GRÖBNER BASIS ATTACK ON STARK-FRIENDLY SYMMETRIC-KEY PRIMITIVES: JARVIS, MiMC AND GMiMC$_{\text{erf}}$

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

GİZEM KARA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
CRYPTOGRAPHY

FEBRUARY 2021

Approval of the thesis:

GRÖBNER BASIS ATTACK ON STARK-FRIENDLY SYMMETRIC-KEY PRIMITIVES: JARVIS, MiMC AND GMiMC$_{\mathbf{erf}}$

submitted by **GİZEM KARA** in partial fulfillment of the requirements for the degree of **Master of Science in Cryptography Department, Middle East Technical University** by,

Prof. Dr. Ayşe Sevtap Kestel-Selçuk
Director, Graduate School of **Applied Mathematics** ______________

Prof. Dr. Ferruh Özbudak
Head of Department, **Cryptography** ______________

Assoc. Prof. Dr. Ali Doğanaksoy
Supervisor, **Mathematics, METU** ______________

Assist. Prof. Dr. Oğuz Yayla
Co-supervisor, **Cryptography, METU** ______________

Examining Committee Members:

Assoc. Prof. Dr. Murat Cenk
Cryptography Department, METU ______________

Assoc. Prof. Dr. Ali Doğanaksoy
Mathematics Department, METU ______________

Assoc. Prof. Dr. Fatih Sulak
Mathematics Department, Atılım University ______________

Assoc. Prof. Dr. Zülfükar Saygı
Mathematics Department, TOBB ETU ______________

Assist. Prof. Dr. Ahmet Sınak
Mathematics and Computer Sciences Department,
Necmettin Erbakan University ______________

Date: ______________

ABSTRACT

GRÖBNER BASIS ATTACK ON STARK-FRIENDLY SYMMETRIC-KEY PRIMITIVES: JARVIS, MiMC AND GMiMC$_{\text{erf}}$

Kara, Gizem
M.S., Department of Cryptography
Supervisor : Assoc. Prof. Dr. Ali Doğanaksoy
Co-Supervisor : Assist. Prof. Dr. Oğuz Yayla

February 2021, 63 pages

A number of arithmetization-oriented ciphers emerge for use in advanced cryptographic protocols such as secure multi-party computation (MPC), fully homomorphic encryption (FHE) and zero-knowledge proofs (ZK) in recent years. The standard block ciphers like AES and the hash functions SHA2/SHA3 are proved to be efficient in software and hardware but not optimal to use in this field, for this reason, new kind of cryptographic primitives proposed. However, unlike traditional ones, there is no standard approach to design and analyze such block ciphers and the hash functions, therefore their security analysis needs to be done carefully. In 2018, StarkWare launched a public STARK-Friendly Hash (SFH) Challenge to select an efficient and secure hash function to be used within ZK-STARKs, transparent and post-quantum secure proof systems. The block cipher JARVIS is one of the first ciphers designed for STARK applications but, shortly after its publication, the cipher has been shown vulnerable to Gröbner basis attack. This master thesis aims to describe a Gröbner basis attack on new block ciphers, MiMC, GMiMC$_{\text{erf}}$ (SFH candidates) and the variants of JARVIS. We present the complexity of Gröbner basis attack on JARVIS-like ciphers, results from our experiments for the attack on reduced-round MiMC and a structure we found in the Gröbner basis for GMiMC$_{\text{erf}}$.

Keywords: Gröbner Basis, Jarvis, MiMC, GMiMC, Secure Multi-party Computation (MPC), ZK-STARKs

ÖZ

STARK DOSTU SİMETRİK ANAHTAR İLKELLERİNE KARŞI GRÖBNER BAZ SALDIRISI: JARVİS, MiMC VE GMiMC$_{\text{erf}}$

Kara, Gizem
Yüksek Lisans, Kriptografi Bölümü
Tez Yöneticisi : Doç. Dr. Ali Doğanaksoy
Ortak Tez Yöneticisi : Yrd. Doç. Dr. Oğuz Yayla

