Y_1 + Y_6Y_0 + Y_5Y_3 + Y_5Y_2 + Y_5Y_1 + Y_4Y_0 + Y_8 + Y_7 + Y_6 + Y_5
\end{aligned}
$$

$$
\begin{aligned}
red_4 = {} & Y_8Y_3Y_2Y_0 + Y_8Y_2Y_1Y_0 + Y_7Y_3Y_2Y_1 + Y_7Y_3Y_2Y_0 + Y_6Y_3Y_2Y_1 + Y_6Y_3Y_2Y_0 + Y_6Y_2Y_1Y_0 + Y_5Y_3Y_2Y_1 + \\
& Y_5Y_3Y_1Y_0 + Y_5Y_2Y_1Y_0 + Y_4Y_3Y_2Y_1 + Y_4Y_3Y_2Y_0 + Y_4Y_3Y_1Y_0 + Y_4Y_2Y_1Y_0 + Y_9Y_1Y_0 + Y_8Y_3Y_0 + \\
& Y_7Y_3Y_0 + Y_7Y_2Y_1 + Y_7Y_1Y_0 + Y_6Y_3Y_0 + Y_6Y_2Y_1 + Y_6Y_1Y_0 + Y_5Y_2Y_1 + Y_4Y_3Y_0 + Y_4Y_2Y_1 + Y_4Y_1Y_0
\end{aligned}
$$

Table 14: LILI-128 degree-reductor polynomials 1-4.

$$\begin{aligned}
red_5 =& Y_8Y_3Y_2Y_1 + Y_8Y_2Y_1Y_0 + Y_7Y_3Y_2Y_1 + Y_7Y_2Y_1Y_0 + Y_6Y_3Y_2Y_0 + Y_6Y_3Y_1Y_0 + Y_4Y_3Y_2Y_0 + Y_4Y_3Y_1Y_0 + \\
& Y_9Y_3Y_2 + Y_9Y_3Y_1 + Y_9Y_2Y_0 + Y_9Y_1Y_0 + Y_8Y_3Y_1 + Y_8Y_2Y_1 + Y_8Y_1Y_0 + Y_7Y_3Y_2 + Y_7Y_2Y_1 + Y_7Y_2Y_0 + \\
& Y_6Y_3Y_2 + Y_6Y_3Y_1 + Y_6Y_2Y_0 + Y_6Y_1Y_0 + Y_5Y_3Y_2 + Y_5Y_3Y_1 + Y_5Y_2Y_0 + Y_5Y_1Y_0 + Y_9Y_2 + Y_9Y_1 + \\
& Y_8Y_1 + Y_7Y_2 + Y_6Y_2 + Y_6Y_1 + Y_5Y_2 + Y_5Y_1
\end{aligned}$$

$$\begin{aligned}
red_6 =& Y_8Y_3Y_2Y_1 + Y_8Y_3Y_2Y_0 + Y_7Y_3Y_2Y_1 + Y_7Y_3Y_2Y_0 + Y_6Y_3Y_2Y_1 + Y_6Y_3Y_2Y_0 + Y_5Y_3Y_2Y_1 + Y_5Y_3Y_2Y_0 + \\
& Y_5Y_2Y_1Y_0 + Y_4Y_3Y_1Y_0 + Y_4Y_2Y_1Y_0 + Y_9Y_2Y_1 + Y_9Y_2Y_0 + Y_9Y_1Y_0 + Y_8Y_3Y_2 + Y_8Y_3Y_1 + \\
& Y_8Y_3Y_0 + Y_7Y_3Y_2 + Y_7Y_3Y_1 + Y_7Y_3Y_0 + Y_7Y_2Y_1 + Y_7Y_2Y_0 + Y_7Y_1Y_0 + Y_6Y_3Y_2 + Y_6Y_3Y_1 + Y_6Y_3Y_0 + \\
& Y_6Y_2Y_1 + Y_6Y_2Y_0 + Y_6Y_1Y_0 + Y_5Y_3Y_2 + Y_5Y_3Y_1 + Y_5Y_3Y_0 + Y_5Y_2Y_1 + Y_5Y_2Y_0 + Y_4Y_3Y_1 + Y_4Y_3Y_0 + \\
& Y_4Y_1Y_0 + Y_9Y_2 + Y_8Y_3 + Y_7Y_3 + Y_7Y_2 + Y_6Y_3 + Y_6Y_2 + Y_5Y_3 + Y_5Y_2 + Y_4Y_3
\end{aligned}$$

$$\begin{aligned}
red_7 =& Y_9Y_2Y_1Y_0 + Y_7Y_3Y_2Y_0 + Y_6Y_3Y_2Y_0 + Y_5Y_2Y_1Y_0 + Y_4Y_3Y_1Y_0 + Y_9Y_2Y_1 + Y_8Y_2Y_1 + Y_8Y_2Y_0 + Y_8Y_1Y_0 + \\
& Y_7Y_1Y_0 + Y_6Y_2Y_0 + Y_6Y_1Y_0 + Y_5Y_2Y_0 + Y_5Y_1Y_0 + Y_4Y_3Y_1 + Y_4Y_3Y_0 + Y_4Y_2Y_0 + Y_9Y_2 + Y_8Y_2 + \\
& Y_8Y_1 + Y_8Y_0 + Y_7Y_1 + Y_7Y_0 + Y_6Y_1 + Y_6Y_0 + Y_5Y_1 + Y_5Y_0 + Y_4Y_3 + Y_8 + Y_7 + Y_6 + Y_5
\end{aligned}$$

$$\begin{aligned}
red_8 =& Y_9Y_2Y_1Y_0 + Y_8Y_3Y_2Y_1 + Y_8Y_3Y_1Y_0 + Y_7Y_3Y_2Y_1 + Y_7Y_3Y_2Y_0 + Y_7Y_3Y_1Y_0 + Y_7Y_2Y_1Y_0 + Y_5Y_3Y_2Y_0 + \\
& Y_5Y_2Y_1Y_0 + Y_4Y_2Y_1Y_0 + Y_9Y_2Y_0 + Y_9Y_1Y_0 + Y_8Y_2Y_0 + Y_8Y_1Y_0 + Y_7Y_2Y_1 + Y_7Y_2Y_0 + Y_6Y_3Y_0 + \\
& Y_6Y_2Y_1 + Y_6Y_2Y_0 + Y_6Y_1Y_0 + Y_5Y_3Y_0 + Y_5Y_2Y_1 + Y_5Y_1Y_0 + Y_4Y_2Y_1 + Y_8Y_0 + Y_7Y_0 + Y_5Y_0 + Y_4Y_0
\end{aligned}$$

$$\begin{aligned}
red_9 =& Y_9Y_3Y_1Y_0 + Y_9Y_2Y_1Y_0 + Y_8Y_3Y_1Y_0 + Y_8Y_2Y_1Y_0 + Y_7Y_3Y_2Y_0 + Y_7Y_2Y_1Y_0 + Y_6Y_3Y_1Y_0 + Y_6Y_2Y_1Y_0 + \\
& Y_5Y_3Y_2Y_0 + Y_4Y_3Y_1Y_0 + Y_9Y_2Y_0 + Y_8Y_2Y_0 + Y_8Y_1Y_0 + Y_7Y_2Y_0 + Y_7Y_1Y_0 + Y_6Y_3Y_0 + Y_6Y_2Y_0 + \\
& Y_5Y_3Y_0 + Y_5Y_1Y_0 + Y_4Y_1Y_0 + Y_8Y_0 + Y_7Y_0 + Y_5Y_0 + Y_4Y_0
\end{aligned}$$

$$\begin{aligned}
red_{10} =& Y_9Y_3Y_2Y_0 + Y_8Y_3Y_1Y_0 + Y_8Y_2Y_1Y_0 + Y_7Y_3Y_2Y_0 + Y_7Y_3Y_1Y_0 + Y_7Y_2Y_1Y_0 + Y_6Y_3Y_1Y_0 + Y_5Y_3Y_2Y_0 + \\
& Y_5Y_3Y_1Y_0 + Y_4Y_3Y_2Y_0 + Y_9Y_1Y_0 + Y_7Y_1Y_0 + Y_6Y_1Y_0 + Y_4Y_1Y_0
\end{aligned}$$

$$\begin{aligned}
red_{11} =& Y_9Y_3Y_2Y_1 + Y_7Y_2Y_1Y_0 + Y_6Y_3Y_1Y_0 + Y_6Y_2Y_1Y_0 + Y_5Y_3Y_2Y_0 + Y_5Y_3Y_1Y_0 + Y_5Y_2Y_1Y_0 + Y_4Y_3Y_2Y_1 + \\
& Y_4Y_3Y_2Y_0 + Y_4Y_2Y_1Y_0 + Y_9Y_3Y_2 + Y_9Y_2Y_1 + Y_9Y_2Y_0 + Y_8Y_2Y_0 + Y_8Y_1Y_0 + Y_7Y_3Y_2 + Y_7Y_1Y_0 + \\
& Y_6Y_3Y_2 + Y_6Y_3Y_0 + Y_6Y_2Y_0 + Y_5Y_3Y_2 + Y_5Y_3Y_0 + Y_5Y_1Y_0 + Y_4Y_2Y_1 + Y_4Y_2Y_0 + Y_4Y_1Y_0 + \\
& Y_9Y_2 + Y_8Y_0 + Y_7Y_2 + Y_7Y_0 + Y_6Y_2 + Y_5Y_2 + Y_5Y_0 + Y_4Y_0
\end{aligned}$$

Table 15: LILI-128 degree-reductor polynomials 6-11.

