FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Algebraic Cryptanalysis of Small Scale Variants of the AES
Student:	Bc. Marek Bielik
Supervisor:	Mgr. Martin Jureček
Study Programme:	Informatics
Study Branch:	System Programming
Department:	Department of Theoretical Computer Science
Validity:	Until the end of summer semester 2020/21

Instructions

Algebraic cryptanalysis using Groebner bases is a modern and effective approach used for finding secret key. Good results were obtained especially for block ciphers. Student will get familiar with the area of Groebner bases and apply it on the chosen cipher.

Detailed instructions:

1) Describe Groebner bases and algorithms for their computation.
2) Convert the chosen cipher into a system of polynomial equations.
3) Propose and implement guess and determine attack on the cipher.
4) Apply a suitable program (e.g. Magma) to compute Groebner bases for the system of equations and discuss the results.

Due to the high computational complexity of the attack, a simplified version of the cipher can be considered.

References

Will be provided by the supervisor.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague October 16, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Algebraic Cryptanalysis of Small Scale Variants of the AES

Marek Bielik

Department of Theoretical Computer Science
Supervisor: Mgr. Martin Jureček

January 6, 2021

Acknowledgements

I thank my supervisor for introducing me to the realm of algebraic cryptanalysis and my parents for their generosity.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work for nonprofit purposes only, in any way that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on January 6, 2021 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Bielik, Marek. *Algebraic Cryptanalysis of Small Scale Variants of the AES.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstract

This work proposes and demonstrates new advances in algebraic cryptanalysis of small scale derivatives of the Advanced Encryption Standard (AES). We model the AES as a system of polynomial equations over GF(2), which involves only the variables of the initial key, and we subsequently attempt to solve such a system. We show, for example, that one of the attacks can recover the secret key for one round of the AES-128 under one minute on a contemporary CPU. This attack requires only two known plaintexts and their corresponding ciphertexts.

Keywords small scale variants of the AES, algebraic cryptanalysis, Gröbner bases

Abstrakt

Tato práce navrhuje a demonstruje nové postupy v algebraické kryptoanalýze zmenšených verzí šifry s názvem Advanced Encryption Standard (AES). Tuto šifru modelujeme jako systém polynomiálních rovnic nad GF(2), který zahrnuje pouze proměnné počátečního klíče, a následně se pokoušíme takový systém vyřešit. Ukážeme například, že jeden z útoků může na současném CPU obnovit tajný klíč pro jedno kolo AES-128 za méně než jednu minutu. Tento útok vyžaduje pouze dva známé otevřené texty a jejich odpovídající šifrové texty.

Klíčová slova AES a její zmenšené verze, algebraická kryptoanalýza, Gröbnerovy báze

Contents

Introduction

Gaius Julius Caesar, the Roman dictator and one of the very first cryptographers, was assassinated by his peers and stabbed to death. Alan Mathison Turing, a successful cryptanalyst of the Enigma cipher, died of cyanide poisoning caused by his own hand. The study of cryptography and cryptanalysis is up to us now.

We will begin our work by discussing the elementary algebraic structures and gradually progress towards Gröbner bases. We will then show how Gröbner bases can be used to solve systems of multivariate polynomial equations.

In the second chapter, we will describe the AES, its small scale variants, and we will discuss some algebraic properties of this cipher. We will then derive polynomial systems over GF(2) for the small scale versions of the AES. As we will show, a solution to a polynomial system describing the cipher will in fact represent the secret key for the cipher. We will see that besides solving the polynomial systems, Gröbner bases are also useful for modeling the cipher so that the resulting equations contain only the variables of the initial secret key.

Since we will be leveraging the algebraic properties of the cipher to model it as a system of equations, and we will be using algebraic techniques to obtain the solutions of the system, we are referring to this form of analyzing the cipher as algebraic cryptanalysis.

The last chapter discusses the results of our experiments. We will demonstrate the current capabilities of Gröbner bases in solving the polynomial systems from the second chapter, and we will compare their performance to a SAT solver. We will also present some techniques for reducing the polynomial systems before solving them, and we will discuss the progress of diffusion within the reduced versions of the AES.

Chapter 1

Algebraic essentials

> It is almost impossible for me to read contemporary mathematicians who, instead of saying "Petya washed his hands," write simply: "There is a $t_1 < 0$ such that the image of t_1 under the natural mapping $t_1 \mapsto Petya(t_1)$ belongs to the set of dirty hands, and a $t_2, t_1 < t_2 \leq 0$, such that the image of t_2 under the above-mentioned mapping belongs to the complement of the set defined in the preceding sentence."
>
> Vladimir Igorevich Arnol'd [1, p. 30]

The goal of this chapter is to provide an introduction into the theory of Gröbner bases for ideals in polynomial rings. This theory was introduced by Bruno Buchberger [2], who named the concept in honor of his advisor Wolfgang Gröbner (1899–1980). Buchberger also developed the fundamental algorithm for the computation of a Gröbner basis known as Buchberger's algorithm. A similar concept for ideals in power series rings was introduced by Heisuke Hironaka [3], [4].

Gröbner bases are nowadays discussed in multiple books including [5] and [6]. We will follow these books along the way as we gradually unveil the elegance and power of Gröbner bases in solving systems of polynomial equations. Further information can be also found in [7] and [8].

As we progress, we will also define necessary algebraic structures and objects that will allow us to model the AES as a system of multivariate polynomial equations involving only the variables of the initial key.

1.1 Fields, Polynomial Rings, Ideals and Varietes

Let us introduce the rudiments of abstract algebra that will allow us to progress towards the application of Gröbner bases in algebraic cryptanalysis and towards the algebraic description of the AES.

Definition 1.1.1. Let $A_1, \ldots, A_n$ be sets. Then the **Cartesian product** $A_1 \times \cdots \times A_n$ is the set of all ordered n-tuples $(a_1, \ldots, a_n)$ such that $a_i \in A_i$ for $1 \leq i \leq n$.

Definition 1.1.2. Let A and B be sets. A **map** is a set $\varphi \subseteq A \times B$ such that for each $a \in A$ there is exactly one $b \in B$ with $(a, b) \in \varphi$.

Definition 1.1.3. Let A be a set. A **binary operation** is a map from $A \times A$ to A.

Let us use the definition of a group in order to define the structures we are going to operate with throughout the rest of our work — rings, ideals, and fields. Such an approach should make the definitions of these structures shorter and emphasize their relations.

We first start with the definition of a simpler structure than a group:

Definition 1.1.4. A **monoid** is a set M with a binary operation $(a, b) \mapsto a \circ b$ such that the following two axioms hold:

(i) $(a \circ b) \circ c = a \circ (b \circ c)$ for all $a, b, c \in M$,

(ii) there is $e \in M$ such that $e \circ a = a \circ e = a$ for all $a \in M$.

A monoid is called a **commutative monoid** if, in addition to (i) and (ii), the following axiom also holds:

(iii) $a \circ b = b \circ a$ for all $a, b \in M$.

Note that since $\circ$ is a binary operation, the resulting element, $a \circ b$ is always in M for all $a, b \in M$. We say that M is closed under $\circ$ or that $\circ$ is closed on M. Also note that the first axiom is the associative property. The element e is called the **identity element** or simply the **identity**. For simplicity, we will refer to the set M as the monoid with the associated operation being implicit. We will also use this convention for all the subsequent algebraic structures, even when there will be multiple operations associated with the structure.

Definition 1.1.5. A **group** G is a monoid in which for all $a \in G$, there is $b \in G$ with $a \circ b = b \circ a = e$. A group G is an **Abelian group** if it is also a commutative monoid.

The element b in the definition above is called the **inverse** of a. Note that Abelian groups are commutative groups.

Definition 1.1.6. A **ring** is a set R with two binary operations $(a, b) \mapsto a + b$ and $(a, b) \mapsto a \cdot b$, referred to as addition and multiplication, such that the following axioms hold:

(i) the set R is an Abelian group under addition with the **additive identity** 0,

(ii) the set R is a monoid under multiplication with the **multiplicative identity** 1,

(iii) $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(a + b) \cdot c = a \cdot c + b \cdot c$ for all $a, b, c \in R$.

A ring is a **commutative ring** if, under multiplication, R is a commutative monoid.

The inverse under addition in a ring is called the **additive inverse**, and the inverse under multiplication is the **multiplicative inverse**. Note that the axiom (iii) describes the left and right distributive laws. We will usually omit the symbol for multiplication, and instead of $a \cdot b$, we will write ab. Also note that subtraction in a ring can be thought of as addition of the additive inverse.

Example 1.1.7.

(i) The sets $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$ are rings with their standard addition and multiplication.

(ii) The natural numbers do not form a ring since not all elements have their additive inverse in this set.

(iii) The set of integers modulo $n \in \mathbb{Z}$, denoted $\mathbb{Z}_n$, is a ring.

Definition 1.1.8. Let R be a ring and $\emptyset \neq I \subseteq R$. Then I is an **ideal** of R if:

(i) $a + b \in I$ for all $a, b \in I$, and

(ii) $ar \in I$ for all $a \in I$ and $r \in R$.

The ideal I is **proper** if $I \neq R$.

Note that an ideal I of a ring R is closed under addition. It is also closed under multiplication by any $r \in R$.

Proposition 1.1.9. *Let I be an ideal of a commutative ring R, then:*

(i) $a \cdot 0 = 0 \cdot a = 0$ *for all* $a \in R$.

(ii) $0 \in I$, *and*

(iii) *if* $1 \in I$ *then* I *is not proper.*

Proof.

(i) Suppose $a \in R$. Then

$$\begin{aligned} a + a \cdot 0 &= a \cdot 1 + a \cdot 0 \\ &= a\,(1 + 0) \\ &= a \cdot 1 \\ &= a. \end{aligned}$$

Adding the additive inverse of a on both sides gives $a \cdot 0 = 0$. Since R is commutative, $0 \cdot a = 0$ also holds.

(ii) Considering the previous proof, by (i) of Definition 1.1.6, we know that $0 \in R$ and by (ii) of Definition 1.1.8, we get $0 \cdot a = 0 \in I$ for any $a \in I$.

(iii) Since 1 is the multiplicative identity, we have $1 \cdot r = r \in I$ for all $r \in R$ and thus $I = R$. □

Remark 1.1.10. There is an analogy from modular arithmetic that illustrates an intuitive view of ideals — they can be regarded as a generalization of a zero in a number set such as the integers. Consider the ring $\mathbb{Z}_n$ of integers modulo a given integer $n \in \mathbb{Z}$. The exact set of integers that we identify with 0 in $\mathbb{Z}_n$ is the set $n\mathbb{Z} = \{nm \mid m \in \mathbb{Z}\}$. This set meets the criteria for being an ideal ((i) and (ii) of Definition 1.1.8) of $\mathbb{Z}$ and its elements "behave" like 0 in $\mathbb{Z}$: adding two elements of $n\mathbb{Z}$ yields another element of $n\mathbb{Z}$ and multiplying any element of $n\mathbb{Z}$ again yields an element of $n\mathbb{Z}$.

Considering our definition of rings, note that an ideal might not be a ring itself. For example, consider the ring of integers and its ideal consisting of even numbers. This ideal is not a ring since it has no multiplicative identity.

Definition 1.1.11. Let I be an ideal of a ring R. The **coset** of I defined by $a \in R$ is the set $\{b + a \mid b \in I\}$ and denoted $I + a$ or $[a]$.

Let us define the addition of cosets by $(I + a) + (I + a') = I + (a + a')$ and multiplication by $(I + a)\,(I + a') = I + aa'$. It can be shown that the set of all cosets forms a ring under these operations. The proof can be found in [9, Chapter 7.3]. This ring is denoted R/I and called the **quotient ring**.

Definition 1.1.12. A **field** $\mathbb{F}$ is a ring where the set $\mathbb{F} \setminus \{0\}$ is an Abelian group under multiplication with the **multiplicative identity** 1.

Fields with a finite number of elements are **finite fields** and are often denoted $\mathbb{F}_q$ or $\mathrm{GF}(q)$ (in honor of Évariste Galois, 1811–1832), where the number of elements q is the **order** of the field. Since we will not encounter any rings that are not commutative, we will adopt the convention that by a ring, we will mean a commutative ring. Then, the only difference between rings

and fields is that in a field, every element other than 0 has its multiplicative inverse. Note that every field is a ring as well.

Example 1.1.13.

(i) The sets $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$ are fields with their standard addition and multiplication.

(ii) The integers do not form a field since not all elements have their multiplicative inverse in this set.

(iii) The set of integers modulo $p \in \mathbb{Z}$, denoted $\mathbb{Z}_p$, is a field whenever p is prime. The primality of p ensures that each non-zero element has its multiplicative inverse. We can find the inverses by leveraging the extended Euclidean algorithm.

Remark 1.1.14. We will often work with the finite field $\mathbb{Z}_2$, which merits a short comment. We will denote this field $\mathbb{F}_2$ or GF(2). The additive and multiplicative identities are 0 and 1, respectively. The additive inverse of 0 is 0. The element 1 is also its additive and multiplicative inverse. Note that addition in this field corresponds to the *exclusive or* operation denoted XOR or $\oplus$. Also note that subtraction is identical to addition. Owing to these nice properties, we will model single bits (binary digits) as elements of $\mathbb{F}_2$.

Definition 1.1.15. If $\mathbb{F}$ is a subfield of a field $\mathbb{E}$, then we call $\mathbb{E}$ an **extension field** of $\mathbb{F}$.

For example, GF(2^2) is an extension field of GF(2). We give a more detailed way of constructing extension fields in the proof of Proposition 1.3.6.

Definition 1.1.16. Let $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}_0^n$ be an n-tuple of non-negative integers. A **monomial** in $x_1, \ldots, x_n$ is a product of the form

$$\prod_{i=1}^{n} x_i^{\alpha_i} = x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdots x_n^{\alpha_n}.$$

Let us simplify the notation by setting

$$x^{\alpha} = \prod_{i=1}^{n} x_i^{\alpha_i}.$$

The **total degree** or **degree** of a monomial x^α is the sum $\sum_{i=1}^{n} \alpha_i$. We simplify the notation again an let $|x^\alpha|$ denote the total degree of x^α. We will call the symbols $x_1, \ldots, x_n$ variables. Note that $x^\alpha = 1$ when $\alpha = (0, \ldots, 0)$ and also when $|x^\alpha| = 0$. Also note that any monomial is fully determined by α.

Definition 1.1.17. Let x^α be a monomial and let $\mathbb{F}$ be a field. A **term** with a non-zero **coefficient** $c_\alpha \in \mathbb{F}$ is the product $c_\alpha x^\alpha$.

Definition 1.1.18. A **polynomial** f with coefficients in a field $\mathbb{F}$ is a finite sum of terms in the form

$$f = \sum_{\alpha} c_\alpha \cdot x^\alpha, \quad c_\alpha \in \mathbb{F}.$$

The zero polynomial will be denoted 0.

The standard addition and multiplication of polynomials can be defined in the following way.

Definition 1.1.19. Let

$$f = \sum_{\alpha \in \mathbb{N}_0^n} c_\alpha x^\alpha \quad \text{and} \quad g = \sum_{\alpha \in \mathbb{N}_0^n} d_\alpha x^\alpha$$

be two polynomials.

(i) Their **sum** is defined as

$$f + g = \sum_{\alpha \in \mathbb{N}_0^n} (c_\alpha + d_\alpha) x^\alpha,$$

(ii) and their **product** as

$$f \cdot g = \sum_{\gamma \in \mathbb{N}_0^n} \left(\sum_{\alpha+\beta=\gamma} c_\alpha d_\beta \right) x^\gamma.$$

Note that addition consists of adding the coefficients of like powers of x.

The set of all polynomials in $x_1, \ldots, x_n$ with coefficients in a field $\mathbb{F}$ will be denoted $\mathbb{F}[x_1, \ldots, x_n]$. When the particular variables are of no relevance, we will denote the set by $\mathbb{F}[\mathbf{x}]$ for short. We will also employ the standard letters x, y and z instead of x_1, x_2 and x_3 when we discuss illustrative polynomials. Polynomials of one variable, called univariate polynomials, will be denoted by $f(x) \in \mathbb{F}[x]$.

Definition 1.1.20. Let $f = \sum c_\alpha x^\alpha \neq 0 \in \mathbb{F}[\mathbf{x}]$ be a non-zero polynomial. The **total degree** or **degree** of f, denoted $\deg(f)$, is the maximum $|x^\alpha|$ such that the corresponding coefficient c_α is nonzero. The degree of 0 is undefined.

Let $f, g \in \mathbb{F}[\mathbf{x}]$ be polynomials. We say that f *divides* g and write $f \mid g$ if $g = fh$ for some polynomial $h \in \mathbb{F}[\mathbf{x}]$. One can show that the set $\mathbb{F}[\mathbf{x}]$ satisfies all of the ring axioms under standard polynomial addition and multiplication. We will therefore refer to $\mathbb{F}[\mathbf{x}]$ as a *polynomial ring*. Not all polynomials in this ring have their multiplicative inverses, e.g., even the elementary polynomial x_1 does not have its multiplicative inverse and so $\mathbb{F}[\mathbf{x}]$ does not form a field. A proof that $\mathbb{F}[\mathbf{x}]$ forms a ring can be found in [5, Chapher 2], the authors also provide a broader outlook on polynomials by defining them in a more abstract way.

Definition 1.1.21. Let $f(x) \in \mathbb{F}[x]$ be a univariate polynomial of positive degree. The polynomial $f(x)$ is **irreducible** over $\mathbb{F}[x]$ if there is no factorization of the form $f(x) = p(x)\,q(x)$, where $p(x)$ and $q(x)$ are also univariate polynomials of positive degree in $\mathbb{F}[x]$.

Definition 1.1.22. Let $\{f_1, \ldots, f_s\} \subset \mathbb{F}[\mathbf{x}]$ be a set of polynomials. Then we set

$$\langle f_1, \ldots, f_s \rangle = \left\{ \sum_{i=1}^{s} h_i f_i \;\middle|\; h_1, \ldots, h_s \in \mathbb{F}[\mathbf{x}] \right\}.$$

Lemma 1.1.23. *If $\{f_1, \ldots, f_s\} \subset \mathbb{F}[\mathbf{x}]$ is a set of polynomials, then $\langle f_1, \ldots, f_s \rangle$ is an ideal of $\mathbb{F}[\mathbf{x}]$.*

Proof. Assume $f = \sum_{i=1}^{s} p_i f_i$ and $g = \sum_{i=1}^{s} q_i f_i$ are polynomials, and let also $h \in \mathbb{F}[\mathbf{x}]$ be a polynomial. Then the equations

$$f + g = \sum_{i=1}^{s} (p_i + q_i)\, f_i \quad \text{and}$$
$$hf = \sum_{i=1}^{s} (hp_i)\, f_i$$

show that $\langle f_1, \ldots, f_s \rangle$ meets the criteria for being an ideal of $\mathbb{F}[\mathbf{x}]$. □

Definition 1.1.24. Let $\{f_1, \ldots, f_s\} \subset \mathbb{F}[\mathbf{x}]$ be a set of polynomials and let I be an ideal such that $I = \langle f_1, \ldots, f_s \rangle$. The set $\{f_1, \ldots, f_s\}$ is a **basis** of I. We will also call $\langle f_1, \ldots, f_s \rangle$ the **ideal generated by** $\{f_1, \ldots, f_s\}$.

