FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Algebraic Cryptanalysis of Small Scale Variants of the AES Based on Lattice Basis Reduction

Ing. Tomáš Faikl

Department of Information Security
Supervisor: Mgr. Martin Jureček, Ph.D.

January 9, 2025

Acknowledgements

I would like to thank my close ones for their support and motivation. I also thank my supervisor for discussions and corrections.

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 non-profit 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 9, 2025

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Faikl, Tomáš. *Algebraic Cryptanalysis of Small Scale Variants of the AES Based on Lattice Basis Reduction.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2025.

FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of master's thesis

Title:	Algebraic Cryptanalysis of Small Scale Variants of the AES Based on Lattice Basis Reduction
Student:	Ing. Tomáš Faikl
Supervisor:	Mgr. Martin Jureček, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Security
Department:	Department of Information Security
Validity:	until the end of summer semester 2025/2026

Instructions

Algebraic cryptanalysis is one of the modern methods for analyzing ciphers. A cipher is first converted into a system of polynomial equations, which is then solved to obtain the secret key of the cipher. This work explores the possibility of using lattice basis reduction algorithms, which can transform the original set of polynomial equations into a form more suitable for their calculation.

Instructions:

1. Study the Advanced Encryption Standard (AES) and its small-scale variants.
2. Review literature on algebraic cryptanalysis of small-scale variants of ciphers focusing on the AES.
3. Understand lattice basis reduction techniques, especially LLL (Lenstra–Lenstra–Lovász) or BKZ (Block Korkine-Zolotarev) algorithms.
4. Convert the small-scale variants of AES to a system of polynomial equations.
5. Implement scripts for applying the lattice basis reduction algorithm (or its modified version if necessary) to the systems of polynomial equations.
6. Apply a suitable program (e.g., Magma) to compute Groebner bases for the system of equations from step 5.
7. Discuss the effectiveness of lattice basis reduction in simplifying overdefined polynomial systems derived from AES.

Electronically approved by prof. Ing. Róbert Lórencz, CSc. on 10 September 2024 in Prague.

Abstrakt

Tato práce zkoumá algebraickou kryptoanalýzu malých variant AES, přičemž představuje nové techniky předzpracování pro zjednodušení polynomiálních systémů. Hlavními příspěvky jsou nové metody založené na mřížkách a lineárních kódech, které vylepšují řídkost systému a snižují stupeň polynomů. Je nově představena Gilbert-Varshamova mez pro zhodnocení účinnosti lineárního předzpracování pro pseudo-náhodné šifry. Dále je stručně popsán algoritmus Lenstra-Lenstra-Lovász (LLL) adaptovaný pro lineární kódy z nedávné práce. Jsou navrženy tři nové strategie předzpracování, které lze kombinovat pro zvýšení efektivity. Experimentální výsledky ukazují značná zlepšení při řešení polynomiálních systémů — bylo dosaženo až čtyřicetkrát nižšího výpočetního času oproti předchozím pracím v oblasti algebraického předzpracování. Pseudo-náhodné šifry zůstávají mimo dosah metody předzpracování, nicméně poskytujeme výsledky s vylepšenou dobou běhu díky lepšímu nastavení parametrů algoritmu F_4.

Klíčová slova AES, Gröbnerovy báze, předzpracování, algebraická kryptanalýza, LLL, lineární kódy

Abstract

This thesis explores algebraic cryptanalysis of small-scale AES variants, introducing novel preprocessing techniques to simplify polynomial systems. Key contributions include new methods based on linear codes and lattice theory to improve sparsity and reduce polynomial degree, along with a novel application of the Gilbert-Varshamov bound to assess preprocessing effectiveness for pseudo-random ciphers. Additionally, the summary of Lenstra-Lenstra-Lovász (LLL) algorithm's adoption for linear codes is described based on a very recent effort in the area. Three new preprocessing strategies are proposed, which can be combined for enhanced efficiency. Experimental results show significant improvements in solving polynomial systems, achieving up to 40-fold runtime reduction compared to previous works in our line research of algebraic preprocessing. Pseudo-random ciphers remain unaffected by preprocessing methods, however we provide results with improved runtime by tuning parameters of the F_4 algorithm.

Keywords AES, Gröbner basis, preprocessing, algebraic cryptanalysis, LLL, linear codes

Contents

Nomenclature

$\mathcal{P}(A)$: set of all subsets of set A

$A \subset B$: A is a subset of B, possibly $A = B$

$\mathbb{N}$: Natural numbers without zero, $\{1, 2, 3, \dots\}$

$\mathbb{N}_0$: $\{0, 1, 2, 3, \dots\}$

$[\![a, b]\!]$: For a, $b \in \mathbb{Z}$, $\{a, a+1, \dots, b-1, b\}$

R: a ring

K: a field

$K[x_1, \dots, x_n]$: a ring of polynomials in n variables with coefficients in K

x^α: in $K[x_1, \dots, x_n]$, $x^\alpha := x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ for $\alpha = (\alpha_1, \dots, \alpha_n)$

$\#A$: Cardinality of set A

R/I: Factor ring of ring R by ideal I, $R/I = \{r + I : r \in R\}$

$\mathcal{R}_n^p$: Factor ring $\mathbb{F}_p[x_1, \dots, x_n] / \left(x_1^p - x_1, \dots, x_n^p - x_n\right)$

$\mathcal{R}_n$: Boolean polynomials $\mathcal{R}_n := \mathcal{R}_n^2 = \mathbb{F}_2[x_1, \dots, x_n] / \left(x_1^2 - x_1, \dots, x_n^2 - x_n\right)$.

LLL: Lenstra–Lenstra–Lovász basis reduction algorithm or preprocessing

HME: Highest monomial elimination preprocessing

INFL: Inflating ideal equations preprocessing

Introduction

The Advanced Encryption Standard (AES) is a vital component of contemporary cryptographic protocols, securing vast amounts of data in various applications. While the robustness of the full-scale AES is well-documented, small-scale variants of AES provide a simplified yet insightful framework for exploring the cipher's algebraic properties and potential vulnerabilities. The AES was designed to be resistant to other types of analysis such as linear and differential cryptanalysis, however algebraic cryptanalysis is conjectured to be its weakness. These small-scale variants serve as ideal objects for developing and refining cryptanalytic techniques, particularly those rooted in algebraic methods — algebraic cryptanalysis.

Central to the algebraic modeling of the AES cipher is the Mapping Theorem, which enables one to study any cipher as a system of polynomial equations over finite field.

Theorem (Universal mapping, [9])**.** *Any map from a finite set S to a finite set T can be written as a polynomial system of equations over the field $\mathbb{F}_p$ with p elements, for any prime p.*

This thesis examines the algebraic cryptanalysis of small-scale AES variants through the application of reduction theory. A key element of this approach is the development and formalization of novel (linear) preprocessing techniques designed to simplify the complex polynomial systems that arise from modeling AES operations. One preprocessing technique in particular works by leveraging the structure of linear codes and applying algorithms from lattice theory for basis-reduction to introduce sparsity to the system. The other preprocessing techniques introduced aim to decrease other metrics, such as the maximal occurring degree in the equations. These techniques can be freely composed to enhance their individual advantages. In addition, by introducing the Gilbert-Varshamov bound to the context of linear preprocessing, we provide previously unknown bounds on the possible deficiency of sparsity-inducing methods for pseudo-random ciphers featured in previous works [13, 39].

The thesis integrates reduction algorithms, such as the Lenstra–Lenstra–Lovász (LLL) algorithm, adapted for linear codes. The experimental results demonstrate the effectiveness of these methods, showing significant improvements in both sparsity and solvability of non-pseudo-random polynomial systems which remain unaffected by the Gilbert-Varshamov bound. This research provides both theoretical insights and practical tools for advancing the field of cryptographic analysis via preprocessing.

The thesis is organized as follows; in the first chapter, we provide the necessary mathematical background for the thesis. It covers fundamental concepts in algebra,

algebraic geometry, and Gröbner bases, which form the foundation for modeling cryptographic problems. Special emphasis is placed on algebraic geometry for polynomial rings in finite fields and their role in cryptographic applications that is not typically invoked in this context despite its importance. The chapter is referenced, although we provide some short proofs for a few application statements.

Second chapter introduces the connection between linear codes and polynomial systems, emphasizing their relevance in simplifying cryptographic equations through techniques featuring in rational lattice theory. It proceeds to apply an important concept of Gilbert-Varshamov bound from linear codes to our context and provides bounds on the achievable sparsity of pseudo-random systems with linear preprocessing.

The next chapter discusses the Lenstra–Lenstra–Lovász (LLL) algorithm and its application to lattice reduction. The chapter introduces concepts from a very recent article [24] which shows how these techniques can be adapted for linear codes, providing an overview of the algorithm's role in preprocessing of polynomial systems and possible extensions of the work to weighted linear codes.

Then, we outline the algebraic representation of AES, related work in algebraic cryptanalysis of small-scale variants of AES and finally we give the strategy that we follow in this thesis. This chapter reviews related work and presents an overview for applying algebraic methods to analyze AES, setting the stage for the experimental work in subsequent chapters.

In the chapter on preprocessing, we first formalise our concept of linear preprocessing as a map operating only on the additive structure of the polynomials and we focus on the novel preprocessing methods developed in this thesis. It introduces techniques based on the LLL algorithm to introduce sparsity, highest monomial elimination (HME) to reduce the maximal degree occurring in the polynomials, and inflating the ideal equations (INFL) to increase the number of polynomials available for linear preprocessing by exploiting the multiplicative structure of the polynomials. It also provides simple theoretical guarantees on the efficacy of minimising their respective metrics.

The final chapter presents the experimental evaluation of the proposed methods with our implementation. It includes an overview of the implementation details, followed by the results of applying the preprocessing techniques to various small-scale AES variants. The chapter discusses the effectiveness of the methods in terms of reducing the computational effort required to solve the polynomial systems and highlights key findings and is compared to previous works in the line of work on algebraic preprocessing as means to computing the Gröbner basis. It also introduces a new metric as the number of solutions which was not present in the previous line of work and allows us to appreciate more nuanced aspects of the experiments.

Chapter 1

Theoretical foundations

This chapter sums up the needed theoretical knowledge to appreciate the main body of the thesis. Most of the following are standard definitions and theorems, although we add subjective phrasing and emphasis on some the theorems and proofs. Simple examples are stated to follow along with the statements. A topic not typically encountered in materials featuring Gröbner basis is the algebraic-geometric theory of polynomial rings in finite fields. More precisely, the polynomial ring $\mathbb{F}_p[x_1, \ldots, x_n]/\left(x_1^p - x_1, \ldots, x_n^p - x_n\right)$ for p prime which is not usually encountered in textbooks and is crucial for our work. We add minor corollaries and small proofs to emphasize the relevance for this thesis and make the general theory more accessible. Some of the statements which enable application to our work are also proven mostly from references, although we simplify or add clarity to the proofs. This is explicitly stated.

We assume familiarity with basic concepts of univariate and multivariate polynomials which can be found in most introductory algebra courses. As an example reference, see [22, 7].

In this chapter, R will typically denote a ring and K a field unless specified otherwise. A finite field with q elements is denoted $\mathbb{F}_q$. For references on asymptotics, see [42].

Definition 1.1. Let $f, g : X \to \mathbb{R}$ be real functions, $g \geq 0$, both defined on an unbounded subset X of positive real numbers. We say that

$$f(x) = \mathcal{O}\left(g(x)\right) \quad \text{as } x \to +\infty \tag{1.1}$$

if there exists $M > 0$ and $x_0 \in \mathbb{R}$ such that

$$|f(x)| \leq Mg(x), \quad \forall x \geq x_0. \tag{1.2}$$

1.1 Algebra

Definition 1.2. Let R be a ring and $I \subset R$ its ideal. The elements $f_1, \ldots, f_k \in I$ are said to *generate* ideal I when

$$I = (f_1, \ldots, f_k) = \left\{ \sum_{i=1}^{k} r_i f_i : r_1, \ldots, r_k \in R \right\}. \tag{1.3}$$

Definition 1.3. For a polynomial $f \in K[x_1, \ldots, x_n]$, non-zero summands in form $c_\alpha x^\alpha$ of f, $c_\alpha \in K$, are called *terms*. *Monomials* of f are defined similarly, only

without the coefficients:

$$\mathsf{Monom}(f) := \left\{ x^\alpha : f = \sum_\alpha c_\alpha x^\alpha,\ c_\alpha \neq 0 \right\}. \tag{1.4}$$

For a sequence $f_1, \ldots, f_k \in K[x_1, \ldots, x_n]$,

$$\mathsf{Monom}(f_1, \ldots, f_k) := \mathsf{Monom}(f_1) \cup \cdots \cup \mathsf{Monom}(f_k) \tag{1.5}$$

and similarly for terms. For completeness, we state the definition also for a polynomial ring,

$$\mathsf{Monom}\left(K[x_1, \ldots, x_n]\right) = \{cx^\alpha : c \in K,\ \alpha \in \mathbb{N}_0^n\} \subset K[x_1, \ldots, x_k]. \tag{1.6}$$

The definition naturally extends to monomials of factor-ring $K[x_1, \ldots, x_n]/I$ for ideal I as for $f \in K[x_1, \ldots, x_n]/I$, we have $f = \sum_\alpha c_\alpha \overline{x}^\alpha$ for $(\overline{x})_i = x_i + I \in K[x_1, \ldots, x_n]/I$.

Definition 1.4. A monomial ordering $\prec$ on $K[x_1, \ldots, x_n]$ is a relation on $\mathbb{N}_0^n$, or equivalently, a relation on the set of monomials x^α, $\alpha \in \mathbb{N}_0^n$, satisfying:

- $\prec$ is a total order on $\mathbb{N}_0^n$ (every element is comparable),
- for $\alpha, \beta, \gamma \in \mathbb{N}_0^n$, $\alpha \prec \beta \implies \alpha + \gamma \prec \beta + \gamma$,
- for all $\alpha \in \mathbb{N}_0^n$, $\alpha \geq 0$.

We also define $\alpha \succ \beta \iff \beta \prec \alpha$.

Example 1.5 (Lexicographic order)**.** Let $\alpha = (\alpha_1, \ldots, \alpha_n)$ and $\beta = (\beta_1, \ldots, \beta_n)$ be in $\mathbb{N}_0^n$. We say $\alpha \prec_{lex} \beta$ if the leftmost non-zero entry of the vector difference $\beta - \alpha \in \mathbb{Z}^n$ is positive. We write $x^\alpha \prec_{lex} x^\beta$ if $\alpha \prec_{lex} \beta$. As an example for monomials in $K[x_1, x_2, x_3]$,

$$x_1^2 \succ_{lex} x_1 \cdot x_2^{10} \succ_{lex} x_1 \cdot x_3 \succ_{lex} x_1 \succ_{lex} x_2 \succ_{lex} x_3 \succ_{lex} 1. \tag{1.7}$$

Example 1.6 (Degrevlex order)**.** A degree reverse-lexicographic (degrevlex) order is defined as following. Let $\alpha, \beta \in \mathbb{N}_0^n$, we say $\alpha \succ_{degrevlex} \beta$ if

$$|\alpha| = \sum_{i=1}^n \alpha_i \succ_{degrevlex} |\beta| = \sum_{i=1}^n \beta_i, \tag{1.8}$$

or $|\alpha| = |\beta|$ and the rightmost non-zero entry of $\alpha - \beta \in \mathbb{Z}^n$ is negative. This ordering has been shown to be the most efficient for polynomial calculations.

Definition 1.7. Let $f = \sum_\alpha c_\alpha x^\alpha$ be a non-zero polynomial in $K[x_1, \ldots, x_n]$ and choose a monomial order.

- The multi-degree of f is

$$\text{multideg}(f) = \max\{\alpha \in \mathbb{N}_0^n : c_\alpha \neq 0\}, \tag{1.9}$$

  and the maximum is taken with respect to the induced monomial order.

- The leading monomial of f is

$$\mathsf{LM}(f) = x^{\text{multideg}(f)} \tag{1.10}$$

- The leading coefficient of f is the coefficient of the leading monomial in f,

$$\mathsf{LC}(f) = c_{\text{multideg}(f)} \in K. \tag{1.11}$$

Definition 1.8. A commutative ring R is Noetherian if it satisfies the ascending chain condition, that is every non-decreasing sequence of ideals $I_1 \subset I_2 \subset \cdots \subset R$ is eventually constant — there exists $n \in \mathbb{N}$ such that

$$I_n = I_{n+1} = \ldots. \tag{1.12}$$

Theorem 1.9. *Ring R is Noetherian to every ideal of R is finitely generated — for every ideal $I \subset R$ there exists $k \in \mathbb{N}$ and $f_1, \ldots, f_k \in R$ such that $I = (f_1, \ldots, f_k)$.*

Theorem 1.10 (Hilbert basis theorem)**.** *Let K be a field. Then every ideal I of $K[x_1, \ldots, x_n]$ is finitely generated.*

Theorem 1.11 (Division theorem)**.** *Let $\prec$ be a monomial order on monomials of $K[x_1, \ldots, x_n]$ and let $\mathcal{F} = (f_1, \ldots, f_s)$ be an ordered s-tuple of polynomials in $K[x_1, \ldots, x_n]$. Then every $f \in K[x_1, \ldots, x_n]$ can be written as*

$$f = q_1 f_1 + \cdots + q_s f_s + r, \tag{1.13}$$

where q_i, $r \in K[x_1, \ldots, x_n]$, and either $r = 0$ or r is a linear combination, with coefficients in K, of monomials, none of which is divisible by any of $\mathsf{LT}(f_1), \ldots, \mathsf{LT}(f_s)$. We call r a remainder of f on division by $\mathcal{F}$.

Proof. Proof is constructive and provides a way to divide polynomials in multiple variables [22]. □

1.2 Algebraic geometry

In the following, we will state theorems from the field of algebraic geometry which is concerned with polynomial systems and their solutions. These theorems are not directly applicable to our work, although the concept of solving a polynomial system via its Gröbner basis rests heavily upon them. Additional reason we state them is that in classical texts, these theorems are not applicable to finite fields and we will later provide their analogues to finite fields which are the base of our thesis. See reference [22] for algebraically closed fields and [30] for details for finite fields.

Definition 1.12. Let K be a field. Let $I = (f_1, f_2, \ldots, f_k) \subset K[x_1, x_2, \ldots, x_n]$ be a polynomial ideal. The common solutions of these polynomials in K^n form a *variety* denoted as $V(I)$,

$$V(I) = \{x \in K^n : \forall f \in I,\ f(x) = 0\}, \tag{1.14}$$

Note that the notion of variety depends on the base field K. Sometimes, we will use the notation V_L for field extension $L \supset K$ as ideal $I \subset K$ is also an ideal in L,

$$V_L(I) = \{x \in L^n : \forall f \in I \subset L,\ f(x) = 0\}. \tag{1.15}$$

Conversely, given a subset $S \subseteq K^n$, the set of all polynomials that vanish on S forms an ideal in the polynomial ring $K[x_1, \ldots, x_n]$.

Definition 1.13. Let $S \subset K^n$. The ideal of polynomials that vanish on S is called the *vanishing ideal* of S and is denoted by

$$I(S) = \{f \in K[x_1, \ldots, x_n] : \forall x \in S,\ f(x) = 0\}. \tag{1.16}$$

The weak and strong Nullstellensatz are the cornerstones of algebraic geometry. In order to use them, one needs to work in algebraically closed fields such as $\mathbb{C}$.

Theorem 1.14 (Weak Nullstellensatz)**.** *Let k be an algebraically closed field and I an ideal in a polynomial ring $K[x_1, \ldots, x_n]$. Then*

$$I = (1) = R \iff V(I) = \emptyset. \tag{1.17}$$

Definition 1.15. Let R be a commutative ring. The *radical* of an ideal $I \subset R$, denoted by $\sqrt{I}$ is defined as

$$\sqrt{I} = \{r \in R : r^n \in I \text{ for some } n \in \mathbb{N}\}. \tag{1.18}$$

Note that $I \subset \sqrt{I}$.

