FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Kristýna Mašková

Preprocessing Techniques in Algebraic Cryptanalysis

Computer Science Institute of Charles University

Supervisor of the master thesis:	Mgr. Pavel Hubáček, Ph.D.
Study programme:	Mathematics for Information Technologies
Study branch:	MITPN

Prague 2024

I am deeply grateful to my supervisor, Mgr. Pavel Hubáček, Ph.D., for his insights and guidance throughout the writing of this thesis, and to Mgr. Martin Jureček, Ph.D. for consulting and his invaluable ideas.

I am also deeply thankful to my parents for their unwavering support throughout my entire studies, and to Filip Chudoba for his patience and encouragement.

Finally, I would like to express my gratitude to FIT CTU and the Department of Information Security at FIT CTU for providing access to the server and license of the Magma software.

Title: Preprocessing Techniques in Algebraic Cryptanalysis

Author: Bc. Kristýna Mašková

Institute: Computer Science Institute of Charles University

Supervisor: Mgr. Pavel Hubáček, Ph.D., Computer Science Institute of Charles University

Abstract: Algebraic cryptanalysis is a standard set of techniques for analyzing and attacking practical symmetric cryptographic primitives. It involves representing the relationship between a pair of plaintext, ciphertext and the key as a system of polynomial equations and then solving the system using Gröbner bases. When the equations depend only on the key, we can generate multiple systems of equations.

This thesis examines preprocessing techniques in algebraic cryptanalysis, reducing large systems of equations to improve the performance of practical solving algorithms. Concentrating on a technique that aims to increase the sparsity of the polynomials, we lay the theoretical foundations for two methods. The first method of exhaustively going over all pairs and the second method of finding candidates for similar pairs using Locality-Sensitive Hashing. Finally, we improve on the latter method by targeting the leading monomials.

Keywords: Algebraic Cryptanalysis Gröbner bases AES Locality-Sensitive Hashing

Contents

Introduction **2**

1 Theoretical Introduction **4**

1.1 Polynomials, Ideals, and Varieties 4

1.2 Gröbner Bases 6

1.2.1 Computing Gröbner Bases 8

1.3 Algebraic Cryptanalysis 11

1.4 Advanced Encryption Standard 12

1.4.1 Small Scale Variants of AES 13

2 Preprocessing in Algebraic Cryptanalysis **14**

2.1 Correctness of Preprocessing 15

3 Theoretical Guarantees for Preprocessing **16**

3.1 Sparsity of Random Polynomial Systems 16

3.2 Locality-Sensitive Hashing 22

3.3 Reduction in Leading Monomials 27

4 Experiments **32**

4.1 Comparative Analysis of LSH and Exhaustive Comparison 32

4.2 Partial LSH Experiments 36

Conclusion **41**

Bibliography **42**

Introduction

In today's interconnected and data-driven world, the importance of cryptography cannot be overstated. Cryptography serves as the cornerstone of secure communication, enabling individuals, organizations, and governments to protect sensitive information from unauthorized access and malicious manipulation. At its core, cryptography is the study of techniques for secure communication in the presence of adversarial behaviour.

To ensure the quality and reliability of encryption algorithms, standardisation becomes essential. One of the most widely adopted cryptographic standards is the Advanced Encryption Standard (AES), a symmetric encryption algorithm designed to secure sensitive data and communications. Although the security of AES has been extensively studied, it is a good practice to test the security of encryption algorithms against new attacks continually.

Algebraic cryptanalysis is a technique used to attack cryptographic schemes by exploiting algebraic techniques such as solving systems of polynomial equations. One of the most widely used techniques involves leveraging Gröbner bases to solve systems of polynomial equations describing the cipher. Although algebraic cryptanalysis is a well-established concept the importance of this technique is sometimes underestimated. This is mostly due to the computational complexity of solving large systems of polynomial equations. However, with the emergence of a new class of symmetric primitives known as arithmetization-oriented ciphers (AOCs), algebraic cryptanalysis and more importantly Gröbner bases attacks are starting to be more relevant.

Arithmetization-oriented ciphers are symmetric primitives designed for optimization with respect to arithmetic complexity. This makes them efficient in protocols such as Zero-Knowledge proofs and secure Multi-party Computation. However, their design also allows for a concise description in terms of polynomials. Consequently, the systems of polynomials used in algebraic cryptanalysis are generally smaller and have a lower degree, rendering them more susceptible.

To illustrate the underestimation of algebraic cryptanalysis we can look at the primitive known as Jarvis, proposed in 2018 by Ashur and Dhooghe in [AD18]. Despite the initial consideration of a wide range of attacks, including some algebraic techniques like interpolation attacks, the security of Jarvis was compromised within a year of its release. A subsequent paper by Albrecht et al. [ACG$^+$19] demonstrated the inadequacy of the proposed parameters when considering Gröbner bases attacks. This serves as a reminder that we should not rely on the fact that some attacks have thus far been less efficient. We need to proactively assess the security of not only the recently proposed ciphers but also of those that have been in use for years.

Building on this principle, Bielik examined the algebraic cryptanalysis of AES in his thesis [Bie21, BJJL22]. He presented an improved algebraic attack against small-scale variants of AES. Using a single plaintext/ciphertext pair, they recovered the secret key for a one-round, weakened variant of full AES-128 in under one minute. They also observed that constructing multiple polynomial systems from additional plaintext/ciphertext pairs helped the performance of the computation. However, after adding more than five systems the time required to obtain the solution started to increase.

As a follow-up, Berušková [Ber22] experimented with generating large quantities of polynomial systems and preprocessing them. The main idea is to reduce the number of polynomial equations by adding pairs of polynomials. This results in sparser polynomials that could be better for solving. Berušková used several methods to find such pairs, the most successful one being Locality-Sensitive Hashing or LSH.

In this work, we continue with the examination of preprocessing in algebraic cryptanalysis by giving a theoretical foundation of thus far heuristic principles. With this foundation, we aim to explain the observed performance of LSH.

We begin our work by giving a succinct theoretical introduction to polynomials, ideals and varieties serving as a foundation for the introduction of Gröbner basis and AES in Chapter 1. The method of using Gröbner bases in algebraic cryptanalysis is also described in this chapter.

The current state of knowledge regarding the method of preprocessing in algebraic cryptanalysis is discussed in Chapter 2. In Section 2.1 we also argue the correctness of preprocessing.

Our main contribution lies in Chapters 3 and 4. In Section 3.1 we examine the primary idea of the method used for preprocessing. We give here some theoretical guarantees of the sparsity of the final system when using random sets of polynomials and we compare this with the values seen in experiments. We continue in Section 3.2 by giving a theoretical foundation to LSH, we deduce the exact parameters of the method used in experiments and we give further estimates for finding sparse polynomials with this method. In Section 3.3 we try to improve the method by modifying the existing methodology to target a specified portion of the polynomials. We encounter here a problem with implementation, give a solution to this problem and argue the error made by such a solution.

The last Chapter 4 discusses the results of our experiments. One of the goals of our experiments was to better understand the methods used. We give here the comparison of the basic method of adding all polynomials exhaustively to the method of using LSH to find candidates for similar polynomials. We present a graph of the sparsity of polynomials when using LSH. Finally, our second goal was to test our new method of targeting the leading monomials. We give results and interpretations of these tests.

1. Theoretical Introduction

In this chapter, we provide a concise background on the theory of Gröbner bases following the standard textbook by Cox, Little, and O'Shea [CLO15].

Gröbner bases were introduced in 1965 in the PhD thesis of Bruno Buchberger [Buc65], who named them in honour of his advisor Wolfgang Gröbner. Since then, many problems regarding polynomial ideals turned out to be reducible to finding the ideal's Gröbner basis. Some examples include ideal membership and solving polynomial equations, which is our main focus.

Buchberger also introduced an algorithm for computing Gröbner bases known as the Buchberger's algorithm. While straightforward to implement, its practical utility proved to be limited. Several proposals were made to improve the practical efficiency, such as the algorithms F4 [Fau99] and F5 [Fau02].

1.1 Polynomials, Ideals, and Varieties

To begin, we provide a succinct introduction to polynomials and ideals as a means to establish the notation in this work. Unless stated otherwise, k denotes a field throughout this whole chapter.

Definition 1.1. A *monomial* in variables $x_1, \ldots, x_n$ is a product of the form

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

where all the exponents α_i are nonnegative integers.

The *total degree* of a monomial $x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdot \ldots \cdot x_n^{\alpha_n}$ is defined as $\alpha_1 + \alpha_2 + \ldots + \alpha_n$.

For simplicity, we shorten the notation of a monomial to $x^\alpha = x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdot \ldots \cdot x_n^{\alpha_n}$ and we denote its degree as $|\alpha| = \alpha_1 + \alpha_2 + \ldots + \alpha_n$. We define the *multidegree* of a monomial x^α as $\operatorname{multideg}(x^\alpha) = \alpha$.

Definition 1.2. A *polynomial* in variables $x_1, \ldots, x_n$ over a field k is a finite sum

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

where the sum is over a finite number of n-tuples $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}_0^n$.

We use the following terminology:

- c_α is the *coefficient* of the monomial x^α,
- the *total degree* of f is defined as $deg(f) = \max_\alpha\{|\alpha| \mid c_\alpha \neq 0\}$,
- if $c_\alpha \neq 0$ then $c_\alpha x^\alpha$ is a *term* of f.

The set of all polynomials in variables $x_1, \ldots, x_n$ with coefficients from k is denoted by $k[x_1, \ldots, x_n]$.

It can be proved that $k[x_1, \ldots, x_n]$ with addition and multiplication is a ring, therefore we refer to it as the polynomial ring.

We have used the notation of monomials as being without coefficients and terms with coefficients. However, be aware that in some literature (for example in [Fau99, Fau02]) the definition of term and monomial can be reversed.

Definition 1.3. For k a field and n a positive integer, we can define an n-dimentional *affine space* over k as the set

$$k^n = \{(a_1, \dots, a_n) \mid a_1, \dots, a_n \in k\}.$$

A polynomial can also be interpreted as a function mapping the n-tuples $(a_1, \dots, a_n) \in k^n$ to elements of k, by replacing each occurrence of x_i in the polynomial with a_i.

Definition 1.4. An *ideal* I is a subset of $k[x_1, \dots, x_n]$ such that

1. $0 \in I$,
2. if $f, g \in I$, then $f + g \in I$,
3. if $f \in I$ and $h \in k[x_1, \dots, x_n]$, then $hf \in I$.

An ideal generated by polynomials $f_1, \dots, f_s \subset k[x_1, \dots, x_n]$ is a set

$$\langle f_1, \dots, f_s \rangle = \{\sum_{i=0}^{s} h_i f_i \mid h_1, \dots, h_s \in k[x_1, \dots, x_n]\}.$$

A proof that $\langle f_1, \dots, f_s \rangle$ is an ideal of $k[x_1, \dots, x_n]$ can be found in [CLO15, p. 29].

We say that an ideal I is *finitely generated* if there exists a finite number of polynomials $f_1, \dots, f_s \in k[x_1, \dots, x_n]$ such that $I = \langle f_1, \dots, f_s \rangle$ and we call $f_1, \dots, f_s$ the *basis* of I. A ring in which every ideal is finitely generated is called a *Noetherian ring.* From Hilbert's Basis Theorem, we get that every ideal of $k[x_1, \dots, x_n]$ is finitely generated.

Theorem 1.1 (Hilbert Basis Theorem)**.** *If R is a Noetherian ring, then $R[x]$ is a Noetherian ring.*

The proof can be found in [CLO15, p. 77].

Definition 1.5. Let $f_1, \dots, f_s$ be polynomials from $k[x_1, \dots, x_n]$. The *affine variety* defined by $f_1, \dots, f_s$ is a set

$$V(f_1, \dots, f_s) = \{(a_1, \dots, a_n) \in k^n \mid f_i(a_1, \dots, a_n) = 0 \,\text{for all}\, 1 \leq i \leq s\}.$$

Considering a set of polynomial equations

$$\begin{gathered} f_1(x_1, \dots, x_n) = 0 \\ f_2(x_1, \dots, x_n) = 0 \\ \vdots \\ f_s(x_1, \dots, x_n) = 0 \end{gathered}$$

the variety $V(f_1, \dots, f_s)$ is the set of all solutions to these equations in k^n.

Due to Hilbert's Basis Theorem, it also makes sense to consider a variety defined by an ideal.

Definition 1.6. Let $I \subset k[x_1, \dots, x_n]$ be an ideal. We denote by $V(I)$ the set

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

Theorem 1.2. *Let $I \subset k[x_1, \dots, x_n]$ be an ideal such that $I = \langle f_1, \dots, f_s \rangle$. Then $V(I)$ is an affine variety and $V(I) = V(f_1, \dots, f_s)$.*

For a proof see [CLO15, p. 81]

1.2 Gröbner Bases

In this section, we introduce Gröbner bases - a fundamental tool when working with polynomial rings with multiple variables. Gröbner basis is a particular basis of an ideal that has "nice" properties. These properties make them useful when working with multivariate polynomials and often allow us to solve problems about polynomial ideals in an algorithmic fashion. They are also one of the main practical tools for solving systems of polynomial equations in multiple variables.