$$
\begin{aligned}
\mathrm{red}_{12} =& Y_9Y_3Y_2Y_1 + Y_8Y_3Y_1Y_0 + Y_7Y_3Y_2Y_1 + Y_7Y_3Y_2Y_0 + Y_7Y_3Y_1Y_0 + Y_6Y_3Y_1Y_0 + Y_5Y_3Y_2Y_0 + Y_4Y_3Y_1Y_0 + \\
& Y_9Y_3Y_2 + Y_9Y_2Y_1 + Y_9Y_1Y_0 + Y_8Y_3Y_2 + Y_8Y_3Y_1 + Y_8Y_2Y_1 + Y_8Y_2Y_0 + Y_8Y_1Y_0 + Y_7Y_3Y_1 + Y_7Y_2Y_1 + \\
& Y_6Y_3Y_1 + Y_6Y_2Y_1 + Y_6Y_2Y_0 + Y_5Y_3Y_1 + Y_5Y_3Y_0 + Y_5Y_2Y_1 + Y_5Y_2Y_0 + Y_5Y_1Y_0 + Y_4Y_3Y_0 + Y_4Y_2Y_1 + \\
& Y_4Y_2Y_0 + Y_4Y_1Y_0 + Y_9Y_2 + Y_8Y_3 + Y_8Y_2 + Y_8Y_1 + Y_8Y_0 + Y_7Y_3 + Y_7Y_1 + Y_7Y_0 + \\
& Y_6Y_3 + Y_6Y_1 + Y_6Y_0 + Y_5Y_3 + Y_5Y_1 + Y_5Y_0 + Y_8 + Y_7 + Y_6 + Y_5
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{red}_{13} =& Y_9Y_7Y_1Y_0 + Y_9Y_6Y_3Y_0 + Y_9Y_6Y_2Y_0 + Y_9Y_5Y_3Y_0 + Y_9Y_5Y_2Y_1 + Y_9Y_5Y_2Y_0 + Y_9Y_4Y_3Y_1 + Y_9Y_4Y_3Y_0 + \\
& Y_9Y_4Y_2Y_0 + Y_9Y_4Y_1Y_0 + Y_9Y_3Y_2Y_1 + Y_9Y_3Y_2Y_0 + Y_9Y_3Y_1Y_0 + Y_8Y_7Y_3Y_2 + Y_8Y_7Y_3Y_1 + Y_8Y_7Y_3Y_0 + \\
& Y_8Y_7Y_2Y_1 + Y_8Y_7Y_2Y_0 + Y_8Y_6Y_3Y_0 + Y_8Y_6Y_2Y_1 + Y_8Y_6Y_2Y_0 + Y_8Y_5Y_3Y_2 + Y_8Y_5Y_3Y_1 + Y_8Y_5Y_2Y_1 + \\
& Y_8Y_5Y_1Y_0 + Y_8Y_4Y_3Y_1 + Y_8Y_4Y_3Y_0 + Y_8Y_4Y_2Y_1 + Y_8Y_4Y_2Y_0 + Y_8Y_3Y_2Y_1 + Y_8Y_3Y_2Y_0 + Y_7Y_6Y_3Y_2 + \\
& Y_7Y_6Y_2Y_1 + Y_7Y_6Y_2Y_0 + Y_7Y_5Y_3Y_1 + Y_7Y_5Y_3Y_0 + Y_7Y_5Y_2Y_1 + Y_7Y_5Y_2Y_0 + Y_7Y_5Y_1Y_0 + Y_7Y_4Y_2Y_0 + \\
& Y_7Y_4Y_1Y_0 + Y_7Y_3Y_1Y_0 + Y_7Y_2Y_1Y_0 + Y_6Y_5Y_3Y_2 + Y_6Y_5Y_3Y_0 + Y_6Y_5Y_2Y_0 + Y_6Y_5Y_1Y_0 + Y_6Y_4Y_3Y_2 + \\
& Y_6Y_4Y_3Y_1 + Y_6Y_4Y_2Y_1 + Y_6Y_4Y_2Y_0 + Y_6Y_4Y_1Y_0 + Y_6Y_3Y_2Y_0 + Y_6Y_3Y_1Y_0 + Y_6Y_2Y_1Y_0 + Y_5Y_4Y_3Y_2 + \\
& Y_5Y_4Y_3Y_0 + Y_5Y_4Y_2Y_0 + Y_5Y_3Y_2Y_0 + Y_4Y_3Y_1Y_0 + Y_9Y_8Y_1 + Y_9Y_7Y_1 + Y_9Y_6Y_2 + Y_9Y_6Y_0 + Y_9Y_4Y_3 + \\
& Y_9Y_4Y_0 + Y_9Y_3Y_1 + Y_9Y_2Y_1 + Y_9Y_1Y_0 + Y_8Y_7Y_3 + Y_8Y_7Y_2 + Y_8Y_6Y_0 + Y_8Y_5Y_3 + Y_8Y_4Y_3 + Y_8Y_4Y_2 + \\
& Y_8Y_4Y_1 + Y_8Y_3Y_1 + Y_8Y_3Y_0 + Y_8Y_1Y_0 + Y_7Y_6Y_3 + Y_7Y_6Y_1 + Y_7Y_6Y_0 + Y_7Y_5Y_2 + Y_7Y_5Y_1 + Y_7Y_4Y_2 + \\
& Y_7Y_4Y_1 + Y_7Y_3Y_2 + Y_7Y_3Y_1 + Y_7Y_2Y_1 + Y_7Y_2Y_0 + Y_7Y_1Y_0 + Y_6Y_5Y_3 + Y_6Y_5Y_2 + Y_6Y_4Y_2 + Y_6Y_4Y_1 + \\
& Y_6Y_4Y_0 + Y_6Y_3Y_0 + Y_5Y_4Y_3 + Y_5Y_4Y_2 + Y_5Y_4Y_1 + Y_5Y_4Y_0 + Y_5Y_3Y_2 + Y_5Y_3Y_0 + Y_5Y_2Y_0 + Y_4Y_3Y_1 + \\
& Y_9Y_8 + Y_9Y_7 + Y_9Y_6 + Y_9Y_5 + Y_9Y_1 + Y_8Y_6 + Y_8Y_5 + Y_8Y_4 + Y_8Y_2 + Y_7Y_6 + Y_7Y_5 + Y_7Y_4 + \\
& Y_7Y_3 + Y_7Y_1 + Y_6Y_4 + Y_5Y_4 + Y_5Y_3 + Y_5Y_2 + Y_4Y_3 + Y_8 + Y_7
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{red}_{14} =& Y_9Y_8Y_2Y_1 + Y_9Y_7Y_3Y_2 + Y_9Y_7Y_3Y_1 + Y_9Y_6Y_3Y_1 + Y_9Y_6Y_3Y_0 + Y_9Y_6Y_2Y_0 + Y_9Y_5Y_3Y_1 + Y_9Y_5Y_2Y_1 + \\
& Y_9Y_5Y_1Y_0 + Y_9Y_4Y_3Y_2 + Y_9Y_4Y_2Y_1 + Y_9Y_4Y_1Y_0 + Y_9Y_3Y_2Y_0 + Y_9Y_3Y_1Y_0 + Y_9Y_2Y_1Y_0 + Y_8Y_7Y_3Y_1 + \\
& Y_8Y_7Y_3Y_0 + Y_8Y_6Y_3Y_2 + Y_8Y_6Y_3Y_1 + Y_8Y_5Y_3Y_2 + Y_8Y_5Y_3Y_1 + Y_8Y_5Y_2Y_1 + Y_8Y_4Y_2Y_0 + Y_8Y_4Y_1Y_0 + \\
& Y_7Y_6Y_3Y_1 + Y_7Y_6Y_3Y_0 + Y_7Y_6Y_2Y_1 + Y_7Y_6Y_2Y_0 + Y_7Y_5Y_3Y_1 + Y_7Y_5Y_3Y_0 + Y_7Y_5Y_2Y_1 + Y_7Y_5Y_1Y_0 + \\
& Y_7Y_4Y_3Y_2 + Y_7Y_4Y_3Y_0 + Y_7Y_4Y_2Y_1 + Y_7Y_4Y_2Y_0 + Y_7Y_3Y_2Y_0 + Y_6Y_5Y_3Y_0 + Y_6Y_5Y_2Y_0 + Y_6Y_5Y_1Y_0 + \\
& Y_6Y_4Y_1Y_0 + Y_6Y_3Y_2Y_0 + Y_6Y_3Y_1Y_0 + Y_5Y_4Y_3Y_1 + Y_5Y_4Y_2Y_0 + Y_5Y_3Y_2Y_1 + Y_5Y_3Y_2Y_0 + Y_4Y_3Y_2Y_1 + \\
& Y_4Y_3Y_1Y_0 + Y_4Y_2Y_1Y_0 + Y_9Y_8Y_3 + Y_9Y_8Y_1 + Y_9Y_8Y_0 + Y_9Y_7Y_3 + Y_9Y_7Y_2 + Y_9Y_7Y_0 + Y_9Y_6Y_3 + \\
& Y_9Y_6Y_2 + Y_9Y_5Y_3 + Y_9Y_5Y_0 + Y_9Y_4Y_3 + Y_9Y_4Y_2 + Y_9Y_4Y_1 + Y_9Y_4Y_0 + Y_9Y_3Y_2 + Y_9Y_3Y_1 + Y_9Y_2Y_0 + \\
& Y_9Y_1Y_0 + Y_8Y_7Y_3 + Y_8Y_6Y_2 + Y_8Y_6Y_0 + Y_8Y_5Y_1 + Y_8Y_5Y_0 + Y_8Y_4Y_3 + Y_8Y_4Y_2 + Y_8Y_3Y_2 + Y_8Y_3Y_1 + \\
& Y_8Y_2Y_0 + Y_7Y_6Y_3 + Y_7Y_6Y_2 + Y_7Y_6Y_1 + Y_7Y_6Y_0 + Y_7Y_5Y_3 + Y_7Y_5Y_2 + Y_7Y_5Y_0 + Y_7Y_4Y_1 + Y_7Y_3Y_2 + \\
& Y_7Y_3Y_1 + Y_7Y_3Y_0 + Y_7Y_1Y_0 + Y_6Y_5Y_1 + Y_6Y_5Y_0 + Y_6Y_4Y_0 + Y_6Y_3Y_2 + Y_6Y_3Y_1 + Y_6Y_2Y_1 + Y_6Y_1Y_0 + \\
& Y_5Y_4Y_3 + Y_5Y_4Y_0 + Y_5Y_3Y_2 + Y_5Y_2Y_1 + Y_4Y_2Y_1 + Y_4Y_1Y_0 + Y_8Y_6 + Y_8Y_4 + Y_8Y_3 + Y_8Y_2 + \\
& Y_8Y_0 + Y_7Y_6 + Y_7Y_4 + Y_7Y_1 + Y_7Y_0 + Y_6Y_5 + Y_6Y_4 + Y_6Y_3 + Y_6Y_2 + Y_5Y_4 + Y_5Y_3 + Y_5Y_2 + \\
& Y_5Y_1 + Y_4Y_0 + Y_8 + Y_7 + Y_5
\end{aligned}
$$