Theorem 1.16 (Strong Nullstellensatz)**.** *Let K be an algebraically closed field. Let I be an ideal in $K[x_1, \ldots, x_n]$. Then $I(V(I)) = \sqrt{I}$. Intuitively, each variety uniquely corresponds to an ideal modulo multiplicities.*

Proof. A proof can be found in [28]. □

However, $\mathbb{F}_p$ is not algebraically closed for any p. By factoring the polynomials by the field equations, we will in fact obtain a ring with very similar properties, as is shown in the following statements. The proofs are rather elegant and short and we encourage the reader to examine them in the linked reference [30]. For the sake of brevity, let us denote

$$\mathcal{R}_n^p := \mathbb{F}_p[x_1, \ldots, x_n] / \left(x_1^p - x_1, \ldots, x_n^p - x_n\right) \tag{1.19}$$

and, as our main interest is the case of $p = 2$,

$$\mathcal{R}_n := \mathcal{R}_n^2 = \mathbb{F}_2[x_1, \ldots, x_n] / \left(x_1^2 - x_1, \ldots, x_n^2 - x_n\right). \tag{1.20}$$

Lemma 1.17. *Let p be prime. Then every ideal $I \subset \mathcal{R}_n^p$ is radical, i.e. $\sqrt{I} = I$.*

Proof. The lemma holds in general [30] but we will prove it for simplicity only for the case $p = 2$, i.e. ideals in $\mathbb{F}_2[x_1, \ldots, x_n] / \left(x_1^2 - x_1, \ldots, x_n^2 - x_n\right)$. The proof is trivial after we show that $f^2 = f$ for any $f \in \mathcal{R}_n^p$. Indeed, the formula can be easily obtained by using

$$(m + g)^2 = m^2 + g^2 = m + g^2 \tag{1.21}$$

where m denotes arbitrary monomial, $g \in \mathcal{R}_n$ and continuing by induction on the number of monomials of g. The inclusion $\sqrt{I} \subset I$ holds always. Assume $f \in \sqrt{I}$ so there exists $m \in \mathbb{N}$ such that $f^m \in I$. Due to the field equations, $f = f^m \in I$. □

Theorem 1.18 (Strong Nullstellensatz for finite fields)**.** *Let p be prime. Then for all ideals $J \subset \mathbb{F}_p[x_1, \ldots, x_n]$,*

$$I(V(J)) = J + (x_1^p - x_1, \ldots, x_n^p - x_n). \tag{1.22}$$

For ideals $J \subset \mathcal{R}_n^p$, it has a simple form

$$I(V(J)) = J. \tag{1.23}$$

Proof. A proof can be found in [30]. □

Corollary 1.19. *Variety map* $V : \mathsf{Ideals}(\mathcal{R}_n^p) \to \mathbb{F}_p^n$ *is a bijection, i.e. for ideals* J_1, J_2 *in* $\mathcal{R}_n^p$,

$$J_1 = J_2 \iff V(J_1) = V(J_2). \tag{1.24}$$

Proof. The implication $J_1 = J_2 \implies V(J_1) = V(J_2)$ is trivial from

$$V\left(J + (x_1^p - x_1, \ldots, x_n^p - x_n)\right) = V(J) \cap \mathbb{F}_p^n = V(J) \tag{1.25}$$

for $J \subset \mathbb{F}_p[x_1, \ldots, x_n]$. The rest follows from applying ideal operation I to both sides of equation and from Theorem 1.18, $I(V(J)) = J$. □

Example 1.20. The equivalence from Corollary 1.19 holds only in the factor-ring $\mathcal{R}_n^p$. Let $R = \mathbb{F}_2[x]$, $I_1 = (x) \subset R$ and $I_2 = (x^2) \subset R$. Then $V(I_1) = V(I_2) = \{0\}$ but $I_2 \neq I_1$.

On the other hand, for $\mathcal{R} = \mathbb{F}_2[x]/(x^2 - x)$, $\bar{x} = x + (x^2 - x)\mathbb{F}_2[x]$ the equivalence class of x in $\mathcal{R}$, we have $(\bar{x}^2) = (\bar{x}) \subset \mathcal{R}$ and the equivalence holds. Intuitively, $\mathcal{R}_n$ factors out multiplicities.

Example 1.21. Let $R = \mathbb{F}_2[x]$, $I_1 = (x^2 + x + 1) \subset R$ and $I_2 = (1) = R$. Then $V(I_1) = V(I_2) = \emptyset$ but $I_2 \neq I_1$. In $\mathcal{R}_n$, $(1) = (x^2 + x + 1)$. This is useful when calculating Gröbner basis in the next section.

For the rest of the thesis, we will mostly restrict ourselves to the case of $p = 2$ for concreteness.

1.3 Gröbner basis

Definition 1.22. Let $I \subset K[x_1, \ldots, x_n]$ be an ideal and fix a monomial ordering on the monomials of $K[x_1, \ldots, x_n]$. Then the set of leading terms of elements of I is denoted

$$\mathsf{LT}(I) = \left\{cx^\alpha : \exists f \in I,\ \mathsf{LT}(f) = cx^\alpha\right\}. \tag{1.26}$$

Additionally, $(\mathsf{LT}(I))$ is the ideal generated by elements of $\mathsf{LT}(I)$.

Definition 1.23. Fix a monomial order on the polynomial ring $K[x_1, \ldots, x_n]$. A finite subset $G \subset \{g_1, \ldots, g_k\}$ of a nontrivial ideal $I \subset K[x_1, \ldots, x_n]$ is said to be a *Gröbner basis* if

$$(\mathsf{LT}(g_1), \ldots, \mathsf{LT}(g_k)) = (\mathsf{LT}(I)). \tag{1.27}$$

Definition 1.24. A *reduced Gröbner basis* of a polynomial ideal I is a Gröbner basis G of I such that:

1. $\mathsf{LC}(f) = 1$ for all $f \in G$,
2. for all $f \in G$, no monomial of f lies in $\left(\mathsf{LT}\left(G \setminus \{f\}\right)\right)$.

Theorem 1.25. *Let* $I \neq \{0\}$ *be a polynomial ideal. Then, for a given monomial ordering,* I *has a unique reduced Gröbner basis.*

Definition 1.26. Given an ideal $I \subset K[x_1, \ldots, x_n]$, the *j-th elimination ideal* is

$$I_j = I \cap K[x_{j+1}, \ldots, x_n]. \tag{1.28}$$

Theorem 1.27. *Let* K *be algebraically closed. If* G *is a Gröbner basis for ideal* $I \subset K[x_1, \ldots, x_n]$ *with respect to the lex order* $x_1 \succ_{lex} \cdots \succ_{lex} x_n$, *then for* $0 \leq j < n$, $G_j = G \cap K[x_{j+1}, \ldots, x_n]$ *is a Gröbner basis for the elimination ideal* I_j.

Note 1.28. Theorem 1.27 gives a recipe to find variety using Gröbner basis; first, compute lex Gröbner basis of ideal I and then compute its last elimination ideal, which is an ideal of univariate polynomials. This can be solved via standard methods for computing roots of univariate polynomials, such as the Berlekamp's factoring algorithm. Subsequently, the solutions are substituted to lower elimination ideals until the system is solved. Note that the theorem remains valid with minor modifications also for $\mathcal{R}_n^p$, see [30].

A Gröbner basis describes solutions of a system over closure of the base field. For a cryptographic polynomial system $I \subset \mathbb{F}_2[x_1, \ldots, x_n]$, we are interested in computing its variety over the base field $\mathbb{F}_2^n$ via Gröbner basis. However, we are only interested in solutions in the base field, which contains the key, and not in the algebraic closure $\overline{\mathbb{F}_2}$. In order to facilitate that, one can compute the Gröbner basis of $I + (x_1^2 - x_1, \ldots, x_n^2 - x_n)$ instead.

Lemma 1.29. *Let $I = (f_1, \ldots, f_k) \subset \mathbb{F}_2[x_1, \ldots, x_n]$. By adding the field equations to I, the Gröbner basis describes exactly the solutions of I in $\mathbb{F}_2^n$. In other words,*

$$V_{\mathbb{F}_2}(I) = V_{\overline{\mathbb{F}_2}}\left(I + (x_1^2 - x_1, \ldots, x_n^2 - x_n)\right) = V_{\overline{\mathbb{F}_2}}\left(f_1, \ldots, f_k, x_1^2 - x_1, \ldots, x_n^2 - x_n\right). \tag{1.29}$$

1.3.1 Number of solutions from Gröbner basis

Definition 1.30. Let $I \subset K[x_1, \ldots, x_n]$ be an ideal. Define the *standard monomials* as

$$SM(I) = \left\{x^\alpha : x^\alpha \notin (\mathsf{LT}(I))\right\}. \tag{1.30}$$

Theorem 1.31. *Let $K = \mathbb{F}_p$, $I \subset K[x_1, \ldots, x_n]$ be an ideal. Then $K[x_1, \ldots, x_n]/I$ is isomorphic as a K-vector space to the span of standard monomials $\mathsf{Span}_K(SM(I))$,*

$$K[x_1, \ldots, x_n]/I \cong \mathsf{Span}_K\left(SM(I)\right). \tag{1.31}$$

Proof. Let $R = \mathbb{F}_p[x_1, \ldots, x_n]$. Define $\phi : \mathsf{Span}\left(SM(I)\right) \subset R \to R/I$ as $\phi = \pi|_{\mathsf{Span}(SM(I))}$ where $\pi : R \to R/I$ is the canonical projection $\pi(f) = f + I$. Now, we will proceed to show that ϕ is bijective.

Map ϕ is injective as $\ker \phi = \{0\}$. Take $f \in \mathsf{Span}(SM(I)) \neq 0$ such that $\phi(f) = 0$. Since f cannot be reduced by I, then by the Division Theorem 1.11, $f = 0$. Also, ϕ is surjective. Fix arbitrary $f' \in R/I$ and assume $f' = f + I$. Then $f \in \mathsf{Span}(SM(I))$ by the same theorem. □

Note 1.32. In the above proof, we have presented a more straightforward proof modified from [29]. The above theorem holds also for algebraically closed fields.

Proposition 1.33. *Let $I \subset \mathcal{R}_n^p$ be an ideal. Then $SM(I)$ is finite and its size is equal to the number of points in the variety,*

$$\#V(I) = \#SM(I). \tag{1.32}$$

Proof. Set of standard monomials $SM(I)$ is finite as $\mathcal{R}_n$ contains the field equation and hence, the set of all monomials $\mathsf{Monom}(\mathcal{R}_n^p)$ is finite. The main part proceeds by constructing an explicit basis of $\mathcal{R}_n/I \cong \mathbb{F}_p[x_1, \ldots, x_n]/\left(I + (x_1^p - x_1, \ldots, x_n^p - x_n)\right)$ via the Legendre interpolation polynomials centered at the points of the variety $V(I + (x_1^p - x_1, \ldots, x_n^p - x_n))$. The proof is then concluded by Theorem 1.31 and the strong Nullstellensatz for finite fields. More details can be found in [29]. □

Note 1.34. This provides an explicit way to count the number of solutions of polynomials system $I \subset \mathcal{R}_n$ via its Gröbner basis $G = \{f_1, \ldots, f_k\}$, $(G) = I$ as it directly related to the standard monomials $SM(I)$. In fact, $SM(I) = SM\Big(\big(\mathsf{LT}(I)\big)\Big) = SM\Big(\big(\mathsf{LT}(f_1), \ldots, \mathsf{LT}(f_k)\big)\Big)$. Determining the number of points in the variety of ideal I from its Gröbner basis might still be ineffective when implemented naïvely and there are algorithms in [22] which encode the information from the Gröbner basis to an object called Hilbert series from which the number of points can be easily extracted. We use this algorithm to compute the Hilbert series of a (homogeneous) ideal corresponding to $LM(G)$. It can be shown that the Hilbert series is a polynomial for $\mathcal{R}_n$ and the number of solutions is the sum of its coefficients.

1.3.2 Complexity of computation of Gröbner basis

Let us summarise the current understanding of the complexity of finding a Gröbner basis of an ideal in $\mathcal{R}_n^p$ or $\mathbb{F}_p[x_1, \ldots, x_n]$.

The problem of solving algebraic equations is known to be an NP-complete problem even for finite fields and even when the input polynomials have degree less than or equal to two (unless they are all linear polynomials). In general, for $\mathcal{R}_n$, the complexity of computing a Gröbner basis is at most a polynomial in 2^n. There is a comprehensive paper [10] on the complexity of computing Gröbner basis (via the more analysable F_5 algorithm) of systems in $\mathcal{R}_n$ under the assumption that the system is regular or semi-regular — the definition of which is outside the scope of this thesis, nevertheless the assumption means that there is the least amount of algebraic relations (syzygies) between the initial set of polynomials. We do not think it is the case for pseudo-random systems with maximal degree equal to number of variables. In addition, they explore the case when the degree is bound to some small number and the number of variables is large. Another foundational assumption upon which the complexity estimate is founded is that the computation time is dominated by linear algebra on the Macaulay matrix corresponding to the largest degree occurring during the computation. Unfortunately, the maximal degree of polynomial equations for diffused systems in our approach is expected to be equal to the number of variables, so the results of the article do not apply. We state these results as they are basically the only practical estimate that exists in the literature for equations for finite fields.

The Macaulay matrix for an ideal $I = (f_1, \ldots, f_k)$ is defined in [35]. In simple terms, its columns correspond to monomials of degree $\leq d$, rows to polynomials $x^\alpha f_i$ of degree $\leq d$ and the entry (i, j) is the coefficient of the monomials corresponding to column j in the polynomial corresponding to row i. By using Gaussian elimination on this matrix, one computes the Gröbner basis — that is an approximate description of the XL algorithm used for computing Gröbner bases.

What follows are basic known estimates on the complexity of computing Gröbner basis via Buchberger-like algorithm like F_4 and F_5. For more advanced information on the complexity of computing Gröbner basis, see papers [26, 6, 20, 34, 46].

Definition 1.35. Let $\mathcal{F} = \{f_1, \ldots, f_n\} \subset \mathcal{R}_n$. The *solving degree* of $\mathcal{F}$, denoted *solv. deg*$(\mathcal{F})$, is the least degree d for which Gaussian elimination in the degrevlex Macaulay matrix of degree d yields a Gröbner basis of $(\mathcal{F}) = (f_1, \ldots, f_k)$.

Note 1.36. The solving degree for systems with some structure, i.e. relations between the initial polynomials, is usually much lower than for generic (semi-regular) systems. The reader is invited to explore the ideas in [10] for more details.

Theorem 1.37. *Let $R = \mathbb{F}_p[x_1, \ldots, x_n]$ for p prime and $\mathcal{F} = \{f_1, \ldots, f_k\}$ containing field equations $\left\{x_1^2 - x_1, \ldots, x_n^2 - x_n\right\}$ ordered such that $deg(f_i) \geq deg(f_{i+1})$ for all $1 \leq i < k$ and $k \geq n + 1$. Then*

$$max.\ GB.\ deg(\mathcal{F}) \leq solv.\ deg(\mathcal{F}) \leq d_1 + \cdots + d_{n+1} - n \tag{1.33}$$

where max. GB. $deg(\mathcal{F})$ is the maximal degree of the polynomials appearing in the Gröbner basis of $(\mathcal{F})$.

Proof. Proof consists of two parts. There is a similar theorem in [35] which assumes that $\mathcal{F}$ is in "generic coordinates", definition of which is outside the scope of this thesis. Proof that $\mathcal{F}$ with field equations satisfies the condition can be found in [20]. □

Note 1.38. It might not always be beneficial to add the field equations in case p is large, as discussed in [20]. Although in the case $p = 2$ it is almost always beneficial. Polynomial systems derived from highly diffused ciphers have typically a small amount of solutions and their Gröbner basis have a small maximal degree. On the other hand, larger systems with high amount of solutions typically have larger maximal Gröbner basis degrees. This estimate gives some, yet not particularly good, bounds on the solving degree. The degree of any polynomial (or more precisely the degree of any equivalence class) in $\mathcal{R}_n$ is not greater that n.

On the other hand, when computing the Gröbner basis of $\{f_1, \ldots, f_k\}$ containing the field equations $\left\{x_1^2 - x_1, \ldots, x_n^2 - x_n\right\}$ in $\mathbb{F}_2[x_1, \ldots, x_n]$ via an algorithm which is oblivious to the field equations, the degree can rise up to the upper bound in the Theorem. The Gröbner basis algorithms implemented in Magma and PolyBori are the only algorithms that are known to us that allow to exploit these relations.

There are a variety of algorithm for computing the Gröbner basis, be it the Buchberger [15] or XL [35] algorithm. However, one of the most performant algorithms today are Buchberger-like algorithm families called F_4 and F_5, references to which can be found in [22]. Same as the Buchberger algorithm, they operate by multiplying the original polynomials by monomials, finding reducing relations (syzygies) and finally producing a Gröbner basis. However, the F_4 and F_5 algorithms also operate by cleverly constructing the Macaulay matrix such that the amount of superfluous polynomials is kept low and fast matrix algebra is applied. The details of the algorithms can be found in the reference as they are rather advanced and simply stating them at this place would not bring any additional value. Also, we do not know the exact implementation of the F_4 algorithm used by the closed-source computer algebra system Magma which we use. These algorithms are hard to analyse, nevertheless, see references above to gather state-of-the-art information.

Chapter 2

Linear codes and lattices

A lattice $\mathcal{L} \subset \mathbb{R}^n$ is defined as the set of all integer linear combinations of a set of linearly independent basis vectors $A := (a_1, \ldots, a_k) \in \mathbb{R}^n$. Specifically,

$$\mathcal{L}(A) := \{z_1 a_1 + \cdots + z_k a_k : z_i \in \mathbb{Z}\}.$$

We study the geometry of lattices as determined by the Euclidean norm, $\|a\| := \sqrt{a_1^2 + \cdots + a_n^2}$. A key geometric property is the *minimum distance* of the lattice, defined as the smallest Euclidean norm of any non-zero lattice vector:

$$\lambda_1(\mathcal{L}) := \min_{y \in \mathcal{L} \setminus \{0\}} \|y\|.$$

At a high level, there are two fundamental computational problems related to lattices:

- The first problem is to find a short non-zero lattice vector. Given a basis for a lattice $\mathcal{L}$, the objective is to identify a non-zero vector $y \in \mathcal{L} \setminus \{0\}$ with a relatively small Euclidean norm $\|y\|$.
- The second problem involves finding a lattice vector close to a target vector $t \in \mathbb{R}^n$. Given a basis for a lattice $\mathcal{L}$ and a target vector $t \in \mathbb{Z}^n$, the goal is to find $y \in \mathcal{L}$ such that $\|y - t\|$ is relatively small.

We will not make elaborate descriptions of lattice theory as we are mainly interested in linear codes. However, lattice theory historically gave rise to the lattice-reduction techniques such as LLL algorithm and the techniques were ported to linear codes only very recently. Hence, we make only a brief summary of lattice theory needed to state the original LLL algorithm in the next chapter.

On the other hand, a linear code $\mathcal{C} \subset \mathbb{F}_q^n$ is a subspace of the vector space $\mathbb{F}_q^n$ over the finite field $\mathbb{F}_q$. Specifically, it is the set of all $\mathbb{F}_q$-linear combinations of a linearly independent list of vectors $B = (b_1, \ldots, b_k)$:

$$\mathcal{C}(B) := \{z_1 b_1 + \cdots + z_k b_k : z_i \in \mathbb{F}_q\}.$$

The vectors $b_1, \ldots, b_k$ form a basis for the code, and k is called the *dimension* of the code.

We analyze the geometry of the code through the *Hamming weight* $|c|$ for $c \in \mathbb{F}_q^n$, which is the number of coordinates of c that are nonzero. A fundamental property of a code is its *minimum distance*, defined as the smallest Hamming weight of any non-zero codeword:

$$d_{\min}(\mathcal{C}) := \min_{c \in \mathcal{C} \setminus \{0\}} |c|.$$

Two important computational problems are associated with linear codes:

- **Finding a short non-zero codeword:** Given a basis for a code $\mathcal{C}$, the objective is to find a non-zero codeword $c \in \mathcal{C} \setminus \{0\}$ with relatively small Hamming weight $|c|$.
- **Finding a codeword close to a target vector:** Given a basis for a code $\mathcal{C}$ and a target vector $t \in \mathbb{F}_q^n$, the goal is to find a codeword $c \in \mathcal{C}$ such that $|c - t|$ is relatively small. (Here, the term "relatively small" is deliberately left vague.)

The relationship between linear codes and lattices is evident: moving from codes to lattices essentially involves replacing the finite field $\mathbb{F}_q$ with the integers $\mathbb{Z}$ and substituting the Hamming weight $|\cdot|$ with the Euclidean norm $\|\cdot\|$.

Unsurprisingly, this analogy extends to various applications. For example, both lattices and codes are used for decoding noisy communication channels. They also play important roles in cryptography, including post-quantum cryptographic schemes (see, e.g., [40, 2, 3, 33, 43] and also some recent post-quantum algorithms in [38], of which one of the authors is also the author of the pioneering article [24]). Furthermore, many complexity-theoretic hardness results for coding and lattice problems have been proven simultaneously or nearly simultaneously, often using similar or identical techniques.

In this thesis, we briefly explore the ideas of a very recent endeavour in transferring basis-reduction of lattice theory to linear codes which was pioneered by [24] for LLL in $\mathbb{F}_2$ and further extended in [31] to $\mathbb{F}_q$ and also to BKZ and other, more powerful reduction techniques.

2.1 Relevance to polynomial rings

Before we delve into the depths of both linear code and lattice theory and algorithms, let us clarify why we are interested in these abstractions when solving a system of polynomial equations.

In previous works [14, 13, 39], it was noted that the polynomial system of some ciphers can be substantially simplified in terms of lower average number of non-zero polynomials. Frequently, this leads to better solving times (computing the Gröbner basis of the system) as the F_4 algorithm might construct matrices with fewer monomials and ultimately lead to sparser systems. In those works, the authors have used techniques from machine learning known as *Partitioning Around Medoids* (PAM) and *Locality Sensitive Hashing* (LSH) and a reference algorithm which eliminates the highest occurring monomial from all polynomials in the system. Without going into much detail, all of these algorithms operate by XORing some preselected pairs of polynomials of the input polynomial system and output a polynomial system with lower amount of polynomials.

As the XOR operation on polynomials is a simple addition of polynomials in $\mathbb{F}_2[x_1, \ldots, x_n]$, we can naturally cast the problem to the domain of linear codes over $\mathbb{F}_2$ as finding the shortest codewords in a given linear code. Each polynomial can be thought of as a vector of zeros and ones, one at position i indicating that a given i-th monomial (in a fixed monomial ordering) is present in the polynomial.

We would like to provide bounds on the effectiveness of reducing the number of monomials by arbitrary process using only addition, as is the case with all the previous algorithms that were examined. Additionally, we would like to apply a very recent algorithm adapted into the domain of linear codes from the theory of integer lattices. The algorithm is based on LLL lattice reduction algorithm in the domain of lattice theory in $\mathbb{Z}^n$ or $\mathbb{R}^n$. A very recent article [24] has allowed the modification

also to the domain of linear codes for $\mathbb{F}_2$ and even more recent paper [31] which extends the techniques to $\mathbb{F}_p$ and also provides more powerful algorithms such as BKZ. Here, we adopt the modified LLL algorithm for linear codes over $\mathbb{F}_2$. This algorithm is practically both faster and produces sparser systems than any of the current algebraic preprocessing techniques presented in [14, 13, 39]. It should be noted that we have not compared the results with improved LSH in [39] as it does not contain detailed experimental results for different ciphers.