Remark 1.1.10 provides an intuitive view of ideals through modular arithmetic. Another analogy comes from linear algebra where the definition of subspaces can be likened to the definition of ideals of polynomial rings. Both are closed under addition. Subspaces are closed under multiplication by scalars while ideals of polynomial rings are closed under multiplication by polynomials. An ideal generated by a set of polynomials also shares similar properties with a span generated by a set of vectors, which is a structure similar to subspaces as well.

Definition 1.1.25. Let $\mathbb{F}$ be a field and n a positive integer. The n-dimensional **affine space** over $\mathbb{F}$ is the set

$$\mathbb{F}^n = \{(a_1, \ldots, a_n) \mid a_1, \ldots, a_n \in \mathbb{F}\}.$$

Remark 1.1.26. A polynomial $f \in \mathbb{F}[x_1, \ldots, x_n]$ can be regarded as a function $f : \mathbb{F}^n \mapsto \mathbb{F}$ that takes in points in the affine space $\mathbb{F}^n$ and produces elements of the field $\mathbb{F}$.

Definition 1.1.27. Let $\mathbb{F}^n$ be an affine space and let $f = f(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial. The **zero point** or **root** of f is a point $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{F}^n$ such that $f(\mathbf{a}) = 0$.

Definition 1.1.28. Let $\{f_1, \ldots, f_s\} \subset \mathbb{F}[x_1, \ldots, x_n]$ be a set of polynomials and $\mathbb{F}^n$ an affine space. The **affine variety** $V(f_1, \ldots, f_s)$ defined by $\{f_1, \ldots, f_s\}$ is the set

$$V(f_1, \ldots, f_s) = \left\{(a_1, \ldots, a_n) \in \mathbb{F}^n \;\middle|\; f_i(a_1, \ldots, a_n) = 0 \text{ for all } 1 \leq i \leq s\right\}$$

of all zero points of all the polynomials in $\{f_1, \ldots, f_s\}$.

Solving an equation that can be expressed as a polynomial in multiple variables can be seen as finding the zero points of the corresponding polynomial. Affine varieties generalize this notion to systems of polynomial equations. Considering Remark 1.1.26, we may also see varieties as geometric objects, which is briefly illustrated by the following example:

Example 1.1.29. Consider the real coordinate space $\mathbb{R}^2$ and the polynomial $f(x, y) = f(x^2 + y^2 - 1)$. The variety $V(f)$ is the unit circle centered at the origin.

We will use the following lemma to show that a given ideal is contained in another one. This is useful for proving the equality of two ideals in Example 1.1.31.

Lemma 1.1.30. *Let $I \subseteq \mathbb{F}[\mathbf{x}]$ be an ideal, and let $\{f_1, \ldots, f_s\} \subset \mathbb{F}[\mathbf{x}]$ be a set of polynomials. Then $\langle f_1, \ldots, f_s\rangle \subseteq I$ if and only if $\{f_1, \ldots, f_s\} \subseteq I$.*

Proof.

$\Longrightarrow$ Assume $\langle f_1, \ldots, f_s\rangle \subseteq I$. Each $f_i \in \{f_1, \ldots, f_s\}$ can be constructed as follows: $f_i = 0 \cdot f_1 + \cdots + 1 \cdot f_i + \cdots + 0 \cdot f_s$, and hence $\{f_1, \ldots, f_n\} \subseteq I$.

$\Longleftarrow$ Assume $\{f_1, \ldots, f_n\} \subseteq I$ and choose any $f \in \langle f_1, \ldots, f_s\rangle$ so that $f = h_1 f_1 + \cdots + h_s f_s$ where each $h_i \in \mathbb{F}[\mathbf{x}]$. We see that $f \in I$ since I is an ideal and so $\langle f_1, \ldots, f_s\rangle \subseteq I$. □

Example 1.1.31. Consider the ideals $\langle x, y\rangle$ and $\langle x + y, x - y\rangle$ in the polynomial ring $\mathbb{Q}[x, y]$. We will show that these two ideals are equal so that $\langle x, y\rangle = \langle x + y, x - y\rangle$.

We see that $x + y \in \langle x, y\rangle$ and $x - y \in \langle x, y\rangle$, so by Lemma 1.1.30, $\langle x + y, x - y\rangle \subseteq \langle x, y\rangle$. Similarly, both $x = \frac{1}{2}(x + y) + \frac{1}{2}(x - y)$ and $y = \frac{1}{2}(x + y) - \frac{1}{2}(x - y)$ are in $\langle x + y, x - y\rangle$ so that by Lemma 1.1.30, $\langle x, y\rangle \subseteq \langle x + y, x - y\rangle$ and the equality follows.

Proposition 1.1.32. *If $\{f_1, \ldots, f_s\}$ and $\{g_1, \ldots, g_t\}$ are two bases of the same ideal in $\mathbb{F}[x_1, \ldots, x_n]$, so that $\langle f_1, \ldots, f_s\rangle = \langle g_1, \ldots, g_t\rangle$, then $V(f_1, \ldots, f_s) = V(g_1, \ldots, g_t)$.*

Proof. Choose any $(a_1, \ldots, a_n) \in V(f_1, \ldots, f_s)$. We know that all polynomials in $\{f_1, \ldots, f_s\}$ are equal to zero at $(a_1, \ldots, a_n)$. Now choose any $g \in \langle g_1, \ldots, g_t \rangle$. Since $\langle g_1, \ldots, g_t \rangle = \langle f_1, \ldots, f_s \rangle$, we can write $g = \sum_{i=1}^{s} h_i f_i$, $h_i \in \mathbb{F}[x_1, \ldots, x_n]$. Then $g(a_1, \ldots, a_n) = \sum_{i=1}^{s} h_i(a_1, \ldots, a_n) \cdot f_i(a_1, \ldots, a_n) = 0$, which shows that $(a_1, \ldots, a_n) \in V(g_1, \ldots, g_t)$, which means that $V(f_1, \ldots, f_s) \subseteq V(g_1, \ldots, g_t)$. The opposite inclusion can be proved in the same way. □

The following theorem shows that every ideal can be generated by a finite basis.

Theorem 1.1.33 (Hilbert Basis Theorem)**.** *For every ideal $I \subseteq \mathbb{F}[\mathbf{x}]$ we have $I = \langle g_1, \ldots, g_m \rangle$ for some $g_1, \ldots, g_m \in I$.*

Proof. See [6, p. 77]. □

So far, we were thinking of varietes as the sets of solutions of finite sets of polynomial equations. The Hilbert Basis Theorem shows that we can also think of varietes defined by ideals.

Definition 1.1.34. Let $I \subseteq \mathbb{F}[x_1, \ldots, x_n]$ be an ideal. We denote by $V(I)$ the set

$$V(I) = \{(a_1, \ldots, a_n) \in \mathbb{F}^n \mid f(a_1, \ldots, a_n) = 0 \text{ for all } f \in I\}.$$

Proposition 1.1.35. *$V(I)$ is an affine variety. In particular, if $I = \langle f_1, \ldots, f_m \rangle$, then $V(I) = V(f_1, \ldots, f_m)$.*

Proof. See [6, p. 81]. □

Example 1.1.31 shows that an ideal may have multiple different bases while propositions 1.1.32 and 1.1.35 reveal that a variety is actually determined by the ideal generated by its basis and not by the basis itself. In combination with the Hilbert Basis Theorem, Proposition 1.1.35 also shows that even though we have infinitely many polynomials in a nonzero ideal, its variety $V(I)$ can still be defined by a finite set of polynomial equations. Proposition 1.1.35 is in fact a generalization of Proposition 1.1.32.

A system of multivariate equations can be seen as an ideal basis. Propositions 1.1.32 and 1.1.35 then give us a potential ability to change the original system to another one while keeping the exact same solution set. We will model our cipher as system of polynomial equations and then we will transform this system into a new one which will be solvable in linear time. We will show that a specific Gröbner basis can be the new system and that the transformation will be the most demanding part of the computation as regards both time and memory.

Definition 1.1.36. Let $V \subseteq \mathbb{F}^n$ be an affine variety. We define

$$I(V) = \{f \in \mathbb{F}[x_1, \ldots, x_n] \mid f(a_1, \ldots, a_n) = 0 \text{ for all } (a_1, \ldots, a_n) \in V\}.$$

Proposition 1.1.37. *If $V \subseteq \mathbb{F}^n$ is an affine variety, then $I(V) \subseteq \mathbb{F}[x_1, \ldots, x_n]$ is an ideal. We call $I(V)$ the* ***ideal of*** *V.*

Proof. We have $0 \in I(V)$ since the zero polynomial vanishes on all points in $\mathbb{F}^n$. Now let $f, g \in I(V), h \in \mathbb{F}[x_1, \ldots, x_n]$ and let $(a_1, \ldots, a_n)$ be an arbitrary point in V. We get $f(a_1, \ldots, a_n) + g(a_1, \ldots, a_n) = 0 + 0 = 0$ and $h(a_1, \ldots, a_n)\, f(a_1, \ldots, a_n) = h(a_1, \ldots, a_n) \cdot 0 = 0$, which shows that $I(V)$ is an ideal. □

We will use Proposition 1.1.37 in order to combine and subsequently reduce our polynomial systems during the experiments. As we will see, this will allow us to compute the solutions of larger systems, which we would be not able to obtain otherwise.

1.2 Monomial Orders

A Gröbner basis always pertains to a particular order on monomials. Let us therefore introduce the most fundamental ones.

Before we actually define a monomial order, let us start with a concise discussion about binary relations so that it is convenient to prove that certain orders are in fact monomial orders.

Definition 1.2.1. Let S be a non-empty set. A **binary relation** on S is a subset r of $S \times S$. The relation $\Delta(S) = \{(a, a) \mid a \in S\}$ is the **diagonal** of S.

We will use only binary relations in our work and so we will refer to them simply as relations. In order to simplify the notation, we will also employ infix notation to denote that two elements are in a relation, i.e., if r is a binary relation on S and $a, b \in S$, then $a\ r\ b$ will mean $(a, b) \in r$.

Definition 1.2.2. Let r and u be relations on S. The relation $r^{-1} = \{(a, b) \,|\, (b, a) \in r\}$ is the **inverse** of r. The **strict part** of r is the relation $r_s = r \setminus r^{-1}$, and

$$u \circ r = \{(a, c) \mid \text{there is } b \in S \text{ such that } (a, b) \in r \text{ and } (b, c) \in u\}$$

is the **product** of r and u.

Definition 1.2.3. Let r be a relation on S. Then r is

(i) **transitive** if $r \circ r \subseteq r$,

(ii) **antisymmetric** if $r \cap r^{-1} \subseteq \Delta(S)$,

(iii) **connex** if $r \cup r^{-1} = S \times S$,

(iv) a **linear order on** S if r is transitive, antisymmetric and connex.

Definition 1.2.4. Let r be a relation on S with strict part r_s and let $R \subseteq S$. An element $a \in R$ is **minimal** if there is no $b \in R$ such that $b\ r_s\ a$. A **strictly descending** (or **strictly decreasing**) **sequence** in S is an infinite sequence of elements $a_n \in S$ such that $a_{n+1}\ r_s\ a_n$ for all $n \in \mathbb{N}_0$. The relation r is **noetherian** if every non-empty subset R of S has a minimal element. The relation r is a **well-order** on S if it is a noetherian linear order on S.

A natural way to think about the strict part of a relation is to consider the natural order on $\mathbb{N}_0$, which is a linear order, where for each $m, n \in \mathbb{N}_0$; $m > n$ means $m \geq n$ and $m \neq n$. The symbol $>$ denotes the strict part of the relation $\geq$. We will also denote our orders on monomials by $\succeq$, the inverse will be $\preceq$ and the strict parts will be denoted $\succ$ and $\prec$.

We will denote by $\mathcal{M}(x_1, \ldots, x_n)$, $\mathcal{M}(\mathbf{x})$ or simply $\mathcal{M}$, the set of all monomials in the variables $x_1, \ldots, x_n$. It turns out that $\mathcal{M}$ forms an Abelian monoid under natural multiplication where we add corresponding exponents of the variables. The multiplicative identity is the monomial 1. Note that we can associate any monomial $x^\alpha \in \mathcal{M}(x_1, \ldots, x_n)$ with its n-tuple of exponents $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}_0^n$ in a one-to-one fashion. Thus, we can use the sets $\mathcal{M}$ and $\mathbb{N}_0^n$ interchangeably.

Lemma 1.2.5. *A linear order $\geq$ on S is a well-order if and only if there is no strictly descending sequence in S.*

Proof. Let us turn the lemma into its contrapositive form: $\geq$ is not a well-order if and only if there is a strictly descending sequence in S; and prove this version of the lemma.

$\Longrightarrow$ Suppose $\geq$ is not a well-order. Then there is a non-empty subset $R \subseteq S$ that has no minimal element. We can choose $a \in R$ and since a is not the minimal element, we can choose again $b \in R$ such that $a > b$, which leads to a strictly descending sequence.

$\Longleftarrow$ Suppose there is a strictly descending sequence in S. The elements of such a sequence form a non-empty subset R of S that has no minimal element. Hence, $\geq$ is not a well-order. □

Definition 1.2.6. A **monomial order** $\succeq$ is a well-order on $\mathcal{M}$, which satisfies the **property of respecting multiplication**: if $m_1 \succeq m_2$, then $n \cdot m_1 \succeq n \cdot m_2$ for all $m_1, m_2, n \in \mathcal{M}$.

The purpose of the property of respecting multiplication is that the relative ordering of monomials in a polynomial does not change when we multiply the polynomial by a monomial. Such behavior is necessary for the division algorithm described in the next section.

Definition 1.2.7 (Lexicographic order). Let $x^\alpha, x^\beta \in \mathcal{M}(x_1, \ldots, x_n)$ be monomials. We say $x^\alpha \succeq_{lex} x^\beta$ if $\alpha = \beta$ or if there is $1 \leq i \leq n$ such that $\alpha_j = \beta_j$ for $1 \leq j < i$ and $\alpha_i > \beta_i$.

Note that $\succ_{lex}$ compares the exponent n-tuples $\alpha, \beta \in \mathbb{N}_0^n$ so that $x^\alpha \succ_{lex} x^\beta$ if the left-most non-zero component of the difference $\alpha - \beta \in \mathbb{N}_0^n$ is positive.

Remark 1.2.8. Also note that the lexicographic order depends on how the underlying variables $x_1, x_2, \ldots, x_n$ are ordered. In general, there are $n!$ ways to order n variables and each of these orders has its respective lexicographic order. We will only assume the standard order where $x_1 > x_2 > \cdots > x_n$, or the alphabetical order where $x > y > z$.

Example 1.2.9.

(i) Let xy^2z^3 and xy^3 be monomials in $\mathcal{M}(x, y, z)$. Then $xy^3 \succ_{lex} xy^2z^3$ since there is $i = 2$ and $j = 1$ such that $\alpha_j = \beta_j$ and $\alpha_i > \beta_i$, where $\alpha = (1, 3, 0)$ and $\beta = (1, 2, 3)$. Also, the left-most non-zero component of the difference $\beta - \alpha = (0, 1, -3)$ is positive.

(ii) Let x, y, z be monomials in $\mathcal{M}(x, y, z)$. Then considering Remark 1.2.8 and example (i), we get $x \succ_{lex} y \succ_{lex} z$.

(iii) In the lexicographic order, note that a monomial that contains the most significant variable (as regards the underlying order) is greater than any other monomial that does not contain such a variable. For example, if x and y^3z^2 are monomials in $\mathcal{M}(x, y, z)$, then $x \succ_{lex} y^3z^2$. The reasoning is the same as in (i) and (ii).

The intuitive outlook on the lexicographic order is that it looks for the most significant variable that appears in one of the monomials and then gives preference to the monomial in which this variable has greater power.

Proposition 1.2.10. *The lexicographic order $\succeq_{lex}$ on $\mathcal{M}$ is a monomial order.*

Proof. Following the definition of the lexicographic order and the fact that the regular numerical order on $\mathbb{N}_0$ is a linear order, it is straightforward to show that for any monomials $x^\alpha, x^\beta, x^\gamma \in \mathcal{M}(x_1, \ldots, x_n)$ and $\alpha, \beta, \gamma \in \mathbb{N}_0^n$, the following conditions hold:

(transitivity) if $x^\alpha \succeq_{lex} x^\beta$ and $x^\beta \succeq_{lex} x^\gamma$, then $x^\alpha \succeq_{lex} x^\gamma$;

(antisymmetry) if $x^\alpha \succeq_{lex} x^\beta$ and $x^\alpha \preceq_{lex} x^\beta$, then $x^\alpha = x^\beta$; and

(connexity) either $x^\alpha \succeq_{lex} x^\beta$ or $x^\alpha \preceq_{lex} x^\beta$.

These properties show that $\succeq_{lex}$ is a linear order on $\mathcal{M}$.

Let us prove the property of respecting multiplication explicitly. If $x^\alpha \succeq_{lex} x^\beta$, then either $\alpha = \beta$, or there is $1 \leq i \leq n$ such that $\alpha_i - \beta_i > 0$ with $\alpha_j = \beta_j$ for $1 \leq j < i$. Also, $x^\alpha \cdot x^\gamma = x^{\alpha+\gamma}$ and $x^\beta \cdot x^\gamma = x^{\beta+\gamma}$. Comparing the results gives us $(\alpha + \gamma) - (\beta + \gamma) = \alpha - \beta$ and we see that $\alpha_i - \beta_i > 0$ with $\alpha_j = \beta_j$ for $1 \leq j < i$ again; or if $\alpha = \beta$, then $(\alpha + \gamma) = (\beta + \gamma)$. This shows that also $x^{\alpha+\gamma} \succeq_{lex} x^{\beta+\gamma}$.

The last part to prove is to show that $\succeq_{lex}$ is also noetherian, i.e a well-order. We will prove this by the following contradiction:

By Lemma 1.2.5, if $\succeq_{lex}$ is not a well-order, then there is a strictly decreasing sequence

$$x^{\alpha_{(1)}} \succ_{lex} x^{\alpha_{(2)}} \succ_{lex} \cdots$$

of elements in $\mathcal{M}(x_1, \ldots, x_n)$, where each $\alpha_{(i)} = \left(\alpha_1^{(i)}, \ldots, \alpha_n^{(i)}\right) \in \mathbb{N}_0^n$. By the definition of $\succeq_{lex}$, we also know that there exists a j such that all the first components of the n-tuples $\alpha_{(k)}$ with $k \geq j$ are equal. Continuing further, there is an $l \geq j$ such that all the second components of the n-tuples $\alpha_{(m)}$ with $m \geq l$ are all equal. We see that there must be a $p \geq l$, for which the whole n-tuples $\alpha_{(p)} = \alpha_{(p+1)} = \cdots$ are all equal. This means that the sequence is not strictly decreasing, which contradicts the lemma. □

Definition 1.2.11 (Reverse Colexicographic Order). Let $x^\alpha, x^\beta \in \mathcal{M}(x_1, \ldots, x_n)$ be monomials. We say $x^\alpha \succeq_{rclex} x^\beta$ if $\alpha = \beta$ or if there is $1 \leq i \leq n$ such that $\alpha_j = \beta_j$ for $i < j \leq n$ and $\alpha_i < \beta_i$.

Observe that $\succ_{rclex}$ compares the exponent n-tuples $\alpha, \beta \in \mathbb{N}_0^n$ so that $x^\alpha \succ_{rclex} x^\beta$ if the right-most non-zero component of the difference $\alpha - \beta \in \mathbb{N}_0^n$ is negative. Remark 1.2.8 also applies.

Example 1.2.12.