Şubat 2021, 63 sayfa

Son yıllarda güvenli çok partili hesaplama (MPC), tamamen homomorfik şifreleme (FHE) ve sıfır bilgi kanıtları (ZK) gibi gelişmiş kriptografik protokollerde kullanılmak üzere bir dizi aritmetizasyon odaklı şifreleme ortaya çıkmıştır. AES veya SHA2/SHA3 gibi standart blok şifreler ve özet fonksiyonlarının yazılım ve donanımda verimli olduğu ancak bu yeni alanda kullanım için uygun olmadığı kanıtlanmıştır bu nedenle, yeni türde kriptografik ilkeller önerilmektedir. Ancak, geleneksel olanların aksine bu tür aritmetizasyon odaklı blok şifreleri veya özet fonksiyonlarını tasarlamak ve analiz etmek için standart bir yaklaşım yoktur, dolayısıyla güvenlik analizlerinin dikkatlice yapılması gerekmektedir. 2018'de StarkWare, şeffaf ve kuantum sonrası güvenli kanıt sistemleri ZK-STARK'larda kullanılacak verimli ve güvenli bir özet fonksiyonu seçmek üzere halka açık bir STARK Dostu Özet (SFH) Mücadelesi başlatmıştır. JARVİS blok şifresi STARK uygulamaları için tasarlanmış ilk şifrelerden biridir, ancak yayınlanmasından kısa bir süre sonra şifrenin Gröbner baz saldırısına karşı savunmasız olduğu görülmüştür. Bu yüksek lisans tezi, yeni blok şifreler MiMC, GMiMC$_{\text{erf}}$ (SFH adayları) ve JARVİS varyantlarına karşı Gröbner baz saldırısını tanımlamayı hedeflemektedir. JARVİS benzeri şifrelere Gröbner baz saldırısının karmaşıklığı, azaltılmış tur sayılı MiMC'ye yönelik Gröbner baz saldırısının deneysel sonuçları ve GMiMC$_{\text{erf}}$ ye ait Gröbner bazda bulunan yapı sunulmaktadır.

Anahtar Kelimeler: Gröbner Baz, Jarvis, MiMC, GMiMC, Güvenli Çok Partili Hesaplama (MPC), ZK-STARKlar

To my family...

ACKNOWLEDGMENTS

I would like to thank my supervisor Assoc. Prof. Dr. Ali Doğanaksoy not only for the supervision of this master thesis but also for his support on my whole study in cryptography.

I would also thank to my co-supervisor Assist. Prof. Dr. Oğuz Yayla for his guidance, support and motivation throughout this thesis. I owe much to him.

I am very grateful to Prof. Dr. Vincent Rijmen for giving me an opportunity to carry out a research visit at COSIC in Leuven, Belgium. I would like to thank Siemen Dhooghe for his supervision and kindness throughout my study in Leuven. It was a pleasure to work with him.

Furthermore, I would like to thank my family and my friends for listening to my concerns, believing in me and their supports.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

$\mathbb{F}$	The base field of the polynomial ring $\mathbb{F}[x_1, \cdots, x_k]$ in k variables x_i
$\mathbb{F}_p$	The finite field or Galois field (GF) of characteristic p where p is prime
$\mathbb{Z}$	The set of integers, $\{\cdots, -2, -1, 0, 1, 2, \cdots\}$
$\mathbb{Z}_{\geq 0}$	The set of nonpositive integers, $\{0, 1, 2, \cdots\}$
$\mathbb{Z}^k$	k-tuple of integers
$\mathbb{Q}$	The set of rational numbers,
I	an ideal in a polynomial ring
$LT(f)$	The leading term of the polynomial f

CHAPTER 1

INTRODUCTION

Block ciphers are the fundamental tools of modern cryptography. They are pseudo-random permutations operating on fixed-size blocks and used to secure different types of data. Their design and security considerations are well understood in the literature. However, the design of symmetric-key primitives for use in advanced cryptographic protocols such as secure multi-party computation (MPC), fully homomorphic encryption (FHE) or new proof systems like SNARKs, STARKs, Bulletproofs studied in the past few years because of the recent progress in practical applications of this field.