Many algorithms need a way to determine an order in which to go through the monomials of a polynomial. For univariate polynomials, we have

$$\cdots > x^{m+1} > x^m > \cdots > x^2 > x > 1.$$

However, for multivariate polynomials things are not as clear. For example, how should the monomials x_1^2 and x_2^2 compare? To define such an order, let us denote Mon the set of all monomials in variables $x_1, \ldots, x_n$.

Definition 1.7. A *monomial ordering* $>$ on $k[x_1, \ldots, x_n]$ is a relation on the set of monomials Mon, such that

1. $>$ is a total ordering on Mon,
2. if $x^\alpha > x^\beta$ and $x^\gamma \in \text{Mon}$, then $x^\gamma x^\alpha > x^\gamma x^\beta$,
3. $>$ is a well-ordering on Mon; meaning that every non-empty subset of monomials has a smallest element.

The first condition allows us to compare every pair of monomials. The second condition then makes it so that multiplying polynomials will not change the relative ordering of terms. Finally, the last condition guarantees that every strictly decreasing sequence of monomials eventually terminates, which is important for proving that certain algorithms terminate.

Now, let us examine several examples of orders that are the most relevant to us.

Definition 1.8 (Lexicographic Order)**.** Let x^α and x^β be two monomials in $k[x_1, \ldots, x_n]$. In lexicographic (Lex) order, we have $x^\alpha >_{\text{Lex}} x^\beta$ if the leftmost nonzero coordinate of $\alpha - \beta$ is positive.

The lexicographic order represents rather a family of orders depending on how we decide to arrange the variables in α. Unless stated otherwise, we consider the most intuitive order of the variables, that is

$$x_1 > x_2 > \cdots > x_n$$

or

$$x > y > z,$$

depending on which labels we use for the variables.

Definition 1.9 (Graded Lexicographic Order)**.** Let x^α and x^β be monomials in $k[x_1, \ldots, x_n]$. In graded lexicographic (GrLex) order, we have $x^\alpha >_{\text{GrLex}} x^\beta$ if either $|\alpha| > |\beta|$ or $|\alpha| = |\beta|$ and $x^\alpha >_{\text{Lex}} x^\beta$.

The main difference between Lex and GrLex orders is that GrLex considers the degrees of the monomials first and if that is not a deciding factor it uses the lexicographic ordering. For example, in lexicographic order, we would have

$$x >_{\text{Lex}} y^5,$$

but in graded lexicographic order

$$x <_{\text{GrLex}} y^5.$$

A monomial order that first orders by degree is called a graded order. The GrLex order is a graded order, while Lex is not. Another example of a graded order is the graded reverse lexicographic order.

Definition 1.10 (Graded Reverse Lexicographic Order)**.** Let x^α and x^β be monomials in $k[x_1, \dots, x_n]$. In graded reverse lexicographic (GrevLex) order, we have $x^\alpha >_{GrevLex} x^\beta$ if either $|\alpha| > |\beta|$ or $|\alpha| = |\beta|$ and the rightmost nonzero coordinate of $\alpha - \beta$ is negative.

To explain the difference between GrLex and GrevLex, first note that both orders first look at the degree of the monomials. When the degrees are the same the GrLex looks for the largest leftmost power, while the GrevLex looks for the lowest rightmost power. You can notice, that

$$x_1 > x_2 > \cdots > x_n$$

is true in both orderings.

So far we have used subscript to indicate what monomial ordering we are working with. However, if the monomial order is clear from the context we omit the subscript.

Once we have chosen a monomial order, we can find a unique leading term of any polynomial. That in turn allows us to define the ideal generated by the leading terms of its polynomials.

Definition 1.11. Let $f = \sum_\alpha c_\alpha x^\alpha$ be a polynomial in $k[x_1, \dots, x_n]$ and $<$ a monomial order on $k[x_1, \dots, x_n]$.

- The *leading monomial* of f is $\mathrm{LM}(f) = \max\{x^\alpha \mid c_\alpha \neq 0\}$, where the maximum is taken with respect to $<$.
- The *leading coefficient* of f is $\mathrm{LC}(f) = c_\alpha$, where $\mathrm{LM}(f) = x^\alpha$.
- The *leading term* of f is $\mathrm{LT}(f) = \mathrm{LC}(f) \cdot \mathrm{LM}(f)$.
- The *multidegree* of f is $\mathrm{multideg}(f) = \mathrm{multideg}(\mathrm{LM}(f))$.

As an example, let us determine the leading monomial, coefficient, term and the multidegree of $f = x^2 + 7y^2 - 3xyz$ with respect to the Grevlex order

$$\begin{aligned} \mathrm{LM}(f) &= xyz, \\ \mathrm{LC}(f) &= -3, \\ \mathrm{LT}(f) &= -3xyz, \\ \mathrm{multideg}(f) &= (1, 1, 1). \end{aligned}$$

Definition 1.12. Let $I \in k[x_1, \ldots, x_n]$ be an ideal other than $\{0\}$. The set of leading terms is defined as

$$\mathrm{LT}(I) = \{\mathrm{LT}(f) \mid f \in I \setminus \{0\}\}.$$

The ideal $\langle \mathrm{LT}(I) \rangle$ is called the *ideal of leading terms.*

Importantly, the leading terms of an arbitrary basis of an ideal do not always generate the ideal of leading terms. The bases that do satisfy this are called Gröbner bases.

Definition 1.13. Let $<$ be a monomial order on $k[x_1, \ldots, x_n]$ and I an ideal of $k[x_1, \ldots, x_n]$ different from $\{0\}$. A finite subset $\mathcal{G} = \{g_1, \ldots, g_t\}$ of I is a *Gröbner basis* of I if $\langle \mathrm{LT}(I) \rangle = \langle \mathrm{LT}(g_1), \ldots, \mathrm{LT}(g_t) \rangle$ holds.

Proof of the following can be found in the fifth section of the second chapter of [CLO15].

As seen in the following theorem, all ideals in $k[x_1, \ldots, x_n]$ have a Gröbner basis.

Theorem 1.3. *For a fixed monomial ordering every ideal $I \in k[x_1, \ldots, x_n]$ has a Gröbner basis. Furthermore, any Gröbner basis is also a basis of that ideal.*

However, this basis does not have to be unique. For example, if we have $g_1, \ldots, g_t$ a Gröbner basis of I, then for any element $h \in I$ the set $g_1, \ldots, g_t, h$ is also a Gröbner basis. To get uniqueness we have to pose some additional restrictions on the basis.

Definition 1.14. A *reduced Gröbner basis* $\mathcal{G}$ of an ideal $I \in k[x_1, \ldots, x_n]$ is a Gröbner basis that satisfies:

1. $\mathrm{LC}(g) = 1$ for all $g \in \mathcal{G}$,
2. for all $g \in \mathcal{G}$, no monomial of g lies in $\langle \mathrm{LT}(\mathcal{G} \setminus \{g\}) \rangle$.

It is worth noting that every polynomial ideal has a unique reduced Gröbner basis. Most computer algebra systems compute the reduced Gröbner basis by default.

1.2.1 Computing Gröbner Bases

Thus far, we have seen that all polynomial ideals have a Gröbner basis. However, we have not seen how to find it. The first algorithm for computing Gröbner bases was introduced by Bruno Buchberger in his PhD thesis introducing Gröbner bases. While this algorithm is simple, it has a few issues limiting its efficiency. In this section, we introduce the main idea behind Buchberger's algorithm as well as its limitations. We also explore other algorithms that aim to overcome these limitations, most notably F4, which is one of the best algorithms implemented today in practical computer algebra software.

Two of the main definitions needed for the Buchberger's algorithm are reductions and S-polynomials.

Definition 1.15. Let $<$ be a monomial order on $k[x_1, \ldots, x_n]$, $I \in k[x_1, \ldots, x_n]$ an ideal, and $\mathcal{G} = \{g_1, \ldots, g_t\}$ its Gröbner basis. Then any polynomial $f \in k[x_1, \ldots, x_n]$ can be written as

$$f = q_1 g_1 + \ldots + q_t g_t + r,$$

where $q_i, r \in k[x_1, \ldots, x_n]$ and either $r = 0$ or no term of r is divisible by any $\mathrm{LT}(g_1), \ldots, \mathrm{LT}(g_t)$. Additionally, r is unique. We call r the *remainder* of f on division by $\mathcal{G}$.

It is worth noting that the uniqueness of r is not guaranteed for a general set of polynomials $g_1, \ldots, g_t$. This aspect is one of the advantageous properties of Gröbner bases. In fact, Gröbner bases can be characterized by the uniqueness of the remainder.

Definition 1.16. The *S-polynomial* of $f, g \in k[x_1, \ldots, x_n]$ is the combination

$$\mathrm{S}(f, g) = \frac{\mathrm{lcm}(\mathrm{LM}(f), \mathrm{LM}(g))}{\mathrm{LT}(f)} \cdot f - \frac{\mathrm{lcm}(\mathrm{LM}(f), \mathrm{LM}(g))}{\mathrm{LT}(g)} \cdot g$$

The S-polynomials then can be used to define a criterion for determining when a set of polynomials is a Gröbner basis.

Theorem 1.4 (Buchberger's Criterion)**.** *Let $I \in k[x_1, \ldots, x_n]$ be an ideal and $\mathcal{G} = \{g_1, \ldots, g_t\}$ its basis. Then $\mathcal{G}$ is a Gröbner basis of I if and only if the remainder of $\mathrm{S}(g_i, g_j)$ on division by $\mathcal{G}$ is zero for all $i \neq j$.*

Buchberger's Criterion is the foundation of the Buchberger's algoritm, whose pseudocode is written in Algorithm 1. In its most basic version, the algorithm continually computes the reduction of the S-polynomials of the partial Gröbner basis and adds the remainders to the Gröbner basis.

Algorithm 1 Buchberger's Algorithm

```
Input: Set of polynomials $F = \{f_1, \ldots, f_s\}$
Output: Gröbner basis $G$ for the ideal $I = \langle F \rangle$
1: $G' \leftarrow F$
2: $G = \emptyset$
3: while $G \neq G'$ do
4:     $G \leftarrow G'$
5:     for all $(g_i, g_j) \in G \times G$ do
6:         Compute the remainder $r$ of $S(f, g)$ on division by $G$
7:         if $r \neq 0$ then
8:             Add $r$ to $G$
9: return $G$
```

While this algorithm is straightforward, it does not perform very well in practice. The primary bottleneck stems from the fact that a considerable amount of the S-polynomials reduce to 0, resulting in substantial time spent on computations that do not add anything to the basis and thus, making them essentially useless.

Two strategies for accelerating the computation taken in algorithms F4 and F5 involve parallelizing the reductions and implementing criteria to identify instances when certain S-polynomials reduce to 0. The parallelization is done by constructing a matrix holding information about the S-polynomials and then reducing that matrix to a reduced echelon form. In this way, we get multiple reduction results simultaneously. The algorithm employing this strategy, introduced by Jean-Charles Faugère in [Fau99], is known as F4 and we present it in Algorithm 2.

Algorithm 2 F4 Algorithm

```
Input: Set of polynomials $F = \{f_1, \dots, f_s\}$
Output: Gröbner basis $G$ for the ideal $I = \langle F \rangle$
 1: $G \leftarrow F$
 2: $t \leftarrow s$
 3: $B = \{\{i, j\} \mid 1 \le i < j \le s\}$
 4: while $B \neq \emptyset$ do
 5:     Select $B' \neq \emptyset, B' \subseteq B$
 6:     $B = B \setminus B'$
 7:     $L = \left\{\frac{\mathrm{lcm}(\mathrm{LM}(f_i), \mathrm{LM}(f_j))}{\mathrm{LT}(f_i)} \cdot f_i \mid \{i, j\} \in B'\right\}$
 8:     $M = \mathrm{ComputeM}(L, G)$
 9:     $N$ = row reduced echelon form of $M$
10:     $N^+ = \{n \in \mathrm{rows}(N) \mid \mathrm{LM}(n) \notin \langle \mathrm{LM}(\mathrm{rows}(M)) \rangle\}$
11:     for all $n \in N^+$ do
12:         $t = t + 1$
13:         $f_t$ = polynomial form of $n$
14:         $G = G \cup \{f_t\}$
15:         $B = B \cup \{\{i, t\} \mid 1 \le i < t\}$
```

The algorithm uses a function ComputeM which is presented in Algorithm 3.

Algorithm 3 ComputeM algorithm from F4

```
Input: $L, G = \{f_1, \dots, f_t\}$
Output: Matrix $M$
 1: $H \leftarrow L$
 2: $done = \mathrm{LM}(H)$
 3: while $done \neq Mon(H)$ do
 4:     select largest $x^\beta \in Mon(H) \setminus done$ with respect to $<$
 5:     $done = done \cup \{x^\beta\}$
 6:     if there exists $f_l \in G$ such that $\mathrm{LM}(f_l) \mid x^\beta$ then
 7:         $H = H \cup \{\frac{x^\beta}{\mathrm{LM}(f_l)} \cdot f_l\}$
 8: $M$ = matrix of coefficients of $H$ with respect to $Mon(H)$, columns in de-
    creasing order according to $<$
 9: return $M$
```

The F4 algorithm depends on several choices like what strategy we employ to select the set B'. It is perhaps more accurate to talk about F4 as a family of algorithms.