A qualitative theoretical analysis of the expected sparsity after preprocessing was already done in [39]. However, we provide alternative bounds which are more appropriate for the type of preprocessing concerning this thesis. The original work [39] was interested in using LSH which intrinsically operates on pairs of polynomials and hence, the theoretical estimates of the sparsity were based on the pairwise distance between all the polynomials of the input set. Here, we are interested in the Hamming distance of the polynomials, i.e. estimating the distance of the polynomials to the zero vector. This is more relevant as we are interested in sparsity of more general systems which do not necessarily operate on pairs of polynomials — for example, the algorithm should be able to add more than two polynomials. Later in this chapter, we compare the two results.

A precise formalization of our approach to study polynomial systems through linear codes is given in Chapter 5.

2.2 Linear codes

As this thesis is concerned with providing bounds on effectiveness of linear preprocessing as a building block for more general and sophisticated preprocessing algorithms, the notion of linear codes is crucial. It models the additive structure of the polynomials and discards its multiplicative nature.

First, let us remind the notions of linear codes. We assume basic familiarity with the concepts and restate some of them here just for completeness. For more details, please see references, e.g. [49].

Definition 2.1. A subspace of the vector space $\mathbb{F}_q^m$ of dimension k is called $[m, k]$-linear code.

Definition 2.2. Let G be a $k \times m$ matrix with linearly independent rows $r_1, \ldots, r_k \in \mathbb{F}_q^n$. The code

$$\mathcal{C} := \{u_1 r_1 + \cdots + u_k r_k : u_1, \ldots, u_k \in \mathbb{F}_q\} \subset \mathbb{F}_q^n \tag{2.1}$$

is said to be generated by the matrix G (and it is a $[m, k]$-linear code).

Note 2.3. It is customary to denote the length of code as n in literature. Here, we use notation of m as n is reserved for a number of variables in a polynomial ring.

Lemma 2.4. *The number of distinct codewords of a linear $[m, k]$ code $\mathcal{C}$ is*

$$|\mathcal{C}| = q^k \tag{2.2}$$

Proof. It can easily be seen from the definition of the generator matrix. Any word of code $\mathcal{C}$ can be expressed as $\sum_{i=1}^{k} \alpha_i g_i$ where $\alpha_i \in \mathbb{F}_q$ and g_i are the code generators for all $1 \leq i \leq k$. At the same time, all of these combinations are distinct. By contradiction, if $x = \sum_i \alpha_i x_i$, $x' = \sum_i \beta_i x_i$ with $\{\alpha_i\}_i$, $\{\beta_i\}_i \subset \mathbb{F}_q$ and $x = x'$ but $\alpha_i \neq \beta_i$ for all $1 \leq i \leq k$ then $\sum_i (\alpha_i - \beta_i) x_i = 0$ and that is a contradiction with linear independence of the basis of the code. □

Definition 2.5 (Minimal distance)**.** Minimal distance of linear code $\mathcal{C}$ is defined as

$$d_{\min}(\mathcal{C}) := \min\{|x| : x \in \mathcal{C} \setminus \{0\}\}, \tag{2.3}$$

where $|\cdot| : \mathbb{F}_2^m \to \mathbb{R}$, $|x| := \sum_i x_i$ is the Hamming weight.

2.2.1 Random codes and Gilbert-Varshamov bound

Definition 2.6. By *uniform distribution* $\mathcal{U}(\mathbb{F}_q)$ on $\mathbb{F}_q$, we understand a discrete probability distribution such that every element is equally likely to be chosen. That is, it has constant discrete density function $f : \mathbb{F}_q \to [0,1]$, $f(i) = 1/q$ for all $i \in \mathbb{F}_q$.

Definition 2.7. By a random linear $[m,k]$-code over $\mathbb{F}_q$, we understand a linear code with generator matrix $G \in \mathbb{F}_q^{m\times k}$ such that its entries are independent and identically distributed according to uniform distribution $\mathcal{U}(\mathbb{F}_q)$ and the rows are linearly independent. I.e. for $q = 2$, each entry of such G has probability $1/2$ to be non-zero independently on other entries.

Note 2.8. The portion of matrices G with statistically independent and uniformly random entries which have linearly dependent rows is exponentially small compared to those with linearly independent rows. Hence, we will ignore the difference in subsequent analysis. For more details, see Proposition 5.11.

Now, let us examine an estimate on the minimal distance $d_{\min}$ of a random linear code.

Theorem 2.9. *For a random linear $[m,k]$-code over $\mathbb{F}_2$, there exists a distance $d_{GV} \in \mathbb{R}$ called* Gilbert-Varshamov distance *such that for all $d \in \mathbb{N}$ there is $\epsilon > 0$ such that*

$$\mathbb{E}\left[\#\{x \in \mathcal{C} : |x| \le d\}\right] = \begin{cases} \mathcal{O}\left(2^{\epsilon m} m\right), & d \in (d_{GV}, m - d_{GV}), \\ \mathcal{O}\left(2^{-\epsilon m}\right), & \text{otherwise}, \end{cases} \tag{2.4}$$

where the asymptotics are for $m \to +\infty$ and

$$\boxed{d_{GV} := \hat{\alpha} m, \quad H(\hat{\alpha}) = 1 - \frac{k}{m},\ \hat{\alpha} \in (0, m/2]}, \tag{2.5}$$

where $H : [0,1] \to [0,1]$ is the binary entropy $H(x) := -x\log_2(x) - (1-x)\log_2(1-x)$.

Proof. Let us fix $d \in \mathbb{N}$. Then for a given $x \in \mathcal{C} \setminus \{0\}$ we obtain a sum of binomial distribution $B(m,p)$ with $p = 1/q$ for field $\mathbb{F}_q$ — in our case $p = 1/2$ for $\mathbb{F}_2$

$$\begin{aligned} \Pr\{|x| \le d\} &= 2^{-m} \sum_{d'=1}^{d} \binom{m}{d'}, \\ \mathbb{E}\left[\#\{x \in \mathcal{C} \setminus \{0\} : |x| \le d\}\right] &= 2^{k-m} \sum_{d'=1}^{d} \binom{m}{d'}, \end{aligned} \tag{2.6}$$

where we have neglected removal of zero code word.

There is a known bound [8] on sum of binomial coefficients, for $\alpha \in (0,1)$ and αm integer in terms of binary entropy function $H(x) = -x\log_2(x) - (1-x)\log_2(1-x)$ given using Stirling's formula as

$$\frac{1}{\sqrt{8m\alpha(1-\alpha)}} \cdot 2^{H(\alpha)\cdot m} \le \sum_{d'=1}^{d} \binom{m}{d'} \le \frac{1}{\sqrt{2\pi m\alpha(1-\alpha)}} \cdot 2^{H(\alpha)\cdot m}. \tag{2.7}$$

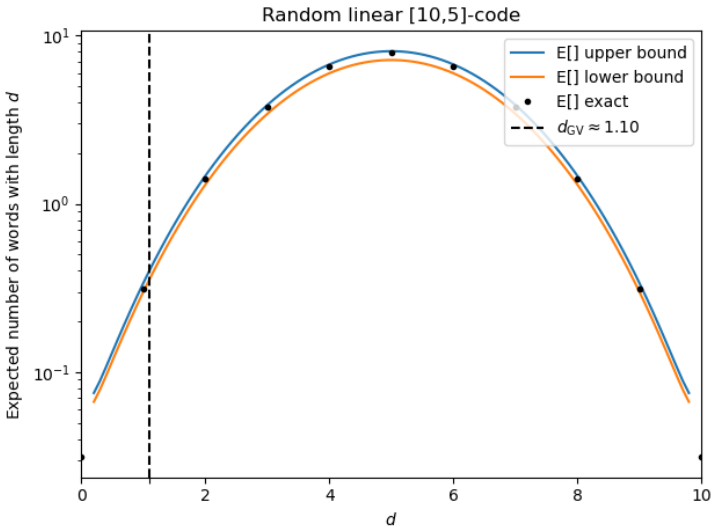

Figure 2.1: Example of a linear $[10, 5]$ code and its expected of number of codewords with weight less than d. We give the Gilbert-Varshamov distance, exact expectation in terms of the binomial coefficient and also bounds as in proof of Theorem 2.9. The bounds are tight.

As we would like to reason about the behaviour of the expected value based on m, we need to obtain uniform bounds in α. Due to the relations $\alpha m \in \mathbb{N}$ and $0 < \alpha < 1$, we can bound the function denominators. As $1/m \leq \alpha \leq 1 - 1/m$, then

$$\frac{1}{\sqrt{8}} 2^{H(\alpha)m} \leq \frac{1}{\sqrt{8\,(1-1/m)}} 2^{H(\alpha)m} \leq \sum_{d'=1}^{d} \binom{m}{d'} \leq \frac{m}{\sqrt{2\pi}} 2^{H(\alpha)m}. \tag{2.8}$$

Now, for the expectation value we have for $\frac{1}{\sqrt{8}} \leq f(m) \leq \frac{m}{\sqrt{2\pi}}$,

$$2^{-m+k} \sum_{d'=1}^{d} \binom{m}{d'} = 2^{m\left(\frac{k}{m} - 1 + H(\alpha)\right)} f(m) \tag{2.9}$$

and the asymptotic behaviour as $m \to \infty$ now depends solely on the sign of the exponent. As $H(\alpha)$ is symmetric around $\alpha = 1/2$, let $\hat{\alpha} \in (0, 1/2]$, d_{GV} be such as defined in the statement of the theorem. □

Remark 2.10. The meaning of Theorem 2.9 is that there exists a special distance $d_{\mathrm{GV}}(k, m)$ such that for random codes and large m, it is exponentially unlikely that there are any *shorter* codewords and exponentially likely that there are *longer* codewords. It is a good estimate on the minimal distance of a code and is used frequently for evaluation of quality of algorithm finding the shortest vector in a code. For more details, see *Decoding Challenge* [5]. Visualisation of the theorem can be found in Figures 2.1 and 2.2.

Typically, the *Gilbert-Varshamov bound* is invoked in a different context — given a code length and minimal distance, what is the maximal dimension k such that there exists a code with that dimension? Nevertheless, the maximal k is computed from the identical relation (2.5) as here, except k is unknown and d fixed.

The derivation given here is known and mostly referenced in [11], although we make sure that the derivation is rigorous by using the bounds in the denominator independent on α in the proof.

To better understand the situation, let us introduce our typical values of parameters. In the case of fully diffused ciphers, $m = 2^n$ and hence we expect $\frac{k}{m} = \frac{R \cdot n}{2^n}$ to

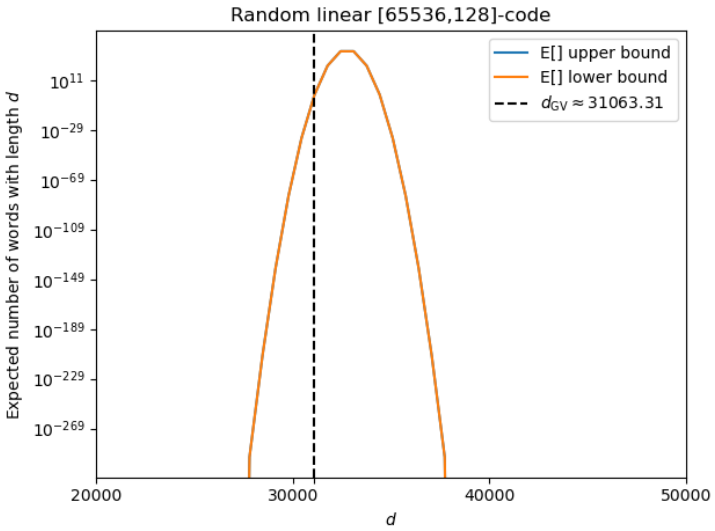

Figure 2.2: Example of the linear code corresponding to the linear span of polynomials for cipher $SR(3,2,2,4)$ with $n = 16$ key bits. Each polynomial is encoded as a vector of its polynomials. Here, $m = 2^{16}$ and we have chosen $k = n$ polynomials. In addition, $E[d \leq d_{\mathrm{GV}}] = 3 \cdot 10^{-3}$. Exact expectation was not directly computable due to operations with too large numbers in the binomial coefficient. However the lower and upper bounds are indistinguishable.

be small and thus $H(\alpha) \approx 1$ for large n. Here, R denotes the number of ciphertext pairs. Further, we present our estimate in that case.

Proposition 2.11. *For $k/m \to 0$, we have*

$$d_{GV}(m,k) = m\left(\frac{1}{2} - \sqrt{\frac{k}{m}\frac{\ln 2}{2}} + \mathcal{O}\left(\frac{k}{m}\right)\right). \tag{2.10}$$

The approximation of (2.10) without the asymptotic term is a rather accurate even for larger values of $\frac{k}{m}$. The relative error monotonically increases and depends only on the ratio k/m. For $\frac{k}{m} = \frac{1}{4}$, the estimate is less than 5 % smaller than the precise value $d_{GV}(m,k)$.

Proof. Let us expand $H(\alpha)$ to Taylor series around $\alpha = \frac{1}{2}$,

$$H(\alpha) = 1 - \frac{2\left(\alpha - \frac{1}{2}\right)^2}{\ln 2} + \mathcal{O}\left((\alpha - 1/2)^4\right). \tag{2.11}$$

By inverting this relation and solving for $\alpha \in (0, \frac{1}{2}]$,

$$\alpha - \frac{1}{2} = -\sqrt{\frac{k}{m}\frac{\ln 2}{2} + \mathcal{O}\left((\alpha - 1/2)^4\right)} \approx -\sqrt{\frac{k}{m}\frac{\ln 2}{2}}. \tag{2.12}$$

By omitting the small term, we obtain an error proportional to $|\hat{\alpha} - \alpha| \propto \frac{k}{m}$. Ultimately, we obtain

$$d_{GV}(m,k) \approx m\left(\frac{1}{2} - \sqrt{\frac{k}{m}\frac{\ln 2}{2}}\right) \tag{2.13}$$

with the error in the approximation of the upper bound proportional to k. The monotonicity of the error term is trivial to show and concrete error values can be trivially computed numerically. □

Note 2.12. As we are more interested in the average number of non-zero monomials in r polynomials, the estimate on minimal d such that $\#\{c \in \mathcal{C} \setminus \{0\} : |c| \leq d\} \geq r$ would be more relevant. However, as the number of codewords with weight higher than Gilbert-Varshamov distance is growing exponentially, it is expected to be very close to it for relevant values of r. Nevertheless, the estimate should be directly obtainable by a similar technique as in Proposition 2.11.

Here, we present the notion of (weighted) Hamming norm which will be needed in further chapters. The proofs can be easily done without reference.

Definition 2.13 (Weighted Hamming norm)**.** Let $n \in \mathbb{N}$ and $w = (w_1, w_2, \ldots, w_n)$ be a finite sequence of positive real numbers, $w_i > 0$ for all $i \leq n$. Such w is called an *n-weight* or simply a *weight* if $n \in \mathbb{N}$ is clear from context. Then the map $|\cdot|_w : \mathbb{F}_2^n \to \mathbb{R}$,

$$|x|_w := \sum_{i=1}^{n} w_i x_i, \quad \forall x \in \mathbb{F}_2^n \tag{2.14}$$

where the components are considered as real numbers and the summation is in $\mathbb{R}$, is called a *weighted Hamming norm* of weight w.

Lemma 2.14. *Weighted Hamming norm $|\cdot|_w$ satisfies relation*

$$\forall x, y \in \mathbb{F}_2^n,\ |x+y|_w = |x|_w + |y|_w - 2|x \wedge y|_w. \tag{2.15}$$

Proof. Remember, that addition in $\mathbb{F}_2$ is given by $x +_{\mathbb{F}_2} y := x + y \mod 2$ or equivalently, via the exclusive-or operation $\oplus$. The following holds when the truth values $\{0,1\}$ and corresponding logical operations are realised in real numbers using usual operations in $\mathbb{R}$ as a Boolean algebra,

$$\forall x, y \in \{0,1\},\ x \oplus y = x + y - 2xy \qquad \text{in } \mathbb{R} \tag{2.16}$$

as can be easily verified via enumeration. Hence, returning to the original relation,

$$\begin{aligned} |x+y|_w \equiv |x \oplus y|_w &= \sum_{i=1}^{n} w_i\,(x_i \oplus y_i) = \sum_{i=1}^{n} w_i\,(x_i + y_i - 2x_i y_i) \\ &= \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} w_i y_i - 2\sum_{i=1}^{n} w_i (x_i y_i) \\ &= |x|_w + |y|_w - 2|x \wedge y|_w \end{aligned} \tag{2.17}$$

where the summands in sums are considered as real numbers. □

Corollary 2.15. *Weighted Hamming norm $|\cdot|_w : \mathbb{F}_2^n \to \mathbb{R}$ is a norm — in a sense it satisfies*

1. *(positive-definiteness):* $\forall x \in \mathbb{F}_2^n,\ |x|_w \geq 0$ *and* $\forall x \in \mathbb{F}_2^n,\ |x|_w = 0 \iff x = 0$,
2. *(absolute homogeneity):* $\forall x \in \mathbb{F}_2^n,\ \alpha \in \mathbb{F}_2,\ |\alpha x|_w = |\alpha||x|_w$,
3. *(triangle inequality):* $\forall x, y \in \mathbb{F}_2^n,\ |x+y|_w \leq |x|_w + |y|_w$.

Proof. Fix a weight $w = (w_1, \ldots, w_n)$. As $w_i > 0$ for any index $i \leq n$, the first two properties are trivial. The triangle inequality follows from Lemma 2.14. □

Remark 2.16. Lemma 2.14 can be considered an analogy of polarization identity for real inner-product vector spaces $\mathcal{V}$ with inner product $\langle \cdot, \cdot \rangle : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ and the corresponding norm $\|x\|^2 = \langle x, x \rangle$, $\forall x \in \mathcal{V}$. The polarization identity is then given as

$$\|x - y\|^2 = \|x\|^2 + \|y\|^2 - 2\langle x, y \rangle, \qquad \forall x, y \in \mathcal{V}. \tag{2.18}$$

Even though the (weighted) Hamming norm can be considered a norm in the precise sense given in Corollary 2.15, the tuple $(\mathbb{F}_2^n, |\cdot|)$ is not a normed vector space as that would require $|\cdot| : \mathbb{F}_2^n \to \mathbb{F}_2$. The field $\mathbb{F}_2$ cannot be ordered in a way which preserves addition and multiplication and this makes the concept of scalar product in finite fields rather not useful.

Chapter 3

LLL reduction and similar algorithms

We have already mentioned the Lenstra–Lenstra–Lovász (LLL) algorithm [36] briefly in the previous chapter on lattices. It is an algorithm which takes as input a basis of a lattice and outputs another lattice which provides approximate shortest vectors of a lattice, the notion of which we will state more precisely later in this chapter, in polynomial time. The algorithm works by iteratively reducing the problem of finding short vectors in a lattice to finding vectors in a two-dimensional sub-lattice — a problem that is easily exactly solvable. By composing the individual short vectors from the sub-lattice, we arrive at a set of vectors which are not necessarily the shortest in the large lattice but at least they provide a good approximation. Note that finding the shortest vector in a lattice or a linear code is NP-hard problem in general.

Furthermore, generalizations of the LLL algorithm provide the most efficient methods for finding short, non-zero lattice vectors across various parameter regimes, including those crucial for cryptography [37].

Lattice basis-reduction technique are not limited to the LLL algorithm. There are other, more powerful algorithm, which produce better[1] bases, although they do no longer run in strictly polynomial time. A notable example is the Block Korkine-Zolotarev (BKZ) basis reduction algorithm, originally proposed by Schnorr [45], which extends the LLL algorithm by operating on larger blocks. The BKZ algorithm iteratively processes sub-lattices of dimension β. Here, the parameter $\beta \geq 2$ represents the block size, with $\beta = 2$ corresponding to the LLL algorithm (neglecting minor technical distinctions). Increasing the block size β results in improved bases consisting of shorter lattice vectors.

However, employing the BKZ algorithm with block size β necessitates finding the shortest non-zero vector in a β-dimensional lattice. This operation requires a running time of $2^{O(\beta)}$ using the best-known algorithms [4, 1, 12]. Consequently, BKZ offers a trade-off between the quality of the reduced basis and the algorithm's computational runtime.

Both of these algorithms and more have recently been studied in the context of linear codes [24, 31] and provide a new, rapidly expanding area of research.

First, we will quickly highlight the features of LLL for lattices and then we more thoroughly examine the situation for linear codes as that is the main interest for this thesis.

[1] "Good" bases are those which allow to better solve SVP and CVP. It can be shown that the basis $(b_1, \ldots, b_k)$ is good if its profile consisting of norms of its Gram-Schmidt vector $\left(\|b_1^*\|_2, \ldots, \|b_k^*\|_2\right)$ is approximately constant.

3.1 In lattices

In this work, we focus on *basis reduction*, a fundamental algorithmic framework in lattice theory. At a high level, given a basis $A := (a_1, \ldots, a_k)$ for a lattice $\mathcal{L}$, basis reduction algorithms aim to compute a "good" basis of $\mathcal{L}$. In particular, they strive to produce a basis where the first vector a_1 is short. This is achieved by making iterative "local changes" to the basis. However, we do not focus on the theory of lattices more than necessary to highlight the similarities with LLL for linear codes. For more details, see reference [17].