Secure multi-party computation (MPC) is a cryptographic protocol that enables to parties securely evaluate output of a function without knowing anything about their private inputs. In MPC systems, the arithmetic operations on secret sharing values are often performed over a finite field with large prime characteristic $\mathbb{F}_p$. The problem of using traditional block ciphers like AES in MPC setting is the hardness of representing such block ciphers using arithmetic over finite fields. Their design strategy aims to provide mostly good performance in hardware or software implementations. Therefore, we have a new area of designing efficient symmetric primitives for use in MPC or ZK-proof systems. We refer reader to [6] that gives detailed information for the design of such primitives.

We know that the first paper which explicitly designs pseudo-random functions (PRFs) for MPC applications is [5] from Eurocrypt 2015. The designers propose a blockcipher LowMC with low multiplicative depth and low multiplicative complexity which operates over GF(2). After that, several bit-oriented primitives have appeared like Kreyvium [13] or FLIP [24] considering the same design strategy as LowMC. Because most of the advanced cryptographic protocols support operations over large prime fields, MiMC family designs [2], include a block cipher and a cryptographic hash function, were presented by offering multiplications over large fields GF(2^n) and GF(p) at the Asiacrypt 2016. The block cipher MiMC was designed mainly for SNARK applications like Zerocash [26] but it is also competitive for use in STARKs and MPC applications. The designers of MiMC improved cipher to Generalized MiMC (GMiMC) [4] in order to provide efficient performance also in the area of PQ-secure signature schemes where MiMC was not so competitive in this area. MARVELlous family cryptographic primitives [7], the block cipher JARVIS and the hash function FRIDAY are the first designs to propose efficiency in STARK applications but after its publication, it has been shown that these designs do not provide adequate security as claimed [3]. The paper [6] calls these new primitives as arithmetization-oriented algorithms.

The design strategies of standard block ciphers like AES (Advanced Encryption Standard) [25] or 3DES (Triple DES, Data Encryption Standard) [23] and the arithmetization–oriented ciphers are different. Therefore, their security analysis and the corresponding attack techniques are also different. Statistical attacks such as differential and linear cryptanalysis are widely used for the cryptanalysis of block ciphers. Algebraic attacks are a different type of cryptanalysis which aims to exploit algebraic structure of the cipher. These kinds of attacks try to represent the cipher as a system of polynomial equations and then solve to recover the key using a suitable method like SAT solvers, Gröbner basis methods, Mixed -Integer Linear Programming (MILP) Solvers or Algebraic higher-order differentials. A common belief is that the statistical attacks are generally faster than the algebraic attacks because of the high complexity of algebraic attacks. *"Not a single proper block cipher has been broken using pure algebraic techniques faster than with other techniques." (Albrecht).* Algebraic techniques were mostly considered against some public-key schemes and stream ciphers because they proved success against them. However, the target applications like MPC/FHE/ZK-STARKs are algebraic systems, and therefore algebraic attacks gain attention again from the cryptographers.

The design of arithmetization-oriented algorithms which are both efficient and secure still in progress. Two design strategies, e.g. Marvellous [6] and Hades [21, 22] provide a generic way for the demand in design space relative to these target applications. After JARVIS shown to be insecure againsts Gröbner basis attack, the designers of MARVELlous together with Ben-Sasson, co-founder and president of StarkWare, propose Marvellous family design strategy which includes two ciphers *Vision* for binary fields and *Rescue* for prime fields. These ciphers were candidates for STARK-Friendly Hash (SFH) Challenge [1]. The HADES design strategy proposed by Grassi et al. [22] at Eurocrypt 2020 and the HadesMiMC family of algorithms, the hash functions *Starkad* and *Poseidon* [21], were also candidate in SFH challenge. In this public competition, the security of four families of algorithms – MiMC, GMiMC, HadesMiMC and MARVELlous, was analyzed by the cryptanalysts. At the end of the selection process of STARK-Friendly hash function, the hash function *Rescue* is recommended by Ben-Sasson et al in. eprint.iacr.org/2020/948.