Table 16: LILI-128 degree-reductor polynomials 12-14.

$$
\begin{aligned}
red_{15} =& Y_9Y_8Y_3Y_0 + Y_9Y_8Y_1Y_0 + Y_9Y_7Y_3Y_2 + Y_9Y_7Y_3Y_0 + Y_9Y_6Y_3Y_2 + Y_9Y_6Y_1Y_0 + Y_9Y_5Y_3Y_1 + Y_9Y_5Y_3Y_0 + \\
& Y_9Y_5Y_1Y_0 + Y_9Y_4Y_3Y_1 + Y_9Y_4Y_3Y_0 + Y_9Y_4Y_1Y_0 + Y_9Y_3Y_2Y_1 + Y_8Y_7Y_3Y_2 + Y_8Y_7Y_3Y_1 + Y_8Y_7Y_2Y_1 + \\
& Y_8Y_6Y_3Y_0 + Y_8Y_6Y_1Y_0 + Y_8Y_5Y_3Y_1 + Y_8Y_5Y_3Y_0 + Y_8Y_5Y_2Y_1 + Y_8Y_4Y_3Y_2 + Y_8Y_4Y_3Y_0 + Y_8Y_3Y_2Y_1 + \\
& Y_8Y_3Y_2Y_0 + Y_8Y_2Y_1Y_0 + Y_7Y_6Y_3Y_2 + Y_7Y_6Y_3Y_1 + Y_7Y_6Y_2Y_1 + Y_7Y_5Y_3Y_1 + Y_7Y_5Y_3Y_0 + Y_7Y_5Y_2Y_1 + \\
& Y_7Y_4Y_3Y_2 + Y_7Y_4Y_3Y_1 + Y_7Y_4Y_2Y_1 + Y_7Y_3Y_2Y_0 + Y_7Y_2Y_1Y_0 + Y_6Y_5Y_3Y_2 + Y_6Y_5Y_3Y_1 + Y_6Y_5Y_3Y_0 + \\
& Y_6Y_5Y_2Y_1 + Y_6Y_5Y_1Y_0 + Y_6Y_4Y_3Y_2 + Y_6Y_4Y_1Y_0 + Y_6Y_3Y_2Y_1 + Y_6Y_3Y_2Y_0 + Y_5Y_4Y_3Y_2 + Y_5Y_4Y_2Y_1 + \\
& Y_5Y_3Y_2Y_1 + Y_5Y_3Y_1Y_0 + Y_5Y_2Y_1Y_0 + Y_4Y_3Y_2Y_0 + Y_4Y_2Y_1Y_0 + Y_9Y_8Y_0 + Y_9Y_7Y_2 + Y_9Y_7Y_0 + \\
& Y_9Y_6Y_2 + Y_9Y_5Y_1 + Y_9Y_5Y_0 + Y_9Y_4Y_1 + Y_9Y_4Y_0 + Y_9Y_3Y_1 + Y_9Y_1Y_0 + Y_8Y_7Y_3 + Y_8Y_7Y_2 + Y_8Y_7Y_1 + \\
& Y_8Y_6Y_0 + Y_8Y_5Y_1 + Y_8Y_5Y_0 + Y_8Y_4Y_3 + Y_8Y_4Y_2 + Y_8Y_4Y_0 + Y_8Y_3Y_2 + Y_8Y_3Y_0 + Y_8Y_2Y_1 + Y_8Y_2Y_0 + \\
& Y_7Y_6Y_3 + Y_7Y_6Y_2 + Y_7Y_6Y_1 + Y_7Y_5Y_3 + Y_7Y_5Y_1 + Y_7Y_5Y_0 + Y_7Y_4Y_3 + Y_7Y_4Y_2 + Y_7Y_4Y_1 + Y_7Y_3Y_2 + \\
& Y_7Y_3Y_0 + Y_7Y_2Y_0 + Y_6Y_5Y_2 + Y_6Y_5Y_1 + Y_6Y_5Y_0 + Y_6Y_4Y_3 + Y_6Y_4Y_2 + Y_5Y_4Y_3 + Y_5Y_4Y_2 + Y_5Y_3Y_2 + \\
& Y_5Y_3Y_1 + Y_5Y_3Y_0 + Y_5Y_2Y_1 + Y_5Y_2Y_0 + Y_4Y_3Y_1 + Y_4Y_3Y_0 + Y_4Y_2Y_1 + Y_4Y_2Y_0 + Y_9Y_2 + Y_9Y_1 + \\
& Y_8Y_7 + Y_8Y_4 + Y_8Y_3 + Y_8Y_1 + Y_8Y_0 + Y_7Y_6 + Y_7Y_5 + Y_7Y_4 + Y_7Y_2 + Y_7Y_1 + Y_7Y_0 + Y_6Y_4 + \\
& Y_6Y_3 + Y_6Y_1 + Y_5Y_4 + Y_5Y_3 + Y_5Y_2 + Y_4Y_3 + Y_4Y_0 + Y_7
\end{aligned}
$$

$$
\begin{aligned}
red_{16} =& Y_9Y_8Y_3Y_1 + Y_9Y_8Y_2Y_1 + Y_9Y_8Y_2Y_0 + Y_9Y_7Y_3Y_2 + Y_9Y_7Y_3Y_1 + Y_9Y_6Y_3Y_1 + Y_9Y_6Y_3Y_0 + Y_9Y_6Y_2Y_1 + \\
& Y_9Y_6Y_2Y_0 + Y_9Y_6Y_1Y_0 + Y_9Y_5Y_3Y_1 + Y_9Y_5Y_3Y_0 + Y_9Y_5Y_2Y_1 + Y_9Y_5Y_2Y_0 + Y_9Y_5Y_1Y_0 + Y_9Y_4Y_3Y_2 + \\
& Y_9Y_4Y_2Y_1 + Y_9Y_4Y_2Y_0 + Y_9Y_3Y_2Y_1 + Y_9Y_3Y_2Y_0 + Y_9Y_3Y_1Y_0 + Y_9Y_2Y_1Y_0 + Y_8Y_7Y_3Y_1 + Y_8Y_7Y_2Y_0 + \\
& Y_8Y_6Y_2Y_1 + Y_8Y_6Y_2Y_0 + Y_8Y_6Y_1Y_0 + Y_8Y_5Y_2Y_1 + Y_8Y_4Y_2Y_1 + Y_8Y_4Y_2Y_0 + Y_8Y_4Y_1Y_0 + Y_8Y_3Y_2Y_1 + \\
& Y_8Y_3Y_2Y_0 + Y_7Y_6Y_3Y_2 + Y_7Y_6Y_3Y_1 + Y_7Y_6Y_3Y_0 + Y_7Y_5Y_3Y_2 + Y_7Y_5Y_3Y_1 + Y_7Y_5Y_3Y_0 + Y_7Y_5Y_1Y_0 + \\
& Y_7Y_4Y_3Y_2 + Y_7Y_4Y_2Y_0 + Y_7Y_4Y_1Y_0 + Y_7Y_3Y_1Y_0 + Y_7Y_2Y_1Y_0 + Y_6Y_5Y_3Y_0 + Y_6Y_4Y_3Y_2 + Y_6Y_4Y_3Y_0 + \\
& Y_6Y_3Y_2Y_1 + Y_6Y_3Y_2Y_0 + Y_5Y_4Y_3Y_2 + Y_5Y_4Y_3Y_0 + Y_5Y_4Y_2Y_0 + Y_5Y_3Y_2Y_1 + Y_5Y_3Y_1Y_0 + Y_5Y_2Y_1Y_0 + \\
& Y_4Y_3Y_2Y_0 + Y_4Y_3Y_1Y_0 + Y_4Y_2Y_1Y_0 + Y_9Y_8Y_1 + Y_9Y_8Y_0 + Y_9Y_7Y_2 + Y_9Y_7Y_1 + Y_9Y_7Y_0 + \\
& Y_9Y_6Y_1 + Y_9Y_5Y_1 + Y_9Y_5Y_0 + Y_9Y_4Y_2 + Y_9Y_4Y_0 + Y_9Y_3Y_1 + Y_9Y_2Y_1 + Y_9Y_1Y_0 + Y_8Y_7Y_1 + Y_8Y_7Y_0 + \\
& Y_8Y_6Y_0 + Y_8Y_4Y_0 + Y_8Y_3Y_0 + Y_7Y_6Y_2 + Y_7Y_6Y_1 + Y_7Y_6Y_0 + Y_7Y_5Y_2 + Y_7Y_5Y_1 + Y_7Y_5Y_0 + Y_7Y_4Y_2 + \\
& Y_7Y_3Y_2 + Y_7Y_3Y_0 + Y_7Y_1Y_0 + Y_6Y_5Y_0 + Y_6Y_4Y_2 + Y_6Y_4Y_0 + Y_6Y_2Y_1 + Y_6Y_2Y_0 + Y_5Y_4Y_2 + Y_5Y_4Y_0 + \\
& Y_5Y_2Y_1 + Y_4Y_3Y_0 + Y_9Y_1 + Y_7Y_2 + Y_7Y_0 + Y_4Y_0
\end{aligned}
$$