1.3 Algebraic Cryptanalysis

Algebraic cryptanalysis is, by now, a classical topic developed by cryptographers in the 2000s [Bar09]. It is a class of cryptanalytic methods against practical symmetric ciphers such as the Advanced Encryption Standard [oST] that exploits the algebraic structure of a given cipher. In its most common form, it consists of two steps:

1. polynomial modelling,
2. solving the equations.

In the first step, given a pair of plaintext and its corresponding ciphertext, the cryptanalyst describes the encryption operation as a system of multivariate polynomial equations with variables corresponding to the bits of the secret key. The second step then consists of recovering the bits of the key by solving these equations.

The problem of solving a polynomial system of equations over a finite field is NP-Hard [Bar09, Corollary 79]. Despite this characterization, individual instances of these problems may not be that difficult to solve. Usually, algorithms for computing Gröbner bases are employed to solve these equations. The polynomials in the equations generate an ideal in an appropriate polynomial ring, and finding a Gröbner basis for that ideal with respect to a suitable monomial order allows solving for the key. This is a consequence of the following.

Definition 1.17. Let $I = \langle f_1, \dots, f_s \rangle$ be an ideal in $k[x_1, \dots, x_n]$ and let $l \in [0, n]$. The *l-th elimination ideal I_l* is an ideal of $k[x_{l+1}, \dots, x_n]$ defined by

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

The Gröbner basis of an l-th elimination ideal can be easily found with the help of the Gröbner basis of the whole ideal, as seen in the next theorem.

Theorem 1.5 (The Elimination Theorem)**.** *Let I be an ideal in $k[x_1, \dots, x_n]$ and let G be a Gröbner basis of I with respect to the lexicographic order. Then for every $l \in [0, n]$ the set*

$$G_l = G \cap k[x_{l+1}, \dots, x_n]$$

is a Gröbner basis of the l-th elimination ideal I_l.

Thus, computing the Gröbner basis in lexicographic order allows us to find a univariate polynomial. After solving such a polynomial we can substitute the solution into the rest of the polynomials and we again get a univariate polynomial. Not all solutions can be extended in such a way. However, if there is a solution, we can find it in this way.

Computing a Gröbner basis in lexicographic order is often costly in practice and the resulting polynomials can be quite large. For example, the Gröbner basis of the ideal $I = \langle x^5 + y^4 + z^3 - 1, x^3 + y^3 + z^2 - 1 \rangle$ in Lex order has 8 polynomials, one of which has 282 terms, is of degree 25 and has a coefficient $\frac{47780995}{7776}$. On the other hand, its Gröbner basis in GrevLex order has just 3 elements, the longest polynomial has 9 terms and the highest degree is 6. In practice, if we require the lexicographic Gröbner basis most computer algebra systems first compute the

Gröbner basis with respect to the GrevLex order and then use an algorithm to convert it to the Gröbner basis with respect to the lexicographic order such as FGLM [FGLM93]. With this in mind, the general process of algebraic cryptanalysis looks as follows:

1. Model the cipher as a system of polynomial equations.
2. Compute the Gröbner basis w.r.t. the GrevLex order.
3. Change the order to lexicographic by using an algorithm like FGLM.
4. Read out the solution.

There are several proposals of optimized algorithms for finding a Gröbner basis, such as F4, F5 and XL [SS21]. However, when applying the algebraic attacks in practice, we are restricted to the algorithm F4 as it is the only one with an efficient implementation provided in the highly optimized software Magma.

1.4 Advanced Encryption Standard

Advanced Encryption Standard, also known by its original name Rijndael, is a widely used symmetric encryption algorithm adopted by the U.S. government for securing sensitive information [oST]. It operates on fixed block sizes of 128 bits and supports key sizes of 128, 192, and 256 bits. Our focus is on the version with key size set to 128, referred to as AES-128. We give here a very high-level description of the AES and refer the reader to the official standard [oST] for a thorough description.

AES employs a substitution-permutation network (SPN) structure, which provides a high degree of confusion and diffusion. It works with the state as a 4×4 matrix, where each entry is 8-bits long. We refer to 32-bits as a word and thus, the matrix has exactly one word in each column or row.

A pseudocode of the structure is illustrated in algorithm 4. The algorithm starts by running a KeySchedule on the original 128-bit key to get an expanded key of 44 words - 4 words for each round and 4 additional words for the initial key addition. The 4 initial words are the original key.

The input of the cipher is divided into 16 blocks of 8 bits and copied into the state and then the initial 4 words of the key are added. The cipher then repeats 10 rounds iterating over 4 operations. The SubBytes operation is a non-linear substitution step, that operates on each element of the matrix separately. We refer to it as an S-box and it is usually implemented as a lookup table. The S-box used is derived from an inverse function combined with an invertible affine transformation. After the SubBytes comes the ShiftRows operation. It is a transposition step that works on the rows of the matrix, shifting each row cyclically by a certain offset. The MixColumns step on the other hand operates on the columns of the matrix, multiplying each column by a matrix. This step is omitted in the last round, which makes the encryption and decryption more similar and it does not reduce the security of the cipher. The last operation is the AddRoundKey, it operates on elements of the matrix, adding to each byte a byte of the expanded key.

Algorithm 4 High-level overview of AES

```
Input: Plaintext $PT$, Key $K$
Output: Ciphertext $CT$
 1: ExpandKey($K$)
 2: $state \leftarrow$ AddRoundKey($PT, RoundKey[0]$)
 3: for $i$ from 1 to 9 do
 4:     $state \leftarrow$ SubBytes($state$)
 5:     $state \leftarrow$ ShiftRows($state$)
 6:     $state \leftarrow$ MixColumns($state$)
 7:     $state \leftarrow$ AddRoundKey($state, RoundKey$[i])
 8: $state \leftarrow$ SubBytes($state$)
 9: $state \leftarrow$ ShiftRows($state$)
10: $state \leftarrow$ AddRoundKey($state, RoundKey[10]$)
11: return $state$
```

1.4.1 Small Scale Variants of AES

In this section, we describe scaled-down variants of AES introduced by Cid, Murphy, and Robshaw in [CMR05, CMR06]. Their modelling differs from the standard AES by keeping the MixColumns operation in the last round. The reductions emerge naturally with the consideration of the state matrix. The new cipher is described by parameters:

- n, the number of rounds,
- r, the number of rows of the state matrix,
- c, the number of columns,
- e, the length of the elements in the matrix in bits,

and we denote the small scale version as $\mathrm{SR}(n, r, c, e)$. A rigorous definition would require us to define each operation for the small scale version. This is out of the scope of this thesis and we refer the reader to [CMR05] for a more precise description.

The parameters we consider are $n \in [1, 10]$ for the number of rounds, both $r, c \in \{1, 2, 4\}$ and $e \in \{4, 8\}$. The $\mathrm{SR}(10, 4, 4, 8)$ would then be the standard AES with the only difference being the added MixColumns operation in the last round.

The small-scale variant, represented as a system of polynomials, is comprehensively described in [Bie21, BJJL22]. Our focus lies primarily on the systems with no auxiliary variables. This configuration enables us to consider multiple plaintext/ciphertext pairs for one key and get multiple polynomial systems.

2. Preprocessing in Algebraic Cryptanalysis

A recent line of research by the group of Martin Jureček, Ph.D. from FIT ČVUT introduced the possibility of using preprocessing of the polynomial system obtained in the first step of algebraic cryptanalysis to improve the efficiency of the whole attack. Recall that the algebraic cryptanalysis consists of two steps:

1. Polynomials modelling,
2. Solving the equations.

In the first step, given a pair of plaintext and its corresponding ciphertext, the cryptanalyst describes the encryption operation as a system of multivariate polynomial equations with variables corresponding to the bits of the secret key. Because the system depends on the plaintext/ciphertext pair generated we can consider generating multiple plaintext/ciphertext pairs for the same key. This gives us more polynomial equations to work with.

The work of Bielik et al. [Bie21, BJJL22] presented an improved algebraic attack against small-scale variants of AES. Using a single plaintext/ciphertext pair, they recovered the secret key for a one-round, weakened variant of full AES-128 in under one minute. More importantly to us, they observed that constructing more equations from additional plaintext/ciphertext pairs helped the performance of the computation. However, after adding more than five systems (i.e., using five plaintext/ciphertext pairs), the time required to obtain the solution started to increase.

As a follow-up, Berušková [Ber22] experimented with preprocessing the systems of polynomial equations created from large quantities of plaintext/ciphertext pairs. She used three methods: clustering based on Partitioning Around Medoids (PAM), Locality-Sensitive Hashing (LSH), and direct elimination of the leading monomial, where LSH proved to be the most effective approach. All of the methods attempted to lessen the number of monomials in the polynomials given as input to the solving algorithm, as this parameter is believed to have the biggest effect on the time complexity.

The first method PAM partitions the polynomials into clusters, then takes two of the most similar polynomials from each cluster and xors them. The LSH approach works by hashing the polynomials in a way that preserves the distance. If two polynomials hash to the same output, they are marked as "possibly similar" and their xor is used in the computation. Finally, the last method simply takes a certain number of polynomials with the lowest leading monomial.

The most effective algorithm out of the three appears to be the LSH. Importantly, the methods used are purely heuristic. Understanding how the preprocessing helps in the computation would allow us to design more effective methods.

Jureček's group primarily investigated the "classical" cipher AES. Although ciphers like the Arithmetization Oriented Ciphers (AOCs) may potentially be more susceptible to algebraic attacks, our analysis will continue with the examination of the AES cipher. Our objective is to establish a theoretical foundation of the LSH method used by Berušková and to leverage the existing codebase as a cornerstone for our experiments.

2.1 Correctness of Preprocessing

Using preprocessing in algebraic cryptanalysis involves modifying the original set of equations by adding pairs of polynomials, followed by selecting a subset of these pairs. Could this process potentially destroy information about the correct solution? In this section, we want to clarify why this process is correct and why the correct key can still be determined as a solution to these equations.

Let us start by examining the original set of polynomials $\mathcal{F}$ in $k[x_1, \dots, x_n]$ created by modelling a cipher and let $K = (k_1, \dots, k_n) \in k^n$ be the secret key. The set of all solutions to $\mathcal{F}$ forms a variety $V(\mathcal{F})$ and it is straightforward to see that the secret key lies in this variety.

Lemma 2.1. *Let $\mathcal{F} \subseteq k[x_1, \dots, x_n]$ be a system of polynomials created by modelling a cipher and let $K = (k_1, \dots, k_n) \in k^n$ be the secret key. Then $K \in V(\mathcal{F})$.*

Proof. We need to show that, for an arbitrary $f \in \mathcal{F}$, the key satisfies $f(K) = 0$. This follows directly from the fact that K is the secret key to the cipher modelled by $\mathcal{F}$ that was used to create the plaintext/ciphertext pair. $\square$

Consider a new system of equations $\mathcal{G}$ created by taking linear combinations of polynomials from $\mathcal{F}$, that is $g \in \mathcal{G}$ then $g = \sum_{i=0}^{s} g_i \cdot f_i,\ g_i \in k[x_1, \dots, x_n], f_i \in \mathcal{F}$.

We want to examine the relationship between the varieties $V(\mathcal{F})$ and $\mathcal{G}$. The next theorem shows that this construction preserves all solutions of the original system $\mathcal{F}$.

Theorem 2.2. *Let $\mathcal{F}$ and $\mathcal{G}$ be two systems of polynomials defined as above. Then $V(\mathcal{F}) \subseteq V(\mathcal{G})$.*

Proof. The original system forms an ideal $\langle \mathcal{F} \rangle$. We have seen in Theorem 1.2 that $V(\mathcal{F}) = V(\langle \mathcal{F} \rangle)$. We know that $\mathcal{G} \subseteq \langle \mathcal{F} \rangle$ from the fact that ideals are closed under addition and multiplication by an element from the parent ring. This gives us $\langle \mathcal{G} \rangle \subseteq \langle \mathcal{F} \rangle$.

From $\langle \mathcal{G} \rangle \subseteq \langle \mathcal{F} \rangle$ we can finally deduce $V(\mathcal{G}) \subseteq V(\mathcal{F})$. $\square$

The other inclusion does not have to be true. To illustrate, let us consider $f_1 = 2x + 2y$ and $f_2 = 3x + 3y$ in $\mathbb{Z}_5[x, y]$ and $g = f_1 + f_2$. We have $a = (1, 1) \in V(g)$ because $g(a) = f_1(a) + f_2(a) = 2 + 2 + 3 + 3 = 0$, but $f_1(a) = 2 + 2 = 4$, $f_2(a) = 3 + 3 = 1$.

This demonstrates that preprocessing is a viable technique that could be used in algebraic cryptanalysis. Although the variety of our new system may be larger than the original, it nevertheless contains the sought-after key.