1.1 Our motivation

The new arithmetic-oriented primitives designed for applications of advanced cryptographic protocols may vulnerable to algebraic attacks, particularly Gröbner basis attacks. The security of these ciphers was examined against various algebraic attacks but not focus directly on Gröbner basis attacks. However as said in [6], it is the common question for these new designs

"Consequently, the question of security against Gröbner basis attacks seems to be the crucial concern raised by arithmetization-oriented ciphers, and no such proposal is complete without explicitly addressing it".

The success of the attack strategy on JARVIS and FRIDAY motivated us to study Gröbner basis attack against variants of JARVIS and the other proposed ciphers, MiMC and $\text{GMiMC}_{\text{erf}}$.

1.2 Structure of the master thesis

Chapter 2-3 will present mathematical background for Gröbner bases and Gröbner basis attacks. In Chapter 4, we will briefly describe the block cipher JARVIS and in

Section 4.2, we will mention successful Gröbner basis attack on JARVIS by Albrecht et al. [3], then we generalize the attack strategy on JARVIS-like ciphers. We will give a formula to estimate the complexity of the attack and using this formula we will show JARVIS with degree 8 polynomials is still vulnerable to Gröbner basis attack in Section 4.3. Furthermore, we will compare S-boxes of JARVIS and AES in Section 4.3.1 and estimate the complexity of the attack on JARVIS with AES S-box in Section 4.3.2. If we replace the S-box of JARVIS with AES S-box, we see that the complexity of the attack with 8-bits input is around ≈ 97 bits for 10 rounds.

Chapter 5 will express our other target cipher MiMC and present results from our experiments for the Gröbner basis attack on reduced round MiMC. We see that MiMC with 82 rounds is resistant against Gröbner basis attack. The following chapter will give a brief description of the block cipher $\text{GMiMC}_{\text{erf}}$ and describe our findings for the Gröbner basis attack against the primitive. We will say that $\text{GMiMC}_{\text{erf}}$ is secure against Gröbner basis attack not because of the high complexity of basis computation but for a different reason. Chapter 7 will conclude our results in this thesis and end up with discussion and future work section.

Note that all the experiments in this thesis are performed in Sage 9.0. "Sage: Software for Algebra and Geometry Experimentation " is a free and open source computational algebra system [28]. The full source codes of the attacks are provided in Appendix A.

CHAPTER 2

MATHEMATICAL BACKGROUND

In this chapter, we will give some main theorems and definitions to understand the concept of Gröbner basis and Gröbner basis attacks. For more detailed information we refer to see "Ideals and Varieties" by Cox et al. [15].

2.1 Monomial Orders and Monomial Ideals

Definition 2.1.1. A **multivarite polynomial** f in k variables $x_0, \cdots, x_{k-1}$ with coefficients $c_0, \cdots, c_{k-1}$ over a field $\mathbb{F}$ can be expressed as

$$f = \sum_{i \in \mathbb{Z}^k_{\geq 0}} c_i x^i$$

where $x^i = x_0^{i_0} \cdot x_1^{i_1} \cdots x_{k-1}^{i_{k-1}}$ is a **monomial** with total degree $i_0 + i_1 + \cdots + i_{k-1}$. The degree of f is defined as the maximum value of the total degrees of the monomials.

Example 2.1.1. The polynomial $f = 4x_1x_2x_4 + \frac{1}{2}x_1x_3x_4 + x_4 \in \mathbb{Q}[x_1, x_2, x_3, x_4]$ has three terms and has degree 3. Two monomials have the maximum degree 3.

For multivariate polynomials, the order of terms *monomial ordering* is not just important to write and read terms but also to decide the leading term of the polynomial and how to store and operate the polynomials in a computer since they affect the complexity.

For example, while using division algorithm on **univariate polynomials**, a polynomial depends only one variable, over $\mathbb{F}[x]$, we write terms in decreasing order on degrees of the terms, $\cdots > x^{t+2} > x^{t+1} > x^t > \cdots > x^2 > x^1 > x > 1$. Also, in row-reduction algorithm for the matrices, we deal with the linear equations in k variables $x_1, \cdots, x_k$ in decreasing order, written as $x_1 > \cdots > x_k$.