We begin by recalling the standard definition of the Gram-Schmidt Orthogonalisation (GSO) process over Euclidean vector spaces. Consider a basis $(b_1, \ldots, b_n)$ of the Euclidean space $(\mathbb{R}^n, \langle \cdot, \cdot \rangle)$. Its Gram-Schmidt orthogonalisation $(b_1^*, \ldots, b_n^*)$ is defined inductively as follows:

Definition 3.1. Let $b_1, \ldots, b_k$ be a basis of $\mathbb{R}^n$. The Gram-Schmidt Orthogonalisation (GSO) of $b_1, \ldots, b_k$ is defined as the basis $b_1^*, \ldots, b_k^*$ constructed as follows:

$$\begin{aligned} b_1^* &= b_1, \\ b_i^* &= \pi_i^{\perp}(x_i) = b_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^* \quad (2 \leq i \leq k), \end{aligned} \tag{3.1}$$

where the coefficients μ_{ij} are given by:

$$\mu_{ij} = \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} \quad (1 \leq j < i \leq k). \tag{3.2}$$

Here, $\pi_i^{\perp}$ denotes the orthogonal projection onto the orthogonal complement of the subspace spanned by the vectors $\{b_1, \ldots, b_{i-1}\}$ in $\mathbb{R}^n$.

Although the GSO of a lattice basis is not itself a basis of the lattice, it plays a fundamental role in the theory of lattice reduction and forms the backbone of many lattice reduction algorithms. Each Gram-Schmidt vector b_i^* can be interpreted as a lattice vector in a lower-dimensional lattice generated by the projected block

$$A[i, j] := \pi_{\{b_1, \ldots, b_{i-1}\}}^{\perp}(b_i, \ldots, b_j). \tag{3.3}$$

Basis reduction algorithms iteratively perform local changes to A, specifically modifying blocks $A[i, j]$ in a way that preserves the lattice $\mathcal{L}(A[i, j])$. The primary goal of these changes is to shorten the earlier Gram-Schmidt vectors. Notably, the first Gram-Schmidt vector $a_1^* = a_1$ is itself a non-zero lattice vector. Therefore, if a_1^* can be made short, a short non-zero lattice vector is found.

This process often involves identifying a short non-zero vector $y \in \mathcal{L}(A[i, j])$ and replacing the first vector in the block with y. (For simplicity, we omit the details of how this replacement is implemented.)

The basis reduction paradigm was introduced in the seminal work of Lenstra, Lenstra, and Lovász [36], which proposed the polynomial-time *LLL algorithm*. In essence (ignoring technical details not central to this discussion), the LLL algorithm iteratively replaces Gram-Schmidt vectors a_i^* with the shortest non-zero vector in the dimension-two lattice generated by the block $A[i, i+1]$. The LLL algorithm has numerous applications and generalizations of the algorithm provide the most efficient methods for finding short non-zero lattice vectors in various parameter settings, including those relevant to cryptography.

Definition 3.2. A lattice basis $b_1, \ldots, b_k$ is size-reduced if

$$|\langle b_i, b_j^* \rangle| \leq \frac{\|b_j^*\|^2}{2}, \quad 1 \leq j < i \leq k, \tag{3.4}$$

or equivalently

$$|\mu_{ij}| \leq 1/2, \quad 1 \leq j < i \leq k. \tag{3.5}$$

Definition 3.3. A lattice basis $b_1, \ldots, b_k$ is LLL-reduced with coefficient $\alpha \in (1/4, 1)$ if it satisfies

$$\begin{aligned} |\mu_{ij}| &\leq \frac{1}{2}, \quad 1 \leq j < i < k, \\ |b_i^* + \mu_{i,i-1} b_{i-1}^*|^2 &\geq \alpha |x_{i-1}^*|^2, \quad 1 < i \leq k. \end{aligned} \tag{3.6}$$

First condition is the size-reduction and the second one is known as *exchange condition*. It can be rewritten as

$$|b_i^*|^2 \geq \left(\alpha - \mu_{i,i-1}^2\right) |x_{i-1}^*|^2, \quad 2 \leq i \leq k. \tag{3.7}$$

Theorem 3.4 (LLL-theorem)**.** *If $b_1, \ldots, b_k$ is an α LLL-reduced basis of a lattice $\mathcal{L}$ in $\mathbb{Q}^n$, and $y \in \mathcal{L}$ is any non-zero lattice vector, then*

$$|b_1| \leq \beta^{(n-1)/2} |y| \tag{3.8}$$

where parameter β is derived from α using

$$\beta = \frac{4}{4\alpha - 1}. \tag{3.9}$$

Note 3.5. Theorem 3.4 proves that the first vector of a LLL-reduced produces (an exponential) approximation to the shortest vector problem. It can be shown that obtaining better, polynomial-factor approximation, is NP-hard problem.

Finally, we present a simplified version of the LLL algorithm for $\alpha = 3/4$ in Algorithm 1 described in study material [47]. It can be proven that it terminates and produces a LLL-reduced basis using a potential defined via Gramm determinants and we leave it here for comparison with LLL for linear codes.

Algorithm 1 LLL algorithm for lattices

Input: basis $b_1, \ldots, b_k$ of lattice $\mathcal{L} \subset \mathbb{Q}^m$
Output: LLL-reduced basis of lattice $\mathcal{L}$

$(b_i^*)_{i=1}^k, (\mu_{ij})_{i,j=1}^k \leftarrow$ GramSchmidt$(b_1, \ldots, b_k)$
for $i = 2, \ldots, k$ **do**
  for $j = i-1, \ldots, 1$ **do** ▷ Size-reduction
    $x \leftarrow \lfloor \mu_{ij} \rceil$
    $b_i \leftarrow b_i - x b_j$
    $\mu_{ij} \leftarrow \mu_{ij} - x$
    for $l = 1, \ldots, j-1$ **do** $\mu_{il} \leftarrow \mu_{il} - x\mu_{jl}$
for $i = 2, \ldots, k$ **do**
  if $\|b_i^*\|^2 < \left(\frac{3}{4} - \mu_{i\ i-1}^2\right) \|b_{i-1}^*\|^2$ **then**
    $b_i \leftrightarrows b_{i-1}$ ▷ Swap
    goto 1
return $b_1, \ldots, b_k$

Definition 3.6. For a basis $(b_1, \ldots, b_k)$, define a potential D as

$$D = \prod_{i=1}^{k} \prod_{j=1}^{i} \|b_j^*\|^2. \tag{3.10}$$

Theorem 3.7. *For a basis $(b_1, \ldots, b_k)$ of a lattice $\mathcal{L}$ and fixed value of parameter α, the LLL Algorithm 1 has complexity*

$$\mathcal{O}(k^6 \log(\max\{\|b_1\|, \ldots, \|b_k\|)^3). \tag{3.11}$$

Proof. Can be found in [47]. □

Note 3.8. There are more optimized versions of LLL algorithm which might require fewer operations. There is still very active research in theoretical guarantees of this type of algorithm as the heuristic and experimental results give both shorter vectors than expected and in shorter time than the current theoretical understanding.

3.2 In linear codes

The recent (2022) adoption of LLL algorithm for codes [24] is currently the best implementation for finding the shortest codewords in certain linear codes which does not use GPU, according to *Decoding Challenge* [5]. It is a novel algorithmic version of the Griesmer's bound [32] for linear codes. Here, we present the adoption of LLL algorithm for lattices into the domain of linear codes in a brief manner and should the reader need more details, the original article can be consulted.

Scalar product in $\mathbb{F}_2$ To continue the analogy between lattices and codes, we could strive to define GSO for codes in terms of a scalar product on $\mathbb{F}_2^m$. However, the field $\mathbb{F}_2$ cannot be equipped with order $\prec$ which would be "well-behaved" in a sense that $a < b$ would imply $a + c \prec b + c$ for all $a, b, c \in \mathbb{F}_2$. In turn, any scalar product $\langle \cdot, \cdot \rangle : \mathbb{F}_2^m \times \mathbb{F}_2^m \to \mathbb{F}_2^m$ does not lead to a meaningful geometric notion of orthogonality. However, we can see a hint on the analogy in Lemma 2.14 and Remark 2.16.

Further on, logical operations on vectors in $\mathbb{F}_2^m$ are applied component-wise. Especially in this chapter, we will denote the addition (XOR) in $\mathbb{F}_2^m$ as $\oplus$ and multiplication (AND) as $\wedge$ to highlight the differences to lattices.

Definition 3.9. Define the support of $x = (x_1, \ldots, x_m) \in \mathbb{F}_2^m$ as the non-zero positions

$$\mathsf{Supp}(x) = \{i : x_i \neq 0\}. \tag{3.12}$$

Definition 3.10. Define $x, y \in \mathbb{F}_2^m$ to be *orthopodal* if their supports are disjoint, that is, defining the orthogonality relation as:

$$x \perp y : \iff x \wedge y = 0. \tag{3.13}$$

Additionally, we can then associate convenient notions of projections onto (the support of) x and orthopodally to (the support of) x as follows:

$$\pi_x : y \mapsto y \wedge x, \quad \pi_x^\perp : y \mapsto y \wedge \overline{x} = \pi_{\overline{x}}(y) \tag{3.14}$$

where $\overline{x}$ denotes the negation of x.

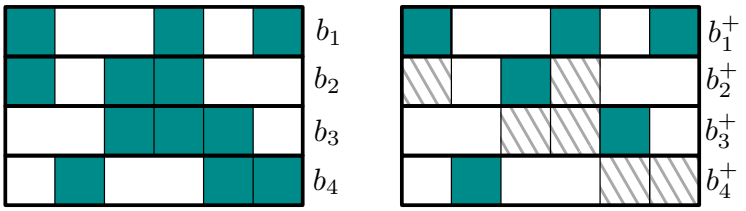

Figure 3.1: Generator matrix $(b_1; b_2; b_3; b_4)$ of a linear binary $[6, 4]$ code and its epipodal matrix. Areas shaded in white or gray represent zeros, colored parts are ones.

Lemma 3.11. *For any $x, y, z \in \mathbb{F}_2^m$ it holds that:*

$$\begin{aligned} \pi_x(y \oplus z) &= \pi_x(y) \oplus \pi_x(z), \\ y = \pi_x(y) \oplus \pi_x^\perp(y), \quad \pi_x^2(y) &= \pi_x(x), \\ \pi_x(x) = x, \quad |\pi_x(y)| &\leq |y| \end{aligned} \tag{3.15}$$

and so π_x is a projection operator for any $x \in \mathbb{F}_2^m$.

Definition 3.12. Let $B = (b_1; \ldots; b_k) \in \mathbb{F}_2^{k \times m}$ be a binary matrix, $B_{ij} = (b_i)_j$. The i-th projection associated to this matrix is defined as $\pi_i := \pi^\perp_{\{b_1, \ldots, b_{i-1}\}}$ where π_1 denotes the identity. Equivalently,

$$\begin{aligned} \pi_i : \quad \mathbb{F}_2^m &\longrightarrow \mathbb{F}_2^m, \\ c &\longmapsto c \wedge \overline{(b_1 \vee \cdots \vee b_{i-1})}. \end{aligned} \tag{3.16}$$

The i-th *epipodal vector* is then defined as

$$b_i^+ := \pi_i(b_i) \tag{3.17}$$

and the matrix $B^+ := (b_1^+; \ldots; b_k^+)$ is called the *epipodal matrix* of B.

Now, in order to define analogue of Gram-Schmidt process for linear codes, notice that the Gram-Schmidt preserves $\mathsf{Span}_{\mathbb{R}}(b_1, \ldots, b_k) = \mathsf{Span}_{\mathbb{R}}(b_1^*, \ldots, b_k^*)$. The orthopodal projection does not preserve $\mathbb{F}_2$-spans, and one might think that the analogy is not possible. However, the lattice analogue of $\mathbb{F}_2$-span is $\mathbb{Z}$-span and the analogue of lattice's $\mathbb{R}$-span is code's support. Example of the construction of epipodal vectors is shown in Figure 3.1.

Lemma 3.13 (Properties of epipodal matrix)**.** *For any binary matrix B, $B = (b_1; \ldots \beta_k) \in \mathbb{F}_2^{k \times m}$, its epopodal matrix B^+ satisfies*

1. *The epipodal vectors are pairwise orthopodal,*

$$\forall i \neq j, \quad b_i^+ \perp b_j^+. \tag{3.18}$$

2. *For all $i \leq k$, $(b_1, \ldots, b_i)$ and $(b_1^+, \ldots, b_i^+)$ have the same supports. That is*

$$\bigcup_{j \leq i} \mathit{Supp}(b_j^+) = \bigcup_{j \leq i} \mathit{Supp}(b_j). \tag{3.19}$$

Proof. We rewrite $b_i^+ = \pi_i(b_i) = b_i \wedge \overline{(b_1 \vee \cdots \vee b_{i-1})}$ and so $\mathsf{Supp}(b_i^+) \subset \mathsf{Supp}(b_i)$ and $\mathsf{Supp}(b_i^+) \cap \mathsf{Supp}(b_j) = \emptyset$ for any $j < i$. Hence, $\mathsf{Supp}(b_i^+) \cap \mathsf{Supp}(b_j^+) = \emptyset$, or equivalently $b_i^+ \perp b_j^+$ for any $j < i$. The second part is equivalent to $\bigvee_{j \leq i} b_j^+ = \bigvee_{j \leq i} b_j$

and by rewriting $b_j^+ = b_j \wedge \bigwedge_{i<j}$, the induction is straightforward. For $i = 1$, $b_1^+ = b_1$ and now assume that the statement hold up-to $i - 1$. Then

$$\begin{aligned} b_i^+ \vee b_{i-1}^+ \vee \cdots \vee b_1^+ &= b_i^+ \vee b_{i-1} \vee \cdots \vee b_1 \\ &= \left(b_i \wedge \overline{(b_{i-1} \vee \cdots \vee b_1)}\right) \vee b_{i-1} \vee \cdots \vee b_1 \\ &= b_i \vee b_{i-1} \vee \cdots b_1. \end{aligned} \tag{3.20}$$

□

Note 3.14. Note the analogy between the induction for epipodal vectors obtained from the definition and the Gram-Schmidt orthogonalization process over $\mathbb{R}$.

- Epipodal vectors $(b_1^+, \ldots, b_k^+)$ satisfy:

$$b_i^+ = b_i + \sum_{j=1}^{i-1} b_i \wedge b_j^+. \tag{3.21}$$

- Gram-Schmidt vectors $(b_1^*, \ldots, b_k^*)$ satisfy:

$$b_i^* = b_i - \sum_{j=1}^{i-1} \frac{\langle b_i, b_j \rangle}{\langle b_j^*, b_j^* \rangle} b_j^*. \tag{3.22}$$

Unlike for the GSO, epipodal vectors can be zero. In order to avoid this problem, we suggest the following definition.

Definition 3.15. A basis $(b_1, \ldots, b_k)$ is proper if all its epipodal vectors b_i^+ are non-zero.

Remark 3.16. Every code has a proper basis and a proper basis can be created from a basis in polynomial time. As an example, basis in systematic form is proper and can be created from a basis via Gaussian elimination in $\mathcal{O}(k^2 m)$ steps.

Definition 3.17. A basis $(b_1, \ldots, b_k)$ is locally size-reduced if for all $i \in [\![0, k-1]\!]$,

$$|b_i^+ \wedge \pi_i(b_{i+1})| \leq \frac{|b_i^+|}{2}. \tag{3.23}$$

Definition 3.18. A lattice basis $b_1, \ldots, b_k$ is LLL-reduced if it is a proper basis and if b_i^+ is the shortest non-zero codeword of the projected sub-code $\pi_i\left(\mathcal{C}(b_i, b_{i+1})\right)$ for all $i \in [\![i, k-1]\!]$.

Note 3.19. The projected sub-code in LLL-reduction definition has only three non-zero words and can be enumerated as

$$\pi_i\left(\mathcal{C}(b_i, b_{i+1})\right) = \mathcal{C}\left(b_i^+, \pi_i(b_{i+1})\right) = \left\{0, b_i^+, \pi_i(b_{i+1}), b_i^+ + \pi_i(b_{i+1})\right\}. \tag{3.24}$$

Theorem 3.20 (Approximation factor bound)**.** *Let $B = (b_1; \ldots; b_k)$ be an LLL-reduced basis of code $\mathcal{C}$. Then we have,*

$$|b_1| \leq 2^{k-1} \cdot d_{min}\left(\mathcal{C}(B)\right). \tag{3.25}$$

Theorem 3.21 (Griesmer bound)**.** *Let $(b_1, \dots, b_k)$ be a LLL-reduced basis of a linear binary $[n, k]$-code $\mathcal{C}$. Denote $\ell_1 := |b_1|$ and then,*

$$n \geq \sum_{i=0}^{k-1} \left\lceil \frac{\ell_1}{2^i} \right\rceil \tag{3.26}$$

where $\ell_1 \geq d_{min}(\mathcal{C})$. In particular, if $k - 1 \geq \log_2(d)$, then it holds for $\ell_1 = |b_1|$

$$\ell_1 - \frac{\lceil \log_2 \ell_1 \rceil}{2} \leq \frac{m-k}{2} + 1. \tag{3.27}$$

Moreover, every code $\mathcal{C}$ admits a LLL-reduced basis.

Note 3.22. A Gaussian elimination on $(b_1, \dots, b_k)$ on a random linear binary code yields an estimate $\ell_1 \approx \frac{m-k}{2} + 1$ *on average*. In worst cases, it can be as large as the Singleton bound $m - k + 1$. The usefulness of the LLL-reduced basis lies in the better worst-case scenario.

In order to be able to define a tiling domain called *fundamental domain*, one encounters a problem; in lattices, the overlap between the fundamental domains (definition of which is not important to our thesis) is of zero measure. However, in finite $\mathbb{F}_2^m$, it is not of zero measure and hence, one needs to offset the domains so that their borders do not overlap in order to tile the space. We present the following technical definition.

Definition 3.23. We define a (non-canonical) tail-break function for $p, y \in \mathbb{F}_2^m$ as

$$\mathsf{TB}_p(y) = \begin{cases} 0, & \text{if } |p| \text{ is odd,} \\ 0, & \text{if } y_j = 0 \text{ where } j = \min \mathsf{Supp}(p), \\ 1/2, & \text{otherwise.} \end{cases} \tag{3.28}$$

Algorithm 2 LLL for linear codes over $\mathbb{F}_2$

Input: A proper basis $B = (b_1; \dots; b_k) \in \mathbb{F}_2^{k \times m}$ of a code $\mathcal{C}$
Output: An LLL-reduced basis for code $\mathcal{C}$

1: **while** $\exists i \in [\![0, k-1]\!]$ s.t.
$\min\left(\left|\pi_i(b_{i+1})\right|, \left|b_i^+ \oplus \pi_i(b_{i+1})\right|\right) < |b_i^+|$ **do** ▷ Ensure b_i^+ is shortest in $\pi_i\left(\mathcal{C}(b_i, b_{i+1})\right)$
2: **if** $|\pi_i(b_{i+1}) \wedge b_i^+| + \mathsf{TB}_{b_i^+}\left(\pi_i(b_{i+1})\right) > |b_i^+|/2$ **then**
3: $b_{i+1} \leftarrow b_{i+1} + b_i$ ▷ Local size-reduction
4: $b_i \leftrightarrows b_{i+1}$ ▷ Swap
return B

Definition 3.24. Let $B = (b_1; \dots, b_k)$ be a basis of an $[m, k]$-code. The potential of B, denoted as $\mathcal{D}_B$ is defined as

$$\mathcal{D}_B := \sum_{i=1} \mathcal{D}_{B,i} = \sum_{i=1}^{k} (k - i + 1)|b_i^+|, \tag{3.29}$$

where $\mathcal{D}_{B,i} = \#\mathsf{Supp}(\mathcal{C}(b_1, \dots, b_i)) = \sum_{j=1}^{i} |b_i^+|$.

Lemma 3.25. *If Algorithm 2 terminates then it outputs an LLL-reduced basis for the code spanned by the basis given as input.*

Proof. The exit condition ensures that the basis is LLL-reduced if it is proper. It can be shown that properness is a loop-invariant. □

Theorem 3.26. *Algorithm 2 terminates and for input* $(b_1; \ldots; b_k) \in \mathbb{F}_2^{k \times m}$*, it performs at most*

$$\frac{k(k+1)}{2} \max |b_i^+| \tag{3.30}$$

vector operations over $\mathbb{F}_2^m$ *where* $|b_i^+| \leq m$ *for all* $1 \leq i \leq k$*.*

Proof. The outline of the proof is the following. First, we show that the potential $\mathcal{D}_B$ decreases by at least one at each swap in the algorithm due to the step that ensures that the sub-code $\pi_i\left(\mathcal{C}(b_i, b_{i+1})\right)$ is size-reduced. That is, if a swap occurs between b_i and b_{i+1}, $\mathcal{D}_{B,i}$ denotes the original potential before the swap and $\mathcal{D}'_{B,i}$ after the swap, then

$$\mathcal{D}'_{B,i} < \mathcal{D}_{B,i}. \tag{3.31}$$

As the potential is an integer, the potential decreases by one at each swap and as it is positive, the number of iterations is limited by $\mathcal{D}_{B,0}$. □

LLL for weighted codes As the main properties of projection π_x for $x \in \mathbb{F}_2^m$ are also valid for weighted Hamming distance in Definition 2.13 as we have shown earlier, we suspect that the majority of the proofs in [24] are also valid for weighted Hamming norm. This would allow us to privilege elimination of higher monomials in a given monomial order as an interpolation between LLL with standard Hamming weight and elimination of monomials with the highest degrees. The main modification needed seems to be in Theorem 3.26 in order to show that the algorithm terminates. The proof relies on the fact that potentials $\mathcal{D}_{B,i}$ are integers[2]. So one would expect that the theorem remains true also for integer weights (w_i). We did not implement this technique as it requires nontrivial modification of the LLL library which is highly optimized. However, let us note the following peculiarity. By choosing appropriate (extreme) weights, we achieve that lower weight means eliminating higher monomials.