3. Theoretical Guarantees for Preprocessing

In algebraic cryptanalysis, we construct polynomial equations describing the cipher we are analysing. These equations encapsulate the relationship between the plaintext, ciphertext, and the encryption key. Solving the system using Gröbner bases then gives us the key.

The polynomial system generated depends on the plaintext/ciphertext pair used. Thus, generating more plaintext/ciphertext pairs for a fixed key gives us more equations. This gives us more information about the key, and, intuitively, it should help the attacker to find the key quicker. However, larger numbers of plaintext/ciphertext pairs help the cipher only initially.

In the experiments carried out by Bielik [Bie21, BJJL22], it was observed that taking two pairs of plaintext/ciphertext leads to generally quicker solving times. However, as the number of plaintext/ciphertext pairs increased, the time required for solving also increased. A natural explanation for such behaviour is that the F4 algorithm necessitates investigating all pairs of input polynomials. Specifically, for s polynomials in the input, the algorithm must process at least $\binom{s}{2}$ pairs. The number s of polynomials on the input is typically quite large, making the quadratic growth significant.

Building on this observation, Berušková [Ber22] experimented with preprocessing the systems of polynomial equations created from large quantities of plaintext/ciphertext pairs. The primary objective was to reduce the number of equations while simultaneously enhancing the suitability of the polynomials for the F4 algorithm.

One approach is to reduce the number of monomials in the polynomials. The intuition behind this is that, in the ComputeM algorithm used in F4, we have to go through all of the monomials present in the input polynomials. Crafting sparser polynomials, containing fewer monomials, could thus lead to having to go through less monomials. One method of decreasing the number of nonzero monomials is adding each pair of polynomials in the system and working with the resulting polynomials with the fewest monomials - hoping for significant cancellations of monomials. Since we work over $\mathbb{Z}_2$ this corresponds to performing elementwise XOR of the vectors representing the polynomials.

In the context of a "good" cipher, the polynomial system derived from encrypting the plaintext/ciphertext pair should exhibit pseudorandom characteristics, resembling the properties of a fully random polynomial system. Newtx, we examine the sparsity that can be achieved by XORing a fully random polynomial system over $\mathbb{Z}_2$.

3.1 Sparsity of Random Polynomial Systems

Let us have a set of polynomials $V = \{v_1, \ldots, v_n\}$ from $\mathbb{Z}_2^m$ and say we want k polynomials as an input for the F4 algorithm. Using the method explained above of XORing every pair of polynomials from this set, then picking the ones with the lowest number of monomials, we want to examine how many polynomials we need in n to pick on average at least k polynomials with a certain level of sparsity.

We can describe a level of sparsity after XORing with the Hamming distance.

Definition 3.1. Let $v, w \in \mathbb{Z}_2^m$ be two vectors. Their *Hamming distance* is

$$d(v, w) = |\{i \in [1, m] \mid v_i \neq w_i\}|,$$

where $v = (v_1, \ldots, v_m)$ and $w = (w_1, \ldots, w_m)$.

Say we have two polynomials $p, q \in \mathbb{Z}_2[x_1, \ldots, x_n]$. We can convert these polynomials into vectors by assigning a number to each monomial from 1 to the number of distinct monomials in both polynomials and then create a vector in $\mathbb{Z}_2^m$ for each polynomial, where there is a 1 in position i if and only if the i-th monomial is in that polynomial.

Now say two vectors $v_p, v_q \in \mathbb{Z}_2^m$ corresponding to two polynomials p, q have a hamming distance $d(v_p, v_q) = m/2$. Then the XOR of those polynomials will have exactly $m/2$ monomials. That is due to the fact that we are working over $\mathbb{Z}_2$ and so if the two vectors differ in an element, meaning in one polynomial the corresponding monomial is present and in the other polynomial it is not then, after XORing, the monomial will be present in the new polynomial. On the other hand, if the vectors agree in an element, the corresponding monomial either is present in both of the polynomials or in neither of them and in both cases the monomial will not be present in the XOR.

We can thus examine the situation where we have a fixed $k \in \mathbb{N}$ and $r \in [0, m]$ where k stands for the number of polynomials we want to find that have a Hamming distance less than or equal to r. A similar concept was explored in [LMRS12], when investigating general methods of finding near-collisions in cryptographic hash functions. They defined an ϵ-near-collision for a pair of messages m, m^* under a hash function H as instances where $d(H(m), H(m^*)) \leq \epsilon$. We aim to investigate when the Hamming distance $d(v, w)$ is less than or equal to r for two random vectors v and w from $\mathbb{F}_2^m$. To accomplish this, we adopt the methodology outlined in Section 3.2 of [LMRS12].

Let $V = \{v_1, \ldots, v_n\}$ be a set of uniformly random polynomials from $\mathbb{Z}_2^m$. For the hamming distance d, we can define a hamming sphere of radius r of a polynomial $x \in \mathbb{Z}_2^m$ as

$$B_r(x) = \{y \in \mathbb{Z}_2^m : d(x, y) \leq r\}$$

and its volume as

$$V(m, r) = |B_r(x)| = \sum_{i=0}^{r} \binom{m}{i}.$$

The $V(m, r)$ function represents the number of all vectors of $\mathbb{Z}_2^m$ that have the hamming weight at most r or that have at most r distance from an arbitrary vector $x \in \mathbb{Z}_2^m$. This means that the probability that two random vectors from $\mathbb{Z}_2^m$ have a Hamming distance at most r is $V(m, r)/2^m$.

For a pair of vectors $v_i, v_j \in \mathbb{Z}_2^m$, let us consider the random variable $d(v_i, v_j)$ and the characteristic function χ of the event $d(v_i, v_j) \leq r$, defined as

$$\chi(d(v_i, v_j) \leq r) = \begin{cases} 1, & \text{if } d(v_i, v_j) \leq r \\ 0, & \text{otherwise.} \end{cases}$$

The number of pairs from V such that their distance is at most r can be expressed as a random variable

$$N_V(r) = \sum_{i=1}^{n} \sum_{j=1}^{i-1} \chi(d(v_i, v_j) \leq r).$$

We are interested in the expected number of pairs of distance at most r.

Theorem 3.1. *With the notation introduced above, the expected value of $N_V(r)$ is*

$$\mathsf{E}(N_V(r)) = \binom{n}{2} \frac{V(m,r)}{2^m}.$$

Proof. Let us first examine the expected value of χ.

$$\mathsf{E}\left(\chi(d(v_i, v_j) \leq r)\right) = 0 \cdot \mathsf{Pr}[d(v_i, v_j) > r] + 1 \cdot \mathsf{Pr}[d(v_i, v_j) \leq r] = \frac{V(m,r)}{2^m}.$$

Using the linearity of expectation, we can now compute the expected value of $N_V(r)$:

$$\begin{aligned} \mathsf{E}\left(N_V(r)\right) &= \mathsf{E}\left(\sum_{i=1}^{n} \sum_{j=1}^{i-1} \chi(d(v_i, v_j) \leq r)\right) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{i-1} \mathsf{E}\left(\chi(d(v_i, v_j) \leq r)\right) \\ &= \binom{n}{2} \frac{V(m,r)}{2^m}. \end{aligned}$$

□

The above was used in [LMRS12] to analyse the complexity of finding ϵ-near-collitions using a Birthday-like ϵ-near-collision search. Building upon this foundation, we can leverage similar techniques to investigate the number of vectors in V necessary to establish a lower bound for the expected value of r-near pairs.

Theorem 3.2. *To have the average number of vectors from V that have distance at most r be at least k, we need the set V of size at least*

$$|V| \geq \frac{\sqrt{k} \cdot 2^{m/2}}{\sqrt{V(m,r)}}.$$

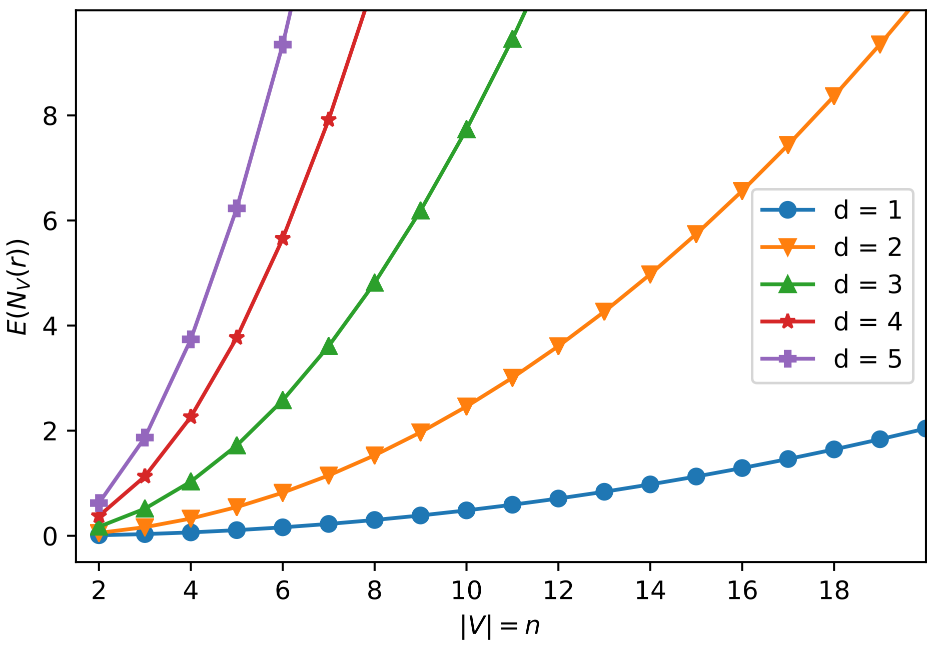

Figure 3.1: Expected value of the number of pairs closer than d, for m = 10

Proof. To see this, let us look at the expected value of $N_V(r)$.

$$\begin{aligned}
\mathsf{E}\left(N_V(r)\right) &\geq k \\
\binom{n}{2}\frac{V(m,r)}{2^m} &\geq k \\
\binom{n}{2} &\geq \frac{k \cdot 2^m}{V(m,r)} \\
n^2 - n &\geq \frac{k \cdot 2^{m+1}}{V(m,r)} \\
n^2 &\geq \frac{k \cdot 2^{m+1}}{V(m,r)} \\
n &\geq \frac{\sqrt{k} \cdot 2^{m/2}}{\sqrt{V(m,r)}}
\end{aligned}$$

□

In Figure 3.1, we can see the value of $\mathsf{E}\left(N_V(r)\right)$ plotted for n from 2 to 20 and different values of d and for a fixed value of $m = 10$. When we fix a value $k = \mathsf{E}\left(N_V(r)\right)$ of how many polynomial pairs we want to find on average we can see that as n increases the distance of the found vectors decreases. For example when we would like to obtain $k = 6$ vector pairs on average, when we have $n = 6$ we would have to set d as 5, meaning we would want the vectors to be different in at most 1/2 elements. When we increase n to 18, we can now set d to 2, meaning at most 1/5 of the vectors is going to be different.

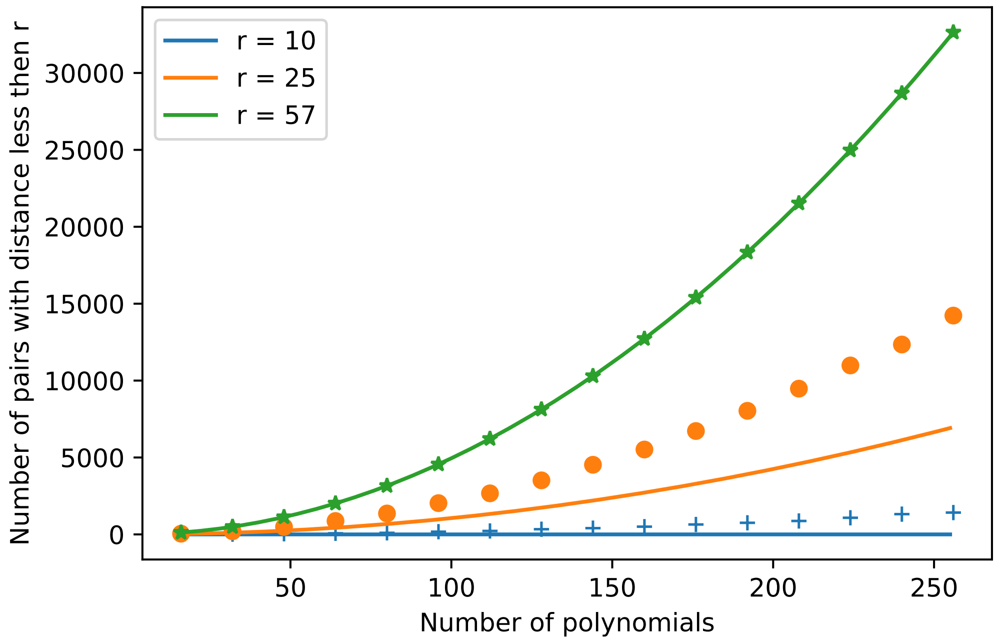

Figure 3.2: Number of polynomial pairs from the SR(1,2,2,4) cipher with distance less than r. The values indicated by lines represent the expected values, while the dots represent the experimental values.