Now, we may define ordering in monomials.

Definition 2.1.2 (Monomial ordering). A monomial ordering on $\mathbb{F}[x_1, \cdots, x_k]$ is a relation $>$ on $\mathbb{Z}^k_{\geq 0}$ (i.e, exponents of monomials) or a relation on monomials $x^a, a \in \mathbb{Z}^k_{\geq 0}$, such that:

1. The relation $>$ is a total ordering on $\mathbb{Z}^k_{\geq 0}$. That means for any pairs of x^a and x^b exactly one of the three statements, $x^a > x^b, x^a = x^b, x^a < x^b$ should be satisfied.
2. If $a > b$ and $c \in \mathbb{Z}^k_{\geq 0}$, then $a + c > b + c$.
3. The relation $>$ has well-ordering which means every non-empty subset has a smallest element under $>$.

For example, the numerical order $t + 1 > t > \cdots > 2 > 1 > 0$ on $\mathbb{N}$, satisfies the above conditions, hence the degree ordering on monomials over $\mathbb{F}[x]$ is a monomial ordering.

In computational algebra, the following three term orderings are mostly used

Definition 2.1.3 (Lexicografic Order). We say $a >_{lex} b$ if the left most non zero entry in $a - b \in \mathbb{Z}^k$ is positive.

Definition 2.1.4 (Graded Lexicografic Order). We say $a >_{grlex} b$ if the total degrees $|a| > |b|$ or if $|a| = |b|$ and $a >_{lex} b$.

Definition 2.1.5 (Graded Reverse Lexicografic Order). We say $a >_{grevlex} b$ if the total degrees $|a| > |b|$ or if $|a| = |b|$ and the rightmost non zero enrty of vector difference $a - b \in \mathbb{Z}^k$ is negative.

For example,

- $a = (1, 0, 0) >_{lex} (0, 3, 4) = b$ since the left most non zero entry of $a - b = (1, -3, -4)$ is positive.

- $(1,1,2) >_{grlex} (1,0,3)$ since $|(1,1,2)| = |(1,0,3)|$ and $(1,1,2) >_{lex} (1,0,3)$.
- Consider the monomials $a = x^3y^5z^2$ and $b = x^2y^7z$, if we have $x > y > z$

$$a >_{lex} b, \; a >_{grlex} b \text{ and } a <_{grevlex} b.$$

Before giving the definition of Gröbner basis, let's first define the *monomial ideals.*

Definition 2.1.6. An ideal $I \subseteq \mathbb{F}[x_1, \cdots, x_k]$ is called a **monomial ideal** if it can be generated by monomials.

For example, $I = \langle x^2y, xy^3 \rangle \subseteq \mathbb{F}[x, y]$ is a monomial ideal generated by the monomials x^2y and xy^3.

Theorem 2.1.1 (Dickson's Lemma). Every monomial ideal I is finitely generated, i.e I has a finite basis.

Proof. See [15, Chapter 2, Section 4, Theorem 5]. $\square$

Definition 2.1.7. Consider an ideal $I \subseteq F[x_1, \cdots, x_k]$ different than zero and fix a monomial ordering. The set $LT(I)$ is the set of leading terms of the polynomials in I

$$LT(I) = \{LT(f) |\; f \in I\}.$$

The ideal generated by the elements of $LT(I)$ is denoted by $\langle LT(I) \rangle$.

Note that for the ideal I say, $I = \langle g_1, \cdots, g_t \rangle$, the ideals $\langle LT(g_1), \cdots, LT(g_t) \rangle$ and $\langle LT(I) \rangle$ may be different. Let's observe the following example.

Example 2.1.2. Consider $I = \langle x^3 + 2xy, x^2y + 2y^2 - 1 \rangle$ and fix a lex ordering on $\mathbb{Q}[x, y]$.