Proposition 3.27. *There exist weights* $w = (w_1, \ldots, w_m)$ *for* $\mathbb{F}_2^m$ *such that for all* $x, y \in \mathbb{F}_2^m$*,*

$$|x|_w \leq |y|_w \iff \min \mathit{Supp}(x) \leq \min \mathit{Supp}(y). \tag{3.32}$$

Proof. Choose $w_i = 2^i$ for all $1 \leq i \leq m$. The proposition follows from $\sum_{i=1}^{\ell} w_i < w_{\ell+1}$ for $1 \leq \ell \leq m$. □

[2] Although it should be possible to bound the final difference between the potentials by the non-zero difference between $|b_i^+|$ and the currently shortest codeword in $\pi_i\left(\mathcal{C}(b_i, b_{i+1})\right)$ to achieve a similar effect.

Chapter 4

Algebraic model of the AES cipher

The small-scale variants of AES, as described in [21], are a way to explore various attacks on the full 128-bit AES block cipher [23]. The small-scale variants $SR(n, r, c, e)$ feature four parameters — number of iterative rounds $n \in [\![1, 10]\!]$, the number of rows of the rectangular grid of the state r, the number of columns in the rectangular grid of the state c and the word size e. The full AES is equivalent to $SR(10, 4, 4, 8)$ with addition of a cryptographically unimportant last round. The number of key bits of each cipher is equal to rce.

We will not describe the details of AES as our technique of generating the corresponding polynomial equations modelling the cipher have been described elsewhere [19, 14, 13] and the design of preprocessing is irrespective of the precise structure of the equations and can be applied to any cipher. Nevertheless, the main features are a key schedule which derives a different key for each round based on the key of the previous round and the initial key and subsequently, the round key is applied to the state variables derived from plaintext in each round. Each round features both linear and non-linear elements in order to be resistant to linear and differential cryptanalysis.

However, AES was not designed to be resistant to algebraic cryptanalysis — in particular, the non-linear element (SBOX) in each round can be represented by a second-degree polynomial in a low number of variables. In fact, the whole cipher can be modelled as a system of linear and second-degree equations in the initial rce key variables, rce round keys for each round $[\![1, n]\!]$ and the rc state variable for each round. More exact statistics and attack estimates can be found in [21].

It has been shown in that solving the original system with both state and key variables can be problematic using Gröbner basis. However, we will adopt a technique from [19, 14] described in detail below. First, we eliminate all variables from the system except for the rce initial key variables and subsequently compute its Gröbner basis.

4.1 Related work

In [19], the authors propose to first eliminate the state variables from the polynomial system over $\mathbb{F}_2$, producing a system with only initial rce key variables. This essentially helps to compute the Gröbner basis by first marking the special key variables — indeed, the original key variables are variables with the property that guessing these variables completely solves the system without any ambiguity — the same cannot generally be said of the state variables. Furthermore, they propose an alternative (already known) meet-in-the-middle technique where the system is divided

into two parts — one for rounds $[\![1, n/2]\!]$ and a second one for rounds $[\![n/2+1, n]\!]$. By propagating the plaintext and ciphertext from both sides and applying lex elimination, the authors obtain a system with two sets of key variables, corresponding to $n/2$ and $n/2+1$, respectively. This allows them to generate and solve larger ciphers than with the original single set of key variables.

In [14, 13, 39], the authors use the approach of generating a system with the initial key-variables only and subsequently, apply preprocessing techniques to reduce the density of the system by various techniques. They have used prominently the PAM (Partitioning Around Medoids) and LSH (Locality Sensitive Hashing), both coming from machine learning, to decrease the average number of monomials in each polynomial of the system. Mainly, they have broken the first round of AES $SR(1, 4, 4, 8)$ and improved the solving time with preprocessing.

Another technique central to the very comprehensive work [50] is by modelling the polynomial system as a system of Multiple Right Hand Sides (MRHS) linear equations. It requires different design for each group of ciphers and requires some amount of key bits to be guessed. However, it gives very competitive results, probably the best we know of, for many small-scale variants of AES.

4.2 Strategy overview

First, we create the polynomials for small-scale AES system and add the field equations $x_i^2 - x_i = 0$ which is equivalent to working in $\mathcal{R}_n$. This is done by first fixing a key of length n bits. Then plaintext-ciphertext pair (PC pair) is computed using the key by choosing a plaintext and encrypting it with the fixed key. This process is repeated as many times as needed, always with the fixed key. The polynomials are then generated from the PC pairs without using the key — simulating a real known plaintext-ciphertext attack scenario.

These polynomials are at most quadratic and more information about their counts, number of variables and their degrees can be found in [21]. These polynomials contain the key variables as well as state variables.

As a next step, we eliminate all the state variables. This can be done when we obtain a Gröbner basis of the original system with respect to *lex* ordering due to Theorem 1.27. However, *degrevlex* order is more suitable and faster to compute and hence we first compute it in this preferred order and then convert to *lex* via the FGLM [27] algorithm. The initial *degrevlex* computation usually takes the majority of computation time.

Finally, we will explore various preprocessings, mainly one based on lattice-reduction algorithms, on the system of polynomials in key variables only. At last, we again compute its *degrevlex* Gröbner basis. In order to obtain the solution, another execution of FGLM is needed. However, here we are interested only in some Gröbner basis as we want to obtain the number of solutions and we can easily check if the original key is present by simple substitution.

In the rest of the thesis, $n = rce$ will denote the key length.

Chapter 5

Preprocessing

In previous chapters, we have highlighted a possible simplified framework for the notion of *linear preprocessing* as a basic building block for preprocessing a system of polynomials equations in such a way that the resulting Gröbner algorithm produces the Gröbner basis faster that with the original system. Here, we state our devised formalisation of the concept and its proven theoretical guarantees. First, the computation of Gröbner basis can be thought of as a map

$$\mathsf{ComputeGB} : \mathcal{P}\left(\mathbb{F}[x_1, \ldots, x_n]\right) \to \mathcal{P}\left(\mathbb{F}[x_1, \ldots, x_n]\right) \tag{5.1}$$

such that $I = (f_1, \ldots, f_k)$, $\mathsf{ComputeGB}(f_1, \ldots, f_k) \subset (f_1, \ldots, f_k)$ and the result $\mathsf{ComputeGB}(f_1, \ldots, f_k)$ is a Gröbner basis of I. Subsequently, we present the following formalisation of linear preprocessing.

Definition 5.1 (Linear preprocessing)**.** Let K be a field. Let $K[x_1, \ldots, x_n]$ be a polynomial ring in n variables and coefficients in K which naturally forms a vector space over K. A map $\phi : \mathcal{P}(K[x_1, \ldots, x_n]) \to \mathcal{P}(K[x_1, \ldots, x_n])$ is called *linear preprocessing* if $\forall \mathcal{F} \subset \mathcal{P}\left(K[x_1, \ldots, x_n]\right)$,

$$\phi(\mathcal{F}) \subset \mathsf{Span}_K(\mathcal{F}). \tag{5.2}$$

In other words, let $d := \dim \mathsf{Span}(\mathcal{F})$ and $\mathcal{F} = \{f_1, \ldots, f_d\}$. Then for any $g \in \phi(\mathcal{F})$, there are $\alpha_1, \ldots, \alpha_d \in K$ such that $g = \sum_{j=1}^{d} \alpha_j f_j$. The definitions hold analogously for $\mathcal{R}_n^p$.

Example 5.2. The preprocessing techniques introduced in previous work [14, 13, 39] such as adding polynomials with similar monomials, PAM, LSH and RMSM are all examples of linear preprocessing.

Proposition 5.3. *Let ϕ be linear preprocessing. The solutions of polynomial system represented as an ideal $I = (f_1, \ldots, f_k)$, $I \subset K[x_1, \ldots, x_n]$ are still solutions after linear preprocessing in $I' = \phi(I) = (g_1, \ldots, g_l)$, i.e. the following holds for their varieties,*

$$V\left(I\right) \subset V\left(\phi(I)\right). \tag{5.3}$$

In particular, if $\mathsf{Span}_K\left\{f_1, \ldots, f_k\right\} = \mathsf{Span}_K\left\{g_1, \ldots, g_l\right\}$, then

$$V\left(I\right) = V\left(\phi(I)\right). \tag{5.4}$$

Proof. The first part follows more generally from $J \subset I \implies V(J) \supset V(I)$ for $J = I'$. The second part holds as when the spans of $\{f_1, \ldots, f_k\}$ and $\{g_1, \ldots, g_l\}$ are equal and without loss of generality $l \geq k$, then there exist $\alpha_{ij}, \beta_{ij} \in \mathbb{F}_2$ such that

$g_i = \sum_{j=1}^{k} \alpha_{ij} f_j$ and also $f_i = \sum_{j=1}^{l} \beta_{ij} g_i$ for all $i \leq k$. This proves $(f_1, \ldots, f_k) = (g_1, \ldots, g_k)$ as any $f = \sum_i r_i f_i \in I$, $r_i \in I$ can be written as

$$f = \sum_i r_i \sum_j \beta_{ij} g_i \in I'. \tag{5.5}$$

□

For illustration of our upcoming sections, we will try to construct preprocessing ϕ of the polynomial system such that the preprocessing ϕ decreases an objective function ρ.

Definition 5.4 (ρ-non-expanding)**.** Let M be a set and $\rho : \mathcal{P}(M) \to \mathbb{R}$ a function on subsets of M. A map $\phi : M \to M$ is said to be *ρ-non-expanding* if $\rho\left(\phi(M)\right) \leq \rho(M)$.

Definition 5.5 (ρ-contracting)**.** Let M be a set and $\rho : \mathcal{P}(M) \to \mathbb{R}$ a function on subsets of M. A map $\phi : M \to M$ is said to be *ρ-contracting* if $\rho\left(\phi(M)\right) < \rho(M)$.

We explore various objective functions ρ, ρ being either maximal degree of the resulting polynomials, number of nonzero monomials and others.

Let $\mathcal{R}_n := \mathbb{F}_2[x_1, \ldots, x_n] / \left(x_1^2 - x_i, \ldots, x_n^2 - x_n\right)$ be boolean polynomials in n variables and fix a monomial ordering $\prec$. There is an induced ordering $\prec'$ on multi-indices such that $x^\alpha \prec x^\beta$ iff $\alpha \prec' \beta$. We will usually neglect the different notations and denote both simply as $\prec$.

Now, every two monomials can be compared and there are $\sum_{i=0}^{n} \binom{i}{k} = 2^n$ distinct monomials in $\mathsf{Monom}(\mathcal{R}_n)$. As $\prec$ is a total ordering, we can canonically number the monomials or multi-indices by numbers between 1 and 2^n. By α_i, we will denote the i-th monomial multi-index.

More importantly, there is a canonical $\mathbb{F}_2$–vector space isomorphism $\mathsf{Coeff}_\prec : \mathcal{R}_n \to \mathbb{F}_2^{2^n}$ simply outputting polynomial coefficients with respect to $\prec$ such that

$$\mathsf{Coeff}_\prec \left(\sum_{i=1}^{2^n} a_i x^{\alpha_i} \right)_j = a_j, \quad \forall j \in [\![1, 2^n]\!]. \tag{5.6}$$

When ordering $\prec$ is known from context, then it is implicitly assumed. When $m \ll 2^n$ then in order to achieve efficient implementation, we usually restrict the $\mathsf{Coeff}_\prec$ map to output a lower-dimensional vector, omitting some or all of the zero coefficients. This is typically to construct a matrix $\mathbb{F}_2^{k \times m_{all}}$ out of the coefficients of polynomials $f_1, \ldots, f_k$. Here, $m_{all} = \#\mathsf{Monom}(f_1, \ldots, f_k)$ and we would want no columns which are all zero in the matrix.

5.1 LLL

We will use a variety of algorithms, mainly LLL with additional methods, mentioned in the previous chapters and the paper [24]. By using this preprocessing, we strive to minimize the average number of monomials in the polynomials which could help the F_4 algorithm to decrease solving time. This preprocessing performs the following.

- First, create a matrix from the polynomial system using the aforementioned function Coeff (here, the ordering does not matter).
- Subsequently, make sure that it contains no linearly dependent rows. If there are linearly dependent rows, they are removed by performing row reduction.

- A variety of sub-methods is applied to the original basis. For each method, a metric ρ is computed and basis with the lowest value of the metric is chosen. In our tests, we have chosen to minimize the number of non-zero monomials and the metric ρ for matrix $B = (b_1; \ldots; b_k) \in \mathbb{F}_2^{k \times m}$ is given according to

$$\rho(B) = \sum_{i=1}^{k} \sum_{j=1}^{m} (b_i)_j. \tag{5.7}$$

  The number of rows of resulting matrix in each method is the same and hence, it is equivalent to minimizing the average number of non-zero monomials in a polynomial.

- Finally, the rows of the matrix are sorted according to their weight and the first r polynomials are returned, r being a parameter. Alternatively, a LLL algorithm for linearly dependent input vectors from lattices can be adapted for linear codes [17, Section 6.1].

Remark 5.6. It should be noted that following this metric ρ too closely might be problematic when non-linear preprocessing is taken into account. For a cipher with n bits, one can easily construct a system with a single monomial — the highest monomial with degree n by simply multiplying all the polynomials by this monomial. However, this system has $2^n - 1$ solutions.

Here, we note the used sub-methods. Note, that due to inclusion of identity, this preprocessing is ρ-non-expanding. Additional steps of the following sub-methods are described in the original article [24] and are only briefly mentioned here:

1. Identity: for input codewords $\{f_i\}_{i=1}^k$ output $\{f_i\}_{i=1}^k$.
2. Systematize: Perform Gaussian elimination according to random information set; pivots are chosen at random.
3. LLL: Algorithm 2.
4. Systematize → LLL: systematization ensures that the input base is proper and the LLL algorithm has the proven qualities, mainly it outputs a LLL-reduced basis.
5. Systematize → EpiSort → LLL: EpiSort greedily optimizes the epipodal length of basis vectors.
6. Systematize → EpiSort → LLL → SizeRedBasis: SizeRedBasis performs a global size-reduction on the basis vectors.
7. Systematize → EpiSort → LLL → SizeRedBasis → KillTwos: KillTwos attempts to flatten the resulting epipodal profile[3] in order to make the profile more balanced.

Finally, we have decided not to implement finding the r smallest codewords using LeeBrickelBabai algorithm of the original paper. Supposedly, it would find a much better shortest vector but we are interested not in a single codeword but r linearly independent codewords. As LeeBrickelBabai iteratively finds shorter and shorter vectors, the problem could be remedied by remembering the history of the shortest

[3] Profile of a basis $(b_1, \ldots, b_k)$ of a linear code is equal to $\left(|b_1^+|, \ldots, |b_k^+|\right)$ and for many applications it is advantageous when all $|b_i^+|$ are approximately equal.

vectors in a queue. However, for diffused ciphers (which are the main concern when breaking large-round AES variants), the estimates on the minimal distance using Gilbert-Varshamov estimate is expected to be near $m/2 = 2^{n-1}$ and the difference in using LeeBrickelBabai instead of the methods above is small under assumption of pseudo-random ciphers.

5.2 Highest monomial elimination (HME)

Definition 5.7. Let $\mathcal{F} \subset \mathcal{R}_n$ be a set of boolean polynomials and $\prec$ a monomial ordering. Subset $\mathcal{I} \subset \{1, \ldots, 2^n\}$ is called *monomial indexing*. Define the *zero-monomial subspace* $\mathcal{F}_0^{\mathcal{I}}$ of $\mathsf{Span}_{\mathbb{F}_2}(\mathcal{F})$ as

$$\mathcal{F}_0^{\mathcal{I}} := \left\{ p \in \mathsf{Span}_{\mathbb{F}_2}(\mathcal{F}) : \mathsf{Coeff}_{\prec}(p)_i = 0,\ \forall i \in \mathcal{I} \right\} \tag{5.8}$$

Proposition 5.8. *Let $\mathcal{F} = \{f_1, \ldots, f_k\} \subset \mathcal{R}_n$ be $\mathbb{F}_2$–linearly independent boolean polynomials and let $\mathcal{I}$ be monomial indexing on $\mathcal{R}_n$. Then the dimension of $\mathcal{F}_0^{\mathcal{I}}$ as a $\mathbb{F}_2$–vector space satisfies*

$$\dim \mathcal{F}_0^{\mathcal{I}} = k - \mathsf{rank}\left(F_{\bullet,\mathcal{I}}\right) \geq k - |\mathcal{I}|, \tag{5.9}$$

where $F \in \mathbb{F}_2^{k \times m}$, $m = \#\mathsf{Monom}(f_1, \ldots, f_k) \leq 2^n$ is a matrix formed by term coefficients of polynomials in $\mathcal{F}$,

$$F_{ij} = \mathsf{Coeff}(f_i)_j \tag{5.10}$$

and $F_{\bullet,\mathcal{I}} \in \mathbb{F}_2^{k \times |\mathcal{I}|}$ is a matrix formed from F by selecting only columns (monomials) given by $\mathcal{I}$.

Proof. We will proceed by explicitly constructing the space as a solution to a linear system.

We want to find such $x \in \mathbb{F}_2^k$ that

$$\left(\sum_{i=1}^{k} x_i f_i \right)_j = 0, \quad \forall j \in \mathcal{I}. \tag{5.11}$$

By reformulating, we arrive at $\sum_{i=1}^{k} x_i f_i = x^\top F$ and ultimately,

$$F_{\bullet,\mathcal{I}}^\top\, x = 0. \tag{5.12}$$

Hence, the set of sought solutions is precisely $S := \ker F_{\bullet,\mathcal{I}}^\top$.

Now, let $\phi : S \to \mathcal{F}_0^{\mathcal{I}}$ be given by

$$\phi(x) = \sum_{i=1}^{k} x_i f_i. \tag{5.13}$$

Then ϕ is $\mathbb{F}_2$–isomorphism as $\phi(x) = 0$ implies $x = 0$ by linear independence of $\mathcal{F}$ and ϕ is an epimorphism due to above construction and hence,

$$\mathcal{F}_0^{\mathcal{I}} = \phi(S) = \left\{ \sum_{i=1}^{k} x_i f_i : x \in S \right\}. \tag{5.14}$$

Statement of the proposition then follows from application of rank-nullity theorem and known property that for a field $\mathbb{F}$ and $A : \mathbb{F}^k \to \mathbb{F}^l$, we have $\mathsf{rank}(A) \leq \min\{k, l\}$ where equality is reached for A full-rank. □

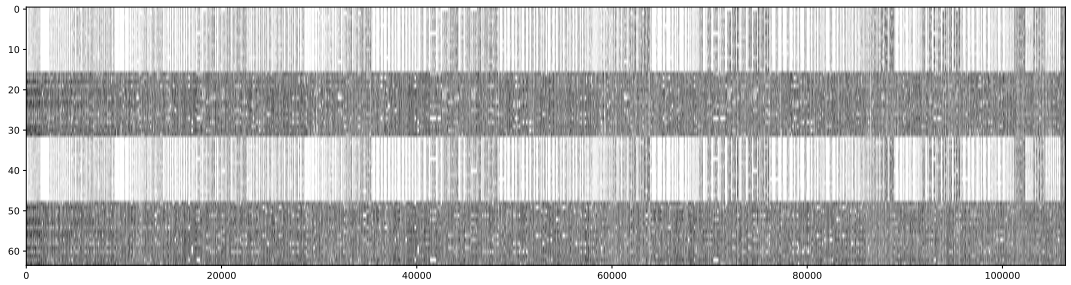

Figure 5.1: Matrix for cipher $SR(2,4,2,4)$ and two PC pairs. The x-axis represents present monomials sorted in descending degrevlex order and the y-axis corresponds to each polynomial of the system. Note that the matrix is full rank, although it displays noticeable structure. This is reflected in the fact that the sub-matrices in Corollary 5.9 are not full-rank as can be seen in Figure 5.3.

Corollary 5.9 (Eliminating first l monomials)**.** *We can try to eliminate monomials* $[\![1,l]\!]$, $l \in \mathbb{N}$, *of* $\mathcal{F} = \{f_1, \dots, f_k\}$ *by various strategies. A natural strategy is to fix dimension* $d \in [\![1,k]\!]$ *and maximize* $l \in \mathbb{N}$ *such that* $d = \dim \mathcal{F}_0^{[\![1,l]\!]}$.

Start with $l_0 := k-d$ *and then find the largest* l *such that* $\dim \mathcal{F}_0^{[\![1,l]\!]} = d$. *This can be done by binary search when degeneracy is expected or by increasing the index one by one when degeneracy is not expected. The algorithm terminates and is correct.*

Proof. From Proposition 5.8 is clearly $\dim \mathcal{F}_0^{[\![1,l_0]\!]} \geq d$ and $\left(\dim \mathcal{F}_0^{[\![1,l]\!]}\right)_{l\in\mathbb{N}}$ is a non-increasing sequence in l. Denote as $l_{\max}$ such $l \in \mathbb{N}$ that

$$l_{\max} = \max\left\{l \in \mathbb{N} : \dim \mathcal{F}_0^{[\![1,l]\!]} \geq d\right\}. \tag{5.15}$$

Then, surely, $l_{\max} \leq 2^n$ and the process terminates. □

Remark 5.10. The proof of Proposition 5.8 also gives an effective algorithm to compute $\mathcal{F}_0^{\mathcal{I}}$ as simply computing kernel of a $\mathbb{F}_2^{|\mathcal{I}|\times k}$ matrix. In light of Corollary 5.9, the kernel has to be computed only at the final step of $l_{\max}$ as the dimension is given by rank which is usually faster to compute. The algorithm is given in Algorithm 3 and for $k < d$, it is LM-contracting in the sense that the leading monomials of the output polynomials $\{g_i\}$ from this preprocessing are strictly smaller than the maximal leading monomial of the original polynomials.