$$
\begin{aligned}
red_{17} =& Y_9Y_8Y_3Y_2 + Y_9Y_8Y_1Y_0 + Y_9Y_7Y_3Y_2 + Y_9Y_7Y_3Y_1 + Y_9Y_7Y_1Y_0 + Y_9Y_6Y_3Y_2 + Y_9Y_6Y_3Y_1 + Y_9Y_6Y_2Y_0 + \\
& Y_9Y_6Y_1Y_0 + Y_9Y_5Y_3Y_2 + Y_9Y_5Y_3Y_1 + Y_9Y_5Y_3Y_0 + Y_9Y_5Y_2Y_1 + Y_9Y_5Y_2Y_0 + Y_9Y_4Y_2Y_0 + Y_9Y_4Y_1Y_0 + \\
& Y_9Y_3Y_2Y_1 + Y_9Y_3Y_2Y_0 + Y_9Y_2Y_1Y_0 + Y_8Y_7Y_3Y_2 + Y_8Y_7Y_3Y_1 + Y_8Y_7Y_3Y_0 + Y_8Y_7Y_2Y_0 + Y_8Y_6Y_3Y_1 + \\
& Y_8Y_6Y_3Y_0 + Y_8Y_6Y_2Y_1 + Y_8Y_6Y_2Y_0 + Y_8Y_6Y_1Y_0 + Y_8Y_5Y_3Y_2 + Y_8Y_5Y_2Y_1 + Y_8Y_5Y_2Y_0 + Y_8Y_5Y_1Y_0 + \\
& Y_8Y_4Y_3Y_2 + Y_8Y_4Y_3Y_1 + Y_8Y_4Y_3Y_0 + Y_8Y_3Y_1Y_0 + Y_7Y_6Y_3Y_2 + Y_7Y_6Y_2Y_0 + Y_7Y_5Y_3Y_1 + Y_7Y_5Y_3Y_0 + \\
& Y_7Y_4Y_3Y_2 + Y_7Y_4Y_3Y_1 + Y_7Y_4Y_2Y_1 + Y_7Y_3Y_2Y_0 + Y_7Y_3Y_1Y_0 + Y_6Y_5Y_3Y_2 + Y_6Y_5Y_1Y_0 + Y_6Y_4Y_3Y_1 + \\
& Y_6Y_4Y_1Y_0 + Y_6Y_3Y_2Y_1 + Y_6Y_3Y_2Y_0 + Y_6Y_3Y_1Y_0 + Y_5Y_4Y_2Y_1 + Y_5Y_4Y_1Y_0 + Y_5Y_3Y_2Y_1 + Y_5Y_3Y_1Y_0 + \\
& Y_4Y_3Y_2Y_1 + Y_4Y_3Y_2Y_0 + Y_4Y_3Y_1Y_0 + Y_4Y_2Y_1Y_0 + Y_9Y_8Y_3 + Y_9Y_8Y_2 + Y_9Y_8Y_1 + Y_9Y_7Y_3 + \\
& Y_9Y_7Y_2 + Y_9Y_6Y_3 + Y_9Y_6Y_1 + Y_9Y_5Y_3 + Y_9Y_5Y_2 + Y_9Y_5Y_1 + Y_9Y_4Y_3 + Y_9Y_4Y_1 + Y_9Y_3Y_1 + Y_9Y_2Y_0 + \\
& Y_8Y_7Y_3 + Y_8Y_7Y_2 + Y_8Y_6Y_1 + Y_8Y_6Y_0 + Y_8Y_5Y_3 + Y_8Y_5Y_1 + Y_8Y_5Y_0 + Y_8Y_4Y_1 + Y_8Y_4Y_0 + Y_8Y_2Y_0 + \\
& Y_8Y_1Y_0 + Y_7Y_6Y_3 + Y_7Y_6Y_1 + Y_7Y_6Y_0 + Y_7Y_5Y_2 + Y_7Y_5Y_1 + Y_7Y_5Y_0 + Y_7Y_4Y_3 + Y_7Y_4Y_0 + Y_7Y_3Y_2 + \\
& Y_7Y_3Y_1 + Y_7Y_3Y_0 + Y_7Y_2Y_1 + Y_7Y_2Y_0 + Y_7Y_1Y_0 + Y_6Y_5Y_3 + Y_6Y_5Y_2 + Y_6Y_4Y_3 + Y_6Y_4Y_1 + Y_6Y_2Y_1 + \\
& Y_6Y_2Y_0 + Y_5Y_4Y_1 + Y_5Y_4Y_0 + Y_5Y_3Y_2 + Y_5Y_2Y_1 + Y_4Y_3Y_0 + Y_4Y_2Y_0 + Y_4Y_1Y_0 + Y_9Y_2 + Y_9Y_1 + \\
& Y_8Y_6 + Y_8Y_5 + Y_8Y_0 + Y_7Y_6 + Y_7Y_5 + Y_7Y_3 + Y_7Y_0 + Y_6Y_1 + Y_6Y_0 + Y_5Y_3 + Y_5Y_2 + Y_5Y_1 + \\
& Y_6 + Y_5
\end{aligned}
$$

Table 17: LILI-128 degree-reductor polynomials 15-17.

$$
\begin{aligned}
ann_1 =& Y_7Y_3Y_2Y_1 + Y_7Y_2Y_1Y_0 + Y_6Y_3Y_2Y_0 + Y_6Y_3Y_1Y_0 + Y_6Y_2Y_1Y_0 + Y_5Y_3Y_2Y_0 + Y_5Y_3Y_1Y_0 + Y_4Y_3Y_2Y_1 + \\
& Y_4Y_3Y_1Y_0 + Y_3Y_2Y_1Y_0 + Y_9Y_3Y_2 + Y_9Y_3Y_1 + Y_9Y_2Y_1 + Y_9Y_1Y_0 + Y_8Y_3Y_2 + Y_8Y_2Y_1 + Y_8Y_1Y_0 + \\
& Y_7Y_3Y_1 + Y_7Y_2Y_1 + Y_6Y_3Y_2 + Y_6Y_3Y_1 + Y_6Y_2Y_1 + Y_6Y_2Y_0 + Y_5Y_3Y_2 + Y_5Y_3Y_1 + Y_5Y_3Y_0 + Y_5Y_2Y_0 + \\
& Y_4Y_1Y_0 + Y_3Y_2Y_1 + Y_9Y_2 + Y_9Y_1 + Y_8Y_3 + Y_8Y_2 + Y_7Y_3 + Y_7Y_1 + Y_6Y_3 + Y_6Y_2 + Y_6Y_1 + \\
& Y_5Y_2 + Y_5Y_1 + Y_5Y_0 + Y_4Y_3 + Y_1Y_0 + Y_8 + Y_7 + Y_6 + Y_4 + Y_3 + 1
\end{aligned}
$$

$$
\begin{aligned}
ann_2 =& Y_8Y_2Y_1Y_0 + Y_7Y_3Y_2Y_1 + Y_7Y_3Y_2Y_0 + Y_7Y_2Y_1Y_0 + Y_6Y_3Y_1Y_0 + Y_6Y_2Y_1Y_0 + Y_5Y_3Y_2Y_0 + Y_5Y_3Y_1Y_0 + \\
& Y_5Y_2Y_1Y_0 + Y_4Y_3Y_2Y_1 + Y_4Y_3Y_1Y_0 + Y_3Y_2Y_1Y_0 + Y_9Y_3Y_2 + Y_9Y_3Y_1 + Y_9Y_2Y_1 + Y_9Y_2Y_0 + \\
& Y_9Y_1Y_0 + Y_8Y_3Y_2 + Y_8Y_1Y_0 + Y_7Y_3Y_2 + Y_7Y_3Y_1 + Y_7Y_2Y_1 + Y_7Y_2Y_0 + Y_6Y_3Y_1 + Y_6Y_2Y_1 + Y_6Y_2Y_0 + \\
& Y_5Y_3Y_2 + Y_5Y_3Y_1 + Y_5Y_3Y_0 + Y_5Y_2Y_1 + Y_4Y_2Y_0 + Y_4Y_1Y_0 + Y_3Y_2Y_1 + Y_9Y_1 + Y_8Y_3 + Y_8Y_2 + \\
& Y_7Y_3 + Y_7Y_2 + Y_7Y_1 + Y_6Y_3 + Y_6Y_2 + Y_6Y_1 + Y_5Y_1 + Y_5Y_0 + Y_4Y_3 + Y_4Y_2 + Y_2Y_0 + Y_1Y_0 + \\
& Y_8 + Y_7 + Y_6 + Y_4 + Y_3 + Y_2 + 1
\end{aligned}
$$