(i) Let xy^2z^3 and xy^3 be monomials in $\mathcal{M}(x, y, z)$. Then $xy^3 \succ_{rclex} xy^2z^3$ as well as in Example 1.2.9 (i), but for a different reason. There is $i = 3$ such that $\alpha_i < \beta_i$, where $\alpha = (1, 3, 0)$ and $\beta = (1, 2, 3)$. Also, the right-most non-zero component of the difference $\beta - \alpha = (0, 1, -3)$ is negative.

(ii) The lexicographic order coincides with the reverse colexicographic order for monomials in one and two variables. These orders may differ for monomials in three and more variables, as shown by the following example: let xz and y^2 be monomials in $\mathcal{M}(x, y, z)$. Then $xz \succ_{lex} y^2$, as explained in Example 1.2.9 (i), but $y^2 \succ_{rclex} xz$, as explained in example (i).

The intuitive outlook on the reverse colexicographic order is that it looks for the least significant variable that appears in one of the monomials and

then gives preference to the monomial in which this variable has lesser power. It can be thought of as a double reversal of the lexicographic order — we first reverse the underlying order of the variables and then their powers.

Equivalently to the lexicographic order, it is straightforward to show that the reverse colexicographic order is a linear order as well. However, it is not a well-order since it is possible to define the following strictly decreasing sequence

$$x_1x_2 \succ_{rclex} x_1x_2^2 \succ_{rclex} x_1x_2^3 \succ_{rclex} \cdots$$

of monomials in $\mathcal{M}(x_1, x_2)$. In this sequence, let $x^\alpha = x^{(1,n)}$ and $x^\beta = x^{(1,n+1)}$ for $n \in \mathbb{N}_{>0}$. We see that it is always the case that $x^\alpha \succ_{rclex} x^\beta$ since $\alpha_1 = \beta_1$ and $\alpha_2 < \beta_2$, and we get a strictly decreasing sequence. Hence, by Lemma 1.2.5, $\succeq_{rclex}$ is not a well-order and by Definition 1.2.6, $\succeq_{rclex}$ cannot be a monomial order either. For this reason, we will not use it to order monomials on its own, but we will use it as a "sub-order" in the definition of the next order, which will be a monomial order.

Examples 1.2.9 and 1.2.12 show that the lexicographic and reverse colexicographic orders do not take into consideration the total degree of monomials. Later in our work, we will see that in certain cases, it is desirable to order the monomials in a polynomial according to their total degree. Let us therefore introduce the following order, which allows for the total degree.

Definition 1.2.13 (Graded Reverse Lexicographic Order). Let $x^\alpha, x^\beta \in \mathcal{M}(x_1, \ldots, x_n)$ be monomials. We say $x^\alpha \succeq_{grlex} x^\beta$ if $|x^\alpha| > |x^\beta|$, or $|x^\alpha| = |x^\beta|$ and $x^\alpha \succeq_{rclex} x^\beta$.

Notice that despite its name, the graded reverse lexicographic order actually makes use of the reverse colexicographic order. There is a general consensus on such a name, so we will follow it.

Example 1.2.14.

(i) Let $x, y^2, xz \in \mathcal{M}(x, y)$ be monomials. Then $y^2 \succeq_{grlex} x$ since $|y^2| = 2 > |x| = 1$; and $y^2 \succeq_{grlex} xz$ since $|xz| = |y^2|$ and $y^2 \succeq_{rclex} xz$.

(ii) Let $x, y, z \in \mathcal{M}(x, yz)$ be monomials. Then $x \succeq_{grlex} y \succeq_{grlex} z$ since $|x| = |y| = |z|$ and $x \succeq_{rclex} y \succeq_{rclex} z$.

Proposition 1.2.15. *The graded reverse lexicographic order $\succeq_{grlex}$ on $\mathcal{M}$ is a monomial order.*

Proof. Since $\succeq_{grlex}$ first uses the usual well-order order on the total degree of monomials $|x^\alpha| \in \mathbb{N}_0$ and when $|x^\alpha| = |x^\beta|$, it decides ties using the reverse colexicographic order (which is a linear order), grlex is also linear.

It is also straightforward to show that $\succeq_{grlex}$ is a well-order since we consider only the strict part $\succ_{grlex}$, which is solely the well-order on $|x^\alpha| \in \mathbb{N}_0$.

In order to show that the property of respecting multiplication holds, consider the monomials $x^\alpha, x^\beta, x^\gamma \in \mathcal{M}(x_1, \ldots, x_n)$ with the n-tuples $\alpha, \beta, \gamma \in \mathbb{N}_0^n$. Also, $x^\alpha \cdot x^\gamma = x^{\alpha+\gamma}$ and $x^\beta \cdot x^\gamma = x^{\alpha+\gamma}$. Assume $x^\alpha \succeq_{grles} x^\beta$. If $|x^\alpha| > |x^\beta|$, then $x^{\alpha+\gamma} \succ_{grlex} x^{\beta+\gamma}$ since $|x^{\alpha+\gamma}| = |x^\alpha| + |x^\gamma| > |x^\beta| + |x^\gamma| = |x^{\beta+\gamma}|$. Also, if $|x^\alpha| = |x^\beta|$, we get $|x^{\alpha+\gamma}| = |x^{\beta+\gamma}|$ by the same argument as above and we use the reverse colexicographic order. So if $|x^\alpha| = |x^\beta|$, then $x^\alpha \succeq_{rclex} x^\beta$ (since we have assumed that $x^\alpha \succeq_{grlex} x^\beta$), which means that either $\alpha = \beta$, or there is $1 \leq i \leq n$ such that $\alpha_i - \beta_i < 0$ with $\alpha_j = \beta_j$ for $i < j \leq n$. As in the proof of Proposition 1.2.10, comparing the results gives us $(\alpha + \gamma) - (\beta + \gamma) = \alpha - \beta$ and we see that $\alpha_i - \beta_i < 0$ with $\alpha_j = \beta_j$ for $i < j \leq n$ again; or if $\alpha = \beta$, then $(\alpha + \gamma) = (\beta + \gamma)$. This shows that $x^{\alpha+\gamma} \succeq_{grlex} x^{\beta+\gamma}$ and completes the proof. □

Definition 1.2.16 (Block Order). Let $x^\alpha, x^\mathrm{A} \in \mathcal{M}(x_1, \ldots, x_n)$ and $y^\beta, y^\mathrm{B} \in \mathcal{M}(y_1, \ldots, y_m)$ be monomials, $\succeq_1$ a monomial order on $\mathcal{M}(x_1, \ldots, x_n)$ and $\succeq_2$ a monomial order on $\mathcal{M}(y_1, \ldots, y_m)$. We say $x^\alpha y^\beta \succeq_{1,2} x^\mathrm{A} y^\mathrm{B}$ on $\mathcal{M}(x_1, \ldots, x_n, y_1, \ldots, y_m)$ if $x^\alpha \succeq_1 x^\mathrm{A}$, or $x^\alpha = x^\mathrm{A}$ and $y^\beta \succeq_2 y^\mathrm{B}$.

Considering a block order from the definition above, note that $x^\alpha \succeq_{1,2} x^\mathrm{A}$ implies $x^\alpha y^\beta \succeq_{1,2} x^\mathrm{A} y^\mathrm{B}$. In combination with Gröbner bases, this observation will allow us to eliminate the variables x_i from a system of polynomials. We will leverage this in order to express the output bits of an S-box only in terms of the input bits.

1.3 Polynomial Division

Let us now present the algorithms for univariate and multivariate polynomial division. Before we start, let us first introduce leading terms and related objects which we will use in our further discussion — for example, in the definition of Gröbner bases.

Definition 1.3.1. Let $h = \sum c_\alpha x^\alpha \in \mathbb{F}[\mathbf{x}] \setminus \{0\}$ be a nonzero polynomial, F a subset of $\mathbb{F}[\mathbf{x}] \setminus \{0\}$ and let $\succeq$ be a monomial order on $\mathcal{M}(\mathbf{x})$.

(i) The **multidegree** of h is $\mathrm{multideg}(h) = \max(\alpha \in \mathbb{N}_0^n \mid c_\alpha \neq 0)$. The maximum is taken with respect to $\succeq$.

(ii) The **leading coefficient** of h is $\mathrm{LC}(h) = c_{\mathrm{multideg}(h)} \in \mathbb{F}$.

(iii) $\mathrm{LC}(F) = \{\mathrm{LC}(f) \mid f \in F\}$.

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

(v) $\mathrm{LM}(F) = \{\mathrm{LM}(f) \mid f \in F\}$.

(vi) The **leading term** of h is $\mathrm{LT}(h) = \mathrm{LC}(h) \cdot \mathrm{LM}(h)$.

(vii) $\mathrm{LT}(F) = \{\mathrm{LT}(f) \mid f \in F\}$.

Furthermore, $M(f)$ denotes the set of all monomials in f and

$$M(F) = \bigcup_{f \in F} M(f)$$

denotes the set of all monomials contained in all $f \in F$.

Example 1.3.2. Let $f = xy^2z^3 + xy^3 \in \mathbb{F}[x, y, z]$ be a polynomial and $\succeq$ a lexicographic order. Then

(i) $\mathrm{multideg}(f) = (1, 3, 0)$,

(ii) $\mathrm{LC}(f) = 1$,

(iii) $\mathrm{LM}(f) = xy^3$,

(iv) $\mathrm{LT}(f) = xy^3$ and

(v) $M(f) = \{xy^2z^3, xy^3\}$.

Univariate Division and Construction of Finite Fields

Univariate division is more straightforward and provides an introduction to multivariate division. We will also utilize univariate division in univariate polynomial rings for construction of finite fields.

Theorem 1.3.3 (Univariate Division). *Let $g(x) \in \mathbb{F}[x]$ be a univariate polynomial. Then every $f(x)$ in $\mathbb{F}[x]$ can be written as $f(x) = q(x)\, g(x) + r(x)$, where $q(x), r(x) \in \mathbb{F}[x]$ are unique and either $r(x) = 0$ or $deg(r(x)) < deg(g(x))$.*

Proof. We can use Algorithm 1.3.1 for finding the polynomials q and r. The proof of correctness and termination can be found in [6, p. 39]. □

Example 1.3.4. Let $f(x) \in \mathbb{F}[x]$ be a univariate polynomial of degree $d = \deg(f(x))$ and $\langle f(x) \rangle$ the ideal generated by $f(x)$. The elements of the quotient ring $\mathbb{F}[x]/\langle f(x) \rangle$ can be written as univariate polynomials

$$c_{d-1}x^{d-1} + \ldots + c_1x + c_0$$

in $\mathbb{F}[x]$ of degree less than d. In this quotient ring, we can use addition from Definition 1.1.19 (i). However, we define multiplication by applying Theorem 1.3.3 in the following way. Let us have two univariate polynomials $g(x), h(x) \in \mathbb{F}(x)$. There must exist another two univariate polynomials $q(x), r(x) \in \mathbb{F}(x)$ such that $g(x)\, h(x) = q(x)\, f(x) + r(x)$, where $\deg(r(x)) < \deg(f(x)) = d$. We now take $r(x)$ as the product of $g(x)$ and $h(x)$.

Algorithm 1.3.1 Univariate polynomial division

Input: univariate polynomials $\boldsymbol{g}, \boldsymbol{f}$,
Output: univariate polynomials $\boldsymbol{q}, \boldsymbol{r}$

```
1: function UNIVARIATE_DIVISION(g, f)
2:     q ← 0
3:     r ← f
4:     while r ≠ 0 & LT(g) divides LT(r) do
5:         q ← q + LT(r) / LT(g)
6:         r ← r − (LT(r) / LT(g))g
7:     return q, r
```

Proposition 1.3.5. *Let $\mathbb{F}$ be a finite field of order $q = p^n$ and $f(x) \in \mathbb{F}[x]$ a univariate polynomial of degree d. The quotient ring $\mathbb{F}[x]/\langle f(x)\rangle$, as constructed in Example 1.3.4, is a finite field of order $q^d = p^{nd}$ if and only if $f(x)$ is irreducible over $\mathbb{F}[x]$.*

Proof. The proof requires introducing further concepts and can be found in [10, Chapter 1.3]. Let us provide an intuitive insight though. We can think of irreducible polynomials as a generalization of primes. Recall Example 1.1.13 (iii), where the primality of the modulus ensured that each non-zero element had its multiplicative inverse. The situation is similar in this case as well and we can find the inverses by leveraging the extended Euclidean algorithm for univariate polynomials, see e.g. [11, p. 81]. □

Proposition 1.3.6. *Let $f \in \mathbb{F}[\mathbf{x}]$ be irreducible over the field $\mathbb{F}$. Then there exists an extension of $\mathbb{F}$ with a root of f.*

Proof. Consider the finite field $\mathbb{E} = \mathbb{F}[\mathbf{x}]/\langle f\rangle$. The elements of $\mathbb{E}$ are the cosets $[r] = r + \langle f\rangle$ with $r \in \mathbb{F}[\mathbf{x}]$. For any $a \in \mathbb{F}$, we can obtain a coset $[a]$ determined by the constant polynomial a. The mapping $a \mapsto [a]$ gives an isomorphism from $\mathbb{F}$ onto a subfield $\mathbb{F}'$ of $\mathbb{E}$, so that $\mathbb{F}'$ may be identified with $\mathbb{F}$. Particularly, $\mathbb{E}$ can be seen as an extension of $\mathbb{F}$. Now, for every $r(x) = a_0 + a_1x + \cdots + a_mx^m \in \mathbb{F}[\mathbf{x}]$ we have $[r] = [a_0 + a_1x + \cdots + a_mx^m] = [a_0] + [a_1][x] + \cdots + [a_m][x]^m = a_0 + a_1[x] + \cdots + a_m[x]^m$ by the rules for operating with cosets and the identification $[a_i] = a_i$. Hence, we can write every element of $\mathbb{E}$ as a polynomial expression in $[x]$ with coefficients in $\mathbb{F}$. We say that the extension field $\mathbb{E}$ is obtained by **adjoining** $[x]$ and we denote this field by $\mathbb{F}([x])$. Note that if $f(x) = b_0 + b_1x + \cdots + b_nx^n$, then $f([x]) = b_0 + b_1[x] + \cdots + b_n[x]^n = [b_0 + b_1x + \cdots + b_nx^n] = [f] = [0]$, so that $[x]$ is a root of f. □

Multivariate Division

We will use the following theorem in the description of algorithms for computing Gröbner bases.

Theorem 1.3.7 (**Multivariate Division**). *Let $\succeq$ be a monomial order on $\mathcal{M}(\mathbf{x})$ and let $F = (f_1, \ldots, f_m)$ be an ordered m-tuple of polynomials in $\mathbb{F}[\mathbf{x}]$. Then every $f \in \mathbb{F}[\mathbf{x}]$ can be written as $f = q_1 f_1 + \cdots + q_m f_m + r$, where $q_i, r \in \mathbb{F}[\mathbf{x}]$, and either $r = 0$ or r is a linear combination, with coefficients in $\mathbb{F}$, of monomials, none of which is divisible by any of $\mathrm{LT}(f_1), \ldots, \mathrm{LT}(f_m)$. We call r a **remainder** of f on division by F. Furthermore, if $q_i f_i \neq 0$, then $\mathrm{LM}(f) \succeq \mathrm{LM}(q_i f_i)$.*

Proof. We can use Algorithm 1.3.2 for finding the polynomials q_i and r. The proof of correctness and termination can be found in [6, p. 64]. □

Algorithm 1.3.2 Multivariate polynomial division

Input: polynomials $\boldsymbol{f_1, \ldots, f_m, f}$,
Output: polynomials $\boldsymbol{q_1, \ldots, q_m, r}$

```
1:  function MULTIVARIATE_DIVISION(f_1, ..., f_m, f)
2:      q_i ← 0
3:      r ← 0
4:      p ← f
5:      while p ≠ 0 do
6:          i ← 1
7:          divisionoccured ← False
8:          while i ≤ m & divisionoccured = False do
9:              if LT(f_i) | LT(p) then
10:                 q_i ← q_i + LT(p)/LT(f_i)
11:                 p ← p − (LT(p)/LT(f_i))f_i
12:                 divisionoccured ← True
13:             else
14:                 i ← i + 1
15:         if divisionoccured = False then
16:             r ← r + LT(p)
17:             p ← p − LT(p)
18:     return q_1, ..., q_m, r
```

We will denote by $\overline{f}^F$ the remainder on division of f by the ordered m-tuple $F = (f_1, \ldots, f_m)$.

1.4 Gröbner Bases and Systems of Equations

Let us now define Gröbner bases and show that they can be used for solving systems of equations.

Definition 1.4.1. Let $I \subseteq \mathbb{F}[\mathbf{x}]$ be an ideal different from $\{0\}$. We denote by $\mathrm{LT}(I) = \{\mathrm{LT}(f) \mid f \in I\}$ the set of leading terms of nonzero elements of I. The **ideal of leading terms** of I, generated by $\mathrm{LT}(I)$, will be denoted by $\langle \mathrm{LT}(I) \rangle$.

Definition 1.4.2. Fix a monomial order on $\mathcal{M}(\mathbf{x})$. A finite basis $G \subseteq I$ of a nonzero ideal $I \subseteq \mathbb{F}[\mathbf{x}]$ is a **Gröbner basis** if

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

The definition above says that a set $\{g_1, \ldots, g_m\} \subseteq I$ is a Gröbner basis if and only if the leading term of any element of I is divisible by some of the $\mathrm{LT}(g_i)$.

Proposition 1.4.3. *Fix a monomial order on $\mathcal{M}(\mathbf{x})$. Then every ideal $I \subseteq \mathbb{F}[\mathbf{x}]$ has a Gröbner basis.*

Proof. It can be shown that the proof can be seen as a corollary of Theorem 1.1.33. Further details can be found in [6, p. 78]. □

Definition 1.4.4. Let $G \subseteq I$ be a Gröbner basis of I. We call G a **reduced Gröbner basis** if for all $g \in G$:

(i) $\mathrm{LC}(g) = 1$.

(ii) No monomial of g is in $\langle \mathrm{LT}(G \setminus \{g\}) \rangle$.

Most computer algebra systems actually compute reduced Gröbner bases by default. It can be shown that any ideal has its reduced Gröbner basis and this basis is unique. When the second condition of Definition 1.4.4 holds for a polynomial $g \in G$, we say that g is fully reduced for G. We can obtain the reduced Gröbner basis as follows. Given $g \in G$, let $g' = \overline{g}^{G \setminus \{g\}}$ and let $G' = (G \setminus \{g\}) \cup \{g'\}$. Observe that g' is fully reduced for G'. If we keep applying this process to all elements of G until all of them are fully reduced, we end up with the reduced Gröbner basis.

Definition 1.4.5. Let $I = \langle f_1, \ldots, f_m \rangle \subseteq \mathbb{F}[x_1, \ldots, x_n]$ be an ideal. The l-th **elimination ideal** I_l is the ideal of $\mathbb{F}[x_{l+1}, \ldots, x_n]$ given by $I_l = I \cap \mathbb{F}[x_{l+1}, \ldots, x_n]$.

Theorem 1.4.6 (The Elimination Theorem)**.** *Let $G \subseteq I \subseteq \mathbb{F}[x_1, \ldots, x_n]$ be a Gröbner basis of I so that $x_1 \succeq_{lex} x_2 \succeq_{lex} \cdots \succeq_{lex} x_n$. Then, for every $0 \leq l < n$, the set $G_l = G \cap \mathbb{F}[x_{l+1}, \ldots, x_n]$ is a Gröbner basis of the l-th elimination ideal I_l.*

Proof. We know that $G_l \subseteq I_l$ by construction, so we only need to show that $\langle \mathrm{LT}(I_l)\rangle = \langle \mathrm{LT}(G_l)\rangle$ for a fixed l between 0 and n. The inclusion $\langle \mathrm{LT}(I_l)\rangle \supseteq \langle \mathrm{LT}(G_l)\rangle$ is evident and to prove $\langle \mathrm{LT}(I_l)\rangle \subseteq \langle \mathrm{LT}(G_l)\rangle$, we need to show that $\mathrm{LT}(g) \mid \mathrm{LT}(f)$ for an arbitrary $f \in I_l$ and some $g \in G_l$,.