Note that finding a non-zero polynomial of $\mathsf{Span}_{\mathbb{F}_2} \mathcal{F}$ with as many first monomials zero as possible is precisely the statement of Corollary 5.9 for dimension $d = 1$. A visualisation is given in Figure 5.3 for a modified algorithm where dimension is computed for every $l \geq 0$ such that $\dim \mathcal{F}_0^{[\![1,l]\!]} \geq 1$. This is better for visualisation purposes but is less efficient than binary search. Note that cipher $SR(2,4,2,4)$ is not pseudo-random and its matrix is depicted in Figure 5.1.

Proposition 5.11 (Proportion of full-rank random matrices)**.** *Probability* $p(t,s)$ *that a uniformly random matrix* $A \in \mathbb{F}_2^{t\times s}$ *(with coefficient chosen from* $\mathcal{U}(\mathbb{F}_2)$*) is full-rank for* $s \geq t$, *i.e.* $\mathsf{rank}\, A = t$ *is*

$$P\left[\mathsf{rank}\, A = t\right] = \prod_{i=s-t+1}^{s} \left(1 - \frac{1}{2^i}\right). \tag{5.16}$$

Proof. Fix a first row of the matrix. It is non-zero with probability $1 - 1/2^s$ By adding another row, there is a probability $1 - 2/2^s$ that the two rows are linearly independent. Continuing the process, one arrives at the proposition. More details can be found in [44]. □

Algorithm 3 HighestMonomialElimination

Input:

$\{f_i\}_{i=1}^k$	set of k $\mathbb{F}_2$-linearly independent polynomials from $\mathcal{R}_n$
m	number of distinct monomials, $\#\mathsf{Monoms}(f_1, \ldots, f_k)$
F	a $\mathbb{F}_2^{k \times m}$ matrix such that $F_{ij} = \mathsf{Coeff}_{\prec_{\text{degrevlex}}}(f_i)_j$
d	required number of linearly independent output polynomials, $d < k$

Output:

l_{max}	number of eliminated monomials
$\{g_i\}_{i=1}^d$	polynomials with first l_{max} monomials zeroed, basis of $\mathcal{F}_0^{[\![1,l]\!]}$, $\mathcal{F} = \{f_i\}_{i=1}^k$

1: $r \leftarrow k - d$

2: $l_{max} \leftarrow \max\left\{l \in [\![r, m]\!] : \mathsf{rank}(F_{\bullet,[\![1,l]\!]}) \leq r\right\}$ ▷ binary search using at most $\log_2(m - r) \leq n$ rank evaluations

3: $\{g_1, \ldots, g_d\} \leftarrow$ LinearBasis(Kernel($F_{\bullet,[\![1,l_{max}]\!]}{}^\top$)) ▷ in M4RI as `right_kernel_matrix`

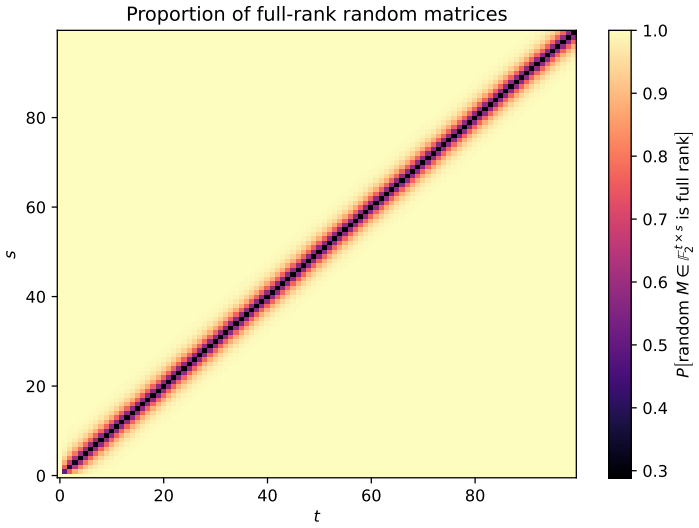

Figure 5.2: Plot of probability that a random matrix over $\mathbb{F}_2$ has full rank as in Proposition 5.11.

Note 5.12. The probability $p(t, t)$ that a $t \times t$ matrix over $\mathbb{F}_2$ with uniformly random coefficients is full-rank is quite small by Proposition 5.11 and as can be seen on Figure 5.2— in fact, for large t there is $\lim_{t\to\infty} p(t, t) \approx 0.28879$ ([41]). However, $p(t, s)$ is very close to one for $|t - s| \geq 4$ as can be seen from the figure. This means that for polynomials with random coefficients, the number of eliminated monomials as in Corollary 5.9 is very close to $k - d$ in notation of the corollary.

5.3 Inflating the ideal equations (INFL)

Let $\mathsf{key} = (k_1, \ldots, k_n) \in \mathbb{F}_2^n$, I_{key} denote the key ideal $(x_1 - k_1, \ldots, x_n - k_n) \subset \mathcal{R}_n$, $V(I_{\mathsf{key}}) = \{\mathsf{key}\}$. Let the ideal of polynomials in key variables be denoted by $I = (f_1, \ldots, f_k) \subset \mathcal{R}_n$. The key is in the variety by definition and hence $I \subset I_{\mathsf{key}}$. By generating more systems with the same encryption key but different plaintexts, we obtain another ideal $I' = (g_1, \ldots, g_l)$. Then $I + I' = (f_1, \ldots, f_k, g_1, \ldots, g_l) \subset I$ is the ideal obtained by concatenating the polynomials into a bigger polynomial system.

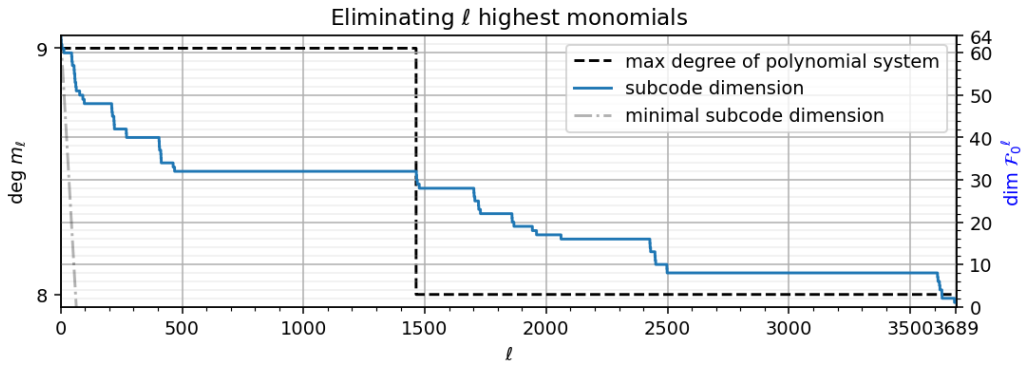

Figure 5.3: Example of Corollary 5.9 for cipher $SR(2,4,2,4)$. On the x-axis is the number of eliminated highest monomials in degrevlex order, the left y-axis is the corresponding degree of the resulting polynomials and the right y-axis is the number of linearly independent polynomials with highest ℓ monomials zero. Here, we have chosen the same strategy to maximize $\ell > 0$ such that $\dim \mathcal{F}_0^{[\![1,\ell]\!]} \geq d$ for $d = 1$. And so output of the algorithm is a single vector with as many highest entries zeroed (here 3689) as possible without being the zero vector.

Now, let us fix the secret key, generate a plaintext and the corresponding ideal I_1. Then generate another plaintext and the corresponding polynomials system and add the corresponding polynomials to I_1 to obtain ideal I_2 and continue the procedure a for a specified number of p steps, in i-th step generating an ideal I_i. Then we have

$$I_1 \subset I_2 \subset \cdots \subset I_p \subset I_{\mathsf{key}} \tag{5.17}$$

where $I_i \subset I_{\mathsf{key}}$ for all $1 \geq i \geq p$ is due to Corollary 1.19.

Due to the ascending chain condition (Theorem 1.9), the sequence of ideals eventually collapses, albeit not necessarily to I_{key}. At this point, adding new equations coming from different PC pairs does not affect the resulting ideal and the time to compute the Gröbner basis is not expected to change by adding more polynomials. Instead, the time to generate the polynomials is dominant time factor in most cases and hence, it is expected to lead to longer overall computation times. Still, it might be beneficial to gather more linearly independent polynomials when linear preprocessing is applied before computing the Gröbner basis. If ideal I_l has a single solution, then all the succeeding ideals collapse and we have $I_l = I_{l+1} = \cdots = I_{\mathsf{key}}$.

As $I_l = (f_1, \ldots, f_k) = \left\{\sum_{i=1}^k r_i f_i : r_i \in \mathcal{R}_n\right\}$, one can simply multiply the original polynomials $f_1, \ldots, f_k$ by monomials from $\mathcal{R}_n$ and add them to the original set of polynomials. This process does not change the generated ideal and creates non-trivial relations between the generators (non-trivial syzygies).

Proposition 5.13. *Multiplication by monomials has the following property. Let* $m \in \mathsf{Monom}(\mathcal{R}_n)$. *Then*

$$\#\mathsf{Monom}(m \cdot \mathcal{R}_n) = \frac{\#\mathsf{Monom}(\mathcal{R}_n)}{2^{\deg m}} = 2^{n-\deg m} \tag{5.18}$$

where $\mathsf{Monom}(m \cdot \mathcal{R}_n) = \{m \cdot \overline{x}^\alpha : \alpha \in \mathbb{N}^n\}$.

Proof. Monomials in $m \cdot \mathcal{R}_n$ are those divisible by m. Every monomial in $\mathcal{R}_n$ is uniquely described by the presence or absence of the variables x_i, $1 \leq i \leq n$ in the monomial. Hence, the monomials in $m \cdot \mathcal{R}_n$ are those containing the fixed $\deg m$ variables present in m and there are $2^{n-\deg m}$ of them. The second follows by multiplication in $\mathbb{F}_2[x_1, \ldots, x_n]$. □

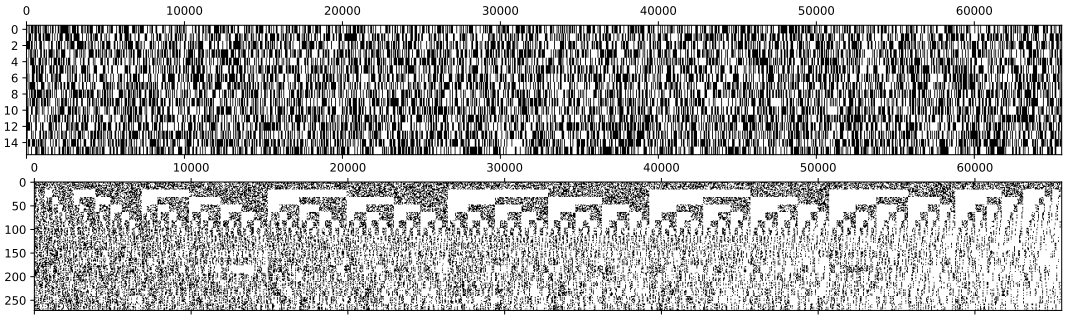

Figure 5.4: Upper figure is a visual representation of $SR(3,2,2,4)$ for a single PC pair, the cipher has $n = 16$ key bits. Rows of the matrix correspond to individual polynomials and columns to its monomials. The monomials are sorted according to descending degrevlex order and a black pixel indicates one. The lower figure is the result of concatenating polynomials created by multiplying by monomials $1, x_1, \ldots, x_n$ in order. Note that the images has been down-scaled.

We can set the algorithm to multiply the original generators by all monomials up-to degree $D \leq n$, apply linear preprocessing and finally compute the Gröbner basis. However, when the degree D is high, we are doing basically the same work as the original Gröbner algorithm although much less efficiently. For exploratory purposes, we will set $D = 1$ which provides additional $\binom{n}{1} = n$ polynomials and the new polynomial system contains $n+1$ times more polynomials than the original system. Visual representation of Proposition 5.13 for $m \in \{1, x_1, \ldots, x_n\}$ can be seen in Figure 5.4.

As a contrast, consider a non-diffused cipher, for example $SR(2,2,4,4)$ with 32 key bits. Maximal degree in $\mathcal{R}_{32}$ is 32, however the maximal degree of polynomials encountered in $SR(2,2,4,4)$ is 9. When we multiply the system by each x_i and concatenate the result, the number of monomials might raise up-to n times in the extreme case when all the systems have disjoint monomials sets. The resulting elements of matrix then increases up-to n^2 times.

Another way to inflate the system would be to multiply by large monomials and discard the original polynomials. In $\mathcal{R}_n$, multiplying by the monomial m with degree $\deg m = n$ reduces any polynomial $f \in \mathcal{R}_n$ to either zero or m. This polynomial then has either $2^n - 1$ or 2^n solutions which is not very useful when solving the original system. Similar discussion can be made for slightly lower multiplying monomials.

Chapter 6

Experiments

6.1 Overview

We have conducted several types of experiments, each probing a different preprocessing technique. Overall, we have implemented

- LLL reduction 5.1 and subsequent search for shortest vectors in the basis
- Elimination of the highest monomials in degrevlex order as in Algorithm 3 and Corollary 5.9.
- Expansion of ideal equations as in Section 5.3.
- Combination of the three above. When present, expansion (3) is applied first, possibly followed by elimination (2) and LLL (1) at last.

All experiments were performed on a machine with operating system Ubuntu 20.04.4 LTS running on two Intel XeonGold 6136 processors, each containing 12 cores. In total, the machine has 24 cores and 768 GB of RAM.

6.2 Implementation

We have written the collection of scripts to run the experiments in order to be fast, parallelized and extendable. There was old implementation of similar experiments in [14], however we did not find the code suitable for our purposes except for generating the system with key variables only as in [19, 14]. Our implementation is available at https://github.com/rampaq/aesalgae. In our scripts, we have used the following libraries.

6.2.1 Libraries

PolyBoRi

The boolean polynomials of $\mathcal{R}_n$ are implemented using PolyBoRi [18]. It uses binary decision diagrams, a variant known as ZDD, to store and operate on the polynomials. Each polynomial in $\mathcal{R}_n$ contains up-to 2^n terms and the PolyBoRi uses the diagrams to store it using $3 \cdot n - C$ for some positive constant C. This specialized implementation is the main reason that makes this approach possible. Various operations, such as converting the system to a matrix, computing density of the system, etc., in our work are implemented using PolyBoRi, or rather by its SageMath interface.

M4RI

Linear algebra was implemented in SageMath via M4RI library for fast operations over dense $\mathbb{F}_2$ matrices. The library provides fast operations for dense matrices over $\mathbb{F}_2$. It is named after *Method of Four Russians* and provides improvement for basic linear-algebra operations like matrix multiplication, Gaussian elimination/row reduction and system solving. For matrix in $\mathbb{F}_2^{m\times n}$, the algorithm provides the above operations with complexity $\mathcal{O}(mn\min(m,n)/\log_2\max(m,n))$ and $\mathcal{O}(mn\min(m,n)/\log_2 m)$ for Gaussian elimination/row reduction. This is a logarithmic improvement over naïve algorithm with complexity $\mathcal{O}(mn\min(m,n))$ field operations.

It also features *Strassen-like* algorithms that operate on square $n\times n$ matrices. Most operations like multiplication of two matrices, Gaussian elimination and system solving run with complexity $\mathcal{O}(n^{\log_2 7n})$, $\log_2 7 \approx 2.807$. For rectangular matrices, there are scheduling algorithms which reduce the problem for large rectangular matrices to operations on smaller square matrices. It can be combined with the Method of Four Russians to obtain a constant-factor speedup. The Method of Four Russian provides faster experimental results for dimensions up-to 64000×64000, depending on hardware.

Magma

We have conducted the experiments with SageMath [48] computer algebra system to generate the polynomial system, handle the polynomials and implement preprocessing. Finally, we have used Magma [16] (v2.28-14) to compute its Gröbner basis via optimized F_4 algorithm.

Magma offers various tunable parameters when computing the Gröbner basis using the F_4 algorithm. The algorithm uses linear algebra on large matrices and one such parameter is to turn on sparse/dense matrices which can lead to significant differences in memory consumption and computation times. Both variants allow for parallelization on GPU and/or via multithreading. Additionally, there is a heuristic parameter which is proclaimed by the author of the Gröbner basis code to significantly speed up the computation *if tuned correctly*. There is no automatic way to determine the parameter and hence, we do not use this feature.

We have implemented automatic switching of dense/sparse linear algebra based on the density of the system matrix. We have utilized threading and GPU for the computation of Gröbner basis and in some cases, it leads to up-to 50% speedup over non-paralellized code, although the parallel efficiency drops for larger systems.

Remark 6.1 (Monomial packing)**.** In order to transfer the polynomials of $\mathcal{R}_n$ from SageMath to Magma, one has to use a representation of the polynomials. In previous works [14, 13], a textual representation was used — key variables are typically denoted as `k001`, `k002`, `k003`, ... and the representation used is simply a textual transcription: polynomial $k_{001}\cdot k_{002}\cdot k_{003}+k_{001}\in\mathcal{R}_n \to$ an ASCII-encoded string `k001*k002*k003+k001`. This is quite inefficient for systems with high density of high-degree monomials and poses a significant bottleneck. This can be improved for the case of boolean polynomials from $\mathcal{R}_n$ by *packing*. Each monomial of $\mathsf{Monom}(\mathcal{R}_n)$ can be represented by a number between 0 and 2^n-1 — a monomial $m=\prod_{i=0}^{n-1} {x_i}^{\alpha_i}$, $\alpha_i\in\{0,1\}$ for all $i\in[\![0,n-1]\!]$ is represented by an n-bit word $\alpha_0\alpha_1\ldots\alpha_n$ — if variable x_i is present in monomial m, then the bit-representation of m contain 1 at the position i. Subsequently, a polynomial is represented as a list of n-bit numbers. This offers reduction by a linear factor in space complexity of the representation of

a polynomial. As this bit packing seems to be an internal representation of polynomials in $\mathcal{R}_n$ in Magma, there is very little additional preprocessing needed on the Magma interface and the function `BooleanPolynomial`.

As an example, for $SR(3,2,2,4)$ and two PC pairs, the transfer using textual representation is completed after ca. 70 s and scales linearly with the number of PC pairs. By using packing, the time was reduced to 5 seconds. This approach is naturally parallelizable and this leads to even faster sub-second transfer times. For larger ciphers, the time difference is even more pronounced.

Additionally, we have reported a bug report with inputting polynomials via the function `BooleanPolynomial` for $n = 31, 32$ to Magma developers. In the meantime, we have added a workaround via dummy variables to artificially increase the number of variables.

CodeRed

CodeRed is an optimized C++ library from the original authors of the LLL algorithm for codes [24] with a Python frontend. It provides all algorithms presented in their paper and is available online at https://github.com/lducas/CodeRed.

FGb

Fast Gröbner basis (FGb) is a free closed-source library. It supposedly provides the only mainstream implementation of F_5 algorithm, as stated in the introductory paper [25]. The presence of F_5 is also advertised on the main page of the project although the tutorial and mention only F_4. We think that the implemented algorithm is perhaps some hybrid version of F_4 or F_5 and we have not found any indication that it natively supports reducing by the field equations. Nevertheless, we have tried to use the library to compute the Gröbner basis for $SR(3,2,2,4)$ and it did not finish in the allotted 5 hours. Hence, we have not used the library in further experiments.

Pandas

For processing experiment data and statistical analysis.

6.2.2 Gröbner basis and preprocessing

First, we generate the polynomial system for the given cipher and then eliminate non-key variables as stated in Chapter 4. This is done for each PC pair separately and then the systems are concatenated. Subsequently, we collect some statistics about the system, such as the maximal occurring degree, average number of monomials, density of the system and if preprocessing takes place later on, then we record also the dimensions of the matrices and their ranks.

If preprocessing is applied, then (if needed) the corresponding matrices (with correct matrix backend) are supplied to it along with the original polynomial system. Note that the matrices are always dense for simplicity. Computation is done and the result is converted back to polynomials. Preprocessing outputs a predefined number of polynomials. If there are not enough linearly-independent polynomials, the computation is aborted. Finally, additional post-processing statistics are collected.

Finally, the Gröbner basis via F_4 in Magma is computed. Based on the density of the last matrix, a dense or sparse matrix format for Gröbner basis is supplied. We have chosen density threshold to be 30 %, although the only cipher we tested with density above this treshold is $SR(3,2,2,4)$. If below, sparse variant of F_4 is computed, otherwise its dense variant. Multithreading and/or GPU support can be

enabled for the linear algebra in F_4 to speed up the computation. We have done all the experiments with 24 threads and GPU enabled. Some experiments have crashed Magma and we then restart without the GPU which solves the problems most of the time. See more possible problems in the Magma subsection above. Still, some unexplained crashes still occur and sometimes, computation finishes but gives wrong solution. These experiments are not included in final results.

In data processing, outliers are detected, all values are averaged and standard deviations are computed. For some positive quantities such as timings, standard deviation is sometimes larger that mean signifying that the values are not normally distributed.