While having the expected value of a number of pairs closer than r is important, the above results assume uniformly random polynomial systems. We want to compare these findings to systems generated for AES. We ran a series of experiments on the cipher SR(1,2,2,4) for various values of plaintext/ciphertext pairs, looking at the distances of pairs of polynomials. The results can be found in Figure 3.2, where they are presented in the same way as Figure 3.1, that is on the x-axis we have the number of polynomials in the system (or $|V| = n$) and the y-axis indicates two distinct instances. The dots, stars and pluses indicate the number of pairs found in these experiments with distance at most r, while the lines represent the expected value $\mathsf{E}\left(N_V(r)\right)$ of the number of pairs. With the vector length of $m = 57$ we picked $r = 10, 25, 57$ to illustrate different values while ensuring legibility.

Observing Figure 3.2 we note that for lower and higher values of r the experimental values closely align with the expected values. However, for the intermediate values of r, there are discrepancies between the experimental and expected values. We suspect this arises from the fact that for parameter as low as in SR(1,2,2,4) the polynomial system is not pseudorandom. With more complex ciphers like SR(10,4,4,8), we would expect a closer alignment between the experimental values and the expected values. It is, however, important to acknowledge that due to time constraints, we could only perform one run of the experiments, which may contribute to the observed variance.

3.2 Locality-Sensitive Hashing

As we mentioned at the start of the chapter, our goal is to reduce the number of polynomials from multiple polynomial systems describing a cipher. We aim to do this by adding all of the polynomials and using a certain number of the shortest polynomials.

However, if we have s polynomials in our system then adding every pair would require going through $\binom{s}{2}$ pairs, leading to a quadratic overhead. To address this challenge, Berušková [Ber22] explored a few strategies on how to mitigate this, with the most promising approach being Locality-Sensitive Hashing or LSH. Next, we introduce the necessary definitions for LSH following the presentation in [Wyl].

Definition 3.2. Let $\mathcal{F}$ be a family of functions $h : M \to S$, d a distance measure on M and $d_1 < d_2$ two distances in this measure. We say that $\mathcal{F}$ is $(d_1, d_2, p_1, p_2)-$ sensitive if for all $h \in \mathcal{F}$ and all $x, y \in M$:

- if $d(x, y) \leq d_1$ then $\Pr[h(x) = h(y)] \geq p_1$,
- if $d(x, y) \geq d_2$ then $\Pr[h(x) = h(y)] \leq p_2$.

A $(d_1, d_2, p_1, p_2)-$ sensitive family is called an *LSH family.*

Locality-Sensitive Hashing is a method of determining which items in a given set are similar. It accomplishes this by hashing similar items to the same elements with high probability. An element of the output is generally called a bucket. Since similar items tend to end up in the same buckets, we can use this technique to find vectors that are similar without going through all of the pairs. This enables us to reduce the number of comparisons to just the pairs in the same bucket.

LSH is versatile in its application, as it can accommodate various distance measures to quantify similarity. In our context, we leverage the Jaccard similarity measure, which assesses the degree of similarity between two sets.

Definition 3.3. Let A and B be finite sets. Then the *Jaccard similarity* is defined as

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

Jaccard similarity indicates what portion of two sets is the same. It is 1 if the sets are the same, 0 if they have no elements in common and somewhere in between otherwise.

This however does not meet the definition of distance. To rectify this, we need to reverse the interval to adhere to the conventions of distance measures.

Definition 3.4. Let A and B be finite sets and $J(A, B)$ their Jaccard similarity. Then the *Jaccard distance* is defined as $1 - J(A, B)$.

With this, we can prove that Jaccard distance is a distance.

Lemma 3.3. *Jaccard distance is a distance measure on the collection of all finite sets.*

Proof. A distance measure d has to meet these four definitions:

1. $d(x, x) = 0$,
2. $d(x, y) > 0$ when $x \neq y$
3. $d(x, y) = d(y, x)$,
4. $d(x, z) \leq d(x, y) + d(y, z)$.

Let d be the Jaccard distance and A, B, C three finite sets. For the first equation, we have $d(A, A) = 1 - J(A, A) = 1 - \frac{|A \cap A|}{|A \cup A|} = 0$.

The second equation holds because $|A \cap B| < |A \cup B|$ when $A \neq B$. The third equation holds because $|A \cap B| = |B \cap A|$ and $|A \cup B| = |B \cup A|$.

Proving the last equation can be quite cumbersome, thus we refer to [Kos19] for a complete proof. □

We can interpret our polynomials as sets of monomials. Because we are working with polynomials in $\mathbb{Z}_2$ this also incorporates the coefficients. Comparing two polynomials then reduces to computing the Jaccard distance of the corresponding sets.

The LSH method using Jaccard distance as a measure uses a family of functions called MinHash. MinHash was first introduced by Andrei Broder in [Bro97] to detect duplicate web pages.

Definition 3.5. Let h be a hash function that maps the members of a set U to distinct integers, let p be a random permutation of the elements of U, and for any subset $S \subseteq U$ define $h_{\min}(S)$ to be the minimal member of S with respect to $h \circ p$—that is, the member $x \in S$ with the minimum value of $h(p(x))$. The family of functions h_{min} defined as above is called a MinHash family.

Let us provide an example to better illustrate the MinHash family.

Example 3.6. Let $f_1 = xy + z$ and $f_2 = x^2 + z$ be two polynomials in $\mathbb{Z}_2[x, y, z]$. The set U in this case would represent the set of all present monomials, hence $U = \{x^2, xy, z\}$ and we can equate the subsets $S_1 = \{xy, z\}$ and $S_2 = \{x^2, z\}$ to the polynomials f_1, f_2.

Let us take a hash function $h : U \to \mathbb{Z}$ such that $h(x^2) = 1, h(xy) = 2$ and $h(z) = 3$ and a permutation $p : U \to U$ such that $p(x^2) = z, p(xy) = xy$ and $p(z) = x^2$. Then

$$h_{\min}(S_1) = \min\{h(p(xy)), h(p(z))\} = \min\{2, 1\} = 1,$$
$$h_{\min}(S_2) = \min\{h(p(x^2)), h(p(z))\} = \min\{3, 1\} = 1.$$

Hence, we get the same hash for both polynomials.

To see that MinHash functions are an LSH family we need the following lemma.

Lemma 3.4. *Let A and B be two finite sets and $J(A, B)$ their Jaccard similarity. Than the probability that $h_{min}(A) = h_{min}(B)$ is equal to $J(A, B)$.*

Proof. We can see that $h_{min}(A) = h_{min}(B)$ if and only if among all elements of $A \cup B$ the element with the minimum hash value lies in both A and B. Therefore we have $\Pr[h_{min}(A) = h_{min}(B)] = J(A, B)$. □

Corollary 3.5. *A family of MinHash functions is $(d_1, d_2, 1-d_1, 1-d_2)-$sensitive.*

Proof. Let $h : M \to S$ be a hash function from the MinHash function family.

Suppose that the Jaccard similarity of two elements $A, B \in M$ is at least s, then their Jaccard distance is at most $d_1 = 1 - s$. By 3.4 we have that $\Pr[h(A) = h(B)] \geq s = 1 - d_1$.

On the other hand, suppose that the Jaccard distance of A and B is at least d_2. Then their Jaccard similarity is at most $s = 1 - d_2$ and thus $\Pr[h(A) = h(B)] \leq s = 1 - d_2$. □

The overarching idea, as explained above, is to hash polynomials in such a way that similar polynomials are more likely to end up in the same bucket and dissimilar polynomials are more likely to end up in different buckets. To have a way of altering that probability we use multiple hash functions from MinHash on a single polynomial. By adjusting the parameters of the combinations of the MinHash functions, we can affect the likelihood of polynomials being grouped together based on their similarity.

Suppose we have a (d_1, d_2, p_1, p_2)-sensitive family of hash functions $\mathcal{F}$. We can construct new families of functions in two ways. An AND-family of functions $\mathcal{G}$ where each $g \in \mathcal{G}$ is constructed by using k independently uniformly chosen functions $f_1, \ldots, f_k \in \mathcal{F}$ and we say that $g(x) = g(y)$ if and only if $f_i(x) = f_i(y)$ for all $i \in [1, k]$.

We can similarily construct an OR-family $\mathcal{G}$ where each $g \in \mathcal{G}$ is again constructed by using k independently uniformly chosen functions $f_1, \ldots, f_k \in \mathcal{F}$ and $g(x) = g(y)$ if and only if $f_i(x) = f_i(y)$ for at least one $i \in [1, k]$.

The properties of these constructions are summarised in the next theorem, for which we present a self-contained proof.

Theorem 3.6. *Let $\mathcal{F}$ be a (d_1, d_2, p_1, p_2)-sensitive family of hash functions. An AND-family of hash functions constructed from $\mathcal{F}$ by using k functions is a (d_1, d_2, p_1^k, p_2^k)-sensitive family.*

An OR-family of hash functions constructed from $\mathcal{F}$ by using k functions is $(d_1, d_2, (1 - (1 - p_1)^k), (1 - (1 - p_2)^k))$-sensitive.

Proof. When $d(x, y) \leq d_1$ we have for all $i \in [1, k]$ that $\mathsf{Pr}[f_i(x) = f_i(y)] \geq p_1$. Let us first look at the sensitiveness of the AND-construction.

Because all f_i are chosen independently uniformly from $\mathcal{F}$ we have

$$\begin{aligned}\mathsf{Pr}[g(x) = g(y)] &= \mathsf{Pr}[f_i(x) = f_i(y) \text{ for all } i \in [1, k]] \\ &= \prod_{i=1}^{k} \mathsf{Pr}[f_i(x) = f_i(y)] \geq p_1^k.\end{aligned}$$

For the OR-construction we have

$$\begin{aligned}\mathsf{Pr}[g(x) = g(y)] &= \mathsf{Pr}[f_i(x) = f_i(y) \text{ for at least one } i \in [1, k]] \\ &= 1 - \mathsf{Pr}[f_i(x) \neq f_i(y) \text{ for all } i \in [1, k]] = 1 - \prod_{i=1}^{k} \mathsf{Pr}[f_i(x) \neq f_i(y)] \\ &\geq 1 - (1 - p_1)^k.\end{aligned}$$

The proof of the case when $d(x, y) \geq d_2$ is analogous to the preceding proofs. □

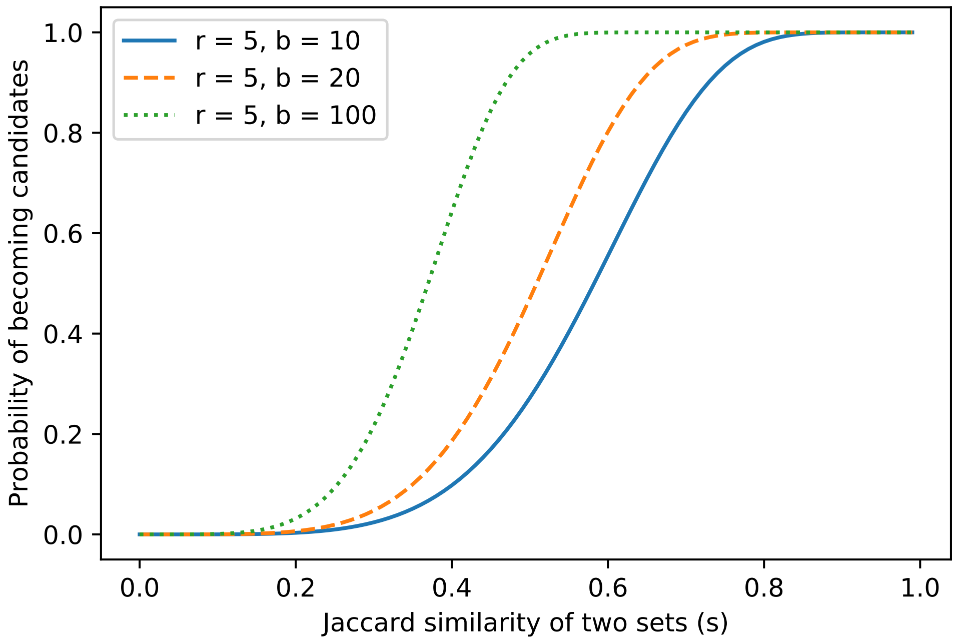

Figure 3.3: Probability of two vectors becoming candidates for an (r, b)-MinHash construction

Using the above theorem, we can formulate the level of sensitivity of families of functions constructed as a combination of the AND-construction and the OR-construction.

Corollary 3.7. *A family of hash functions constructed by first using r MinHash functions with the AND-construction and then using b functions from this new family for the OR-construction is $(d_1, d_2, 1-(1-(1-d_1)^r)^b, 1-(1-(1-d_2)^r)^b$-sensitive. We call this family an (r, b)-MinHash construction.*

In our experiments, the LSH is performed as follows. We interpret each polynomial as a set of monomials, then we fix a set of MinHash functions of size $n = r \cdot b$ and use them on each set. The results of each hash function concatenated together is called a signature of that set. We divide each signature into b parts, called bands. If two polynomials have the same signature in at least one band, we mark them as candidates. Then we can go through all of the candidates, add them and sort all of the new polynomials according to the number of monomials. The polynomials with the lowest number of monomials are the polynomials we take for F4.