Note that

$$y \cdot (x^3 + 2xy) - x \cdot (x^2y + 2y^2 - 1) = x,$$

therefore $LT(x) = x \in \langle LT(I) \rangle$. However, $x \notin \langle LT(f), LT(g) \rangle$ since x is not divisible by $x^3 = LT(f) = LT(x^3 + 2xy)$ or $x^2y = LT(g) = LT(x^2y + 2y^2 - 1)$. Hence, $\langle LT(f), LT(g) \rangle \neq \langle LT(I) \rangle$.

CHAPTER 3

GRÖBNER BASES AND GRÖBNER BASIS ATTACKS

3.1 Gröbner Bases

The concept of Gröbner basis and the algorithm to construct it introduced by Buchberger [11] in 1965. Gröbner bases have many applications in computational algebra such as, ideal membership problem, ideal description problem and the problem of solving polynomial equations. We will mainly focus on the solving polynomial equations.

Definition 3.1.1 (Polynomial Systems Solving (PoSSo) Problem). Given a set of polynomial equations $P = \{f_1, f_2, \cdots, f_m\} \in \mathbb{F}[x_1, \cdots, x_k]$. Find -if any- common solutions of the polynomial system such that:

$$f_1(x_1, \cdots, x_k) = f_2(x_1, \cdots, x_k) = \cdots = f_m(x_1, \cdots, x_k) = 0.$$

When the number of variables is high, this problem is hard to solve.

Definition 3.1.2 (Gröbner Basis). Fix a monomial ordering on $\mathbb{F}[x_1, \cdots, x_k]$ and an ideal I. A finite subset $G = \{g_1, \cdots, g_t\}$ of an ideal I is a **Gröbner basis** of I if the ideal generated by the leading term of every element of I is generated by the leading terms of the g_i, i.e.

$$\big\langle LT(I)\big\rangle = \big\langle LT(g_1), \cdots, LT(g_t)\big\rangle$$

or informally, if any element of I is divisible by one of $LT(g_i)$.

Consider $I = \big\langle x^3 + 2xy, x^2y + 2y^2 - 1\big\rangle$ from our previous Example 2.1.2. The set $F = \{f, g\} = \{x^3 + 2xy, x^2y + 2y^2 - 1\}$ is not a Gröbner basis for ideal $I = \big\langle F\big\rangle$

with respect to lex order since $x \in \langle LT(I) \rangle$ but $x \notin \langle LT(f), LT(g) \rangle$.

Example 3.1.1. Let P be the set of polynomials in $\mathbb{Q}[x, y, z]$ where $P = \{x^3y - z, x^2 + z, x + y + z\}$. The following SAGE code may be used to compute Gröbner basis:

```
sage: P.<x,y,z>=PolynomialRing(QQ)
sage: I = P.ideal([x^3*y-z,x^2+z,x+y+z])
sage: gb=I.groebner_basis()
[y*z^2 + y*z + z^2, z^3 - y*z + z, y^2 + 2*y*z + z^2 + z, x + y + z]
sage: Ideal(gb).basis_is_groebner()
True
```

Theorem 3.1.1. Every ideal I has a Gröbner basis $G = \{g_1, \cdots, g_t\}$ for a fixed monomial order. Furthermore, any Gröbner basis for the ideal I is a basis of I.

Proof. See [15, Chapter 2, Section 5, Corollary 6]. □

Buchberger formulated an algorithm, known as *Burchberger's algorithm*, for computing Gröbner basis. This algorithm comes from the idea behind Buchberger's criterion and used to determine if a given basis for an ideal is Gröbner or not.

Definition 3.1.3 (S-polynomial). Let $f, g \in \mathbb{F}[x_1, \cdots, x_k]$ be two non zero polynomials. The S-polynomial of f and g is defined as the combination

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

where x^γ is the least common multiple of the leading monomials of f and g, written as $x^\gamma = lcm(LM(f), LM(g))$.

Example 3.1.2. Consider $f = x^3y - xy^2$ and $g = 2x^2y^2 + y$ in $\mathbb{R}[x, y]$ with respect to the lex order. Then $x^\gamma = lcm(x^3y, x^2y^2) = x^3y^2$ and

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

Observe that the leading terms of the polynomials f and g are cancel each other.