For basic experiments, we have set a timeout to two hours and 30 minutes. Regarding longer experiments, it was 8 hours. The longer experiment was exclusively run on $SR(2, 4, 4, 4)$ cipher.

Preprocessing

The preprocessing algorithm usually have at least two parameters: number of ($\mathbb{F}_2$-linearly independent) input polynomials k and number of ($\mathbb{F}_2$-linearly independent) output polynomials r. We have:

- $r < k$ for HighestMonomialElimination,
- $r \leq k$ for LLL,
- $r \geq k$ for Inflate.

In our experiments, we work with polynomials in $\mathcal{R}_n$ for various values of n depending on the cipher and we usually work in multiples of n when choosing the values of k and r. We have typically chosen r/n and k/n to be powers of two in order to explore large parameter space in constrained time limit for this thesis.

LLL

The original paper already contains an optimized C++ implementation of all the aforementioned LLL-related algorithms for linear codes. It has a Python frontend which expects input as a Numpy array with `bool` type. We have created parallelized methods to convert polynomial systems to Numpy. We have also tried to make it faster by using shared memory between the processes without success.

Note that checking the linear independence of the input polynomials is crucial as some weaker ciphers give rise to this condition and the implementation of LLL in [24] expects a linearly independent vectors and otherwise, it loops forever in certain methods. However, checking the linear independence of the input polynomials turned out to be a problem. The only package able to operate on Numpy arrays which provides linear algebra over finite fields is the package `galois`. Even though its implementation uses a partially vectorized operations and should be effective, we have discovered it to be a bottleneck when operating on larger matrices. This is probably due to its usage of naïve implementation of row reduction which uses $\mathcal{O}(k^2m)$ operation for $k \times m$ binary matrix and not enough performance optimizations. In the end, we opted to create both Numpy and M4RI matrices and use the M4RI library to determine the dependent rows and remove them from the Numpy matrix subsequently.

As this method in fact performs multiple various sub-methods and evaluates the best one, it is expected that a certain family of sub-methods will be consistently

faster for a given cipher with varying round parameter n and hence this preprocessing could be sped up by a large margin.

Highest monomial elimination

Let $\mathcal{F} = \{f_1, \ldots, f_k\} \subset \mathcal{R}_n$ and fix a number of output linearly independent polynomials $r \in \mathbb{N}$.

According to Algorithm 3 and Corollary 5.9, the computation of $\mathcal{F}_0^{[\![1,l]\!]}$ (elimination of l leading monomials of $f_1, \ldots, f_k \in \mathcal{R}_n$) can be accomplished by a binary search on the ranks of matrices computing a kernel of certain $l \times k$ matrix. This was done using M4RI in SageMath.

6.2.3 Inflating the ideal equations

Implementing this preprocessing is straightforward as we just multiply the original polynomials by n monomials $x_1, \ldots, x_n$ in sequence and it is trivially parallelizable. We have considered implementing this preprocessing using PolyBori at the lowest level possible - the binary decision diagrams (ZDD). Multiplying by a single variable amounts to adding a single node to the diagram and would potentially be very cheap. However, the diagram needs to remain in a particular order and adding the node to the top would destroy its structure. Hence, we have opted to use a higher-level API to simply multiply the polynomials with the monomial.

However, we have encountered technical problems. Due to paralelization of our code, the matrix sometimes needs to be serialized. SageMath uses 1-bit PNG to store the matrix as it is abundant and efficient. However, for unknown reasons, exporting to PNG fails when one of the dimensions is greater than exactly a million (1,000,000). This seems like a very odd number and rather artificial as all used libraries (libPNG, libGD and M4RI) library support structures with dimensions a, b such that $ab < 2^{31}$. It is indeed the case and the libPNG manual states that they have imposed an arbitrary limit of a million rows/columns. However, we do not have low level access through the Python interface and we would have to manually increase the limit by compiling our own version of Sage which uses libGD which in turn uses libPNG. We have also filed a bug report with possible solution. Due to time concerns, we did not modify the library.

6.3 Results

First, we will explore reference solution and each preprocessing separately and then consider their combinations when possible. In each experiment, we note the quantities from Table 6.1 and some additional quantities are collected for preprocessing.

Note that all statistics collected were first grouped by the repeated experiments with the same parameters. Subsequently, we attempted to identify outliers for each variable and subsequently compute mean and standard deviation. We then show the mean in the tables and plots show both mean and standard deviation as error bars with respective color. Note that for ca. five experiments for each parameter set the practically all outlier detection methods are not very reliable for obvious reasons. Furthermore, the given times in Table 6.1 do not add up to t_{all} as time to construct the matrices and collect statistics is not included as it is not directly relevant to cryptographic purposes even though the matrix construction sometimes takes considerable time.

symbol	description
`nrce`	used small-scale AES $SR(n, r, c, e)$, given as `nrce` string
n	number of bits of the secret key, $n = rce$
PC	the number of plaintext-ciphertext pairs, $k = n \cdot \text{PC}$
r	after preprocessing, $r \cdot n$ is the number of output polynomials
t_{elim}	time to generate the system for AES and eliminate non-key variables
t_{pre}	time to compute the preprocessing
t_{key}	time to compute the final Gröbner basis
t_{all}	total running time of the script
RAM	peak RAM consumption during GB computation
#V	cardinality of the variety/number of solutions

Table 6.1: Basic quantities collected and used in results.

`nrce`	n_{bits}	PC	$\#V$	t_{all} [s]	t_{key}	t_{elim}	RAM [MB]
1224	16	2	2^7	1.4	< 1	1.1	33
1228	32	8	1	9.9	< 1	8.4	33
1244	16	2	1	2.3	< 1	1.8	33
1248	64	2	1	21	< 1	19	33
1424	32	2	2^{14}	2.6	< 1	2.0	33
1428	64	8	1	23	< 1	19	99
1444	64	2	1	5.1	< 1	4.5	33
1448	128	16	1	66	9.8	41	579
2224	16	4	1	3.0	< 1	2.3	33
2244	32	32	1	37	4.3	18	860
2424	32	1	—	—	—	5.3	—
2444	64	1	—	—	—	10	—
3224	16	1	2	44	36	7.5	1157
10224	16	1	1	167	33	133	918
2448	128	—	—	—	—	—	—

Table 6.2: Reference timings without preprocessing. Legend is given in Table 6.1. More statistics, like average number of monomials in a polynomial, number of distinct monomials needed to represent the system and maximal degrees, on the corresponding polynomials are given in preprocessing tables. Some quantities like the number of key bits n_{bits} of the cipher will be referenced only in this table as they are invariant to preprocessing. Systems $SR(2, 4, 2, 4)$, $SR(2, 4, 4, 4)$ and $SR(2, 4, 4, 8)$ were not solvable by the reference solution.

6.3.1 Reference solution

The reference solution consists of generating the polynomial system modelling the cipher over $\mathbb{F}_2$, eliminating the state variables to obtain polynomials in key variables only. Finally, the Gröbner basis is computed via Magma's F_4 algorithm with parameters outlined above.

We have collected at least five measurements of all different combinations which are stated below. For reference solution, we have conducted measurements for all ciphers listed for PC pairs up-to 64 at points corresponding to powers of two. One exception is the cipher $SR(3, 2, 2, 4)$ for which we have conducted experiments up-to 256 PC pairs. The results are given in Table 6.2.

We conjecture that when the polynomial system is sufficiently diffused (pseudo-random) and it has a single solution, adding more polynomials/PC pairs is not going to decrease the solving time; on the contrary, adding more polynomials usually leads to gradual rise in the solving time. A notable anti-example is the $SR(1,4,4,8)$ cipher which has a single solution already with a single PC pair but achieves minimal solving time around 16 PC pairs. This is probably due to some additional structure of $SR(1,4,4,8)$.

Regarding the differences in older Magma version (v2.25) without GPU and the new one (v2.28-15) with otherwise same parameters, we have obtained mixed results. Reference solutions of some ciphers were faster to solve using the older version, such as $SR(1,4,4,8)$ in 44 seconds total time and the system $SR(2,4,2,4)$ was solved in ca. 8000 seconds. On the other hand, $SR(3,2,2,4)$ is solved with the old version in ca. 100 seconds. As Magma is closed source, we cannot make any statements on why this happens, however we assume that some optimizations were applied in the newer versions however as polynomial systems can have widely varying properties, it is impossible that the changes lead to improvement on all possible systems.

We have observed that memory consumption strongly correlates with the computation time of Gröbner basis. The consumed RAM in Table 6.2 seems very low compared to previous works, however it should be noted that only values with the minimal time are given. For other parameters, the consumed RAM was much higher. For example, for the simplest cipher in our experiments, $SR(1,2,4,4)$, for $PC = 1$ the consumed RAM was 770 MB, a 23x increase from the optimal solution for $PC = 2$. Regarding the cipher $SR(2,2,4,4)$, the fastest solution consumed 860 MB ($PC = 32$) of RAM and for the slowest ($PC = 4$) it was 75 GB. Experiments which timed out in the 2 hour 30 minutes window sometimes consumed hundreds of gigabytes of RAM.

We have drastically reduced memory usage for $SR(3,2,2,4)$ by using dense matrices in F_4 while previous works were (unintentionally) using sparse matrices to model a matrix with density ca. 50 %.

We have tried to solve also the second round of 128-bit AES $SR(2,4,4,8)$. However, the first step of generating the system — generating the polynomials and eliminating key variables — did not complete after 32 hours and consumed almost all available RAM of the 755 GB available. Hence, to experiments like applying preprocessing or computing its Gröbner basis could not be done.

Overall and compared to previous work, we have achieved faster reference solving times for larger ciphers and we have observer slight delay with smaller ciphers due to collecting statistics overhead and probably due to more precise measurement. However, this could be simply eliminated by switching based on cipher size. The major contribution of our faster solve times is not due to the new version of Magma (almost on the contrary) but rather by using the monomial packing when loading the polynomials to Magma and by manually operating the dense/sparse linear algebra for F_4, however the dense variant was used only for $SR(3,2,2,4)$. A rather significant contribution is also deploying threaded/GPU accelerated linear algebra for F_4, decreasing the solving time by approximately 10-20 % for larger ciphers and by ca. 50 % for dense variant with $SR(3,2,2,4)$.

It should be noted that we have solved both $SR(2,2,4,4)$ and first round of AES $SR(1,4,4,8)$ already with the reference solution which was not previously achieved in this line of work. Also note that previous works [14, 13] have included the time to transfer the polynomials to Magma as part of the t_{key} time — it could be argued that by using preprocessing, they unknowingly optimized the transfer time instead of the Gröbner basis computation.

6.3.2 LLL

In this section, we present the experimental results on the LLL preprocessing. The main goal of LLL is to reduce the average number of monomials per polynomial in order to compute with sparser matrices in F_4. This effort was mostly successful as can be seen in Table 6.3.

Even though the average number of monomials in the polynomials is a crude heuristic, we have obtained better results than with reference solutions in most cases with notable exceptions to $SR(1,4,4,8)$ and $SR(3,2,2,4)$, the second of which is a pseudo-random cipher as indicated by $\overline{\text{MON}} \approx 2^n/2$ and was discussed in more detail in [14]. This can be expected as the Gilbert-Varshamov distance is very close to $2^n/2$ as discussed earlier and it is impossible to obtain much better results under assumption of pseudo-randomness with linear preprocessings. One obvious way to decrease d_{GV} is to provide more PC pairs as this (linearly) increases the number of polynomials and decreases d_{GV}. However, for $PC = 256$, $r = 1$, we have $d_{\text{GV}} \approx 23000$ and $\overline{\text{MON}}_{pp} \approx 30000$. Later, we will show that it is possible to achieve sub-Gilbert-Varshamov results with $\overline{\text{MON}}_{pp} < d_{\text{GV}}$, however it will be for non-linear preprocessing. Even after decreasing the average number of monomials as low as possible, we have not observed any change in t_{key} times.

On the other hand, we were able to achieve significantly better results with preprocessing for non-diffused[4] ciphers. However, in a single instance of $SR(1,2,4,8)$, the LLL algorithms could not reduce the average number of monomials and so the best algorithm was to "do nothing" and select the shortest vector in the original polynomial system. Compared to previous works [13], we have obtained both better overall timings and lower average monomials — for the main cipher $SR(2,4,2,4)$ and $SR(2,2,4,4)$, $\overline{\text{MON}}_{pp}$ is twice as lower and the preprocessing is ca. 40x faster, total solve time approximately 20x faster. A dependence on number of PC pairs is in Figure 6.1. Note that the Gilbert-Varshamov distance for non-diffused cipher is much larger than the obtained $\overline{\text{MON}}_{pp}$.

From Table 6.3, we can see that the number of solutions did not change much from the reference. However, some experiments with LLL (which do not appear here as they did not produce optimal solve times) have shown that choosing the shortest vectors might susceptible to create highly undetermined systems in the meaning that the resulting system has numerous solutions. This might be due to the fact that when the input system is very structured, LLL might not be able to produce shorter polynomial that in the original system without preprocessing and when more PC pairs are present, the preprocessing might choose the same part of the system from each PC pair sub-system. This sometimes leads to the final system having many solutions which is unfavorable.

6.3.3 HME

In this section, we state results for highest monomial elimination preprocessing as given in Algorithm 3 for various numbers r of linearly independent output polynomials. Preprocessing's main goal is to reduce the degrees of polynomials. For pseudo-random ciphers with n bits, degree drops from d to $d-1$ after eliminating approximately $\binom{n}{d}/2$ leading monomials and so reducing more and more degrees requires roughly exponentially many monomial eliminations up-to degree $n/2$. Hence, we see only small drops in degree which is expected.

A secondary effect is that it also reduces the average number of polynomials either by eliminating the leading monomials or by other effects. Even though it was not

[4] Ciphers whose polynomials' coefficients are not uniformly distributed in $\mathbb{F}_2$.

nrce	PC	r	$\overline{\text{Mon}}$	$\overline{\text{Mon}}_{pp}$	d_{GV}	$\#V$	t_{all} [s]	t_{key}	t_{pre}	t_{elim}	Ram [MB]	Meth
1228	2	2	315	278	359	1	10	< 1	< 1	7.7	33	7
1244	4	1	24	1	1.4	$2^{16.7}$	3.6	< 1	< 1	2.3	33	4
1248	2	2	348	353	718	1	16	< 1	1.4	12	33	1 (id)
1424	2	1	37	5.2	11	$2^{8.2}$	3.7	< 1	< 1	2.2	33	7
1428	2	2	566	421	718	1	19	< 1	1.3	14	69	7
1444	2	2	40	20	20	1	6.6	< 1	< 1	3.8	33	7
1448	16	2	599	57	787	1	71	< 1	14	49	33	7
2244	4	2	6687	1336	21255	1	23	8.1	5.6	4.7	426	7
2424	16	2	33128	5745	51891	2^{16}	433	323	60	20	11345	6
3224	1	1	32772	32503	32164	1.8	52	40	2.2	7.4	1388	7

Table 6.3: Experiments for LLL preprocessing. Here, $\overline{\text{Mon}}$ denotes average number of monomials per polynomial and $\overline{\text{Mon}}_{pp}$ denotes the same quantity after preprocessing, d_{GV} is Gilbert-Varshamov distance and for pseudo-random ciphers, this is the expected length of the polynomial with minimal amount of monomials (with linear preprocessing). The prevalent method in last column is selected as the method which was selected as the most effective in minimizing $\overline{\text{Mon}}_{pp}$ with the largest frequency for the given cipher and the number refers to the list in Section 5.1 on LLL.

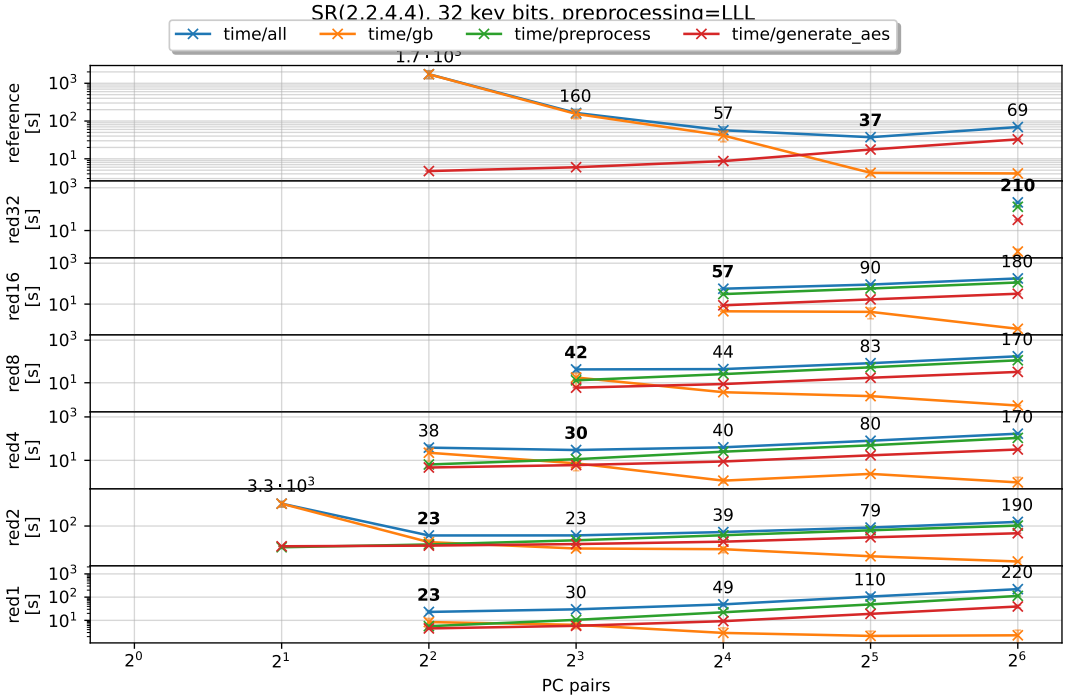

Figure 6.1: Time measurements for cipher $SR(2, 2, 4, 4)$. There is the number of PC pairs on the x-axis and the y-axis corresponds to time. Each block (red1–red32) corresponds to different reduction value r for LLL preprocessing and a reference without preprocessing is on top. Notice that for the lowest PC pairs, Gröbner basis computation time (yellow) is dominant until the cost of generating the system (red) dominates. For preprocessing, the situation is similar although for more PC pairs, the time is dominated by LLL preprocessing.

nrce	PC	r	deg	$\deg_{pp}$	#Mon	ΔMon	$\overline{\text{Mon}_{pp}}$	$\#V$	t_{all} [s]	t_{key}	t_{pre}	t_{elim}	Ram [MB]
1228	2	1	7	6	1017	45	256	7	8.1	< 1	< 1	7.1	33
1244	2	1	3	2	113	29	7.1	$2^{6.6}$	2.8	< 1	< 1	1.8	33
1248	4	2	7	6	2033	161	184	1	18	< 1	1	14	33
1424	4	2	3	2	113	42	14	2^9	4.2	< 1	< 1	2.3	33
1428	4	2	7	6	2033	159	385	1	20	< 1	< 1	16	33
1444	2	1	3	2	225	66	15	2^{17}	5.3	< 1	< 1	3.8	33
1448	4	2	7	6	4065	328	348	1	39	< 1	1.6	30	33
2244	8	1	9	6	45689	8003	1768	2^{15}	35	19	3.1	6.1	949
2424	32	1	9	7	112723	15545	6483	2^{16}	405	284	10	35	10067
3224	2	1	16	15	65536	16	32748	1.3	50	36	< 1	9	882

Table 6.4: Results for HME (highest monomial elimination) preprocessing. The common legend is in Table 6.1, deg and $\deg_{pp}$ denote maximal occurring degree of polynomials before and after preprocessing, respectively. #Mon is the number of distinct monomials in the system, $\#\text{Mon} = \#\mathsf{Coeff}(f_1, \ldots, f_{n \cdot \text{PC}})$, ΔMon is the number of eliminated leading monomials by the preprocessing in degrevlex order and $\overline{\text{Mon}_{pp}}$ is the average number of monomials per polynomial after preprocessing.

primarily designed to do, it produces often lower averages than the best specialized algorithms in [13], in some cases by a factor of two as for cipher $SR(2, 4, 2, 4)$. The average number of monomials is usually higher than for LLL but for cipher $SR(2, 2, 4, 4)$ it is lower on average, see Figure 6.2.

We can see from Table 6.4 that the preprocessing is generally faster than LLL. This is understandable as both algorithms have roughly $\mathcal{O}(k^2 m)$ complexity for processing k polynomials containing m distinct monomials and both methods are optimized, however LLL runs multiple sub-experiments to determine the best method. For some combination of parameters, the Gröbner basis computation has consumed as much as 720 GB of RAM, experiments which did not finish properly sometimes consumed even more memory.

6.3.4 Combined preprocessing

We have also experimentally explored composition of different preprocessings. In Table 6.5, we have explored first applying HME to reduce the system from $\text{PC} \cdot n$ to $r \cdot n$ polynomials and subsequently apply LLL without any dimensional reduction. As this involves generating two types of matrices for each preprocessing, it tends to get slower than each individual preprocessing and in addition, the gain in Gröbner basis computation is small. Thus, this combination usually does not lead to faster solving times. However, there is a very notable exception with cipher $SR(2, 4, 2, 4)$ — the combination of preprocessings is almost twice as fast overall than previous preprocessings, however the average number of monomials is not special compared to the two basic preprocessings.

As next preprocessing, we have tested the non-linear preprocessing — inflating the ideal equations (INFL). Due to the aforementioned implementation difficulties in the library, we were able to run this preprocessing only with Numpy matrices for more than $n \geq 32$ bits. However, we could run both LLL and HME on $SR(3, 2, 2, 4)$ as the most notable cipher in hopes of performing better than linear preprocessing.