We know that f is also in I, so $\mathrm{LT}(g) \mid \mathrm{LT}(f)$ for some $g \in G$ since G is a Gröbner basis of I. Since $f \in I_l$, $\mathrm{LT}(g)$ must consist only of $x_{l+1}, \ldots, x_n$. Now comes the crucial observation: since $x_1 \succeq_{lex} \cdots \succeq_{lex} x_n$, any monomial involving any $x_1, \ldots, x_l$ is greater than all monomials in $\mathbb{F}[x_{l+1}, \ldots, x_n]$. We see that $g \in G_l$, which proves the theorem. □

Let us present the power of the Elimination Theorem on the following example.

Example 1.4.7. Consider a system of equations where $f_1 = f_2 = f_3 = 0$ are polynomials in $\mathbb{R}[x, y, z]$ with

$$\begin{aligned} f_1 &= -16x^2 - 4xy^2 + 4xz, \\ f_2 &= 4x^2z + 2xy^2z + z, \\ f_3 &= xy^2 + 2y^2z + 1. \end{aligned}$$

If we compute the reduced Gröbner basis with $z \succeq_{lex} y \succeq_{lex} x$, we get the following polynomials:

$$\begin{aligned} g_1 &= x + \frac{1}{4}y^2 - \frac{1}{4}z, \\ g_2 &= y^4 - z^2 - 4, \\ g_3 &= y^2z - \frac{1}{9}z^2, \\ g_4 &= z^3 + \frac{81}{20}z. \end{aligned}$$

We see that the last polynomial involves only the variable z and that its only solution in $\mathbb{R}$ is $z = 0$. We can now substitute this solution into $g3$ and $g2$ and obtain two solutions, namely $y = \pm\sqrt{2}$. We can proceed further and get $x = -\frac{1}{2}$. We see that the reduced Gröbner basis allowed us to solve the system.

Definition 1.4.8. Let $\mathbb{F}_q[x_1, \ldots, x_n]$ be a polynomial ring over the finite field $\mathbb{F}_q$ with order $q = p^m$ where $p \in \mathbb{P}$ and $m \in \mathbb{N}_{>0}$. The **field equations** of $\mathbb{F}_q$ are the polynomials $x_i^q - x_i$ for every $x_i \in \{x_1, \ldots, x_n\}$.

Theorem 1.4.9 (Finiteness Theorem)**.** *Let $f_1, \ldots, f_m \in \mathbb{F}[x_1, \ldots, x_n]$ be polynomials. If we have $\langle f_1, \ldots, f_m\rangle \cap \mathbb{F}[x_i] \neq 0$ for all x_i, then $V(\langle f_1, \ldots, f_m\rangle) \subseteq \mathbb{F}^n$ is finite.*

Proof. See [6, p. 251]. □

Considering the Finiteness Theorem above, adding the field equations into our polynomial system ensures that the system will have finitely many solutions.

Theorem 1.4.10 (Hilbert's Weak Nullstellensatz)**.** *Let $f_1, \ldots, f_m \in \mathbb{F}[\mathbf{x}]$ be polynomials. Then the following are equivalent:*

(i) There exists an extension field $\mathbb{E}$ of $\mathbb{F}$ and $\mathbf{a} \in \mathbb{E}^n$ such that for all f_i we have $f_i(\mathbf{a}) = 0$.

(ii) $1 \notin \langle f_1, \ldots, f_m \rangle$.

Proof. See [5, p. 281]. □

Observe that whenever 1 belongs to any ideal $I \subseteq \mathbb{F}[\mathbf{x}]$, we immediately get $I = \mathbb{F}[\mathbf{x}]$, as shown in the proof of Proposition 1.1.9.

It can be shown that if we have a finite field $\mathbb{F}$ and its extension $\mathbb{E}$, all elements from $\mathbb{F}$ satisfy all of the field equations of $\mathbb{F}$ and no element in $\mathbb{E} \setminus \mathbb{F}$ satisfies any of these equations. Therefore, if we add the field equations into a polynomial system that consists of polynomials in $\mathbb{F}[\mathbf{x}]$, we restrict our solutions to the field $\mathbb{F}$. Let us demonstrate this fact in combination with the Hilbert's Weak Nullstellensatz on the following two examples.

Example 1.4.11. Consider a system of equations where $f_1 = f_2 = f_3 = 0$ are polynomials in GF(2) with

$$\begin{aligned} f_1 &= x + y + z, \\ f_2 &= xy + xz + yz, \\ f_3 &= xyz + 1. \end{aligned}$$

If we compute the reduced Gröbner basis with $z \succeq_{lex} y \succeq_{lex} x$, we get the following polynomials:

$$\begin{aligned} g_1 &= x + y + z, \\ g_2 &= y^2 + yz + z^2, \\ g_3 &= z^3 + 1. \end{aligned}$$

We see that the only solution to the last polynomial is $z = 1$. When we substitute this solution into g_2, we get $g_2' = y^2 + y + 1$, which has no solution in GF(2), and therefore the initial polynomial system has no solution in GF(2) either. We note that g_2' is also irreducible over GF(2)[y]. Consider the proof of Proposition 1.3.6, and let α be a root of g_2'; that is, the coset $y + \langle g_2' \rangle$

in $\mathbb{E} = \mathrm{GF}(2)[y]/\langle g_2' \rangle$. We get the finite field $\mathrm{GF}(2)(\alpha) \cong \mathrm{GF}(2^2)$ with the elements $0, 1, \alpha, \alpha+1$. If $z = 1$, the polynomial g_2 has two solutions in $\mathrm{GF}(2^2)$, namely $y = \alpha$ and $y = \alpha + 1$, since $(\alpha + 1)^2 = \alpha$. The polynomial g_1 has then also two solutions in $\mathrm{GF}(2^2)$, namely $x = \alpha$ and $x = \alpha + 1$. All of these solutions also satisfy our initial system f_1, f_2, f_3. We could also obtain further solutions if we set $z = \alpha$.

Example 1.4.12. Considering the previous example, if we add the field equations of $\mathbb{F}$ into the system, we get the following reduced Gröbner basis:

$$g_1 = 1.$$

According to the Hilbert's Weak Nullstellensatz, we can already see that the initial polynomial system f_1, f_2, f_3 has no solutions in $\mathrm{GF}(2)$.

1.5 Algorithms for Computing Gröbner Bases

This section discusses an educational version of the F4 algorithm for computing Gröbner bases. Such a version is described in [6, Chapter 10, §3] which we will follow. The algorithm was introduced in [12] by Jean-Charles Faugère in 1999. It was tailored for the degree reverse lexicographic order. The author recommends computing a Gröbner basis via F4 using this order and then convert it to the lexicographic order by another algorithm, for example the FGLM algorithm [13], in order to apply the Elimination Theorem afterwards. We will also state the Buchberger's criterion which is one of the main claims about Gröbner bases.

Definition 1.5.1. Let $f, g \neq 0 \in \mathbb{F}[\mathbf{x}]$ be polynomials. If $\mathrm{multideg}(f) = \alpha$ and $\mathrm{multideg}(g) = \beta$, then let $\gamma = (\gamma_1, \ldots, \gamma_n)$, where $\gamma_i = \max(\alpha_i, \beta_i)$ for each i. We call x^γ the **least common multiple** of $\mathrm{LM}(f)$ and $\mathrm{LM}(g)$, and we write $x^\gamma = \mathrm{lcm}(\mathrm{LM}(f), \mathrm{LM}(g))$. The **S-polynomial** of f and g is the combination

$$S(f, g) = \frac{x^\gamma}{\mathrm{LT}(f)} \cdot f - \frac{x^\gamma}{\mathrm{LT}(g)} \cdot g.$$

Example 1.5.2. Let $f = x^2y^3 + x$ and $g = 2xy^5 + y$ be polynomials in $\mathbb{R}[x, y]$ under $\succeq_{grlex}$. The least common multiple is x^2y^5 and

$$\begin{aligned} S(f, g) &= \frac{x^2y^5}{x^2y^3}\left(x^2y^3 + x\right) - \frac{x^2y^5}{2xy^5}\left(2xy^5 + y\right) \\ &= x^2y^5 + xy^2 - x^2y^5 - \frac{1}{2}xy \\ &= xy^2 - \frac{1}{2}xy. \end{aligned}$$

Observe that the S-polynomial is designed so that the leading terms of f and g cancel each other out. If both f and g consisted only of the leading terms, $S(f, g)$ would be 0.

Theorem 1.5.3 (Buchberger's Criterion)**.** *A basis $G = \{g_1, \ldots, g_m\}$ of an ideal I is a Gröbner basis if and only if $\overline{S(g_i, g_j)}^G = 0$ for all $i \neq j$.*

Proof. See [6, p. 86]. □

The theorem above, introduced by Bruno Buchberger in [2], is one of the key results about Gröbner bases. It allows us to test whether a given basis is a Gröbner basis in polynomial time. It also naturally leads to an algorithm, called the Buchberger's algorithm, which constructs a Gröbner basis for an ideal by adding nonzero remainders $\overline{S(g_i, g_j)}^G$ to G until the Buchberger's criterion eventually holds.

Definition 1.5.4. Let $\succeq$ be a monomial order on $\mathcal{M}(\mathbf{x})$ and let $G = \{g_1, \ldots, g_m\} \subseteq \mathbb{F}[\mathbf{x}]$ be a set of polynomials. For any $f \in \mathbb{F}[\mathbf{x}]$, we say that f **reduces to zero modulo** G, and write $f \to_G 0$, if f has a **standard representation**

$$f = \sum_{i=1}^{m} h_i g_i, \ h_i \in \mathbb{F}[\mathbf{x}],$$

which means that whenever $h_i g_i \neq 0$, we have $\mathrm{LM}(f) \succeq \mathrm{LM}(h_i g_i)$.

We note that $\overline{f}^G = 0$ implies $f \to_G 0$, but the converse does not hold.

Proposition 1.5.5. *A basis $G = \{g_1, \ldots, g_m\}$ of an ideal I is a Gröbner basis if and only if $S(g_i, g_j) \to_G 0$ for all $i \neq j$.*

Proof. This is a more general version of the Buchberger's criterion. The proof can be found in [6, p. 105]. □

Le us now describe Algorithm 1.5.1. The value of m records the cardinality of G throughout the whole run of the algorithm. The set B represents an unordered list of pairs of polynomials for which the corresponding S-polynomials are not known to reduce to zero. Observe that we add further pairs into B on line 17. The algorithm ends when all pairs in B have been processed. In our case, the while loop starts by selecting all pairs with the minimal degree in the set B. The degree of a pair $\{i, j\}$ is defined as

$$\deg\{i, j\} = \deg(\mathrm{lcm}(\mathrm{LM}(f_i), \mathrm{LM}(f_j))) \in \mathbb{N}_0.$$

This selection is called the *normal selection strategy* and it is recommended by Faugère. Other strategies are possible too though. The value d is used only for selecting the pairs with the minimal degree on line 7 and nowhere else. We note that d is the same as the total degree of the leading monomial of both halves

$$\frac{\mathrm{lcm}(\mathrm{LM}(f_i),\ \mathrm{LM}(f_j))}{\mathrm{LT}(f_i)} \cdot f_i \quad \text{and} \quad \frac{\mathrm{lcm}(\mathrm{LM}(f_i),\ \mathrm{LM}(f_j))}{\mathrm{LT}(f_j)} \cdot f_j \tag{1.5.1}$$

of the S-polynomial $S(f_i, f_j)$. This value might not be the same as the total degree of the S–polynomial itself — it can be smaller or greater or the same, depending on the other terms in f_i and f_j and on the monomial order.

The Buchberger's algorithm would now compute the remainders $\overline{S(f_i, f_j)}^G$ for $\{i, j\} \in B'$ and add them to G so that the Buchberger's criterion eventually holds. The F4 algorithm uses a generalized version of the Buchberger's criterion stated in Proposition 1.5.5 and computes $\overline{S(f_i, f_j)}$ defined by the equation

$$\overline{S(f_i, f_j)} = S(f_i, f_j) - c_1 x^{\alpha(1)} f_{k_1} - c_2 x^{\alpha(2)} f_{k_2} - \cdots . \tag{1.5.2}$$

When $\overline{S(f_i, f_j)}$ is included in G, we get a standard representation of $S(f_i, f_j)$ so that Proposition 1.5.5 holds.

Let us now show how $\overline{S(f_i, f_j)}$ is obtained. We start by creating the set L which contains the two polynomials from (1.5.1) for each pair $\{i, j\} \in B'$. Since $\{i, j\}$ is unordered, both are included. Recall that the difference of the two polynomials for the pair $\{i, j\}$ gives $S(f_i, f_j)$. The sets L and G are then passed to the SymbolicPreprocessing function. This function returns the matrix M representing a set of polynomials as described in the following remark.

Remark 1.5.6. A tuple F of polynomials can be represented by a matrix A and a vector v in the following way. Consider the set $M_\succeq(F) = \{m_n, \ldots, m_0\}$ of all monomials in F under the monomial order $\succeq$. Let $v = (m_n, \ldots, m_0)^T$ be the vector containing all the monomials in F ordered in decreasing order, and let $A[i, j]$ contain the coefficient of the monomial $m_j \in M_\succeq(F)$ in $f_i \in F$. The rows of the product Av then represent the original polynomials in F. We can think of v as being implicit and consider only the matrix A. The rows of A then represent the polynomials in F as well.

Let us now discuss the function SymbolicPreprocessing that creates the matrix M. The function is described in Algorithm 1.5.2. We see that M actually represents the set H. This set has the following two properties:

(i) $L \subseteq H$, and

(ii) whenever x^β is a monomial in some $f \in H$ and there exists some $f_k \in G$ with $\mathrm{LM}(f_k) \mid x^\beta$, then H contains a product $x^\alpha f_k$ with $\mathrm{LM}(x^\alpha f_k) = x^\beta$.

The algorithm starts by including all polynomials from L into H. This satisfies property (i). The algorithm then continues until all monomials in H are processed. The processed monomials are put into the set *done*. The monomials in both *done* and $M_\succeq(H)$ are ordered in decreasing order according to $\succeq$. Note that when x^β is equal to the leading monomial of one of the polynomials in L, property (ii) is satisfied, so we put all the leading monomials in H into *done* before we start the while loop. The algorithm then repeatedly

selects the largest monomial $x^\beta \in M_\succeq(H)$. If there exists some $f_k \in G$ such that $\mathrm{LM}(f_k) \mid x^\beta$, then we include $\frac{x^\beta}{\mathrm{LM}(f_k)} f_k$ into H so that property (ii) is satisfied for x^β. When there are no further monomials to consider, the algorithm returns the matrix M corresponding to H.

The next step in Algorithm 1.5.1 is to compute the row reduced echelon form of M. It can be shown that this computation gives the equations of the form (1.5.2). The reason for including the products $x^\alpha f_k$ into M was that the monomials x^β cannot appear on the left-hand side of (1.5.2) and must be canceled by something on the right-hand side. On line 12, we pick those polynomials (rows) whose leading terms are not divisible by the leading terms of any of the polynomials in M. These new polynomials correspond to the $\overline{S(f_i, f_j)}$ in (1.5.2) and are included in G so that Proposition 1.5.5 eventually holds.

For a better intuitive insight, consider again the matrix M which represents the set H. Recall that $L \subseteq H$ so M contains both halves of $S(f_i, f_j)$ for any $\{i, j\}$ that is currently in B'. The S-polynomial $S(f_i, f_j)$ is the difference of the two halves so it can be represented as a linear combination of the rows of M. The rows of N form a basis for the vector space spanned by the rows of M. This means that $S(f_i, f_j)$ can be also represented by a linear combination of the rows of N. It can be shown that this gives $S(f_i, f_j) \to_G 0$ when we include the new polynomials from N into G as described at the end of the previous paragraph.

Algorithm 1.5.1 F4 Algorithm

Input: a tuple of polynomials $\boldsymbol{F = (f_1, \ldots, f_m)}$,
Output: a Gröbner basis $\mathbf{G}$ of $\boldsymbol{I = \langle f_1, \ldots, f_m \rangle}$

function F4(F)
  $G \leftarrow F$
  $m \leftarrow |F|$
  $B \leftarrow \{\{i, j\} \mid 1 \leq i < j \leq m\}$
  while $B \neq \emptyset$ **do**
    $d \leftarrow \min(\deg(\{i, j\}) \in \mathbb{N}_0 \mid \{i, j\} \in B)$
    $B' \leftarrow \{\{i, j\} \in B \mid \deg(\{i, j\}) = d\}$
    $B \leftarrow B \setminus B'$
    $L \leftarrow \left\{ \frac{\mathrm{lcm}(\mathrm{LM}(f_i),\ \mathrm{LM}(f_j))}{\mathrm{LT}(f_i)} \cdot f_i \;\middle|\; \{i, j\} \in B' \right\}$
    $M \leftarrow$ SymbolicPreprocessing(L, G)
    $N \leftarrow$ row reduced echelon form of M
    $N^+ \leftarrow \{n \in \mathrm{rows}(N) \mid \mathrm{LM}(n) \notin \langle \mathrm{LM}(\mathrm{rows}(M)) \rangle\}$
    for $n \in N^+$ **do**
      $m \leftarrow m + 1$
      $f_m =$ polynomial form of n
      $G \leftarrow G \cup \{f_m\}$
      $B \leftarrow B \cup \{\{i, m\} \mid 1 \leq i < m\}$
  return G

Algorithm 1.5.2 Symbolic Preprocessing

Input: a set of polynomials $\boldsymbol{L}$,
a set of polynomials $\boldsymbol{G}$,
Output: a matrix $\mathbf{M}$

function SymbolicPreprocessing(L, G)
  $H \leftarrow L$
  $done \leftarrow \mathrm{LM}(H)$
  while $done \neq M_{\succeq}(H)$ **do**
    $x^\beta \leftarrow$ the largest monomial in $(M_{\succeq}(H) \setminus done)$
    $done \leftarrow done \cup \{x^\beta\}$
    if there exists $f_k \in G$ such that $\mathrm{LM}(f_k) \mid x^\beta$ **then**
      select any one $f_k \in G$ such that $\mathrm{LM}(f_k) \mid x^\beta$
      $H \leftarrow H \cup \left\{ \frac{x^\beta}{\mathrm{LM}(f_k)} f_k \right\}$
  return matrix of coefficients of H with respect to $M_{\succeq}(H)$ with
    columns in decreasing order according to $\succeq$

Chapter 2

The Advanced Encryption Standard

> Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away.
>
> Antoine de Saint-Exupéry,
> Airman's Odyssey

The Advanced Encryption Standard (referred to as the AES cipher or simply the AES) is presently one of the most popular block cipher used for symmetric encryption. The original name of the cipher is Rijndael, based on the names of two cryptographers—Joan Daemen and Vincent Rijmen—who originally designed the cipher. In 1997, the U.S. National Institute of Standards and Technology (NIST) announced the development of the AES and subsequently organized an open competition, which the Rijndael cipher won. NIST published the cipher as the Federal Information Processing Standard (FIPS) 197 [14] in 2001. The AES effectively superseded the previous Data Encryption Standard (DES).