$$
\begin{aligned}
ann_3 =& Y_8Y_3Y_1Y_0 + Y_8Y_2Y_1Y_0 + Y_7Y_3Y_1Y_0 + Y_6Y_3Y_1Y_0 + Y_5Y_3Y_2Y_0 + Y_4Y_3Y_2Y_0 + Y_4Y_2Y_1Y_0 + Y_9Y_3Y_2 + \\
& Y_9Y_3Y_1 + Y_9Y_2Y_1 + Y_8Y_2Y_1 + Y_7Y_3Y_2 + Y_7Y_3Y_1 + Y_6Y_3Y_2 + Y_6Y_3Y_0 + Y_6Y_2Y_1 + Y_6Y_2Y_0 + Y_5Y_3Y_1 + \\
& Y_5Y_3Y_0 + Y_5Y_2Y_1 + Y_5Y_2Y_0 + Y_5Y_1Y_0 + Y_4Y_3Y_2 + Y_4Y_3Y_1 + Y_4Y_3Y_0 + Y_2Y_1Y_0 + Y_9Y_2 + Y_8Y_2 + \\
& Y_8Y_1 + Y_7Y_1 + Y_6Y_3 + Y_6Y_2 + Y_6Y_1 + Y_5Y_3 + Y_5Y_2 + Y_5Y_0 + Y_4Y_1 + Y_3Y_2 + Y_3Y_1 + Y_8 + \\
& Y_7 + Y_6 + Y_4 + Y_1 + 1
\end{aligned}
$$

$$
\begin{aligned}
ann_4 =& Y_8Y_3Y_2Y_0 + Y_8Y_3Y_1Y_0 + Y_8Y_2Y_1Y_0 + Y_7Y_3Y_2Y_1 + Y_7Y_3Y_1Y_0 + Y_6Y_3Y_2Y_1 + Y_6Y_3Y_1Y_0 + Y_6Y_2Y_1Y_0 + \\
& Y_5Y_3Y_2Y_1 + Y_5Y_3Y_2Y_0 + Y_5Y_3Y_1Y_0 + Y_4Y_3Y_2Y_1 + Y_4Y_3Y_1Y_0 + Y_9Y_3Y_2 + Y_9Y_3Y_1 + Y_9Y_2Y_1 + \\
& Y_9Y_2Y_0 + Y_9Y_1Y_0 + Y_8Y_3Y_2 + Y_8Y_3Y_0 + Y_8Y_2Y_1 + Y_8Y_2Y_0 + Y_7Y_3Y_2 + Y_7Y_3Y_1 + Y_7Y_3Y_0 + Y_7Y_1Y_0 + \\
& Y_6Y_3Y_2 + Y_6Y_1Y_0 + Y_5Y_3Y_0 + Y_5Y_2Y_1 + Y_4Y_2Y_1 + Y_3Y_2Y_1 + Y_3Y_2Y_0 + Y_3Y_1Y_0 + Y_9Y_1 + Y_8Y_3 + \\
& Y_8Y_1 + Y_8Y_0 + Y_7Y_3 + Y_7Y_0 + Y_6Y_0 + Y_5Y_3 + Y_5Y_1 + Y_5Y_0 + Y_4Y_3 + Y_4Y_1 + Y_4Y_0 + Y_3Y_0 + \\
& Y_2Y_1 + Y_3 + Y_1 + Y_0
\end{aligned}
$$

$$
\begin{aligned}
ann_5 =& Y_8Y_3Y_2Y_1 + Y_8Y_3Y_2Y_0 + Y_8Y_3Y_1Y_0 + Y_7Y_3Y_1Y_0 + Y_7Y_2Y_1Y_0 + Y_6Y_3Y_2Y_1 + Y_6Y_3Y_2Y_0 + Y_6Y_2Y_1Y_0 + \\
& Y_5Y_3Y_2Y_1 + Y_5Y_3Y_2Y_0 + Y_5Y_3Y_1Y_0 + Y_4Y_3Y_2Y_1 + Y_4Y_3Y_2Y_0 + Y_9Y_2Y_1 + Y_8Y_3Y_2 + Y_8Y_3Y_1 + \\
& Y_8Y_3Y_0 + Y_8Y_2Y_0 + Y_8Y_1Y_0 + Y_7Y_3Y_1 + Y_7Y_3Y_0 + Y_7Y_2Y_1 + Y_7Y_2Y_0 + Y_7Y_1Y_0 + Y_6Y_3Y_2 + Y_6Y_1Y_0 + \\
& Y_5Y_3Y_2 + Y_5Y_3Y_1 + Y_5Y_3Y_0 + Y_5Y_2Y_1 + Y_5Y_2Y_0 + Y_5Y_1Y_0 + Y_4Y_3Y_2 + Y_4Y_3Y_1 + Y_4Y_2Y_1 + Y_4Y_2Y_0 + \\
& Y_4Y_1Y_0 + Y_3Y_2Y_1 + Y_3Y_2Y_0 + Y_3Y_1Y_0 + Y_9Y_2 + Y_8Y_3 + Y_8Y_0 + Y_7Y_3 + Y_7Y_2 + Y_7Y_0 + Y_6Y_0 + \\
& Y_5Y_3 + Y_5Y_2 + Y_5Y_0 + Y_4Y_3 + Y_4Y_2 + Y_4Y_0 + Y_3Y_2 + Y_3Y_1 + Y_3Y_0 + Y_2Y_1 + Y_2Y_0 + Y_1Y_0 + \\
& Y_3 + Y_2 + Y_0
\end{aligned}
$$

Table 18: LILI-128 annihilator polynomials 1-5.

$$
\begin{aligned}
ann_6 =& Y_9Y_2Y_1Y_0 + Y_8Y_3Y_2Y_1 + Y_8Y_2Y_1Y_0 + Y_7Y_3Y_2Y_0 + Y_6Y_3Y_2Y_0 + Y_6Y_2Y_1Y_0 + Y_5Y_3Y_2Y_0 + Y_5Y_3Y_1Y_0 + \\
& Y_5Y_2Y_1Y_0 + Y_4Y_3Y_2Y_1 + Y_4Y_3Y_2Y_0 + Y_4Y_3Y_1Y_0 + Y_3Y_2Y_1Y_0 + Y_9Y_2Y_1 + Y_9Y_2Y_0 + Y_8Y_3Y_2 + \\
& Y_8Y_3Y_1 + Y_8Y_2Y_1 + Y_8Y_2Y_0 + Y_8Y_1Y_0 + Y_7Y_3Y_1 + Y_7Y_1Y_0 + Y_6Y_3Y_1 + Y_6Y_2Y_1 + Y_6Y_2Y_0 + Y_6Y_1Y_0 + \\
& Y_5Y_3Y_0 + Y_5Y_2Y_1 + Y_5Y_2Y_0 + Y_4Y_3Y_2 + Y_4Y_3Y_1 + Y_4Y_3Y_0 + Y_4Y_1Y_0 + Y_3Y_2Y_1 + Y_9Y_2 + Y_8Y_3 + \\
& Y_8Y_2 + Y_8Y_1 + Y_8Y_0 + Y_7Y_3 + Y_7Y_1 + Y_7Y_0 + Y_6Y_3 + Y_6Y_2 + Y_6Y_1 + Y_6Y_0 + Y_5Y_2 + Y_4Y_3 + \\
& Y_4Y_1 + Y_4Y_0 + Y_3Y_2 + Y_3Y_1 + Y_1Y_0 + Y_8 + Y_7 + Y_6 + Y_4 + Y_3 + Y_1 + Y_0 + 1 \\
ann_7 =& Y_9Y_3Y_1Y_0 + Y_9Y_2Y_1Y_0 + Y_8Y_3Y_2Y_1 + Y_8Y_3Y_1Y_0 + Y_8Y_2Y_1Y_0 + Y_7Y_2Y_1Y_0 + Y_6Y_3Y_2Y_0 + Y_6Y_3Y_1Y_0 + \\
& Y_5Y_3Y_1Y_0 + Y_5Y_2Y_1Y_0 + Y_4Y_3Y_2Y_1 + Y_4Y_3Y_2Y_0 + Y_4Y_3Y_1Y_0 + Y_3Y_2Y_1Y_0 + Y_9Y_3Y_1 + Y_9Y_2Y_0 + \\
& Y_9Y_1Y_0 + Y_8Y_3Y_2 + Y_8Y_2Y_0 + Y_7Y_3Y_2 + Y_7Y_3Y_1 + Y_7Y_2Y_1 + Y_7Y_1Y_0 + Y_6Y_3Y_0 + Y_6Y_1Y_0 + Y_5Y_3Y_2 + \\
& Y_5Y_2Y_1 + Y_4Y_3Y_2 + Y_3Y_2Y_1 + Y_3Y_1Y_0 + Y_9Y_1 + Y_8Y_3 + Y_8Y_1 + Y_8Y_0 + Y_7Y_3 + Y_7Y_0 + \\
& Y_6Y_0 + Y_5Y_3 + Y_5Y_1 + Y_5Y_0 + Y_4Y_3 + Y_4Y_1 + Y_4Y_0 + Y_3Y_2 + Y_3 + Y_1 + Y_0
\end{aligned}
$$