On its own, it performs underwhelmingly, as expected — as it purely increases the number of polynomials with closely related ones, the F_4 algorithm simply has to go through $n + 1$ times more polynomials. Indeed, total time for $SR(3, 2, 2, 4)$ with INFL for $\text{PC} = 1, 2, 4, 8$ takes approximately 450 seconds overall and is vastly

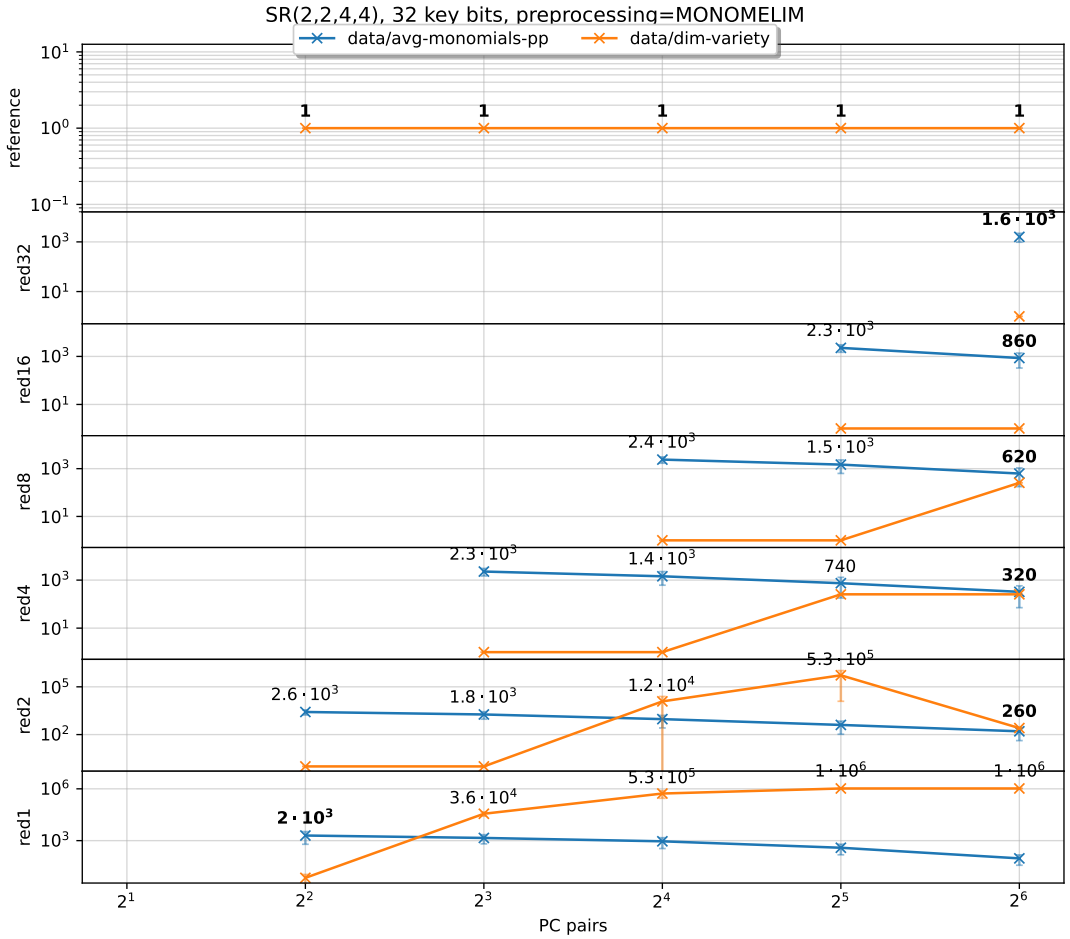

Figure 6.2: A plot of dependence of average number of monomials per polynomial after HME preprocessing, $\overline{\text{MON}}_{pp}$ (blue) and number of solutions (yellow) of the final polynomial system for cipher $SR(2, 2, 4, 4)$. One can observe the rising trend in number of solutions for larger number of PC pairs.

`nrce`	PC	r	$\overline{\text{MON}}$	$\overline{\text{MON}}_{pp}$	d_{GV}	$\#V$	t_{all} [s]	t_{key}	t_{pre}	t_{elim}	RAM [MB]	method
1244	2	1	24	4.7	9	2^7	3.8	< 1	< 1	1.9	33	6
1248	4	2	347	120	645	1	20	< 1	3	14	33	6
1424	4	2	37	1.4	< 1	1	5.4	< 1	1.4	2.4	33	7
1428	4	2	566	119	639	1	28	< 1	4.1	18	33	7
1444	2	1	40	5.7	19	2^{15}	7.4	< 1	1.4	4.3	33	7
1448	4	2	598	131	1283	1	47	< 1	7.5	33	33	6
2244	8	1	6694	1158	2806	1	29	11	4.4	6.1	517	7
2424	32	1	33152	5179	4958	2^{16}	192	60	28	40	772	6
3224	2	1	32746	32514	32157	2	51	34	5.2	8.6	894	7

Table 6.5: Here, HME was applied first to reduce the number of polynomials to $n \cdot r$ and LLL was run subsequently. All legends are already stated in previous tables.

nrce	n_{bits}	PC	r	Pre	t_{all} [s]	t_{key}	t_{pre}	Ram [MB]
3224	16	1	—	Infl	357	339	2.9	17906
3224	16	1	4	Infl → Hme	58	34	14	970
3224	16	1	1	Infl → Hme → Lll	66	40	17	1096

Table 6.6: Results minimizing total computation time for preprocessing involving inflation of ideal equations (INFL). The initial number of polynomials $n \cdot$PC was expanded $(1+n)\times$ by multiplying the original polynomials by monomials $1, x_1, \ldots, x_n$ in order and concatenating the results into a new polynomial system.

dominated by Gröbner basis computation. However, the original purpose is to apply further linear preprocessing after INFL.

The results for the best overall times are given in Table 6.6. Additionally, we have confirmed that it is computationally much less expensive to generate the same number of polynomials via INFL than with corresponding number of PC pairs. We state results only for $SR(3,2,2,4)$. As we had only LLL preprocessing available for larger ciphers and computing time for larger PC values was dominated by LLL preprocessing, we see no value in replacing initial system generation (elimination of key variables) by INFL.

Out of interest, we have tried to apply a special transformation to a 16-bit key system $SR(3,2,2,4)$, PC $= 1$, by a single degree one monomial and replace the original system with the new one. As expected, $\overline{\text{Mon}}$ dropped by half. Indeed, the Gröbner basis computed almost instantly, however the number of solutions of this new system grew rapidly; it was approximately 2^{15}, essentially making the whole process useless. As in order to guess the original intended key, we would have to go through 2^{15} solutions to the modified system — making the approach practically brute-force.

At last, we have tried to increase timeout to eight hours and try to break cipher $SR(2,4,4,4)$ with all available combinations of preprocessing but without success.

6.3.5 Comparison of preprocessing methods

After comparing all the methods above, we state the best methods for each cipher in Table 6.7. Both main preprocessing methods (and their combination) considered were able to solve the cipher $SR(2,4,2,4)$ which was not solvable by the reference experiment in the allotted two hours thirty minutes. In the last column, we compare the results with recent tangent papers [14, 13] and we conclude success in both time and memory domain optimization. We have designed methods which were able to accelerate the whole solving process by a factor of 40 and decrease memory consumption by order of hundreds for memory consumption in certain ciphers.

Both methods HME and LLL achieve similar qualities in reducing the time to compute a Gröbner basis. Even though both were designed with different intentions in mind, LLL for reducing average number of monomials and HME for reducing maximal degree, it turns out that also HME significantly reduces average number of monomials in addition for some ciphers and they have similar characteristics, even though LLL is better in this aspect. On the other hand, reducing maximal degree of polynomials is the feature of HME and we think that this could also lead to provably better GB computation. They could also work well at the same time as demonstrated in the final table.

Alas, we have not been able to reduce the time needed to crack the pseudo-random cipher $SR(3,2,2,4)$ of reference solution even after considerable effort and

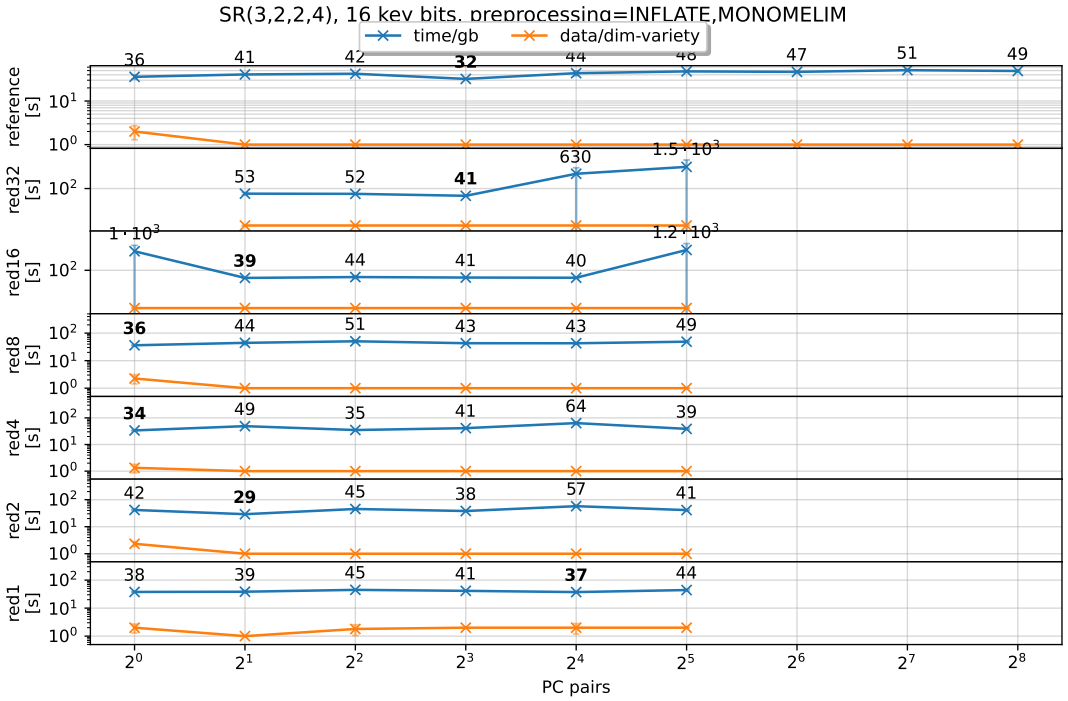

Figure 6.3: Dependence of time needed to compute the Gröbner basis (blue) in seconds and the number of solutions (yellow), dimensionless, for first expanding the system with INFL and subsequently reducing with HME. Note that PC=i indicates that $i \cdot n$ polynomials were created and INFL inflated them to $in(n+1)$ polynomials. Then, HME was applied.

nrce	PC	r	Best Preproc	t_{all} [s]	t_{key}	t_{pre}	t_{elim}	Ram [MB]	$\frac{t_{\text{old}}}{t_{\text{all}}}$	$\frac{\text{Ram}_{\text{old}}}{\text{Ram}}$
1228	2	1	Hme	8.1	< 1	—	7.1	33	1	1
1244	2	—	—	2.3	< 1	—	1.8	33	1	1
1248	2	2	LLL	16	< 1	1.4	12	33	1	13
1424	2	—	—	2.6	< 1	—	2	33	1	1
1428	2	2	LLL	19	< 1	1.3	14	69	2	75
1444	2	—	—	5.1	< 1	—	4.5	33	1	1
1448	4	2	HME	39	< 1	1.6	30	33	4	170
2244	4	2	LLL	23	8.1	5.6	4.7	426	1.5	1
2424	32	1	HME→LLL	192	60	28	40	772	40	320
3224	1	—	—	44	36	—	7.5	1157	13	20

Table 6.7: Selection of best preprocessing methods in our work for each cipher based on total computation time. Legend is given in Table 6.1. In last two columns we give comparison with methods introduced in recent similar work [13]. When preprocessing is not given, reference solution was the fastest.

designing new preprocessing techniques. This seems like a major obstacle in using preprocessing techniques in algebraic cryptanalysis of small-scale variants of AES in general as ciphers with more rounds behave pseudo-randomly which applies also to the original AES-128 with ten rounds similar to $SR(10, 4, 4, 8)$.

Conclusion

In this thesis, we have explored methods for algebraic cryptanalysis of small-scale variants of the symmetric AES cipher.

First, we have collected basic theory of non-standard algebraic geometry in finite fields used to construct Gröbner bases, a central object in the topic of algebraic cryptanalysis. We have also referenced advanced studies concerning complexity of computing such polynomial bases. Then, we have mentioned existing work in the area of algebraic cryptanalysis of AES and discussed our strategy.

In the main chapters, we have discovered a connection between previously used techniques of (linear) preprocessing used to simplify the polynomial systems and the field of lattices and linear codes. This was done by rephrasing the question of reducing the amount of monomials in the system as a decoding problem in linear codes. Subsequently, we have focused on a very recent effort in transferring certain lattice reduction algorithms to the domain of linear codes and hence, making the technique directly available to us.

As one of the main theoretical feats, we have given bounds on effectiveness of a class of preprocessing techniques known from previous works in terms of the Gilbert-Varshamov distance for pseudo-random ciphers. This bound has shown that usefulness of this class of preprocessing is limited in the case of pseudo-random ciphers.

To overcome the limitations imposed by the bound, we have designed three new types of preprocessing which can be used to simplify the polynomial system and aid in computing the Gröbner basis. Two of which are based on linear codes, one optimizing for sparsity of the polynomials and the second minimizing their maximal degrees. The third preprocessing is not linear and operates on the multiplicative structure of the polynomials. Another strength of the design is that these three preprocessings can be freely combined to enhance the strengths of each one. We have also provided and proven simple theoretical qualities for each one and provided visualisations of the various encountered problems.

Finally, we have demonstrated the usefulness of these preprocessing techniques in experiments. We have implemented a new modular system in SageMath to achieve faster execution times and support the variety of methods introduced in the thesis. By optimizing various parameters in the Gröbner basis computation, we have been able to obtain a fairly fast reference solution. Finally, we have compared the techniques designed in this thesis on top of the reference solution — to conclude that we have obtained faster solve times in nearly all aspects of the selected previous work and in some cases obtained faster solutions by a factor of 40, reducing the running time from hours to minutes.

For future work, we propose to incorporate generating the system via the meet-in-the-middle approach of [19], possibly enhanced with guess-and-determine technique in order to generate $SR(2, 4, 4, 8)$, the second round of AES.

Bibliography

[1] D. Aggarwal, D. Dadush, O. Regev, and N. Stephens-Davidowitz. Solving the shortest vector problem in 2^n time via discrete gaussian sampling. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 733–742. ACM, 2015.

[2] M. Ajtai. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 99–108, 1996.

[3] M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 284–293, 1997.

[4] M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 601–610. ACM, 2001.

[5] N. Aragon, J. Lavauzelle, and M. Lequesne. decodingchallenge.org, 2019.

[6] G. Ars, J.-C. Faugere, H. Imai, M. Kawazoe, and M. Sugita. Comparison between XL and Gröbner basis algorithms. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 338–353. Springer, 2004.

[7] M. Artin. *Algebra*. Pearson Education, 2011.

[8] R. B. Ash. *Information theory*. Courier Corporation, 2012.

[9] G. Bard. *Algebraic cryptanalysis*. Springer Science & Business Media, 2009.

[10] M. Bardet, J.-C. Faugere, and B. Salvy. *Complexity of Gröbner basis computation for Semi-regular Overdetermined sequences over F_2 with solutions in F_2*. PhD thesis, INRIA, 2003.

[11] A. Barg and G. D. Forney. Random codes: Minimum distances and error exponents. *IEEE Transactions on Information Theory*, 48(9):2568–2573, 2002.

[12] A. Becker, L. Ducas, N. Gama, and T. Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 10–24. SIAM, 2016.

[13] J. Berušková, M. Jureček, and O. Jurečková. Reducing overdefined systems of polynomial equations derived from small scale variants of the AES via data mining methods. *J. Comput. Virol. Hacking Tech.*, 20(4):885–900, 2024.

[14] M. Bielik, M. Jureček, O. Jurečková, and R. Lórencz. Yet another algebraic cryptanalysis of small scale variants of AES. In *Proceedings of the 19th International Conference on Security and Cryptography, SECRYPT 2022, Lisbon, Portugal, July 11-13, 2022*, pages 415–427. SCITEPRESS, 2022.

[15] N. Bose and N. Bose. *Gröbner bases: An algorithmic method in polynomial ideal theory*. Springer, 1995.

[16] W. Bosma, J. Cannon, and C. Playoust. The magma algebra system i: The user language. *Journal of Symbolic Computation*, 24(3-4):235–265, 1997.

[17] M. Bremner. *Lattice basis reduction*. CRC Press New York, 2011.

[18] M. Brickenstein and A. Dreyer. Polybori: A framework for gröbner-basis computations with boolean polynomials. *Journal of Symbolic Computation*, 44(9):1326–1345, 2009.

[19] S. Bulygin and M. Brickenstein. Obtaining and solving systems of equations in key variables only for the small variants of aes. *Mathematics in Computer Science*, 3:185–200, 2010.

[20] A. Caminata and E. Gorla. Solving multivariate polynomial systems and an invariant from commutative algebra. In *Arithmetic of Finite Fields: 8th International Workshop, WAIFI 2020, Rennes, France, July 6–8, 2020, Revised Selected and Invited Papers 8*, pages 3–36. Springer, 2021.

[21] C. Cid, S. Murphy, and M. Robshaw. *Algebraic aspects of the advanced encryption standard*. Springer Science & Business Media, 2006.

[22] D. Cox, J. Little, D. O'shea, and M. Sweedler. *Ideals, varieties, and algorithms*, volume 3. Springer, 2015.

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

[24] T. Debris-Alazard, L. Ducas, and W. P. van Woerden. An algorithmic reduction theory for binary codes: Lll and more. *IEEE Transactions on Information Theory*, 68(5):3426–3444, 2022.

[25] J.-C. Faugère. Fgb: a library for computing gröbner bases. In *International Congress on Mathematical Software*, pages 84–87. Springer, 2010.

[26] J.-C. Faugere and A. Joux. Algebraic cryptanalysis of hidden field equation (hfe) cryptosystems using gröbner bases. In *Annual International Cryptology Conference*, pages 44–60. Springer, 2003.

[27] J.-C. Faugère and C. Mou. Sparse fglm algorithms. *Journal of Symbolic Computation*, 80:538–569, 2017.

[28] W. Fulton. Algebraic curves. https://dept.math.lsa.umich.edu/~wfulton/CurveBook.pdf, 2008. [Online; accessed 5.1.2025].

[29] S. Gao. *Counting zeros over finite fields using Gröbner bases*. PhD thesis, Carnegie Mellon University, 2009.

[30] R. Germundsson. Basic results on ideals and varieties in finite fields. *Tech. Rep. S-581 83*, 1991.

[31] S. Ghentiyala and N. Stephens-Davidowitz. More basis reduction for linear codes: Backward reduction, bkz, slide reduction, and more. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 2024.

[32] J. H. Griesmer. A bound for error-correcting codes. *IBM Journal of Research and Development*, 4(5):532–542, 1960.

[33] J. Hoffstein. Ntru: A ring based public key cryptosystem. *Algorithmic Number Theory (ANTS III)*, 1998.

[34] M. Kudo and K. Yokoyama. The solving degrees for computing gröbner bases of affine semi-regular polynomial sequences. *Cryptology ePrint Archive*, 2024.

[35] D. Lazard. Gröbner bases, gaussian elimination and resolution of systems of algebraic equations. In *European Conference on Computer Algebra*, pages 146–156. Springer, 1983.

[36] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261:515–534, 1982.

[37] V. Lyubashevsky. Basic lattice cryptography: The concepts behind kyber (ml-kem) and dilithium (ml-dsa). *Cryptology ePrint Archive*, 2024.

[38] V. Lyubashevsky, L. Ducas, E. Kiltz, T. Lepoint, P. Schwabe, G. Seiler, D. Stehlé, and S. Bai. Crystals-Dilithium. *Algorithm Specifications and Supporting Documentation*, 2020.

[39] K. Mašková. Preprocessing techniques in algebraic cryptanalysis. *MSc. thesis*, 2024.

[40] R. J. McEliece. A public-key cryptosystem based on algebraic. *Coding Thv*, 4244:114–116, 1978.

[41] K. E. Morrison. Integer sequences and matrices over finite fields. *Journal of Integer Sequences*, 9(2):3, 2006.

[42] F. Olver. *Asymptotics and special functions*. AK Peters/CRC Press, 1997.

[43] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

[44] D. Salmond, A. Grant, I. Grivell, and T. Chan. On the rank of random matrices over finite fields. *arXiv preprint arXiv:1404.3250*, 2014.

[45] C. P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53(2-3):201–224, 1987.

[46] I. Semaev and A. Tenti. Probabilistic analysis on macaulay matrices over finite fields and complexity of constructing gröbner bases. *Journal of Algebra*, 565:651–674, 2021.

[47] D. Stanovský and L. Barto. *Počítačová algebra*. Matfyzpress, 2017.

[48] W. Stein et al. *Sage Mathematics Software (Version 10.4)*. The Sage Development Team, 2024. http://www.sagemath.org.

[49] J. H. Van Lint. *Introduction to coding theory*, volume 86. Springer Science & Business Media, 1998.

[50] N. Vanatta. Solving multi-variate polynomial equations in a finite field. *PhD. dissertation*, 2013.