2.1 The Structure of the AES

Let us now walk through the structure of the AES, which is thoroughly described in [14]. Further information can be found in [15], a book written by the original authors of Rijndael.

The AES is a symmetric block cipher. The block size is 128 bits. There are three possible key sizes: 128, 192 and 256. The original Rijndael cipher is more liberal as regards the block and keys sizes, as these values can be independently set to any multiple of 32 bits within the range from 128 to 256 bits. This scale is the only difference between Rijndael and the AES. We will focus our attention solely to the version with both the block and key size

set to 128 bits, which is referred to as AES-128. This restriction will have almost no impact on generality since the structure of all the ciphers remains the same up to certain constants. We will also define an even more granular and principally smaller structure of the cipher in the following section.

The input as well as the output of the cipher can be considered as a one-dimensional array of 16 bytes. The cipher operates on a two-dimensional 4×4 array of 16 bytes called the **state**.

Remark 2.1.1. Bytes in the state can be considered as polynomials of the form

$$\sum_{i=0}^{7} b_i x^i,$$

where $b_i \in \mathbb{F}_2$ are the individual bits, see Remark 1.1.14. These polynomials are the elements of the quotient ring $\mathbb{F}_2[x]/\langle f(x)\rangle$, where $f(x) = x^8 + x^4 + x^3 + x + 1 \in \mathbb{F}_2[x]$ is an irreducible polynomial over $\mathbb{F}_2[x]$, so the quotient ring $\mathbb{F}_2[x]/\langle f(x)\rangle$ is in fact a finite field — as described in Example 1.3.4 and Proposition 1.3.5. We will also denote this finite field by $\mathrm{GF}(2^8)$.

Remark 2.1.2. Another way to describe a byte is by its hexadecimal value, e.g. 63_{16} represents 01100011_2, or as described in the previous remark, 03_{16} represents the polynomial $x + 1$. Also note that a byte can be regarded as a vector in an 8-dimensional vector space over $\mathrm{GF}(2)$. We will employ all these views on bytes throughout this chapter. Moreover, a word consisting of four bytes can also be regarded as a vector in an 4-dimensional vector space over $\mathrm{GF}(2^8)$.

The overall structure of the cipher is described in Algorithm 2.1.1. At the beginning, the initial key is expanded into 44 bytes, which are then used throughout the encryption. This expansion is described in Algorithm 2.2.1 and Section 2.2. The plaintext is then copied into the state and the initial key addition is performed. The cipher then performs a cycle with nine iterations. We call these iterations **rounds**. The lines 10 and 11 comprise the tenth round, with the exception of the `MixColumns` operation being omitted. This omission is due to the design of the inverse cipher — it makes the structure of the inverse cipher more consistent with the structure described in Algorithm 2.1.1. The inverse cipher is thoroughly described in [14]. Let us remark that all the operations that manipulate the state have their inverted counterparts and the inverse cipher can be implemented by applying these inverted operations in reverse order where the key schedule is reversed as well.

Let us now go over the individual operations described in the following sections. Note that a prime on a variable (e.g. c') denotes the updated value of the variable. Also note that each byte in the state has two indices: the row index $0 \leq r < 4$ and the column index $0 \leq c < 4$, so that a specific byte in the state can be denoted $s_{r,c}$.

Algorithm 2.1.1 High Level Overview of the AES

Input: ***plaintext*** — array of 16 bytes,
key — array of 16 bytes
Output: ciphertext ***state*** — array of 16 bytes

```
1: function AES(plaintext, key)
2:     expKey ← ExpandKey(key)
3:     state ← plaintext
4:     state ← AddRoundKey(state, expKey[0 : 3])
5:     for round ← 1 to 9 do
6:         state ← MixColumns(ShiftRows(SubBytes(state)))
7:         state ← AddRoundKey(state, expKey[4·round : 4·(round+1)−1])
8:     state ← ShiftRows(SubBytes(state))
9:     state ← AddRoundKey(state, expKey[40 : 43])
10:    return state
```

SubBytes

This transformation operates on the individual bytes of the state and is depicted in Figure 2.1. We will refer to this operation by the term **S-box**. It is a composition of the following two transformations:

(i) Take the multiplicative inverse in GF(2^8), the element 00_{16} is mapped to itself.

(ii) Apply the following affine transformation over GF(2^8):

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i,$$

for $0 \leq i < 8$, where $c = 63_{16}$ and $b_i \in \mathrm{GF}(2)$.

The affine transformation can be also expressed in matrix notation:

$$\begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}. \tag{2.1.1}$$

The S-box actually represents a substitution and it is the only non-linear transformation in the AES. It can be implement as a look-up table containing the substitution values for each byte in GF(2^8).

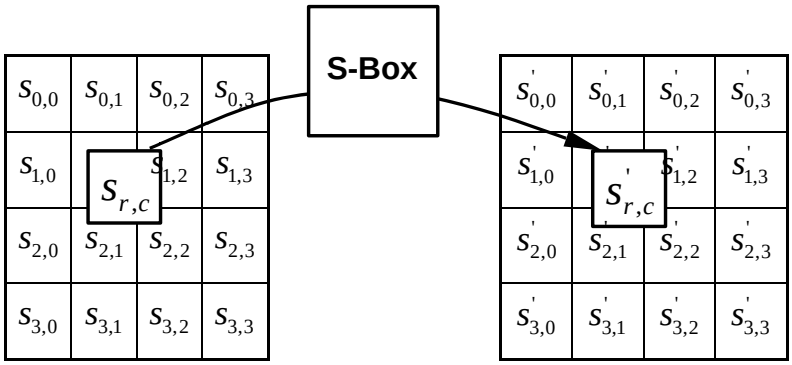

Figure 2.1: The `SubBytes` operation [14].

ShiftRows

This transformation operates on the individual rows of the state and is depicted in Figure 2.2. It is defined by the following expression:

$$s'_{r,c} = s_{r,(r+c) \bmod 4} \quad \text{for} \quad 0 < r < 4 \quad \text{and} \quad 0 \leq c < 4.$$

We can see that each row i of the state is cyclically rotated to the left by i bytes, where $0 \leq i < 4$, so that the first row remains the same and the fourth row is rotated by three bytes.

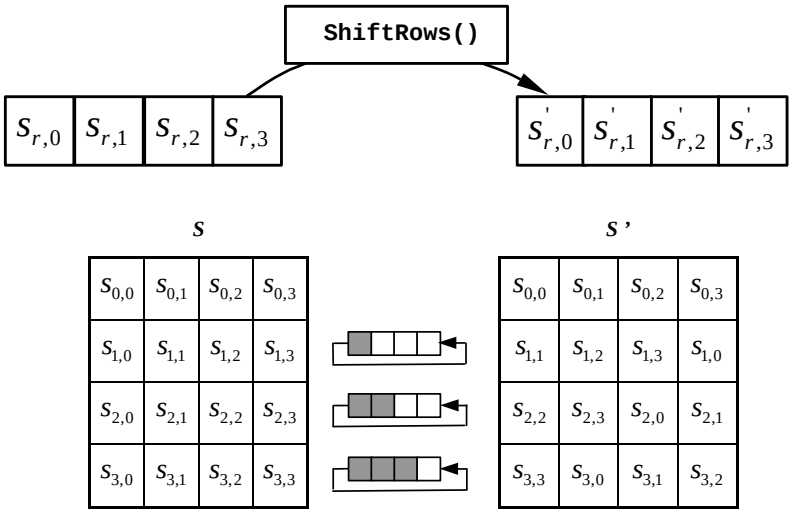

Figure 2.2: The `ShiftRows` operation [14].

Let us now show that the `ShiftRows` operation is in fact a linear transformation. A cyclic rotation by one position of a vector of length four can be

represented by the matrix

$$R_4 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}. \tag{2.1.2}$$

Rotations by more positions can be obtained by taking higher powers of R_4. If we let r_i represent the ith row of the state, we can define the `ShiftRows` operation by the following expression

$$\begin{pmatrix} r'_0 \\ r'_1 \\ r'_2 \\ r'_3 \end{pmatrix} = \begin{pmatrix} I & 0 & 0 & 0 \\ 0 & R_4 & 0 & 0 \\ 0 & 0 & R_4^2 & 0 \\ 0 & 0 & 0 & R_4^3 \end{pmatrix} \begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{pmatrix}. \tag{2.1.3}$$

MixColumns

This transformation operates on the individual columns of the state and is depicted in Figure 2.3. Each column is considered as a four-term polynomial, in which the coefficients are the four bytes constituting the column. These bytes are considered as elements of $\mathrm{GF}(2^8)$. With this set-up, each column is multiplied modulo $x^4 + 1$ with the polynomial

$$a(x) = 03_{16}x^3 + 01_{16}x^2 + 01_{16}x + 02_{16} \in \mathrm{GF}(2^8)[x].$$

The multiplication is performed as in Example 1.3.4.

The coefficients were chosen so that `MixColumns` is fast on 8-bit architectures. The constant 01_{16} requires no processing at all, polynomial multiplication by 02_{16} can be implemented by a shift and a conditional XOR. Multiplication by 03_{16} can be implemented as multiplication by 02_{16} and an additional XOR.

Note that the polynomial x^4+1 is not irreducible over $\mathrm{GF}(2^8)[x]$ since $x^4+1 = (x^2+1)(x^2+1)$. This means that the quotient ring $\mathrm{GF}(2^8)[x]/\langle x^4+1\rangle$ is not a finite field, which means that not each element has its multiplicative inverse so that multiplication by a fixed polynomial is not necessarily invertible. However, the polynomial $a(x)$ has its multiplicative inverse, namely

$$a^{-1}(x) = 0\mathrm{B}_{16}x^3 + 0\mathrm{D}_{16}x^2 + 09_{16}x + 0\mathrm{E}_{16},$$

so that multiplication by this polynomial is invertible and therefore the `MixColumns` operations remains invertible as well.

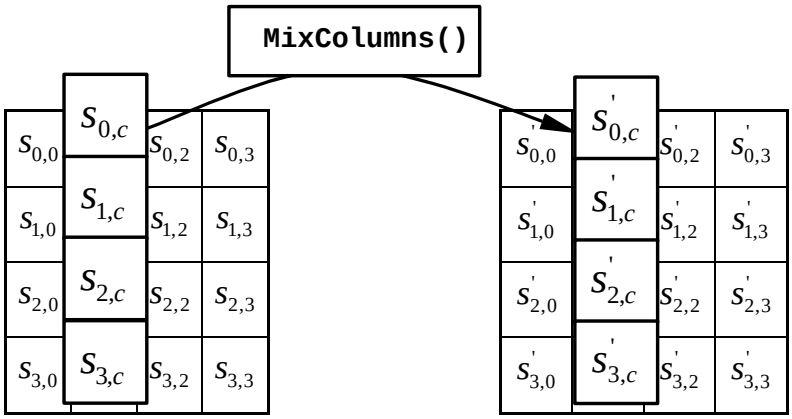

Figure 2.3: The `MicColumns` operation [14].

Remark 2.1.3. Let $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$ and $c(x) = c_3x^3 + c_2x^2 + c_1x + c_0$ be two polynomials in $\mathrm{GF}(2^8)[x]$. Addition of these two polynomials consists of adding the coefficients of like powers of x. These coefficients are the elements of $\mathrm{GF}(2^8)$, so addition of the coefficients effectively corresponds to their XOR and will be denoted by $\oplus$:

$$b(x) + c(x) = (b_3 \oplus c_3)\, x^3 + (b_2 \oplus c_2)\, x^2 + (b_1 \oplus c_1)\, x + (b_0 \oplus c_0)\,.$$

As in Example 1.3.4, multiplication modulo the polynomial $m(x) = x^4 + 1$ is performed in two stages. At first, we obtain the full product $b(x)c(x) = d(x)$ where

$$d(x) = d_6x^6 + d_5x^5 + d_4x^4 + d_3x^3 + d_2x^2 + d_1x + d_0$$

with

$$\begin{aligned}
d_0 &= b_0c_0,\\
d_1 &= b_1c_0 \oplus b_0c_1,\\
d_2 &= b_2c_0 \oplus b_1c_1 \oplus b_0c_2,\\
d_3 &= b_3c_0 \oplus b_2c_1 \oplus b_1c_2 \oplus b_0c_3,\\
d_4 &= b_3c_1 \oplus b_2c_2 \oplus b_1c_3,\\
d_5 &= b_3c_2 \oplus b_2c_3,\\
d_6 &= b_3c_3.
\end{aligned}$$

Now we divide $d(x)$ by $m(x)$ and obtain the quotient $q(x)$ and remainder $r(x)$:

$$d(x) = q(x)m(x) + r(x)$$

where $q(x) = d_6x^2 + d_5x + d_4$ and $r(x) = r_3x^3 + r_2x^2 + r_1x + r_0$ with

$$\begin{aligned} r_0 &= b_0c_0 \oplus b_3c_1 \oplus b_2c_2 \oplus b_1c_3, \\ r_1 &= b_1c_0 \oplus b_0c_1 \oplus b_3c_2 \oplus b_2c_3, \\ r_2 &= b_2c_0 \oplus b_1c_1 \oplus b_0c_2 \oplus b_3c_3, \\ r_3 &= b_3c_0 \oplus b_2c_1 \oplus b_1c_2 \oplus b_0c_3. \end{aligned}$$

Note that we may express the coefficients r_i in matrix notation:

$$\begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{pmatrix} = \begin{pmatrix} b_0 & b_3 & b_2 & b_1 \\ b_1 & b_0 & b_3 & b_2 \\ b_2 & b_1 & b_0 & b_3 \\ b_3 & b_2 & b_1 & b_0 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}. \tag{2.1.4}$$

We take the remainder $r(x)$, as the actual result of the overall multiplication of $b(x)$ and $c(x)$, so we can write

$$b(x)c(x) \equiv r(x) \pmod{m(x)}.$$

Observe that we consider $b(x)$ as a fixed polynomial and use its coefficients to fill the matrix in expression (2.1.4).

The remark above shows that multiplication by a fixed polynomial modulo another fixed polynomial can be regarded as a linear transformation, so that the `MixColumns` operation can be seen as a linear transformation as well. As shown in Remark 2.1.2, the coefficients 03_{16}, 02_{16} and 01_{16} of the polynomial $a(x) \in \mathrm{GF}(2^8)[x]$ can be written as the polynomials $x+1$, x and 1, respectively. If we associate $a(x)$ with the polynomial $b(x)$ from the remark above, we get the following expression

$$\begin{pmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{pmatrix} = \begin{pmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{pmatrix} \begin{pmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{pmatrix} \tag{2.1.5}$$

for $0 \leq c < 4$, which indexes the columns. This expression fully describes the `MixColumns` operation.

2.2 The Key Schedule

The `AddRoundKey` transformation takes in four words (16 bytes) of the expanded key `expKey` and adds them to the state. Each word (4 bytes) is added into a column of the state according to the following formula

$$\left(s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}\right)^T = (s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c})^T \oplus \left(\texttt{expKey}_{4\cdot\mathrm{round}+c}\right)^T \tag{2.2.1}$$

for $0 \leq c < 4$. The value of *round* is 0 during the initial addition and 10 during the last addition. Otherwise, the value is defined by the loop in Algorithm 2.1.1.

The values of the expanded key are defined by Algorithm 2.2.1. We can see that the initial key is copied into the first four words of the resulting `expKey`. Each following word is the sum of the previous word and the word four positions earlier. For words that are in positions that are multiples of four, a transformation is applied to the previous word before the sum. This transformation is defined on line 11 of Algorithm 2.2.1 and described in the following paragraph.

The `RotWord` operation takes a four-byte word $(x_0, x_1, x_2, x_3)^T$, where x_i are individual bytes, and produces a cyclically rotated word $(x_1, x_2, x_3, x_0)^T$. The `SubWord` operation takes a four-byte word and applies the S-box to each byte in the word. The round constant array `Rcon[j]` contains ten four-byte words defined by $\left(02_{16}^{j-1}, 00_{16}, 00_{16}, 00_{16}\right)^T$, where the only effective part is the first byte with $02_{16}^{j-1} \in \mathrm{GF}(2^8)$ (or $x^{j-1} \in \mathrm{GF}(2^8)$) being the powers of $02_{16} \in \mathrm{GF}(2^8)$ (or $x \in \mathrm{GF}(2^8)$).

Algorithm 2.2.1 Key Schedule for the AES

Input: ***key*** — array of 16 bytes
Output: key schedule ***expKey*** — array of 44 words (176 bytes)

```
1:  function ExpandKey(key)
2:      word expKey[44]
3:      for i ← 0 to 3 do
4:          expKey[i] ← word(key[4·i], key[4·i+1], key[4·i+2], key[4·i+3])
5:      for i ← 4 to 43 do
6:          tmp ← expKey[i − 1]
7:          if i ≡ 0 (mod 4) then
8:              tmp ← SubWord(RotWord(tmp)) ⊕ Rcon[i/4]
9:          expKey ← expKey[i − 4] ⊕ tmp
10:     return expKey
```

Figure 2.4 depicts four iterations of the loop defined on line 7 of Algorithm 2.2.1. Each vertical line in the figure represents a 4-byte word in the `expKey` array. The box containing the transformation F_i is defined on line 11 of Algorithm 2.2.1, which is described in the previous paragraph.

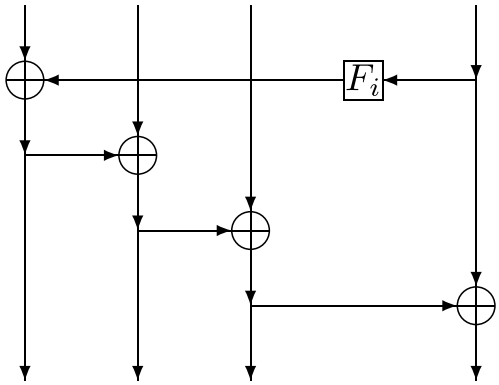

Figure 2.4: A schematic depiction of the key schedule [16].

2.3 Small Scale Variants of the AES

Before we define the scaled-down derivatives of the AES, let us try to estimate how long it would take to attack the full AES-128 by brute force. The actual time complexity of guessing a key with 128 bits can be illustrated by a brief thought experiment.

Suppose we are in possession of a computer cluster with ten billion nodes, each of which runs at 3.3 GHz. Also suppose that one use of the AES-128 takes only one clock cycle on each node. Say that one year has around $3 \cdot 10^7$ seconds. Our cluster will then go through $3 \cdot 10^7 \cdot 3.3 \cdot 10^9 \cdot 10^{10} \approx 10^{27} \approx 2^{90}$ keys in one year. This means that in the worst case, the total time required to guess the correct key will be around 2^{38} years, which is about 250 billion; while the age of the universe is currently estimated to be around 13.8 billion years.

Now suppose that the average consumption of each node is only 1 W and that 1 kWh of energy costs only 0.01 € (the average price of 1 kWh for European household consumers is around 0.2 € in 2020). This means that the energy cost required for our attack is around $10^{10} \cdot 0.001 \cdot 0.01 \cdot 24 \cdot 365 \cdot 2^{38} \approx 10^{20}$ €.

A quick estimate like this immediately leads to the conclusion that the feasibility of the classic brute-force approach is beyond reality. This striking infeasibility of attacking the full AES-128 motivated researchers to come up with scaled down versions of the cipher in order to provide manageable insight into its internals. Carlos Cid et al. introduced such versions in [16] and [17]. The reductions emerge naturally and the new cipher can be described by the following parameters:

(i) the number of rounds n, $1 \leq n \leq 10$;

(ii) the number of rows r of the state, $r = 1, 2, 4$;

(iii) the number of columns c of the state, $c = 1, 2, 4$;

(iv) the number of bits e of the elements of the state, $e = 4, 8$.