If two polynomials have Jaccard similarity s, then with this scheme the probability that we pick them as candidates is exactly $1-(1-s^r)^b$. We can see this function plotted in 3.3 for different values of r and b, used in the experiments by Beruškoyá [Ber22]. Observe that the function has the shape of a S-curve. The shape and steepness of the function suggest the size of the error that can occur when using LSH.

In the Section 3.1, we examined the sparsity of fully random polynomial systems when adding all pairs of polynomials. How does this transfer when using

LSH?

We saw that the (r,b)-MinHash construction is $(d_1, d_2, 1-(1-(1-d_1)^r)^b, 1-(1-(1-d_2)^r)^b$-sensitive. Therefore, if we have two vectors v, w with $d(v,w) \leq d_1$ then, for any h from the MinHash family, $\Pr[h(v) = h(w)] \geq 1-(1-(1-d_1)^r)^b$.

Let us consider the set $W = \{(v_1, w_1), \ldots, (v_k, w_k)\}$ of pairs of vectors such that $d(v_i, w_i) \leq d_1$ for all $i \in [1,k]$ and the set of all candidates after LSH $C = \{(v_i, w_i) \in W \mid h(v_i) = h(w_i)\}$. In Section 3.1 we have examined the expected value of the size of W. We define a characteristic function χ of the event $(v_i, w_i) \in C$, that is

$$\chi((v_i, w_i) \in C) = \begin{cases} 1, & \text{if } h(v_i) = h(w_i) \\ 0, & \text{otherwise.} \end{cases}$$

We want to examine the expected value of the number of candidates in C. We can define a random variable

$$N_C(W) = \sum_{i=1}^{k} \chi((v_i, w_i) \in C).$$

Lemma 3.8. *With the above definitions, the expected value of $N_C(W)$ is*

$$\mathsf{E}(N_C(W)) \geq k \cdot (1-(1-(1-d_1)^r)^b).$$

Proof. By the linearity of expectation

$$\begin{aligned} \mathsf{E}(N_C(W)) &= \mathsf{E}(\sum_{i=1}^{k} \chi((v_i, w_i) \in C)) \\ &= \sum_{i=1}^{k} \mathsf{E}(\chi((v_i, w_i) \in C)) = k \cdot \Pr[h(v_i) = h(w_i)] \\ &\geq k \cdot (1-(1-(1-d_1)^r)^b). \end{aligned}$$

□

To combine this with Section 3.1 we can replace k with $\mathsf{E}(N_V(d_1))$ to get the following theorem.

Theorem 3.9. *Let V be a set of uniformly random polynomials from $\mathbb{Z}_2^m$ and $d_1 \in [0,1]$. When using LSH on this set the expected value of the number of candidates N_C is*

$$\mathsf{E}(N_C) \geq \binom{n}{2} \frac{V(m, d_1)}{2^m} \cdot (1-(1-(1-d_1)^r)^b).$$

Finally, while we have defined LSH for the Jaccard distance, as MinHash is originally defined for this distance, it is worth noting that the Jaccard distance shares similarities with the Hamming distance. To see this, consider two polynomials $p, q \in \mathbb{Z}_2[x_1, \ldots, x_n]$ and two representations of them. Firstly, we can represent them by sets of monomials M_p, M_q present in the respective polynomials. Alternatively, we can represent them using vectors $v_p, v_q \in \mathbb{Z}_2^m$, where m is the number of distinct monomials in both polynomials. In this representation, the vector v_p has a 1 in position i if and only if the i-th monomial is present in p. This, of course, requires us to order the monomials.

Now if we denote by $d(v, w)$ the hamming distance of polynomials v, w, we have

$$1 - \frac{|M_p \cap M_q|}{|M_p \cup M_q|} = \frac{d(v_p, v_q)}{|v_p|}.$$

Consequently, the MinHash is also a method for the normalised hamming distance. This is the typical approach for presenting LSH and MinHash when the main motivation is implementation.

3.3 Reduction in Leading Monomials

In the ComputeM algorithm used in F4, the procedure traverses all monomials from the largest one to the smallest one. We believe that crafting sparser polynomials could enhance the algorithm's performance and we can accomplish that with a method like LSH introduced in Section 3.2. Another potentially beneficial strategy involves considering the sparsity distribution across polynomials. By prioritizing sparsity in larger monomials, we may achieve a reduction in the average leading term, thereby enabling F4 to work with "shorter" polynomials.

To test this hypothesis, we modified the existing experiments programmed by Berušková in [Ber22] to exclusively use LSH on a portion of the biggest monomials. The experiment works as follows:

1. Compute polynomial systems.
2. Sort all present monomials according to GrevLex.
3. Create vector representations of all polynomials.
4. Take only a part of the vector and use the LSH method as before.
5. XOR all candidates and select a subset of the shortest ones.

The existing implementation of LSH skips the second step, as having the monomials sorted when applying the LSH to the whole polynomial is unnecessary. This, however, presents a problem when implementing the new method. The program inherently sorts the monomials according to the Lex ordering and so we need to implement a custom way of sorting the monomials in the GrevLex order. Initially, we attempted to achieve this by adapting a segment of code from the MSM reduction method (another reduction method used by Berušková [Ber22]) to identify the largest leading monomial according to GrevLex. Modifying it as a comparison function, we used it with pythons sort() function to sort the monomials. However, this approach proved to be too time-consuming. The preprocessing time drastically increased from an average of a few seconds to hours.

To mitigate this issue, we opted to utilize the GrLex order instead of GrevLex. Given that the program inherently sorts monomials based on the Lex order, we can leverage this to sort monomials in the GrLex order using the sort() function as before. However, now we let the program sort the monomials only according to the degrees. The sort() function preserves the original order and as a result, monomials are sorted according to the degree lexicographic order.

Next, we need to address the error introduced by using GrLex instead of GrevLex. Let us first recall the definitions of GrLex and GrevLex. Let x^α and x^β be monomials in $k[x_1, \dots, x_n]$.

- Lexicographic Order (Lex):

  $x^\alpha >_{\text{Lex}} x^\beta$ if the leftmost nonzero coordinate of $\alpha - \beta$ is positive.

- Graded Lexicographic Order (GrLex):

  $x^\alpha >_{\text{GrLex}} x^\beta$ if either $|\alpha| > |\beta|$ or $|\alpha| = |\beta|$ and $x^\alpha >_{\text{Lex}} x^\beta$.

- Graded Reverse Lexicographic Order (GrevLex):

  $x^\alpha >_{\text{GrLex}} x^\beta$ if either $|\alpha| > |\beta|$ or $|\alpha| = |\beta|$ and the rightmost nonzero coordinate of $\alpha - \beta$ is negative.

Both GrLex and GrevLex first sort monomials according to their degree. Only when two monomials have the same degree can the orders differ. Thus, we can try to approximate the mistake between these two orders using this characteristic.

Let us first define the problem at hand. Let v_{GrevLex} be the vector containing all monomials up to degree d sorted in the GrevLex order and likewise v_{GrLex} the vector containing all monomials up to degree d sorted in the GrLex order. The question is when we want to use the first, for example, a tenth of the vector v_{GrLex}, what portion of v_{GrevLex} do we need to use to cover all monomials?

When we look at the degree of the lowest monomial in the first tenth of v_{GrevLex} if we add all monomials with that degree then we cover all of the monomials that we wanted to cover in v_{GrLex}. Given that there are $\binom{n+d+1}{d}$ monomials of degree d in n variables, the number of such monomials grows rapidly with an increasing degree d. You can see this growth in Figure 3.4.

We can examine what portion of the vector v_{GrLex} we would have to take to cover everything in a set portion of v_{GrevLex}. You can a graph of this in Figure 3.5. The grey dashed line represents the equality of portions for both vectors, e.g. taking 0.1 portion of v_{GrevLex} and 0.1 of v_{GrLex} lies on the line. Our proportion copying this line would mean we take exactly the same proportion of v_{GrLex} and cover all monomials in v_{GrevLex}. However, you can see, that for smaller portions of v_{GrevLex} like 0.1 we would have to take over half of the v_{GrLex} vector if we were to use this approximation. This is due to the fact that monomials of the highest degree take up as much as half of all the monomials in that vector. For larger amounts, the proportions start to match more closely.

Another way to look at this is to want to know how much of the vectors v_{GrLex} and v_{GrevLex} would overlap were we to take the same portion sizes of both. With our simple approximation in terms of degrees, this does not work for smaller portions. If we take a portion of vectors that contains only the highest degree then with this technique, we have no way to distinguish between the used monomials. As a result, this creates a very similar graph to Figure 3.5.

In conclusion, we can say that without examining the differences between these two orderings within the degrees we cannot say anything about the error with small portions of vectors. However, for bigger portions, we can see that the error between these two orderings is relatively small.

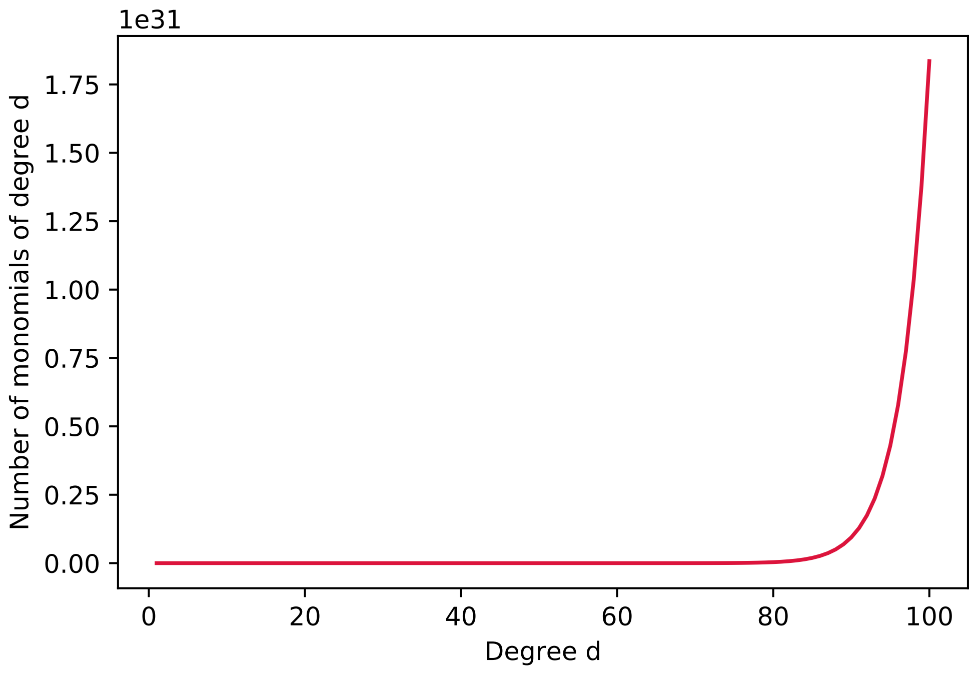

Figure 3.4: Number of monomials of degree d in 32 variables

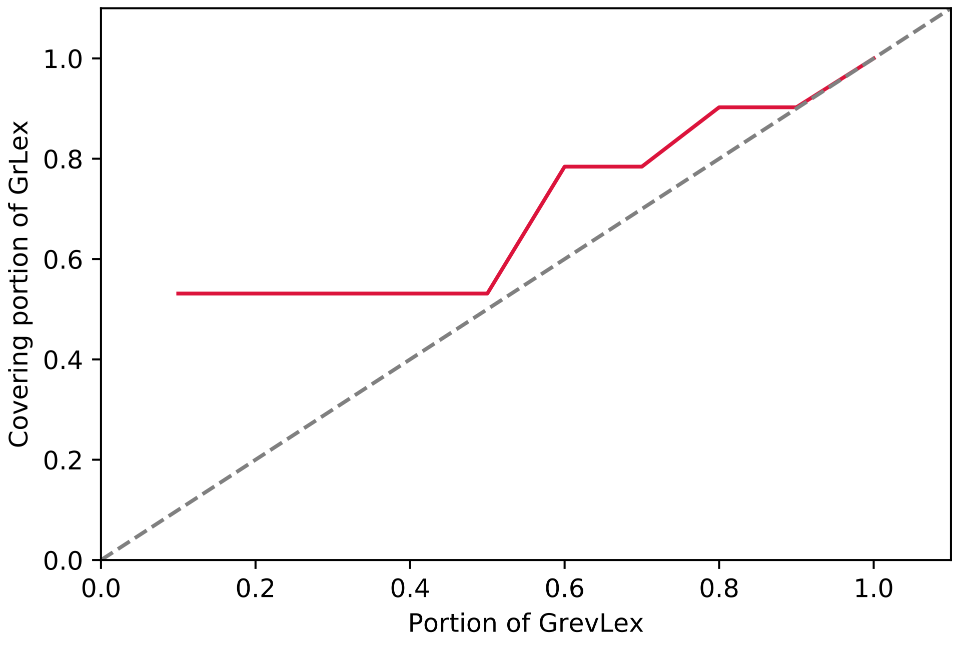

Figure 3.5: Portion of v_{GrLex} needed to cover the monomials in a portion of v_{GrevLex}. The vectors contain all monomials up to degree 30 in 32 variables.