$$
\begin{aligned}
ann_8 =& Y_9Y_3Y_2Y_0 + Y_9Y_2Y_1Y_0 + Y_8Y_3Y_2Y_0 + Y_6Y_3Y_2Y_1 + Y_6Y_3Y_1Y_0 + Y_5Y_3Y_2Y_1 + Y_5Y_3Y_2Y_0 + Y_5Y_3Y_1Y_0 + \\
& Y_5Y_2Y_1Y_0 + Y_4Y_3Y_2Y_0 + Y_4Y_3Y_1Y_0 + Y_3Y_2Y_1Y_0 + Y_9Y_3Y_2 + Y_9Y_1Y_0 + Y_8Y_3Y_1 + Y_8Y_3Y_0 + \\
& Y_8Y_1Y_0 + Y_7Y_3Y_2 + Y_7Y_3Y_1 + Y_7Y_3Y_0 + Y_6Y_3Y_2 + Y_6Y_2Y_1 + Y_6Y_2Y_0 + Y_5Y_2Y_0 + Y_4Y_3Y_2 + Y_4Y_3Y_0 + \\
& Y_4Y_2Y_1 + Y_4Y_1Y_0 + Y_3Y_1Y_0 + Y_9Y_2 + Y_9Y_1 + Y_8Y_2 + Y_7Y_1 + Y_6Y_3 + Y_6Y_2 + Y_6Y_1 + Y_5Y_3 + \\
& Y_5Y_2 + Y_5Y_1 + Y_5Y_0 + Y_3Y_2 + Y_3Y_0 + Y_2Y_1 + Y_1Y_0 + Y_8 + Y_7 + Y_6 + Y_4 + 1
\end{aligned}
$$

$$
\begin{aligned}
ann_9 =& Y_9Y_3Y_2Y_1 + Y_9Y_3Y_2Y_0 + Y_9Y_3Y_1Y_0 + Y_8Y_3Y_2Y_0 + Y_7Y_3Y_2Y_0 + Y_7Y_3Y_1Y_0 + Y_6Y_3Y_2Y_1 + Y_5Y_3Y_2Y_1 + \\
& Y_4Y_3Y_2Y_1 + Y_4Y_3Y_2Y_0 + Y_9Y_3Y_1 + Y_9Y_2Y_1 + Y_9Y_2Y_0 + Y_9Y_1Y_0 + Y_8Y_3Y_2 + Y_8Y_3Y_0 + \\
& Y_8Y_2Y_0 + Y_7Y_3Y_2 + Y_7Y_3Y_1 + Y_7Y_3Y_0 + Y_7Y_2Y_0 + Y_7Y_1Y_0 + Y_6Y_3Y_2 + Y_6Y_3Y_0 + Y_6Y_2Y_1 + Y_5Y_3Y_2 + \\
& Y_5Y_3Y_0 + Y_5Y_2Y_1 + Y_4Y_2Y_1 + Y_4Y_2Y_0 + Y_3Y_2Y_1 + Y_9Y_1 + Y_8Y_3 + Y_8Y_2 + Y_8Y_0 + Y_7Y_3 + \\
& Y_7Y_2 + Y_7Y_1 + Y_7Y_0 + Y_6Y_3 + Y_6Y_2 + Y_6Y_0 + Y_5Y_3 + Y_5Y_2 + Y_5Y_0 + Y_3Y_2 + Y_3Y_0 + Y_2Y_1 + \\
& Y_8 + Y_7 + Y_6 + Y_5 + Y_3 + Y_2 + Y_0 + 1
\end{aligned}
$$

Table 19: LILI-128 annihilator polynomials 6-9.

$$
\begin{aligned}
ann_{10} =& Y_9Y_7Y_1Y_0 + Y_9Y_6Y_3Y_0 + Y_9Y_6Y_2Y_0 + Y_9Y_5Y_3Y_0 + Y_9Y_5Y_2Y_1 + Y_9Y_5Y_2Y_0 + Y_9Y_4Y_3Y_1 + Y_9Y_4Y_3Y_0 + \\
& Y_9Y_4Y_2Y_0 + Y_9Y_4Y_1Y_0 + Y_9Y_3Y_2Y_1 + Y_9Y_2Y_1Y_0 + Y_8Y_7Y_3Y_2 + Y_8Y_7Y_3Y_1 + Y_8Y_7Y_3Y_0 + Y_8Y_7Y_2Y_1 + \\
& Y_8Y_7Y_2Y_0 + Y_8Y_6Y_3Y_0 + Y_8Y_6Y_2Y_1 + Y_8Y_6Y_2Y_0 + Y_8Y_5Y_3Y_2 + Y_8Y_5Y_3Y_1 + Y_8Y_5Y_2Y_1 + Y_8Y_5Y_1Y_0 + \\
& Y_8Y_4Y_3Y_1 + Y_8Y_4Y_3Y_0 + Y_8Y_4Y_2Y_1 + Y_8Y_4Y_2Y_0 + Y_8Y_3Y_2Y_1 + Y_8Y_3Y_2Y_0 + Y_8Y_3Y_1Y_0 + Y_8Y_2Y_1Y_0 + \\
& Y_7Y_6Y_3Y_2 + Y_7Y_6Y_2Y_1 + Y_7Y_6Y_2Y_0 + Y_7Y_5Y_3Y_1 + Y_7Y_5Y_3Y_0 + Y_7Y_5Y_2Y_1 + Y_7Y_5Y_2Y_0 + Y_7Y_5Y_1Y_0 + \\
& Y_7Y_4Y_2Y_0 + Y_7Y_4Y_1Y_0 + Y_7Y_3Y_2Y_1 + Y_7Y_2Y_1Y_0 + Y_6Y_5Y_3Y_2 + Y_6Y_5Y_3Y_0 + Y_6Y_5Y_2Y_0 + Y_6Y_5Y_1Y_0 + \\
& Y_6Y_4Y_3Y_2 + Y_6Y_4Y_3Y_1 + Y_6Y_4Y_2Y_1 + Y_6Y_4Y_2Y_0 + Y_6Y_4Y_1Y_0 + Y_6Y_3Y_2Y_0 + Y_6Y_3Y_1Y_0 + Y_6Y_2Y_1Y_0 + \\
& Y_5Y_4Y_3Y_2 + Y_5Y_4Y_3Y_0 + Y_5Y_4Y_2Y_0 + Y_5Y_3Y_2Y_1 + Y_5Y_3Y_1Y_0 + Y_5Y_2Y_1Y_0 + Y_4Y_3Y_1Y_0 + Y_3Y_2Y_1Y_0 + \\
& Y_9Y_8Y_1 + Y_9Y_7Y_1 + Y_9Y_6Y_2 + Y_9Y_6Y_0 + Y_9Y_4Y_3 + Y_9Y_4Y_0 + Y_9Y_3Y_2 + Y_9Y_2Y_0 + Y_8Y_7Y_3 + \\
& Y_8Y_7Y_2 + Y_8Y_6Y_0 + Y_8Y_5Y_3 + Y_8Y_4Y_3 + Y_8Y_4Y_2 + Y_8Y_4Y_1 + Y_8Y_3Y_0 + Y_8Y_2Y_0 + Y_8Y_1Y_0 + Y_7Y_6Y_3 + \\
& Y_7Y_6Y_1 + Y_7Y_6Y_0 + Y_7Y_5Y_2 + Y_7Y_5Y_1 + Y_7Y_4Y_2 + Y_7Y_4Y_1 + Y_7Y_3Y_2 + Y_7Y_3Y_0 + Y_7Y_2Y_1 + Y_7Y_2Y_0 + \\
& Y_7Y_1Y_0 + Y_6Y_5Y_3 + Y_6Y_5Y_2 + Y_6Y_4Y_2 + Y_6Y_4Y_1 + Y_6Y_4Y_0 + Y_6Y_3Y_2 + Y_6Y_3Y_1 + Y_6Y_3Y_0 + Y_6Y_2Y_1 + \\
& Y_6Y_1Y_0 + Y_5Y_4Y_3 + Y_5Y_4Y_2 + Y_5Y_4Y_1 + Y_5Y_4Y_0 + Y_5Y_3Y_2 + Y_5Y_2Y_1 + Y_4Y_3Y_0 + Y_4Y_2Y_0 + Y_4Y_1Y_0 + \\
& Y_3Y_2Y_1 + Y_3Y_2Y_0 + Y_3Y_1Y_0 + Y_2Y_1Y_0 + Y_9Y_8 + Y_9Y_7 + Y_9Y_6 + Y_9Y_5 + Y_9Y_1 + Y_8Y_6 + Y_8Y_5 + \\
& Y_8Y_4 + Y_8Y_1 + Y_8Y_0 + Y_7Y_6 + Y_7Y_5 + Y_7Y_4 + Y_7Y_1 + Y_7Y_0 + Y_6Y_4 + Y_6Y_1 + Y_6Y_0 + Y_5Y_4 + \\
& Y_5Y_1 + Y_5Y_0 + Y_4Y_2 + Y_4Y_1 + Y_4Y_0 + Y_3Y_2 + Y_3Y_0 + Y_2Y_1 + Y_1Y_0 + Y_9 + Y_8 + Y_7 + \\
& Y_6 + Y_5 + Y_4 + Y_1 + Y_0 + 1
\end{aligned}
$$