We will denote the scaled-down version of the AES by $\mathrm{SR}(n, r, c, e)$. This notation is consistent with [16] and [17]. The standard AES-128 can be then defined by $\mathrm{SR}(10, 4, 4, 8)$ with one subtle difference described in the following paragraph:

As shown in Algorithm 2.1.1, the last round differs from the previous ones inasmuch as the `MixColumns` operation is omitted in it. This omission is due to the design of the inverse of the AES. The new $\mathrm{SR}(n, r, c, e)$ cipher keeps the `MixColumns` operation in the last round. In section 2.1, we saw that this operation is a linear transformation, so the overall complexity of the cryptanalysis of both ciphers remains the same, since a solution of a system of polynomial equations for one cipher would provide a solution for the other cipher. This omission is the only difference between the AES-128 and $\mathrm{SR}(10, 4, 4, 8)$.

Let us now go through the scaled-down versions of the actual encryption operations used in $\mathrm{SR}(n, r, c, e)$. The cipher operates over the field $\mathrm{GF}(2^e)$, defined by the quotient ring $\mathbb{F}_2[x]/\langle f(x)\rangle$ where $f(x) = x^4 + x + 1$ when $e = 4$ and $f(x) = x^8 + x^4 + x^3 + x + 1$ when $e = 8$. Note that the polynomial $f(x)$ is irreducible over $\mathbb{F}_2[x]$ in both cases and when $e = 8$, it is identical to the polynomial used in the original AES-128.

The `SubBytes` operation is also identical to the one used in the AES-128 when $e = 8$. When $e = 4$, the operation is a composition of the following two transformations:

(i) Take the multiplicative inverse in $\mathrm{GF}(2^4)$, the element 0_{16} is mapped to itself.

(ii) Similarly to expression (2.1.1), apply the following affine transformation over $\mathrm{GF}(2^4)$:

$$\begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}. \tag{2.3.1}$$

The `ShiftRows` operation cyclically rotates the row i of the state by i positions, $0 \leq i < r - 1$. Notice that we index the rows from zero so that the first row is always left intact. When $r = 4$, we can use the matrix R_4 from (2.1.2). When $r = 2$, the matrix to rotate a row becomes

$$R_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

When $c = 4$, we can use the matrix from expression (2.1.3) and alternatively use R_2 instead of R_4 when $r = 2$. When $c = 2$, the expression simply becomes

$$\begin{pmatrix} r'_0 \\ r'_1 \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & R \end{pmatrix} \begin{pmatrix} r_0 \\ r_1 \end{pmatrix} \tag{2.3.2}$$

where R is either R_4 or R_2 and I is the identity matrix of corresponding size. When $r = 1$ or $c = 1$, the operation has no effect since either R_2 becomes (1) or the matrix from the expression above becomes I.

The `MixColumns` operation remains the same as in the AES-128 when $r = 4$. When $r = 2$, the operation is defined by the following linear transformation:

$$\begin{pmatrix} s'_{0,j} \\ s'_{1,j} \end{pmatrix} = \begin{pmatrix} x+1 & x \\ x & x+1 \end{pmatrix} \begin{pmatrix} s_{0,j} \\ s_{1,j} \end{pmatrix} \tag{2.3.3}$$

for $0 \leq j < 2$, which indexes the columns, similarly to expression (2.1.5). When $r = 1$, the matrix defining the `MixColumns` operation simply becomes (1), so the operation has no effect.

When $c = 4$, the new cipher uses the same key schedule as in the AES-128. For $c = 2$ and $c = 1$, the structure is naturally reduced and depicted in Figure 2.5 left and right respectively. Similarly to the AES-128, the `AddRoundKey` operation takes in c words of length r. Each word contains the elements of $\mathrm{GF}(2^e)$. These elements are added to the state — each word is added to a column of the state, as shown in formula (2.2.1). The `RotWord` and `SubWord` operations take in r-tuples containing the elements of $\mathrm{GF}(2^e)$. The round constant array also contains r-tuples, in which the only non-zero element is the first one, namely $x^{j-1} \in \mathrm{GF}(2^e)$ being the powers of $x \in \mathrm{GF}(2^e)$ where j is the round number. Notice that the initial key has rce bits. Also recall that this initial key is added to the plaintext before starting the encryption and generating the subsequent sub-keys, just as in the AES-128.

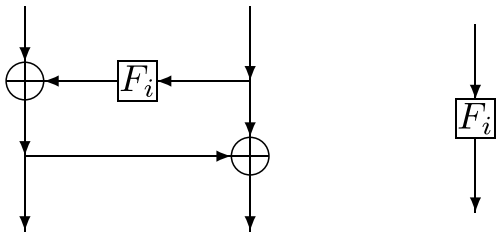

Figure 2.5: A schematic depiction of the scaled-down key schedule [16].

2.4 The AES as a System of Equations

The AES (and its small scale derivatives) is a symmetric block cipher where the block is represented by the state, which is further divided into sub-blocks.

The AES is also an example of an iterated substitution-permutation network where one iteration is split into three stages [18]. The first stage is a local nonlinear transformation (substitution) of the sub-blocks of the state. This transformation is performed by the `SubBytes` operation — the S-box is locally applied to each sub-block in order to substitute its value, while the mutual positions of the sub-blocks are left intact. This stage provides so-called confusion. The next stage is a global linear transformation of the state. This is performed by the `ShiftRows` and `MixColumns` operations, which are linear transformations over GF(2^e), and which also change the mutual positions of the sub-blocks. This stage provides so-called diffusion and it tries to distribute the output bits of the S-boxes in the current iteration to as many S-box inputs as possible in the next iteration. The last stage is the addition of the key.

Let us now model the AES and its scaled-down variants as a system of multivariate polynomial equations over GF(2). We will focus our attention mainly to SR(n, 2, 2, 4) and derive a system of equations for this cipher. A solution to this system will provide us with the encryption key. Other scaled-down derivatives can be modeled in the same way, including the AES itself. Note that we will use one ciphertext with its corresponding plaintext for our model. Our method therefore comes under the known-plaintext type of cryptanalysis.

Non-linear Equations

Let us start by considering the inversion part of the S-box. We know that $bc = 1$, where $b \in \mathrm{GF}(2^e)$ is the input and $c \in \mathrm{GF}(2^e)$ is the output of the S-box. This equation holds unless $b = 0$, in which case we have $b = c = 0$ and we will say that a 0-inversion has taken place. The probability of a 0-inversion occurring is quite low, namely $\frac{1}{16}$ when $e = 4$ and $\frac{1}{256}$ when $e = 8$, so the probability of no 0-inversion occurring is $1 - \frac{1}{16} = \frac{15}{16}$ and $1 - \frac{1}{256} = \frac{255}{256}$. Notice however that these probabilities hold for a single application of the S-box. In SR$(n, 2, 2, 4)$, there are four applications of the S-box during the encryption in one round, so the probability of no 0-inversions occurring during the encryption is $(\frac{15}{16})^{4n}$. There are also two applications of the S-box during the key schedule in one round, so the probability of no 0-inversions occurring during the key schedule is $(\frac{15}{16})^{2n}$. We presume statistical independence of the 0-inversions.

The actual occurrence of a 0-inversion either during the encryption or key schedule is deterministically given by the choice of the plaintext and initial key. If we happen to hit a 0-inversion during the generation of the ciphertext, we can simply disregard the current combination of the plaintext and key, and pick another combination. The issue, as we will see later on, is that one of the equations that model the S-box would have to change, and we as the cryptanalyst, would not know which one it would have to be since we do not know the key anymore. For this reason, we will assume that no

0-inversions have occurred for the given plaintext/key combination when we start generating the equations.

We may regard both $b = \sum_{i=0}^{3} b_i x^i$ and $c = \sum_{i=0}^{3} c_i x^i$ as polynomials in GF(2)[x]. Considering Remark 2.1.3, the product bc modulo the polynomial $m(x) = x^4 + x + 1$ is $r(x) = r_3x^3 + r_2x^2 + r_1x + r_0$ where

$$\begin{aligned} r_0 &= b_0c_0 \oplus b_3c_1 \oplus b_2c_2 \oplus b_1c_3, \\ r_1 &= b_1c_0 \oplus b_0c_1 \oplus b_3c_2 \oplus b_2c_3 \oplus b_3c_1 \oplus b_2c_2 \oplus b_1c_3, \\ r_2 &= b_2c_0 \oplus b_1c_1 \oplus b_0c_2 \oplus b_3c_3 \oplus b_3c_2 \oplus b_2c_3, \\ r_3 &= b_3c_0 \oplus b_2c_1 \oplus b_1c_2 \oplus b_0c_3 \oplus b_3c_3. \end{aligned} \tag{2.4.1}$$

It is important to note that in contradistinction to Remark 2.1.3, the coefficients b_i and c_i are the elements of GF(2). We have $bc = r = 1$. This gives us four multivariate quadratic equations over GF(2): $r_0 = 1$ and $r_i = 0$ where $i = 1, 2, 3$. These equations are bilinear in the b_i and c_i variables. For $e = 8$, we would have got eight multivariate quadratic equations in the variables b_i and c_i instead of four.

If there was a 0-inversion, either during the encryption or key schedule, the first equation would change to $r_0 = 0$. However as already mentioned, we do not consider this case, since we can detect 0-inversions before we start generating the equations and disregard the plaintext/key combinations that produce them.

Along with these equations, it is possible to obtain further quadratic equations from the relation $bc = 1$. Notice that we also have $bc^2 = c$ and $b^2c = b$. Let us focus on the first relation and compute the resulting equations. The equations for $b^2c = b$ can be produced in the same fashion. Since we work over GF(2), we can write $bc^2 + c = 0$. We have already computed the product bc, so we could just multiply it by c and get the result. This computation would require unnecessary steps as it would lead to many intermediate cubic terms which we would have to cross out before obtaining the final coefficients. We can instead compute the square of c and pre-multiply it by b. We are working over a commutative structure, so the order in which we perform the multiplication is of no relevance. In order to work out the square of c, we can use the expression (2.4.1) and substitute c for b. We get the polynomial $d = c^2$ where $d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$ with

$$\begin{aligned} d_0 &= c_0 \oplus c_2 \\ d_1 &= c_2 \\ d_2 &= c_1 \oplus c_3 \\ d_3 &= c_3. \end{aligned}$$

We can now obtain the final result $t = bd + c$ where $t(x) = t_3x^3 + t_2x^2 + t_1x + t_0$

with

$$\begin{aligned}
t_0 &= b_0c_0 \oplus b_0c_2 \oplus b_3c_2 \oplus b_2c_1 \oplus b_2c_3 \oplus b_1c_3 \oplus c_1,\\
t_1 &= b_1c_0 \oplus b_1c_2 \oplus b_0c_2 \oplus b_3c_1 \oplus b_3c_3 \oplus b_3c_2 \oplus b_2c_1 \oplus b_1c_3 \oplus c_1,\\
t_2 &= b_2c_0 \oplus b_2c_2 \oplus b_1c_2 \oplus b_0c_1 \oplus b_0c_3 \oplus b_3c_1 \oplus b_2c_3 \oplus c_2,\\
t_3 &= b_3c_0 \oplus b_3c_2 \oplus b_2c_2 \oplus b_1c_1 \oplus b_1c_3 \oplus b_0c_3 \oplus b_3c_3 \oplus c_3.
\end{aligned}$$

We know that $t = 0$, so we have four equations $t_i = 0$ for $0 \leq i < 4$. Notice that these equations are quadratic as well. We can obtain reciprocal equations from $b^2c = b$. All of these eight equations are biaffine in the b_i and c_i variables.

It is possible to obtain even more quadratic equations by considering the relations $bc^4 = c^3$ and $b^4c = b^3$. As in the previous case, let us focus our attention to the first one and rewrite it to $bc^4 + c^3 = 0$. We can square d to obtain c^4, so let $f = d^2$ where $f(x) = f_3x^3 + f_2x^2 + f_1x + f_0$ with

$$\begin{aligned}
f_0 &= c_0 \oplus c_1 \oplus c_2 \oplus c_3\\
f_1 &= c_1 \oplus c_3\\
f_2 &= c_2 \oplus c_3\\
f_3 &= c_3.
\end{aligned}$$

The polynomial c^3 can be obtained by multiplying d by c. We then get $g = dc$ where $g(x) = g_3x^3 + g_2x^2 + g_1x + g_0$ with

$$\begin{aligned}
g_0 &= c_0 \oplus c_0c_2 \oplus c_1c_2 \oplus c_2c_3\\
g_1 &= c_3 \oplus c_0c_1 \oplus c_0c_2 \oplus c_2c_3\\
g_2 &= c_2 \oplus c_0c_1 \oplus c_0c_2 \oplus c_0c_3 \oplus c_1c_2 \oplus c_1c_3 \oplus c_2c_3\\
g_3 &= c_1 \oplus c_2 \oplus c_3 \oplus c_1c_3 \oplus c_2c_3.
\end{aligned}$$

We are now in a position to obtain the result $u = bf + g$ where $u(x) = u_3x^3 + u_2x^2 + u_1x + u_0$ with

$$\begin{aligned}
u_0 &= b_3c_3 \oplus b_3c_1 \oplus b_2c_3 \oplus b_2c_2 \oplus b_1c_3 \oplus b_0c_3 \oplus b_0c_2 \oplus b_0c_1\\
&\quad \oplus b_0c_0 \oplus c_3c_1 \oplus c_2c_1 \oplus c_2c_0 \oplus c_0,\\
u_1 &= b_3c_2 \oplus b_3c_1 \oplus b_2c_2 \oplus b_1c_2 \oplus b_1c_1 \oplus b_1c_0 \oplus b_0c_3 \oplus b_0c_1\\
&\quad \oplus c_3c_2 \oplus c_2c_0 \oplus c_1c_0 \oplus c_3,\\
u_2 &= b_3c_2 \oplus b_2c_2 \oplus b_2c_1 \oplus b_2c_0 \oplus b_1c_3 \oplus b_1c_1 \oplus b_0c_3 \oplus b_0c_2\\
&\quad \oplus c_3c_2 \oplus c_3c_1 \oplus c_3c_0 \oplus c_2c_1 \oplus c_2c_0 \oplus c_1c_0 \oplus c_2,\\
u_3 &= b_3c_2 \oplus b_3c_1 \oplus b_3c_0 \oplus b_2c_3 \oplus b_2c_1 \oplus b_1c_3 \oplus b_1c_2 \oplus b_0c_3\\
&\quad \oplus c_3c_2 \oplus c_3c_2 \oplus c_3c_1 \oplus c_3 \oplus c_2 \oplus c_1.
\end{aligned}$$

We know that $u = 0$, so we have another four equations $u_i = 0$ for $0 \leq i < 4$. Observe that these equations are still quadratic. We can obtain reciprocal equations from $b^4c = b^3$.

So far, we have derived 20 multivariate quadratic equations from the relation $bc = 1$. A natural question arises whether we have identified all quadratic equations in the b_i and c_i variables. Notice, for example, that we have skipped the relation $bc^3 = c^2$. The reason is that it would produce equations with cubic terms. Relations involving higher powers than c^4 would also lead to equations with higher than quadratic terms. In fact, the 20 equations we have derived are all the quadratic equations over GF(2). A further discussion can be found in [17, p. 77]. As also advised in [17, p. 77], we will focus on the first 12 bilinear and biaffine quadratic equations we have obtained and we will omit the remaining eight ones. For $e = 8$, we would have got 40 multivariate quadratic equations in the variables b_i and c_i instead of 20.

Linear Equations

The equations we have derived for the inversion part of the S-box account for the only non-linear equations in the whole system that models the SR(n, 2, 2, 4) cipher. Let us now derive the remaining linear equations.

The affine transformation of the S-box can be expressed directly by expression (2.3.1) where the input is the polynomial $c(x)$ from the previous subsection. This gives us four linear equations in the c_i variables. These equations together with the non-linear equations from the previous subsection fully describe a single S-box. Let L_s denote the matrix from expression (2.3.1). In order to describe the whole `SubBytes` operation, we can extend the matrix L_s to the whole state array of SR(n, 2, 2, 4), so we have the matrix

$$L = \begin{pmatrix} L_s & 0 & 0 & 0 \\ 0 & L_s & 0 & 0 \\ 0 & 0 & L_s & 0 \\ 0 & 0 & 0 & L_s \end{pmatrix}.$$

We can also extend the S-box constant vector $(0, 1, 1, 0)^T = 6_{16}$ to the vector $\mathbf{6} = (6_{16}, 6_{16}, 6_{16}, 6_{16})$, so that we cover the whole state array. We will use $\mathbf{b}$ to denote the input vector of the `SubBytes` operation, and $\mathbf{b^{-1}}$ to denote its output — the vector of the inverted elements in GF(2^4). Note that each component in these vectors is made of the four coefficients of the polynomials $b(x)$ and $c(x)$, respectively; so we have 12 non-linear equations for each component.

The actual state array is depicted in Figure 2.6. We will represent it as the vector $(s_0, s_1, s_2, s_3)^T$. The `ShiftRows` operation can be then described by the matrix

$$R = \begin{pmatrix} I_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & I_4 \\ 0 & 0 & I_4 & 0 \\ 0 & I_4 & 0 & 0 \end{pmatrix}$$

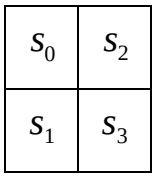

Figure 2.6: The state array of the SR(n, 2, 2, e) cipher.

where I_4 is the identity matrix of size four. Before we describe the `MixColumns` operation, let us rewrite the expression (2.4.1) into matrix form:

$$\begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{pmatrix} = \begin{pmatrix} b_0 & b_3 & b_2 & b_1 \\ b_1 & b_0 \oplus b_3 & b_3 \oplus b_2 & b_2 \oplus b_1 \\ b_2 & b_1 & b_0 \oplus b_3 & b_3 \oplus b_2 \\ b_3 & b_2 & b_1 & b_0 \oplus b_3 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}.$$

If we substitute the binary values of the coefficients of the polynomials $x+1$ and x into the matrix in the expression above, we get the matrices

$$M_{x+1} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \quad \text{and} \quad M_x = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

These matrices represent the multiplication by the polynomials $x+1$ and x modulo the polynomial x^4+x+1. The `MixColumns` operation, defined by the expression (2.3.3), can then be expressed by the matrix

$$M = \begin{pmatrix} M_{x+1} & M_x & 0 & 0 \\ M_x & M_{x+1} & 0 & 0 \\ 0 & 0 & M_{x+1} & M_x \\ 0 & 0 & M_x & M_{x+1} \end{pmatrix}.$$

We can now describe one round of SR(n, 2, 2, 4) by the expression

$$\mathbf{b_i} = MR(L\mathbf{b_{i-1}^{-1}} + \mathbf{6}) + \mathbf{k_i} \quad \text{for} \quad i < 0 \leq n$$

where $\mathbf{k_i}$ is a vector containing 16 binary variables of the round key described in the following subsection and i is the round number. The vector $\mathbf{b_{i-1}^{-1}}$ contains four components — the outputs from the S-boxes — each of which has four binary variables. It is straightforward to check that $R\mathbf{6} = M\mathbf{6} = \mathbf{6}$. We can then write

$$\mathbf{b_i} = MRL\mathbf{b_{i-1}^{-1}} + \mathbf{k_i} + \mathbf{6} \quad \text{for} \quad i < 0 \leq n.$$

The relation above gives 16 linear equations, which represent one round of SR(n, 2, 2, 4). In addition, we have 12 non-linear equations for each component in $\mathbf{b_{i-1}^{-1}}$, so in total, we have $16 + 4 \cdot 12 = 64$ equations describing one

round of encryption in the SR(n, 2, 2, 4) cipher. When $i = n$, we have

$$\mathbf{c_t} = MRL\mathbf{b_{n-1}^{-1}} + \mathbf{k_n} + \mathbf{6}$$

where $\mathbf{c_t}$ is the known ciphertext, which is a vector of 16 binary values. We obtain $\mathbf{b_0}$ by adding the initial unknown key $\mathbf{k_0}$ to the known plaintext $\mathbf{p_t}$, so we have

$$\mathbf{b_0} = \mathbf{p_t} + \mathbf{k_0}.$$

This addition gives us further 16 initial equations where $\mathbf{p_t}$ is a vector of 16 binary values and $\mathbf{k_0}$ is a vector of 16 binary variables. Our goal is to actually compute the values of $\mathbf{k_0}$ since this is the user's key. All other variables are auxiliary.