4. Experiments

All of the experiments were conducted on GNU/Linux 5.4 utilizing two Intel® Xeon® Gold 6136 processors with 769 GB DDR4 memory evenly distributed across 12 modules. To solve the systems of equations we used an F4 implementation in Magma V2.25-5.

The scripts used in our experiments were adapted from those developed by Bielik and Berušková for their respective theses [Bie21, Ber22]. The source code of Bielik's experiments can be found at https://gitlab.com/marek.onl/masters-thesis, while the source code created by Berušková can be found at https://github.com/berusjan/RedPolSysAES.

We added two other preprocessing methods detailed in Section 4.1 and Section 4.2. The source code for these experiments will be made available at https://gitlab.mff.cuni.cz/maskovkr/algcryptopre. Moreover, our repository contains Jupyter notebooks featuring some graphs and statistics presented herein.

Our goal for these experiments was twofold. First, we aimed to confirm that using LSH is better than exhaustively going through all pairs, while also comparing these two methods. We analyze this in Section 4.1. Secondly, we wanted to test our assumption, that using LSH only the first portion of the vectors would further fasten the time required for solving. We describe these experiments and give some results in Section 4.2.

For the sake of simplicity, all experiments were conducted with a single polynomial set as input for F4. The number of polynomials therefore corresponds to the number of key bits.

Some experiments did not finish. This was either due to the fact that the time exceeded a predetermined limit, typically set to 4 hours, or encountering computational errors resulting from an error in Magma. These incomplete runs were excluded from the final analysis.

4.1 Comparative Analysis of LSH and Exhaustive Comparison

In this section, we analyze the efficiency of LSH compared to the exhaustive pairwise comparison. In Chapter 3 we described the primary way of preprocessing in algebraic cryptanalysis we investigate in this work. It involves adding all pairs of polynomials and selecting those resulting in the fewest monomials, which corresponds to performing bitwise XOR operations on the vectors representing the polynomials.

Going through all pairs of polynomials on its own involves traversing $\binom{s}{2}$ pairs, where s is the number of polynomials. This process exhibits quadratic growth and can become prohibitively expensive for a larger number of polynomials. To illustrate, consider SR(1,4,4,8) or one round of AES. The system of polynomials has as many polynomials as there are bits of the key, in this case, that is 128 polynomials. Generating 8 polynomial systems would mean we would have 1024 polynomials and that is 523776 pairs to go through. One way to mitigate this is to use a method like LSH. When experimenting with LSH on SR(1,4,4,8), we

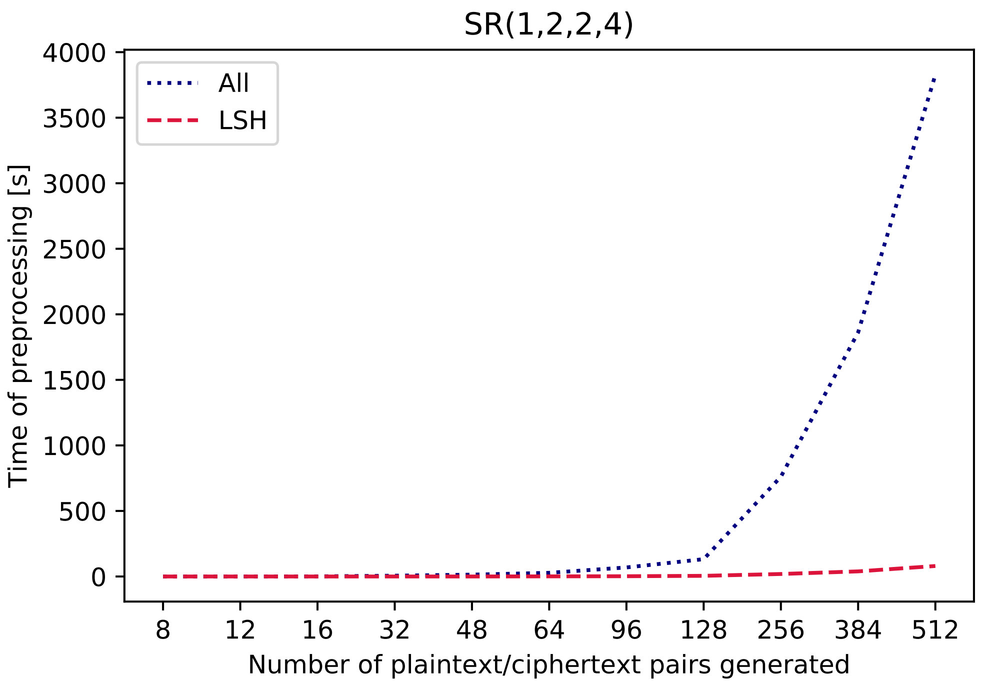

Figure 4.1: Preprocessing times of LSH and All on SR(1,2,2,4)

observed a significant reduction in the number of pairs to just 5826, approximately 0.01 of the original value. When we ran experiments of LSH on SR(1,4,4,8) the number of pairs went down to 5826, or just around 0.01 of the original value.

To give us an idea of how much faster LSH generally is, we implemented a new method of reduction called "All" that first XORs all of the pairs of polynomials and then picks for solving those, that have the lowest amount of monomials. We conducted experiments across various ciphers, each with different numbers of generated plaintext/ciphertext pairs. We examined the proportion of all pairs to candidate pairs when employing LSH, as well as the preprocessing time when using both "All" and LSH. The exact parameters can be found in scripts run_LSH_data-full.sh and run_All_data-full.sh.

For each cipher, we compared the number of all pairs required without LSH to the number of candidate pairs identified by LSH. On average, LSH pairs accounted for a mere 4.9% of all pairs, with the highest value of 38.3% and the lowest of 1.2% of total pairs.

The comparison between the time required to go through all pairs versus the time required when using LSH is illustrated in Figure 4.1. Here the x-axis represents the number of plaintext/ciphertext pairs generated, while the y-axis depicts the time of preprocessing for the SR(1,2,2,4) cipher with that number of plaintext/ciphertext pairs. You can see that for a low number of plaintext/ciphertext pairs the times appear to be comparable. However, the time to go through all pairs grows rapidly as the number of plaintext/ciphertext pairs increases.

We can analyze this further by examining the proportion of the preprocessing time for LSH and "All". On average, the preprocessing time of LSH is approximately 20% of the preprocessing time for "All", indicating that LSH is around

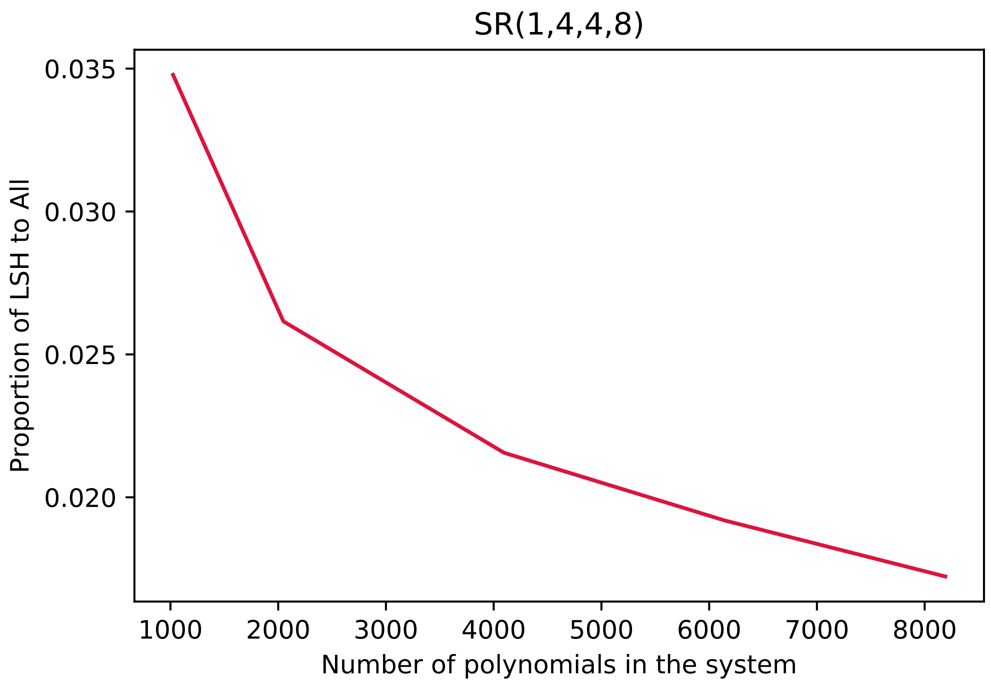

Figure 4.2: Proportion of preprocessing time of LSH to “All” for SR(1,4,4,8)

five times faster. We would expect the LSH preprocessing to be faster the more polynomials are in the system. This is evident for example in Figure 4.2, where we can see the proportion of preprocessing times of LSH to “All” in relation to the number of polynomials in the system.

The number of candidates LSH outputs is related to how many polynomials are on average in a bucket. For instance, if LSH were to put all polynomials into a single bucket, the number of candidates would be the same as going over all pairs. Although we did not delve into this in detail, we wanted to experimentaly examine the distribution of polynomials in buckets. To achieve this, we generated several graphs showing the number of buckets with a certain amount of polynomials. One such graph is presented in Figure 4.3. As depicted, as the amount of polynomials increases, the number of buckets with that amount of polynomials decreases. Notably, the graph illustrates that a large amount of the buckets contain only 2 polynomials, with almost all buckets containing fewer than 40 polynomials. These findings were consistent across all generated graphs, highlighting a distribution pattern.

4.2 Partial LSH Experiments

Our second goal was to test our assumption of using LSH only on a portion of the polynomial, hopefully finding a candidate for the optimal value of that portion. We described this method in detail in Section 3.3. The idea is that polynomials reduced primarily in the bigger monomials would be more suitable for F4.

To test this, we modified the existing code so that the monomials would be

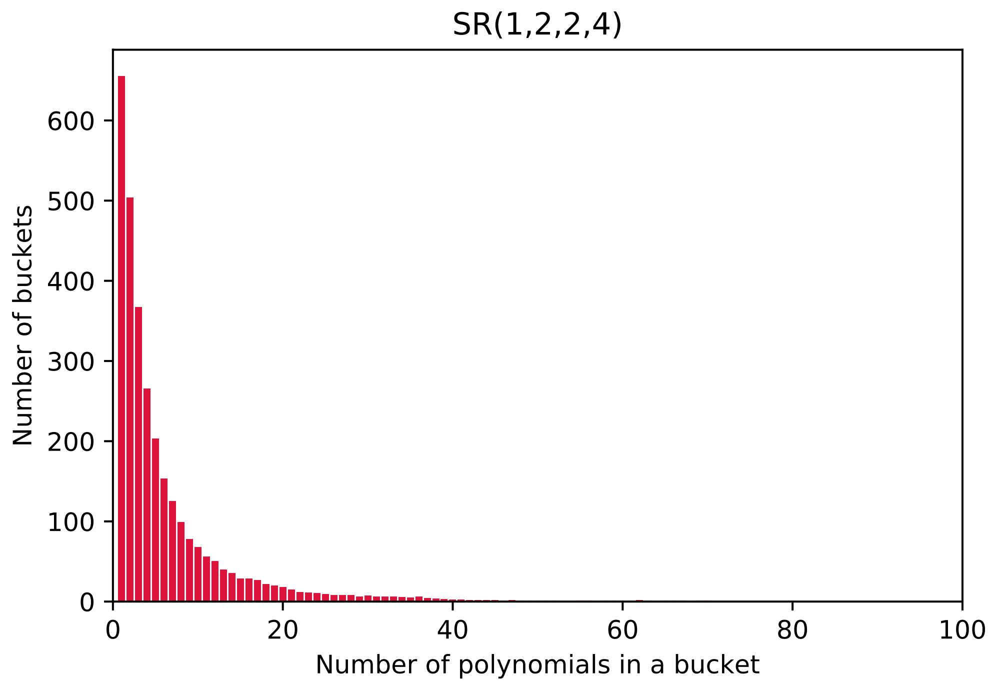

Figure 4.3: Number of buckets with that amount of polynomials in SR(1,2,2,4) for number of generated plaintext/ciphertext pairs set to 128

sorted in GrLex order, we added a new method of reduction named "LSH_part" and we added an argument -LSHpart that takes in a float and interprets it as the portion of the vector that LSH should be used on. We also modified the resolution of the time to be in milliseconds and not seconds, as most of the experiments in our earlier runs either ran for less than a second or did not finish at all.

Running the experiments for all ciphers and all possible values of -LSHpart would be too time-consuming, so we decided to run the experiments only for two ciphers SR(1,2,2,8) and SR(2,2,2,4). We first generated and saved 10 random instances for different numbers of plaintext/ciphertext pairs ranging from 4 to 1024. Then we ran the experiments for values of -LSHpart $0.1, 0.2, \ldots, 0.9, 1$, each 10 times on the pre-generated data. The average times for each value of -LSHpart can be seen in Table 4.1 and in Figure 4.4.