$$
\begin{aligned}
ann_{11} =& Y_9Y_8Y_2Y_1 + Y_9Y_7Y_3Y_2 + Y_9Y_7Y_3Y_1 + Y_9Y_6Y_3Y_1 + Y_9Y_6Y_3Y_0 + Y_9Y_6Y_2Y_0 + Y_9Y_5Y_3Y_1 + Y_9Y_5Y_2Y_1 + \\
& Y_9Y_5Y_1Y_0 + Y_9Y_4Y_3Y_2 + Y_9Y_4Y_2Y_1 + Y_9Y_4Y_1Y_0 + Y_9Y_3Y_2Y_0 + Y_9Y_2Y_1Y_0 + Y_8Y_7Y_3Y_1 + Y_8Y_7Y_3Y_0 + \\
& Y_8Y_6Y_3Y_2 + Y_8Y_6Y_3Y_1 + Y_8Y_5Y_3Y_2 + Y_8Y_5Y_3Y_1 + Y_8Y_5Y_2Y_1 + Y_8Y_4Y_2Y_0 + Y_8Y_4Y_1Y_0 + Y_8Y_3Y_2Y_0 + \\
& Y_8Y_2Y_1Y_0 + Y_7Y_6Y_3Y_1 + Y_7Y_6Y_3Y_0 + Y_7Y_6Y_2Y_1 + Y_7Y_6Y_2Y_0 + Y_7Y_5Y_3Y_1 + Y_7Y_5Y_3Y_0 + Y_7Y_5Y_2Y_1 + \\
& Y_7Y_5Y_1Y_0 + Y_7Y_4Y_3Y_2 + Y_7Y_4Y_3Y_0 + Y_7Y_4Y_2Y_1 + Y_7Y_4Y_2Y_0 + Y_7Y_3Y_2Y_1 + Y_7Y_3Y_2Y_0 + Y_7Y_2Y_1Y_0 + \\
& Y_6Y_5Y_3Y_0 + Y_6Y_5Y_2Y_0 + Y_6Y_5Y_1Y_0 + Y_6Y_4Y_1Y_0 + Y_6Y_3Y_1Y_0 + Y_5Y_4Y_3Y_1 + Y_5Y_4Y_2Y_0 + Y_5Y_3Y_2Y_1 + \\
& Y_5Y_3Y_2Y_0 + Y_5Y_3Y_1Y_0 + Y_5Y_2Y_1Y_0 + Y_4Y_3Y_1Y_0 + Y_4Y_2Y_1Y_0 + Y_3Y_2Y_1Y_0 + Y_9Y_8Y_3 + Y_9Y_8Y_1 + \\
& Y_9Y_8Y_0 + Y_9Y_7Y_3 + Y_9Y_7Y_2 + Y_9Y_7Y_0 + Y_9Y_6Y_3 + Y_9Y_6Y_2 + Y_9Y_5Y_3 + Y_9Y_5Y_0 + Y_9Y_4Y_3 + Y_9Y_4Y_2 + \\
& Y_9Y_4Y_1 + Y_9Y_4Y_0 + Y_9Y_3Y_1 + Y_9Y_1Y_0 + Y_8Y_7Y_3 + Y_8Y_6Y_2 + Y_8Y_6Y_0 + Y_8Y_5Y_1 + Y_8Y_5Y_0 + Y_8Y_4Y_3 + \\
& Y_8Y_4Y_2 + Y_8Y_3Y_2 + Y_8Y_3Y_0 + Y_8Y_2Y_1 + Y_7Y_6Y_3 + Y_7Y_6Y_2 + Y_7Y_6Y_1 + Y_7Y_6Y_0 + Y_7Y_5Y_3 + Y_7Y_5Y_2 + \\
& Y_7Y_5Y_0 + Y_7Y_4Y_1 + Y_7Y_3Y_2 + Y_7Y_3Y_1 + Y_7Y_3Y_0 + Y_7Y_2Y_1 + Y_7Y_2Y_0 + Y_6Y_5Y_1 + Y_6Y_5Y_0 + Y_6Y_4Y_0 + \\
& Y_6Y_3Y_2 + Y_6Y_3Y_1 + Y_6Y_2Y_0 + Y_5Y_4Y_3 + Y_5Y_4Y_0 + Y_5Y_3Y_2 + Y_4Y_3Y_2 + Y_3Y_2Y_1 + Y_3Y_1Y_0 + Y_2Y_1Y_0 + \\
& Y_9Y_3 + Y_9Y_1 + Y_9Y_0 + Y_8Y_6 + Y_8Y_4 + Y_8Y_1 + Y_8Y_0 + Y_7Y_6 + Y_7Y_4 + Y_7Y_0 + Y_6Y_5 + Y_6Y_4 + \\
& Y_6Y_1 + Y_5Y_4 + Y_5Y_1 + Y_5Y_0 + Y_4Y_3 + Y_3Y_2 + Y_3Y_0 + Y_1Y_0 + Y_4 + Y_1 + Y_0
\end{aligned}
$$

Table 20: LILI-128 annihilator polynomials 10-11.

$$
\begin{aligned}
ann_{12} =& Y_9Y_8Y_3Y_0 + Y_9Y_8Y_1Y_0 + Y_9Y_7Y_3Y_2 + Y_9Y_7Y_3Y_0 + Y_9Y_6Y_3Y_2 + Y_9Y_6Y_1Y_0 + Y_9Y_5Y_3Y_1 + Y_9Y_5Y_3Y_0 + \\
& Y_9Y_5Y_1Y_0 + Y_9Y_4Y_3Y_1 + Y_9Y_4Y_3Y_0 + Y_9Y_4Y_1Y_0 + Y_9Y_3Y_2Y_0 + Y_8Y_7Y_3Y_2 + Y_8Y_7Y_3Y_1 + Y_8Y_7Y_2Y_1 + \\
& Y_8Y_6Y_3Y_0 + Y_8Y_6Y_1Y_0 + Y_8Y_5Y_3Y_1 + Y_8Y_5Y_3Y_0 + Y_8Y_5Y_2Y_1 + Y_8Y_4Y_3Y_2 + Y_8Y_4Y_3Y_0 + Y_8Y_3Y_2Y_0 + \\
& Y_8Y_2Y_1Y_0 + Y_7Y_6Y_3Y_2 + Y_7Y_6Y_3Y_1 + Y_7Y_6Y_2Y_1 + Y_7Y_5Y_3Y_1 + Y_7Y_5Y_3Y_0 + Y_7Y_5Y_2Y_1 + Y_7Y_4Y_3Y_2 + \\
& Y_7Y_4Y_3Y_1 + Y_7Y_4Y_2Y_1 + Y_7Y_3Y_1Y_0 + Y_6Y_5Y_3Y_2 + Y_6Y_5Y_3Y_1 + Y_6Y_5Y_3Y_0 + Y_6Y_5Y_2Y_1 + Y_6Y_5Y_1Y_0 + \\
& Y_6Y_4Y_3Y_2 + Y_6Y_4Y_1Y_0 + Y_6Y_3Y_2Y_0 + Y_6Y_2Y_1Y_0 + Y_5Y_4Y_3Y_2 + Y_5Y_4Y_2Y_1 + Y_5Y_3Y_2Y_0 + Y_9Y_8Y_0 + \\
& Y_9Y_7Y_2 + Y_9Y_7Y_0 + Y_9Y_6Y_2 + Y_9Y_5Y_1 + Y_9Y_5Y_0 + Y_9Y_4Y_1 + Y_9Y_4Y_0 + Y_9Y_3Y_1 + Y_9Y_3Y_0 + Y_9Y_2Y_1 + \\
& Y_9Y_1Y_0 + Y_8Y_7Y_3 + Y_8Y_7Y_2 + Y_8Y_7Y_1 + Y_8Y_6Y_0 + Y_8Y_5Y_1 + Y_8Y_5Y_0 + Y_8Y_4Y_3 + Y_8Y_4Y_2 + Y_8Y_4Y_0 + \\
& Y_8Y_3Y_2 + Y_8Y_2Y_1 + Y_8Y_2Y_0 + Y_8Y_1Y_0 + Y_7Y_6Y_3 + Y_7Y_6Y_2 + Y_7Y_6Y_1 + Y_7Y_5Y_3 + Y_7Y_5Y_1 + Y_7Y_5Y_0 + \\
& Y_7Y_4Y_3 + Y_7Y_4Y_2 + Y_7Y_4Y_1 + Y_7Y_3Y_2 + Y_7Y_3Y_1 + Y_7Y_2Y_1 + Y_7Y_2Y_0 + Y_7Y_1Y_0 + Y_6Y_5Y_2 + Y_6Y_5Y_1 + \\
& Y_6Y_5Y_0 + Y_6Y_4Y_3 + Y_6Y_4Y_2 + Y_6Y_3Y_2 + Y_6Y_3Y_0 + Y_6Y_1Y_0 + Y_5Y_4Y_3 + Y_5Y_4Y_2 + Y_5Y_3Y_2 + Y_5Y_3Y_0 + \\
& Y_5Y_2Y_0 + Y_4Y_3Y_2 + Y_4Y_3Y_1 + Y_4Y_3Y_0 + Y_4Y_2Y_0 + Y_4Y_1Y_0 + Y_3Y_1Y_0 + Y_9Y_2 + Y_9Y_1 + Y_9Y_0 + \\
& Y_8Y_7 + Y_8Y_4 + Y_8Y_3 + Y_8Y_1 + Y_7Y_6 + Y_7Y_5 + Y_7Y_4 + Y_7Y_3 + Y_7Y_2 + Y_6Y_4 + Y_6Y_3 + Y_6Y_2 + \\
& Y_6Y_1 + Y_6Y_0 + Y_5Y_4 + Y_5Y_3 + Y_5Y_2 + Y_5Y_1 + Y_4Y_2 + Y_4Y_0 + Y_3Y_2 + Y_2Y_0 + Y_4 + Y_3 + \\
& Y_2 + Y_1
\end{aligned}
$$