Key Schedule

The generation of round keys for SR(n, r, c, e) is thoroughly described in Appendix A of [16]. Let us now describe the equations for SR(n, 2, 2, 4). Let $\mathbf{k_i} = (k_{i,0}, k_{i,1}, k_{i,2}, k_{i,3},)^T \in GF(2^4)^4$ be the round key of round i. The round key can be then defined by

$$\begin{pmatrix} k_{i,2q} \\ k_{i,2q+1} \end{pmatrix} = \begin{pmatrix} Lk_{i-1,3}^{-1} \\ Lk_{i-1,2}^{-1} \end{pmatrix} + \begin{pmatrix} 6_{16} \\ 6_{16} \end{pmatrix} + \begin{pmatrix} x^{i-1} \\ 0 \end{pmatrix} + \sum_{t=0}^{q} \begin{pmatrix} k_{i-1,2t} \\ k_{i-1,2t+1} \end{pmatrix}$$

for $0 \leq q < 2$ where x^{i-1} is an element of GF(2^4). This expression gives 16 linear equations for each $\mathbf{k_i}$. Note that $\mathbf{k_0}$ is not provided by the user — it is a vector of 16 binary variables that we, as the cryptanalyst, are trying to compute. We also get $2 \cdot 12 = 24$ non-linear equations since the computation of each $\mathbf{k_i}$ requires two applications of the S-box. One round of the key schedule in SR(n, 2, 2, 4) is then described by 40 equations.

Equations without auxiliary variables

We can also derive equations that contain only the variables of the initial key. In order to obtain such a system, we can eliminate the auxiliary variables by a gradual substitution of the initial key variables since we know that the cipher starts by adding the initial key to the known plaintext. It is straightforward to perform this substitution for the linear equations. For the non-linear equations, which model the S-box, we can leverage Gröbner bases. Consider the four polynomials $r_0, \ldots, r_3$ from (2.4.1) as polynomials in $\mathbb{F}[c_0, \ldots, c_3, b_0, \ldots, b_3]$. We see that it is not straightforward to express the output bits c_i in terms of the input bits b_i by ordinary manipulation techniques. If we impose, for example, a block order $\succeq_{grlex,grlex}$ on $\mathbb{F}[c_0, \ldots, c_3, b_0, \ldots, b_3]$ with $\succeq_{grlex}$ on both $\mathbb{F}[c_0, \ldots, c_3]$ and $\mathbb{F}[b_0, \ldots, b_3]$, and compute the reduced Gröbner basis, we get the following polynomial system:

$$
\begin{aligned}
f_1 &= c0 \oplus b_2b_1b_0 \oplus b_3b_2b_1 \oplus b_2b_0 \oplus b_2b_1 \oplus b_0 \oplus b_1 \oplus b_2 \oplus b_3,\\
f_2 &= c1 \oplus b_3b_1b_0 \oplus b_1b_0 \oplus b_2b_0 \oplus b_2b_1 \oplus b_3b_1 \oplus b_3,\\
f_3 &= c2 \oplus b_3b_2b_0 \oplus b_1b_0 \oplus b_2b_0 \oplus b_3b_0 \oplus b_2 \oplus b_3,\\
f_4 &= c3 \oplus b_3b_2b_1 \oplus b_3b_0 \oplus b_3b_1 \oplus b_3b_2 \oplus b_1 \oplus b_2 \oplus b_3,\\
f_5 &= b_3b_2b_1b_0 \oplus b_2b_1b_0 \oplus b_3b_1b_0 \oplus b_3b_2b_0 \oplus b_3b_2b_1 \oplus b_1b_0 \oplus b_2b_0 \oplus b_2b_1 \oplus b_3b_0\\
&\quad \oplus b_3b_1 \oplus b_3b_2 \oplus b_0 \oplus b_1 \oplus b_2 \oplus b_3 \oplus 1
\end{aligned}
$$

We see that the last polynomial f_5 involves only the variables b_i. Notice that this polynomial is not satisfied only if all $b_i = 0$ and it holds whenever we have at least one $b_i = 1$. Recall that we do not consider 0-inversions. This polynomial is therefore always satisfied and we can omit it from the system. We also see that in the remaining polynomials, the output variables $c_0, \ldots, c_3$ are expressed solely by the input variables b_i. This allows us to perform the gradual substitution of the unknown variables of the initial key $\mathbf{k_0}$ throughout the whole polynomial system. Notice that we obtain $|\mathbf{k_0}| = 16$ polynomials after we finish the substitution. We note that the size of the polynomials is close to $2^{|\mathbf{k_0}|-1}$ at full diffusion of the cipher. The diffusion grows rapidly with each round. For example, as our experiments will reveal, the cipher SR(n, 2, 2, 4) reaches its full diffusion at round $n = 3$. This way of generating the polynomials is therefore suitable only for low values of n. A different method for obtaining polynomials without auxiliary variables is described in [19].

Yet another way of obtaining polynomials involving only the variables of the initial key $\mathbf{k_0}$ is to regard the cipher as a set of boolean functions of $\mathbf{k_0}$, one function per one bit of $\mathbf{k_0}$. We can then convert such functions into an Algebraic Normal Form (ANF) in order to obtain the polynomials. Such a conversion can be found in [20]. However, the complexity of this approach requires at least $|\mathbf{k_0}|2^{|\mathbf{k_0}|}$ encryptions even if we consider only one round of the cipher. For this reason, we will not examine this method any further.

Chapter 3

Experiments

> I feel I am nibbling on the edges of this world when I am capable of getting what Picasso means when he says to me — perfectly straight-facedly — later of the enormous new mechanical brains or calculating machines: "But they are useless. They can only give you answers."
>
> William Fifield, Pablo Picasso: A Composite Interview, The Paris Review 32

The experiments were carried out on GNU/Linux 5.4 running on on two Intel® Xeon® Gold 6136 processors with 768 GB DDR4 memory evenly split up into 12 modules. The baseboard was Supermicro X11DPi-NT. The initial polynomial systems containing auxiliary variables were generated by utilizing Martin Albrecht's implementation of the small scale variants of the AES in SageMath 9.1 [21] which also uses Python 3.7.3 and PolyBoRi [22]. The systems were solved in Magma V2.25-5 [23] and CryptoMiniSat [24]. The source code for the experiments can be found at `https://gitlab.com/marek.onl/masters-thesis`. The generation and preprocessing of the polynomial systems was implemented in parallel utilizing all 24 available cores. Magma, however, was able to solve one system on one core only, so in order to keep the comparison even, we explicitly restricted CryptoMiniSat to one core only as well.

As stated in Definition 1.1.24, we may regard a system of polynomials as a basis of an ideal I. We can then compute the reduced Gröbner basis of I under the lexicographic order, and by applying the Elimination Theorem, we can easily obtain the solution. We have demonstrated the use of this theorem in examples 1.4.7 and 1.4.11, and as we have discussed in Section 2.4, the solution represents the secret key.

Table 3.1 shows the results of initial experiments with systems of equations containing auxiliary variables. We generated the systems in SageMath for various versions of $\mathrm{SR}(n, r, c, e)$, and we subsequently attempted to solve these systems by the F4 algorithm implemented in Magma and by CryptoMiniSat.

Since we work over GF(2), the polynomials can be seen as logical formulas in algebraic normal form (ANF). SageMath supports a conversion from ANF to CNF (conjunctive normal form). Formulas in CNF can be passed to CryptoMiniSat and the initial key can be then easily recovered from the solution. We have included the SAT solver so that we can compare it to the performance of the F4 algorithm and we can see in the table that the solver performs much better. The SAT solver also takes a negligible amount of memory, so this value is not stated in the table.

The average number of monomials per polynomial is between 6 and 8 when $e = 4$ and between 18 and 20 when $e = 8$. Both the average and highest degrees of monomials are equal to two, so all polynomials are quadratic or linear, as the case may be. In our experiments, we do not consider the ciphers with $r < 2$ or $c < 2$ as these have the matrices for the operations `MixColumns` and `ShiftRows` reduced to (1). Recall that the dimensions of the state array r and c are restricted to the values 1, 2 and 4; the exponent e can be either 4 or 8; and for the number of rounds n, we have $1 \leq n \leq 10$.

The column named Vars contains the number of variables in the whole polynomial system and the column named Polys contains the number of polynomials in the system. We measured the runtime and memory consumption only during the solving of the polynomials since the preparation of the system takes only a fraction of the resources relative to solving it.

Recall that the key size for $\mathrm{SR}(n, r, c, e)$ is given by the product rce. Notice that we were not able to compute the solution for even one round of $\mathrm{SR}(n, 4, 4, 8)$, the key size of which is 128 bits. On the other hand, the SAT solver could quickly compute the solution for all ten round of $\mathrm{SR}(n, 2, 2, 4)$.

Table 3.1: Initial experiments with systems containing auxiliary variables.

Cipher	Key bits	Vars	Polys	F4		SAT
				Time	Mem.	Time
SR(1, 2, 2, 4)	16	72	120	1 s	33 MB	2 s
SR(2, 2, 2, 4)	16	128	224	19 s	848 MB	12 s
SR(3, 2, 2, 4)	16	184	328	4 h	76 GB	17 s
SR(4, 2, 2, 4)	16	240	432	—	—	27 s
SR(10, 2, 2, 4)	16	576	1056	—	—	50 s
SR(1, 4, 2, 4)	32	144	240	48 s	981 MB	9 s
SR(2, 4, 2, 4)	32	256	448	—	—	1.5 m
SR(3, 4, 2, 4)	32	368	656	—	—	63 h
SR(1, 2, 4, 4)	32	136	216	3 s	67 MB	11 s
SR(2, 2, 4, 4)	32	240	400	—	—	33 s
SR(3, 2, 4, 4)	32	344	584	—	—	15.5 m
SR(4, 2, 4, 4)	32	448	768	—	—	34 h
SR(1, 4, 4, 4)	64	272	432	—	—	2.5 m
SR(1, 2, 2, 8)	32	144	240	1 m	2.2 GB	22 s
SR(2, 2, 2, 8)	32	256	448	—	—	11.5 m
SR(1, 4, 2, 8)	64	288	480	—	—	41.5 m
SR(1, 2, 4, 8)	64	272	432	—	—	4 m
SR(1, 4, 4, 8)	128	544	864	—	—	—

Table 3.2 contains the results of experiments with systems that contain only the variables of the initial secret key. We eliminated the auxiliary variables by a gradual substitution of the variables of the initial key through the system, starting by adding the known plaintext bits and ending by adding the known ciphertext bits. The time required for this substitution is stated in the column named PT. This system always contains k polynomials in k variables where k is the number of the key bits. Since k is the number of variables and we work over GF(2), k is also the maximal limit of the total degree of the polynomials.

All further experiments will be carried out with systems of polynomials involving only the variables of the initial key. In systems with auxiliary variables, the structure of the polynomial systems derived from different plaintexts remains unchanged. Only the initial and final polynomials that add the bits of the plaintext and ciphertext differ by this bitwise addition. Since we have eliminated the auxiliary variables by a gradual substitution of the initial key bits starting from the initial plaintext addition, each of the k polynomials now depends on the choice of plaintext and its corresponding ciphertext. Since the structure of each polynomial system is now different, the time and memory

required for obtaining the solution started to differ as well, especially the time required by the SAT solver. For this reason, all the following tables contain average results of five different runs for each experiment. We can still see that the results for the SAT solver differ across tables for the same experiment, so even more than five runs would be required for further investigation. Nevertheless, we restricted ourselves to such number due to limited time resources.

The column named AMP contains the average number of monomials per polynomial in the whole system. We can see that this number grows fast as n increases. The maximal limit of the number of monomials in one polynomial is $2^k - 1$. When $n = 1$ and $e = 4$, the average degree of monomials is 2 and the highest degree is 3. When $n = 2$, the average and highest degrees are 5 and 9, respectively. Note that the average degree has its maximum at $\frac{k}{2}$. We were not able to generate systems with $n > 2$ and $r, c > 2$ for $e = 4$. For $n = 1$ and $e = 8$, the average degree is 4 and the maximal degree is 7. We were not able to generate systems with $e = 8$ and $n > 1$ (recall that we do not consider the cases when $r < 2$ or $c < 2$). We can see in the table that the overall performance is worse compared to the previous table and that the SAT solver still outperforms the F4 algorithm. Moreover, we were able to solve less systems than in the previous experiments.

Table 3.2: Experiments with systems with no auxiliary variables.

Cipher	Key bits	PT [a]	AMP [b]	F4		SAT
				Time	Mem.	Time
SR(1, 2, 2, 4)	16	1 s	20	1 s	33 MB	1 s
SR(2, 2, 2, 4)	16	1 s	2475	2.5 m	4.8 GB	1 m
SR(3, 2, 2, 4)	16	8 s	32784	8.5 m	18.5 GB	13 m
SR(10, 2, 2, 4)	16	2.5 m	32814	9 m	19.5 GB	14 m
SR(1, 4, 2, 4)	32	1 s	37	55 s	1.2 GB	1 s
SR(1, 2, 4, 4)	32	1 s	23	13 s	671 MB	1 s
SR(1, 4, 4, 4)	64	4 s	40	—	—	2 m
SR(1, 2, 2, 8)	32	8 s	314	—	—	1.5 m
SR(1, 4, 2, 8)	64	18 s	567	—	—	33 m
SR(1, 2, 4, 8)	64	14 s	348	—	—	1.5 h

[a] Preprocessing Time — the time required to obtain the system
[b] Average number of Monomials per Polynomial

In the table above, we can see that the the AMP value and the solving time and memory are almost the same for SR(3, 2, 2, 4) and SR(10, 2, 2, 4). This means that the maximal diffusion for SR$(n, 2, 2, 4)$ is reached in the third round of the cipher and the subsequent rounds do not provide any further security as regards the algebraic cryptanalysis, except for a longer time required for

the generation of the polynomial system. This observation seems to be in line with the statements made in [25]. The following table provides a deeper insight into the distribution of monomials in SR(3, 2, 2, 4). At full diffusion, the expected degree of monomials should be equal to $\frac{1}{2}\binom{k}{d}$ where k is the number of variables and d is the degree. Since we have SR(3, 2, 2, 4), we get $k = 2 \cdot 2 \cdot 4 = 16$, recall that we also have k polynomials in the whole system. In Table 3.3, the expected value is stated in the last row. We see that all the polynomials follow this value very closely, meaning that it is not possible to get much closer to the expected value in the subsequent rounds. For this reason, we do not consider the rounds between the third and tenth one. The table also shows that the average monomial degree is 8 for each polynomial, which is half of the maximal degree, and that no polynomial significantly differs from the expected values for monomial degrees. The second last row shows the average value for all of the polynomials — the average of the whole column above.

The last column contains the number of all monomials in the polynomial. At full diffusion, this number should be equal to $\sum_{d=0}^{16} \frac{1}{2}\binom{k}{d} = \frac{2^{16}}{2} = 32768$, so that every polynomial contains half of all of the possible monomials. We see that the number of monomials is close to the expected value for each of the polynomials as well. We may also be interested in the frequency of the variables in the polynomial system. Considering the full diffusion again, each variable should be contained in half of the monomials in every polynomial, so the expected value is $\frac{2^{16}}{4} = 16384$. In the actual system described in Table 3.3, the most frequent variable had 16446 occurrences and the least frequent variable had 16393 occurrences, these are aggregated values.

Table 3.3: Distribution of monomials of a given degree in SR(3, 2, 2, 4).

Poly	Number of monomials of the given degree																	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	all
1	0	6	62	277	939	2177	3965	5820	6500	5769	4001	2201	893	262	55	9	1	32937
2	1	9	55	295	906	2154	3931	5780	6519	5790	3990	2212	902	279	51	4	0	32878
3	1	7	57	277	908	2199	4010	5653	6510	5685	3978	2232	912	277	60	7	0	32773
4	0	7	66	276	940	2159	3997	5759	6514	5737	3979	2260	939	268	59	13	0	32973
5	1	11	69	268	940	2244	4023	5701	6440	5738	4009	2142	892	262	69	6	1	32816
6	1	5	59	286	940	2169	4067	5692	6399	5830	4074	2152	917	269	59	8	1	32928
7	0	8	67	276	904	2236	3990	5634	6407	5764	4034	2164	914	259	58	2	1	32718
8	1	11	61	281	908	2201	4045	5637	6305	5775	3974	2208	919	285	53	7	1	32672
9	1	6	57	277	869	2202	4064	5775	6359	5676	4053	2182	925	302	58	8	1	32815
10	1	9	47	277	937	2185	4012	5718	6359	5713	4027	2191	907	260	56	10	1	32710
11	1	8	68	293	907	2226	3985	5698	6490	5747	4023	2139	903	287	57	6	0	32838
12	0	8	54	293	926	2167	3948	5693	6330	5665	4038	2172	935	294	62	10	0	32595
13	1	7	59	287	918	2230	4067	5804	6505	5700	4035	2208	879	267	58	11	1	33037
14	1	10	46	260	902	2173	3957	5789	6446	5739	4080	2237	885	270	65	10	1	32871
15	1	7	57	275	922	2189	4037	5793	6358	5721	3989	2224	880	294	61	3	0	32811
16	0	10	57	260	905	2174	4057	5741	6533	5824	3942	2180	939	264	76	7	1	32970
Avg.	0.7	8	59	279	917	2193	4010	5730	6436	5742	4014	2194	909	275	60	8	0.6	32834
Exp.	0.5	8	60	280	910	2184	4004	5720	6435	5720	4004	2184	910	280	60	8	0.5	32768

Let us see if we can obtain any better results than those in Table 3.2. Let $\mathbf{k}$ be the initial key. By Proposition 1.1.37, we know that $I(\mathbf{k})$ is an ideal. Now

let $\{f_1, \ldots, f_k\}$ and $\{g_1, \ldots, g_k\}$ be two polynomial systems generated from two different pairs of plaintext and ciphertext under the same key $\mathbf{k}$. Since each $f_i(\mathbf{k}) = 0$ and $g_j(\mathbf{k}) = 0$, we have $I = \langle f_1, \ldots, f_k, g_1, \ldots, g_k \rangle \subseteq I(\mathbf{k})$. In general, in order to obtain the ideal I, we may combine any number of polynomial systems instead of two. We can now compute the Gröbner basis for I and we still get the initial key $\mathbf{k}$. The ideal I represents an overdefined system for which it could be easier to obtain the solution. We will call one pair of plaintext and its corresponding ciphertext a **PC pair**. In our further experiments, we assume that all PC pairs use the same key.