In the Table 4.1, you can also see the value Mon representing the average number of monomials in a polynomial. The value stays roughly the same for all parts. However, we can see that it is lowest for values close to the edge while being higher for values in the middle. This could suggest that using LSH is most effective when done on the whole vector or only on a small part, but not on half of the vector.

In Figure 4.4 we can see that the optimal value of -LSHpart in terms of time spent in the solver is around 0.7. Another dip can be seen in -LSHpart 0.2, however, we could not figure out a reason why that might happen.

While the average number of monomials in a polynomial does not change much between the values of -LSHpart, it changes drastically with larger numbers of plaintext/ciphertext pairs generated. This can be seen in Figure 4.5, where we

-LSHpart	Mon	Time [ms]
0.1	87.9	25374
0.2	88.2	10518
0.3	88.5	25072
0.4	88.5	34072
0.5	89.4	11773
0.6	89.2	11603
0.7	88.8	9290
0.8	87.8	11674
0.9	87.9	16248
1.0	86.2	17321

Table 4.1: Average times of partial LSH used on SR(1,2,2,8).

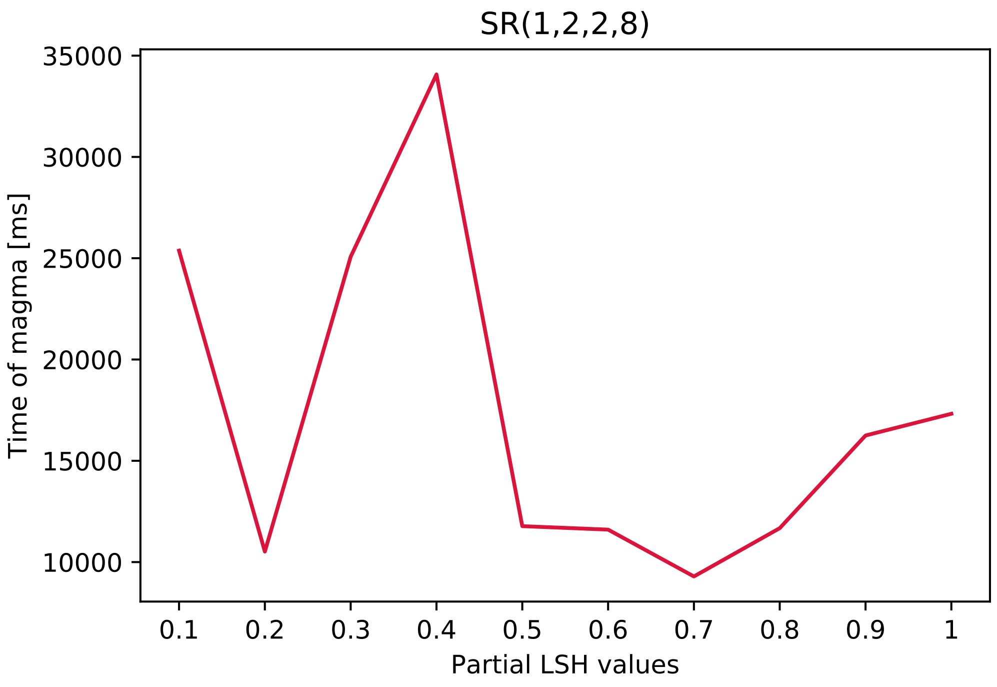

Figure 4.4: Average times of LSH_part for SR(1,2,2,8)

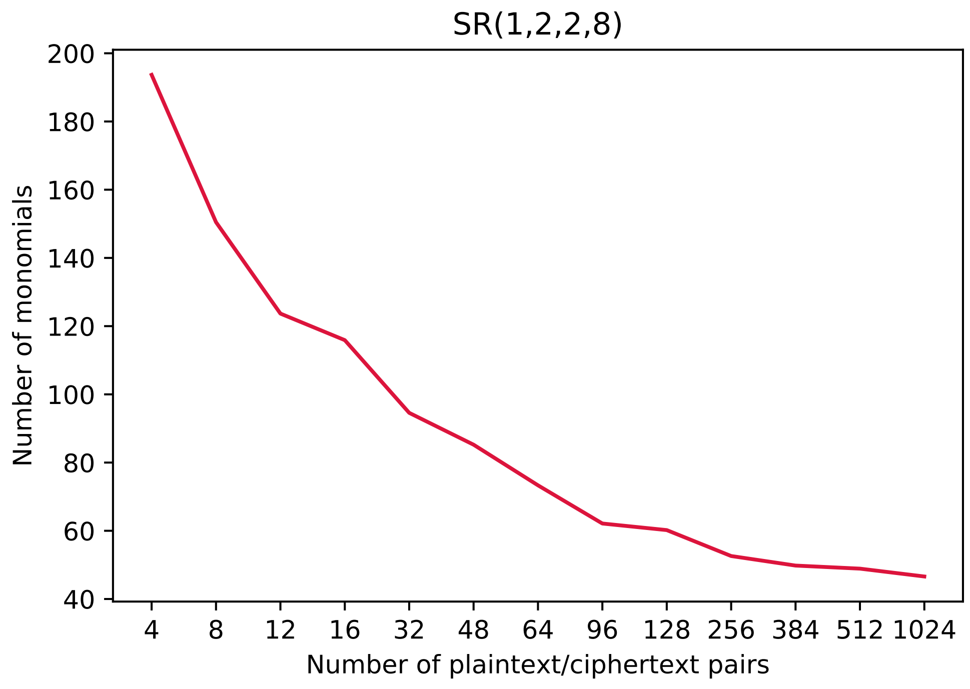

Figure 4.5: Average number of monomials in a polynomial for SR(1,2,2,8)

plotted the average number of monomials in a polynomial for changing numbers of plaintext/ciphertext pairs.

We ran the same experiments on the cipher SR(2,2,2,4), you can see the average times of LSH_part in Figure 4.6. For this cipher, the lowest value of time in the solver is around -LSHpart of 0.6 and there is no dip around the 0.2 as in the previous graph.

Due to the time constraint, we did not manage to run additional experiments for LSH_part. Running experiments for more ciphers with a wider range of parameters could yield more conclusive results.

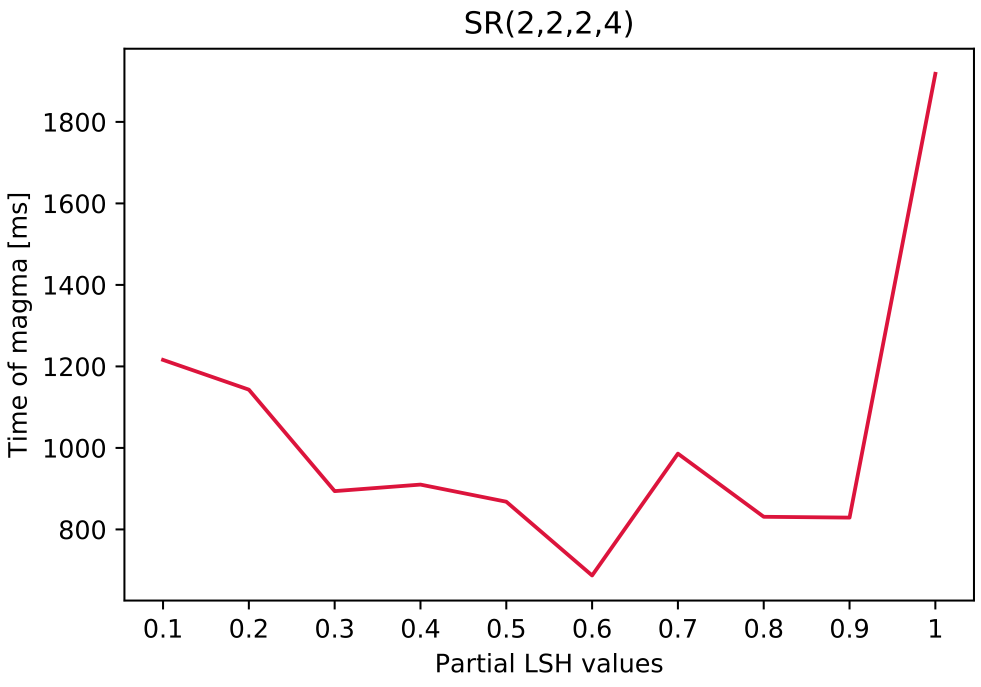

Figure 4.6: Average times of LSH_part for SR(2,2,2,4)

Conclusion

Preprocessing in algebraic cryptanalysis is a new topic, by now only explored by Bielik et al. [Bie21, BJJL22] and Berušková [Ber22]. Both of works focused on implementing the ideas and experimenting with them. While Berušková suggested a few techniques for preprocessing, the suggestions were purely heuristic. Our goal was to provide a theoretical framework for explaining the observed performance in the above experiments and that provides a guide on how to refine and optimize the performance of preprocessing techniques.

We showed the correctness of preprocessing in Chapter 2 together with giving an overview of the current state of knowledge. In Chapter 3 we first examined the sparsity of fully random systems. We presented a foundation of the theory on Locality-Sensitive Hashing used in preprocessing, and, finally, we suggested a version of preprocessing that focuses on the leading monomials in a polynomial. We concluded this thesis with a discussion of the results of our experiments. We showed that LSH is most useful when working with larger sets of polynomials and that using LSH only on the leading monomials could further help the computation.

In the introduction, we mentioned the new class of primitives known as Arithmetization Oriented Ciphers. These primitives are designed to be efficient in protocols using arithmetization like Zero Knowledge proofs. This also makes them vulnerable to algebraic cryptanalysis. Exploring the power of the preprocessing techniques in algebraic cryptanalysis of these ciphers could yield interesting insights on their security.

Bibliography

[ACG^{+}19] Martin R. Albrecht, Carlos Cid, Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, and Markus Schofnegger. Algebraic cryptanalysis of STARK-friendly designs: Application to MARVELlous and MiMC. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 371–397. Springer, 2019.

[AD18] Tomer Ashur and Siemen Dhooghe. MARVELlous: a STARK-friendly family of cryptographic primitives. Cryptology ePrint Archive, Paper 2018/1098, 2018.

[Bar09] Gregory V. Bard. *Algebraic Cryptanalysis.* Springer New York, NY, 2009.

[Ber22] Jana Berušková. Redukování předefinovaných systémů polynomiálních rovnic odvozených ze zjednodušených variant AES, 2022. `https://github.com/berusjan/RedPolSysAES`.

[Bie21] Marek Bielik. Algebraic cryptanalysis of small scale variants of the AES, 2021. `https://gitlab.com/marek.onl/masters-thesis`.

[BJJL22] Marek Bielik, Martin Jurecek, Olha Jurecková, and Róbert Lórencz. Yet another algebraic cryptanalysis of small scale variants of AES. In Sabrina De Capitani di Vimercati and Pierangela Samarati, editors, *Proceedings of the 19th International Conference on Security and Cryptography, SECRYPT 2022, Lisbon, Portugal, July 11-13, 2022*, pages 415–427. SCITEPRESS, 2022.

[Bro97] Andrei Z. Broder. On the resemblance and containment of documents. In Bruno Carpentieri, Alfredo De Santis, Ugo Vaccaro, and James A. Storer, editors, *Compression and Complexity of SEQUENCES 1997, Positano, Amalfitan Coast, Salerno, Italy, June 11-13, 1997, Proceedings*, pages 21–29. IEEE, 1997.

[Buc65] Bruno Buchberger. Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. Dissertation an dem Math. Inst. der Universität von Innsbruck, 1965.

[CLO15] David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms.* Undergraduate Texts in Mathematics. Springer Cham, 2015.

[CMR05] Carlos Cid, Sean Murphy, and Matthew J. B. Robshaw. Small scale variants of the AES. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005,* *Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*, pages 145–162. Springer, 2005.

[CMR06] Carlos Cid, Sean Murphy, and Matthew J. B. Robshaw. *Algebraic aspects of the advanced encryption standard*. Springer, 2006.

[Fau99] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, June 1999.

[Fau02] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, 2002.

[FGLM93] Jean-Charles Faugère, Patrizia M. Gianni, Daniel Lazard, and Teo Mora. Efficient computation of zero-dimensional Gröbner bases by change of ordering. *J. Symb. Comput.*, 16(4):329–344, 1993.

[Kos19] Sven Kosub. A note on the triangle inequality for the Jaccard distance. *Pattern Recognit. Lett.*, 120:36–38, 2019.

[LMRS12] Mario Lamberger, Florian Mendel, Vincent Rijmen, and Koen Simoens. Memoryless near-collisions via coding theory. *Des. Codes Cryptogr.*, 62(1):1–18, 2012.

[oST] National Institute of Standards and Technology. Advanced encryption standard (AES). Federal Information Processing Standards Publication (FIPS) NIST FIPS 197-upd1, updated 2023.

[SS21] Jan Ferdinand Sauer and Alan Szepieniec. Sok: Gröbner basis algorithms for arithmetization oriented ciphers. *IACR Cryptol. ePrint Arch.*, page 870, 2021.

[Wyl] Andrew Wylie. Locality-sensitive hashing. Online lecture notes.