$$
\begin{aligned}
ann_{13} =& Y_9Y_8Y_3Y_1 + Y_9Y_8Y_2Y_1 + Y_9Y_8Y_2Y_0 + Y_9Y_7Y_3Y_2 + Y_9Y_7Y_3Y_1 + Y_9Y_6Y_3Y_1 + Y_9Y_6Y_3Y_0 + Y_9Y_6Y_2Y_1 + \\
& Y_9Y_6Y_2Y_0 + Y_9Y_6Y_1Y_0 + Y_9Y_5Y_3Y_1 + Y_9Y_5Y_3Y_0 + Y_9Y_5Y_2Y_1 + Y_9Y_5Y_2Y_0 + Y_9Y_5Y_1Y_0 + Y_9Y_4Y_3Y_2 + \\
& Y_9Y_4Y_2Y_1 + Y_9Y_4Y_2Y_0 + Y_9Y_3Y_2Y_1 + Y_9Y_3Y_2Y_0 + Y_8Y_7Y_3Y_1 + Y_8Y_7Y_2Y_0 + Y_8Y_6Y_2Y_1 + Y_8Y_6Y_2Y_0 + \\
& Y_8Y_6Y_1Y_0 + Y_8Y_5Y_2Y_1 + Y_8Y_4Y_2Y_1 + Y_8Y_4Y_2Y_0 + Y_8Y_4Y_1Y_0 + Y_8Y_3Y_2Y_1 + Y_8Y_3Y_1Y_0 + Y_8Y_2Y_1Y_0 + \\
& Y_7Y_6Y_3Y_2 + Y_7Y_6Y_3Y_1 + Y_7Y_6Y_3Y_0 + Y_7Y_5Y_3Y_2 + Y_7Y_5Y_3Y_1 + Y_7Y_5Y_3Y_0 + Y_7Y_5Y_1Y_0 + Y_7Y_4Y_3Y_2 + \\
& Y_7Y_4Y_2Y_0 + Y_7Y_4Y_1Y_0 + Y_7Y_3Y_1Y_0 + Y_7Y_2Y_1Y_0 + Y_6Y_5Y_3Y_0 + Y_6Y_4Y_3Y_2 + Y_6Y_4Y_3Y_0 + Y_5Y_4Y_3Y_2 + \\
& Y_5Y_4Y_3Y_0 + Y_5Y_4Y_2Y_0 + Y_5Y_3Y_2Y_0 + Y_9Y_8Y_1 + Y_9Y_8Y_0 + Y_9Y_7Y_2 + Y_9Y_7Y_1 + Y_9Y_7Y_0 + \\
& Y_9Y_6Y_1 + Y_9Y_5Y_1 + Y_9Y_5Y_0 + Y_9Y_4Y_2 + Y_9Y_4Y_0 + Y_9Y_3Y_2 + Y_9Y_3Y_1 + Y_9Y_2Y_0 + Y_9Y_1Y_0 + Y_8Y_7Y_1 + \\
& Y_8Y_7Y_0 + Y_8Y_6Y_0 + Y_8Y_4Y_0 + Y_8Y_3Y_2 + Y_8Y_2Y_1 + Y_8Y_1Y_0 + Y_7Y_6Y_2 + Y_7Y_6Y_1 + Y_7Y_6Y_0 + Y_7Y_5Y_2 + \\
& Y_7Y_5Y_1 + Y_7Y_5Y_0 + Y_7Y_4Y_2 + Y_7Y_2Y_0 + Y_6Y_5Y_0 + Y_6Y_4Y_2 + Y_6Y_4Y_0 + Y_5Y_4Y_2 + Y_5Y_4Y_0 + Y_4Y_3Y_2 + \\
& Y_4Y_3Y_0 + Y_4Y_2Y_0 + Y_3Y_2Y_0 + Y_9Y_2 + Y_9Y_1 + Y_9Y_0 + Y_8Y_3 + Y_8Y_2 + Y_7Y_3 + Y_6Y_3 + Y_5Y_3 + \\
& Y_4Y_2 + Y_4Y_0 + Y_8 + Y_7 + Y_6 + Y_5 + Y_3 + 1
\end{aligned}
$$

Table 21: LILI-128 annihilator polynomials 12-13.

$$
\begin{aligned}
ann_{14} =& Y_9Y_8Y_3Y_2 + Y_9Y_8Y_3Y_1 + Y_9Y_8Y_3Y_0 + Y_9Y_8Y_2Y_0 + Y_9Y_7Y_3Y_1 + Y_9Y_7Y_3Y_0 + Y_9Y_7Y_1Y_0 + Y_9Y_6Y_3Y_1 + \\
& Y_9Y_6Y_2Y_1 + Y_9Y_6Y_2Y_0 + Y_9Y_6Y_1Y_0 + Y_9Y_5Y_3Y_2 + Y_9Y_5Y_3Y_0 + Y_9Y_5Y_2Y_1 + Y_9Y_5Y_1Y_0 + Y_9Y_4Y_3Y_1 + \\
& Y_9Y_4Y_3Y_0 + Y_9Y_4Y_1Y_0 + Y_9Y_3Y_1Y_0 + Y_9Y_2Y_1Y_0 + Y_8Y_7Y_2Y_1 + Y_8Y_6Y_3Y_2 + Y_8Y_6Y_1Y_0 + Y_8Y_5Y_3Y_0 + \\
& Y_8Y_5Y_2Y_0 + Y_8Y_5Y_1Y_0 + Y_8Y_4Y_3Y_1 + Y_8Y_4Y_2Y_1 + Y_8Y_3Y_1Y_0 + Y_8Y_2Y_1Y_0 + Y_7Y_6Y_3Y_2 + Y_7Y_6Y_3Y_1 + \\
& Y_7Y_5Y_3Y_2 + Y_7Y_4Y_3Y_0 + Y_7Y_4Y_2Y_1 + Y_7Y_4Y_1Y_0 + Y_7Y_3Y_2Y_1 + Y_7Y_3Y_2Y_0 + Y_6Y_5Y_3Y_1 + Y_6Y_5Y_3Y_0 + \\
& Y_6Y_5Y_2Y_1 + Y_6Y_5Y_2Y_0 + Y_6Y_5Y_1Y_0 + Y_6Y_4Y_3Y_1 + Y_6Y_4Y_3Y_0 + Y_6Y_4Y_1Y_0 + Y_6Y_3Y_2Y_0 + Y_6Y_3Y_1Y_0 + \\
& Y_5Y_4Y_3Y_1 + Y_5Y_4Y_3Y_0 + Y_5Y_4Y_1Y_0 + Y_5Y_3Y_2Y_0 + Y_5Y_3Y_1Y_0 + Y_5Y_2Y_1Y_0 + Y_4Y_3Y_2Y_1 + Y_4Y_3Y_2Y_0 + \\
& Y_4Y_3Y_1Y_0 + Y_4Y_2Y_1Y_0 + Y_3Y_2Y_1Y_0 + Y_9Y_8Y_2 + Y_9Y_8Y_1 + Y_9Y_8Y_0 + Y_9Y_7Y_1 + Y_9Y_7Y_0 + \\
& Y_9Y_5Y_2 + Y_9Y_5Y_1 + Y_9Y_5Y_0 + Y_9Y_4Y_1 + Y_9Y_4Y_0 + Y_9Y_3Y_1 + Y_9Y_3Y_0 + Y_9Y_1Y_0 + Y_8Y_7Y_3 + Y_8Y_7Y_0 + \\
& Y_8Y_6Y_2 + Y_8Y_6Y_1 + Y_8Y_5Y_3 + Y_8Y_5Y_1 + Y_8Y_5Y_0 + Y_8Y_4Y_1 + Y_8Y_4Y_0 + Y_8Y_3Y_0 + Y_8Y_2Y_1 + Y_8Y_2Y_0 + \\
& Y_8Y_1Y_0 + Y_7Y_6Y_3 + Y_7Y_6Y_2 + Y_7Y_6Y_0 + Y_7Y_5Y_2 + Y_7Y_5Y_1 + Y_7Y_4Y_0 + Y_7Y_3Y_2 + Y_7Y_3Y_1 + Y_7Y_3Y_0 + \\
& Y_7Y_2Y_1 + Y_7Y_2Y_0 + Y_6Y_5Y_3 + Y_6Y_5Y_0 + Y_6Y_4Y_1 + Y_6Y_3Y_2 + Y_6Y_2Y_0 + Y_5Y_4Y_1 + Y_5Y_4Y_0 + Y_5Y_3Y_2 + \\
& Y_4Y_3Y_0 + Y_4Y_2Y_0 + Y_3Y_1Y_0 + Y_2Y_1Y_0 + Y_9Y_2 + Y_9Y_0 + Y_8Y_7 + Y_8Y_5 + Y_8Y_3 + Y_8Y_2 + \\
& Y_8Y_0 + Y_7Y_6 + Y_7Y_3 + Y_7Y_2 + Y_7Y_1 + Y_7Y_0 + Y_6Y_5 + Y_6Y_3 + Y_6Y_2 + Y_5Y_3 + Y_5Y_2 + Y_5Y_0 + \\
& Y_4Y_3 + Y_4Y_1 + Y_4Y_0 + Y_3Y_2 + Y_3Y_0 + Y_1Y_0 + Y_3 + Y_2 + Y_0
\end{aligned}
$$

$$
\begin{aligned}
ann_{15} =& Y_8Y_2Y_1Y_0 + Y_7Y_3Y_2Y_0 + Y_6Y_3Y_2Y_0 + Y_5Y_2Y_1Y_0 + Y_9Y_2Y_0 + Y_8Y_2Y_1 + Y_7Y_3Y_2 + Y_7Y_2Y_0 + \\
& Y_6Y_3Y_2 + Y_5Y_2Y_1 + Y_5Y_2Y_0 + Y_4Y_2Y_0 + Y_9Y_2 + Y_7Y_2 + Y_5Y_2 + Y_4Y_2 + Y_2Y_0 + Y_2
\end{aligned}
$$

Table 22: LILI-128 annihilator polynomials 14-15.