Table 3.4 summarizes the experimental results for two combined systems, as described in the previous paragraph. We can see that the results are much better compared to Table 3.2 and that the F4 algorithm often performs better than the SAT solver. We can also see that we are able to solve more polynomial systems and even the system for SR(1, 4, 4, 8) is solved in a few seconds. Recall that we were unable to obtain this solution for systems with auxiliary variables. This practically means that one round of the AES-128 provides no security against this attack. We were not able to obtain any solution for SR$(n, 4, 4, e)$ with $n > 1$ though. Observe that we used two PC pairs in this scenario. Further experiments with more than two pairs were carried out as well, but did not provide any better results. After adding more than five systems, the time required to obtain to solution started increasing.

We note that it would be not possible to combine the systems if we did not eliminate the auxiliary variables. The reason is that the auxiliary variables do not depend on the PC pair — when we use two different PC pairs, we get the same equations, up to the initial additions of the plaintext and ciphertext. On the other hand, when we express the equations only in the variables of the initial key, we get a different system for each PC pair.

Table 3.4: Experiments with two combined systems.

Cipher	Key bits	PT [a]	AMP [b]	F4		SAT
				Time	Mem.	Time
SR(1, 2, 2, 4)	16	1 s	21	1 s	33 MB	1 s
SR(2, 2, 2, 4)	16	2 s	2469	5 s	100 MB	1 m
SR(3, 2, 2, 4)	16	9 s	32798	13 m	19.8 GB	45.5 m
SR(10, 2, 2, 4)	16	3 m	32774	11 m	25.5 GB	31.5 m
SR(1, 4, 2, 4)	32	2 s	37	1 s	33 MB	1 s
SR(2, 4, 2, 4)	32	6 s	33360	—	—	—
SR(1, 2, 4, 4)	32	2 s	23	1 s	33 MB	1 s
SR(2, 2, 4, 4)	32	3 s	6701	—	—	—
SR(1, 4, 4, 4)	64	4 s	39	1 s	33 MB	2 s
SR(1, 2, 2, 8)	32	10 s	316	1 s	33 MB	8 s
SR(1, 4, 2, 8)	64	18 s	568	2 s	33 MB	17 s
SR(1, 2, 4, 8)	64	15 s	348	1 s	33 MB	17 s
SR(1, 4, 4, 8)	128	34 s	599	4 s	33 MB	35 s

[a] Preprocessing Time — the time required to obtain the system
[b] Average number of Monomials per Polynomial

The table above shows that the hardest systems to solve were the ones with high AMP. Let us see if we can reduce this value.

Definition 3.0.1. Let $f, g \in \mathbb{F}[x_1, \ldots, x_n]$ be two polynomials. We define their similarity $\sigma(f, g)$ as $\sigma(f, g) = |M(f) \cap M(g)|$, where $M(h)$ is the set of monomials in h.

Consider again a polynomial system $F = \{f_1, \ldots, f_k\}$ and a set of l polynomial systems $G = \{g_1, \ldots, g_m\}$ where $m = kl$. We will refer to F as the primal system and to G as the reduction set. Each polynomial system is generated from a different PC pair under the same key $\mathbf{k}$. For each f_i we find a g_j so that $\sigma(f_i, g_j)$ is maximal and compute $h_i = f_i + g_j \in I(\mathbf{k})$. We get an ideal $I = \langle h_1, \ldots, h_k \rangle \subseteq I(\mathbf{k})$. Similarly to the previous experiments, we can now compute the Gröbner basis and obtain the solution $\mathbf{k}$. Since we work over GF(2), if the polynomials f_i and g_j are similar enough, the alike monomials cancel each other out and the resulting polynomials h_i might be smaller than f_i. This might allow faster computation.

As already mentioned, we get a different system for each PC pair. How much different depends on the degree of diffusion in the cipher. In Table 3.3, we have shown that the polynomials for SR$(n, 2, 2, 4)$ with $n \geq 3$ are essentially random. This reflects in Table 3.5, which contains the results of experiments with the reduced polynomials h_i. The value l in the table is the

size of the reduction set, as described in the paragraph above. We see that for SR(3, 2, 2, 4), the Average number of Monomials per Polynomial after the Reduction (AMPR) does not differ from the AMP value in the previous table. On the other hand, for example, for SR(2, 4, 2, 4) and $l = 5$, AMPR is reduced by 86 %. Unfortunately, we could still not compute the solution. For SR(2, 2, 4, 4) and $l = 5$, the reduction allowed us to solve the system, but for $l = 2$, it did so only for the SAT solver. For SR(2, 2, 2, 4), the reduction shortened the computation time. We note that we considered only the ciphers that required more than five seconds to solve in the previous table. We can also see that the number of polynomial systems for reduction l considerably lowered the AMPR value only for SR(2, 2, 4, 4) and for other ciphers it had no, or very subtle effect. We have also tried other values of l, all of which were ≤ 50 due to limited time, with no significant effect either, even for SR(2, 2, 4, 4). The column labeled PT now includes the time required for the reduction.

In order to increase the reduction, we have tried generating the plaintexts in the PC pairs for the polynomial systems in G so that each of them would differ only by one bit from the plaintext for F. It emerged that this approach did not bring any significant improvement.

We suspect that the reduction technique proposed above might not be the only one and that other techniques might provide better results.

Table 3.5: Experiments with reduced polynomial systems.

Cipher	Key bits	PT [a]	AMPR [b]	l [c]	F4		SAT
					Time	Mem.	Time
SR(2, 2, 2, 4)	16	5 s	601	1	1 s	33 MB	29 s
SR(2, 2, 2, 4)	16	5 s	519	5	1 s	33 MB	24 s
SR(3, 2, 2, 4)	16	25 s	32592	1	16 m	17.9 GB	37.5 m
SR(3, 2, 2, 4)	16	40 s	32555	5	18 m	23.1 GB	41 m
SR(2, 4, 2, 4)	32	26 s	4938	1	—	—	—
SR(2, 4, 2, 4)	32	1 m	4563	5	—	—	—
SR(2, 2, 4, 4)	32	14 s	3410	1	—	—	83 m
SR(2, 2, 4, 4)	32	18 s	1192	5	60 m	34.5 GB	50 m

[a] Preprocessing Time — the time required to obtain the system
[b] Average number of Monomials per Polynomial after Reduction
[c] Size of the reduction set

Since the F4 algorithm and the SAT solver run in a single thread, and we had a parallel architecture at our disposal, we tried guessing some variables in the reduced polynomial systems with $l = 5$. This means that we determined the values of the guessed variables, we substituted these values into the system, and then we attempted to solve the system. Observe that substituting concrete values of some variables not only eliminates the variables, but also shortens the

polynomials — for example, a 0 occurring in a monomial makes it vanish. On the other hand, substituting a 1 can lead to two equal monomials which cancel each other out. We used a brute-force approach for guessing the variables so we got 2^v different systems to solve where v is the number of guessed variables. Instead of guessing random variables, we tried to guess the most frequent ones in order to shorten the polynomials even further. The reason can be seen in figures 3.1 and 3.2. These figures contain the frequencies of the variables for five instances of SR(2, 2, 4, 4) and SR(2, 4, 2, 4). The variables are ordered in a descending order, so their labels correspond to their relative positions in the plot according to their frequency — the first variable is the most frequent one. It can be seen that the frequencies differ significantly. Recall that, on the other hand, the frequencies of the variables of SR(3, 2, 2, 4) are evenly distributed as we already showed. We have tried guessing the eight most frequent variables, so we had $2^8 = 256$ parallel threads, one thread for each guess. The results are in Table 3.6. We see that we were able to obtain the solution for SR(2, 4, 2, 4) and that the solving time is reduced significantly for the other two ciphers. Note that the F4 algorithm outperforms the SAT solver. Also observe that the preprocessing time for SR(3, 2, 2, 4) is much longer. This is caused by counting the frequencies since each of the 16 polynomials has around 2^{14} monomials. We have also tried guessing eight of the least frequent variables and we were not able to obtain the solutions for SR(2, 4, 2, 4) and SR(2, 2, 4, 4) even though we solved the system for SR(2, 2, 4, 4) in the previous table. This was due to memory limitations as each of the parallel processes allocated dozens of gigabytes — we see in Table 3.5 that the F4 algorithm allocated on average 34.5 GB when solving SR(2, 2, 4, 4) with no guessed variables. We note that each of the threads finished its computation in a different time. The threads that provided no solution usually ended earlier. This could be leveraged in further analysis since this observation also provides information about the correct key. The times stated in the table are always the overall wall times. Recall that each value in the table is the average for five independent experiments. We have also tried guessing different numbers of variables. Guessing more than eight variables produced even longer solving times. This was caused by creating too many threads. On the other hand, we were often unable to obtain the solutions for SR(2, 4, 2, 4) when we guessed less than six variables.

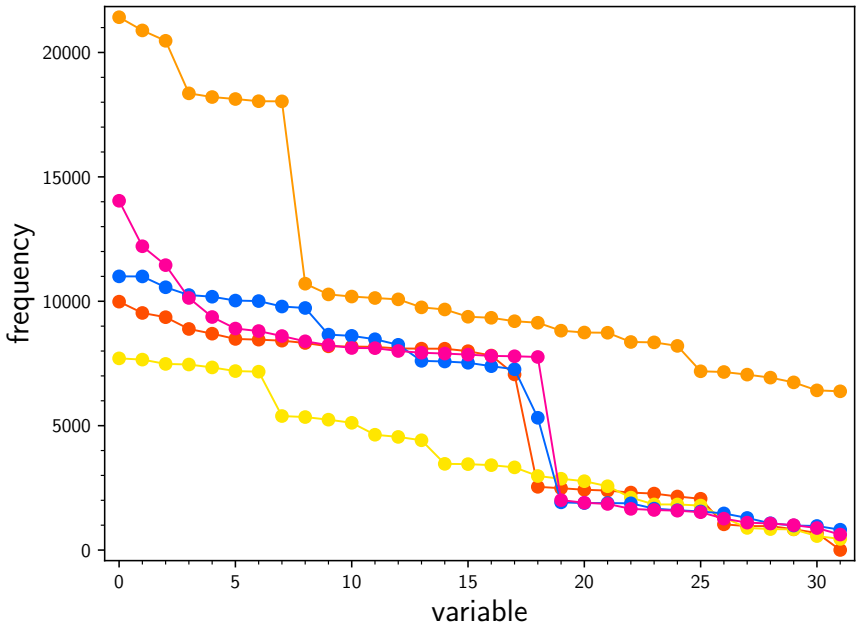

Figure 3.1: Frequencies of the key variables for five instances of SR(2, 2, 4, 4). The variables are ordered according to their frequency.

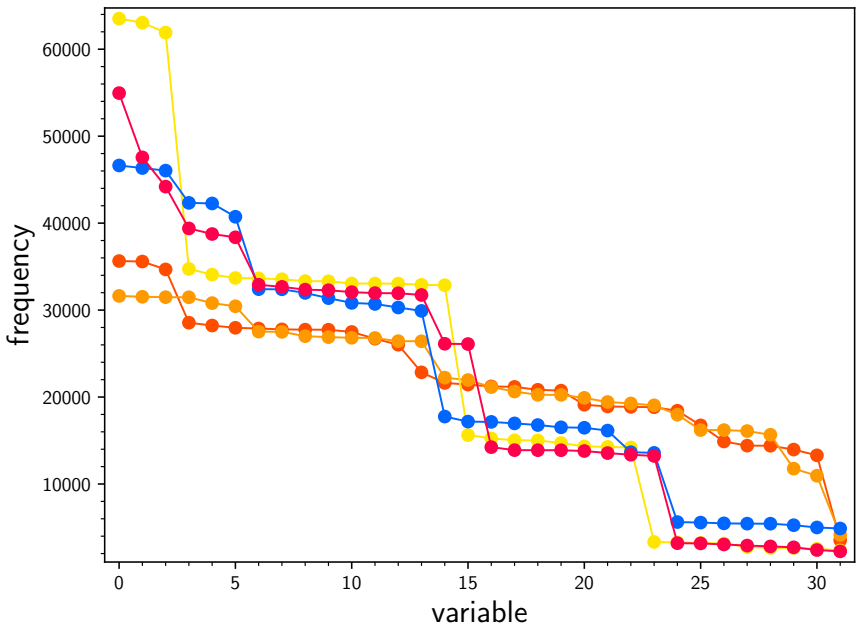

Figure 3.2: Frequencies of the key variables for five instances of SR(2, 4, 2, 4). The variables are ordered according to their frequency.

Table 3.6: Experiments with reduced polynomial systems and guessed variables.

Cipher	Key bits	PT [a]	F4		SAT
			Time	Mem.	Time
SR(3, 2, 2, 4)	16	8 m	6 s	33 MB	35 s
SR(2, 4, 2, 4)	32	2.5 m	43 s	620 MB	9 m
SR(2, 2, 4, 4)	32	31 s	14 s	72 MB	5.5 m

[a] Preprocessing Time — the time required to obtain the system

3.1 Conclusions

In our experiments, we demonstrated the capabilities of solving systems of polynomial equations by means of Gröbner bases and a SAT solver. Initially, we generated systems that contain the auxiliary variables, and we saw that the SAT solver significantly outperformed Gröbner bases. We subsequently eliminated the auxiliary variables by a gradual substitution so that the systems contained only the variables of the initial secret key. We saw that the results were even worse compared to the systems with the auxiliary variables. However, when we combined at least two systems with no auxiliary variables, we got much better results, especially for Gröbner bases. Note, for example, that we were able to obtain the secret key for one round of the AES-128. We also solved one round of all the other ciphers with the state array reduced.

We showed that a 16-bit version of the AES reaches its full diffusion after its third round. We also saw that the polynomial system in the third round has the same properties as the system in the tenth round. This might suggest that the original AES has enough spare rounds as well.

We tried reducing the polynomial systems without auxiliary variables by adding similar polynomials so that equal monomials would cancel each other out, and we also tried guessing the most frequent variables. The combination of these two approaches allowed us to obtain the solutions for some of the systems that we could not solve otherwise.

Bibliography

[1] Zdravkovska, S.; Arnol'd, V. I. Conversation with Vladimir Igorevich Arnol'd. *The Mathematical Intelligencer*, volume 9, no. 4, 1987: pp. 28–32.

[2] Buchberger, B. Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *Journal of symbolic computation*, volume 41, no. 3-4, 2006: pp. 475–511.

[3] Hironaka, H. Resolution of Singularities of an Algebraic Variety Over a Field of Characteristic Zero: I. *Annals of Mathematics*, volume 79, no. 1, 1964: pp. 109–203, ISSN 0003486X. Available from: http://www.jstor.org/stable/1970486

[4] Hironaka, H. Resolution of Singularities of an Algebraic Variety Over a Field of Characteristic Zero: II. *Annals of Mathematics*, volume 79, no. 2, 1964: pp. 205–326, ISSN 0003486X. Available from: http://www.jstor.org/stable/1970547

[5] Becker, T. *Gröbner bases : a computational approach to commutative algebra*. New York: Springer-Verlag, 1993, ISBN 0-387-97971-9.

[6] Cox, D. *Ideals, varieties, and algorithms : an introduction to computational algebraic geometry and commutative algebra*. Cham: Springer, 2015, ISBN 9783319167206.

[7] Adams, W. *An introduction to Gröbner bases*. Providence, R.I: American Mathematical Society, 1994, ISBN 978-0-8218-3804-4.

[8] Hibi, T. *Gröbner bases : statistics and software systems*. Tokyo New York: Springer, 2013, ISBN 978-4-431-54573-6.

[9] Shoup, V. *A computational introduction to number theory and algebra*. Cambridge university press, 2009.

[10] Lidl, R.; Niederreiter, H. *Introduction to finite fields and their applications*. Cambridge university press, 1994.

[11] Menezes, A. J.; Van Oorschot, P. C.; et al. *Handbook of applied cryptography*. CRC press, 2018.

[12] Faugère, J.-C. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, volume 139, no. 1-3, June 1999: pp. 61–88, doi:10.1016/S0022-4049(99)00005-5. Available from: https://hal.archives-ouvertes.fr/hal-01148855

[13] Faugère, J.; Gianni, P.; et al. Efficient Computation of Zero-dimensional Gröbner Bases by Change of Ordering. *Journal of Symbolic Computation*, volume 16, no. 4, 1993: pp. 329 – 344, ISSN 0747-7171, doi:https://doi.org/10.1006/jsco.1993.1051. Available from: http://www.sciencedirect.com/science/article/pii/S0747717183710515

[14] Pub, N. F. 197: Advanced encryption standard (AES). *Federal information processing standards publication*, volume 197, no. 441, 2001: p. 0311.

[15] Daemen, J.; Rijmen, V. *The design of Rijndael*, volume 2. Springer, 2002.

[16] Cid, C.; Murphy, S.; et al. Small scale variants of the AES. In *International Workshop on Fast Software Encryption*, Springer, 2005, pp. 145–162.

[17] Cid, C.; Murphy, S.; et al. *Algebraic aspects of the advanced encryption standard*. Springer Science & Business Media, 2006.

[18] Shannon, C. E. Communication theory of secrecy systems. *The Bell system technical journal*, volume 28, no. 4, 1949: pp. 656–715.

[19] Bulygin, S.; Brickenstein, M. Obtaining and solving systems of equations in key variables only for the small variants of AES. *Mathematics in Computer Science*, volume 3, no. 2, 2010: pp. 185–200.

[20] Saarinen, M.-J. O. Chosen-IV Statistical Attacks on eStream Ciphers. In *SECRYPT*, 2006, pp. 260–266.

[21] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.1)*. 2020. Available from: https://www.sagemath.org

[22] Brickenstein, M.; Dreyer, A. PolyBoRi: A framework for Gröbner-basis computations with Boolean polynomials. *Journal of Symbolic Computation*, volume 44, no. 9, 2009: pp. 1326 – 1345, ISSN 0747-7171, doi: DOI:10.1016/j.jsc.2008.02.017, effective Methods in Algebraic Geometry. Available from: http://dx.doi.org/10.1016/j.jsc.2008.02.017

[23] Bosma, W.; Cannon, J.; et al. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, volume 24, no. 3-4, 1997: pp. 235–265, ISSN 0747-7171, doi:10.1006/jsco.1996.0125, computational algebra and number theory (London, 1993). Available from: http://dx.doi.org/10.1006/jsco.1996.0125

[24] Soos, M.; Nohl, K.; et al. Extending SAT Solvers to Cryptographic Problems. In *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, *Lecture Notes in Computer Science*, volume 5584, edited by O. Kullmann, Springer, 2009, pp. 244–257, doi:10.1007/978-3-642-02777-2_24. Available from: https://doi.org/10.1007/978-3-642-02777-2_24

[25] Aumasson, J.-P. Too Much Crypto. *IACR Cryptol. ePrint Arch.*, volume 2019, 2019: p. 1492.

Appendix **A**

Abbreviations and Symbols

$\mathbb{N}_0 =$ the set of natural numbers including zero

$\mathbb{N}_{>0} =$ the set of natural numbers excluding zero

$\mathbb{Z} =$ the set of integers

$\mathbb{Q} =$ the set of rational numbers (fractions)

$\mathbb{R} =$ the set of real numbers

$\mathbb{C} =$ the set of complex numbers

$\square$ indicates the end of a proof

AES Advanced Encryption Standard

ANF Algebraic Normal Form

CNF Conjunctive Normal Form

SAT SATISFIABILITY (Boolean satisfiability problem)

e.g. (Latin *exempli gratia*) for example

i.e. (Latin *id est*) that is

et al. (Latin *et alii*) and others