S-polynomial is constructed in such a way that the leading terms of two polynomials cancelled.

Theorem 3.1.2 (Buchberger's Criterion). Let I be an ideal. A basis $G = \{g_1, \cdots, g_t\}$ is a Gröbner basis of I if and only if for any pairs $i \neq j$, the remainder on the divison of $S(g_i, g_j)$ by G, listed in some order, is zero, written as

$$\overline{S(g_i, g_j)}^{G} = 0.$$

Proof. See [15, Chapter 2, Section 7, Theorem 2]. □

This criterion leads the Buchberger's algorithm to construct a Gröbner basis for a given ideal, see Algorithm 1.

Algorithm 1 Buchberger's Algorithm

```
Input: $F = (f_1, \cdots, f_t)$                                   ▷ $F \subseteq \mathbb{F}[x_1, \cdots, x_k]$
Output: A gröbner basis $G = (g_1, \cdots, g_s)$ for the ideal $I = \langle F \rangle$
  $G = F$
  $G' = $ set()
  while $G' \neq G$ do
      $G' = G$
      for each pair $\{p, q\}, p \neq q$ in $G'$ do
          $r := \overline{S(p,q)}^{G'}$
          if $r \neq 0$ then
              $G$.add($r$)
          end if
      end for
  end while
  return $G$
```

This algorithm terminates since the Buchberger's criterion, if $G' = G$ then $\overline{S(p,q)}^{G'} = 0$ for any $p, q \in G'$ and for $r \neq 0, G' = G$ in finitely many steps due to the Ascending Chain Condition which stabilizes the ascending chain of ideals, a nested increasing sequence. The runtime of the algorithm is affected by the choice of monomial ordering, the order of which p, q are selected and the unnecessary reductions to 0.

To understand how we construct Gröbner basis using Buchberger's algorithm, let's look at the following example.

Example 3.1.3. We have already seen that $F = \{f_1, f_2\} = \{x^3+2xy, x^2y+2y^2-1\}$ in Example 2.1.2 is not a Gröbner basis for $I = \langle F \rangle$. We compute S-polynomial of f_1 and f_2

$$S(f_1, f_2) = x \in I,$$

and its remainder on the division by F is x which is non-zero. Then, we add the remainder $x = f_3$ to the set F and check if this new extended set $F = \{f_1, f_2, f_3\}$ is a Gröbner basis for I or not. Notice that $\overline{S(f_1, f_2)}^F = 0$ and compute

$$\begin{aligned} S(f_1, f_3) &= (x^3 + 2xy) - (x^2)(x) = 2xy = 2yf_3, \text{ so} \\ \overline{S(f_1, f_3)}^F &= 0, \\ S(f_2, f_3) &= (x^2y + 2y^2 - 1) - (xy)(x) = 2y^2 - 1, \text{and} \\ \overline{S(f_2, f_3)}^F &= 2y^2 - 1 \neq 0, \end{aligned}$$

therefore we need to add the remainder $f_4 = 2y^2 - 1$ to the generating set F. Now, we have $F = \{x^3 + 2xy, x^2y + 2y^2 - 1, x, 2y^2 - 1\}$.

$$\begin{aligned} S(f_1, f_4) &= (1/2)x^3 + 2xy^3 = (1/2)f_1 + (2y^3 - y)f_3, \text{ so} \\ \overline{S(f_1, f_4)}^F &= 0. \end{aligned}$$

Similary, one can easily check that $\overline{S(f_i, f_j)}^F = 0$ for any pairs $i \neq j \in \{1, 2, 3, 4\}$. Hence $\{x^3 + 2xy, x^2y + 2y^2 - 1, x, 2y^2 - 1\}$ is a lex ordered Gröbner basis for I.

One may view Buchberger's algorithm as a generalization of Euclidean algorithm for computing greatest common divisor of polynomials and Gaussian elimination to solve linear equations. There are other algorithms such as F_4 and F_5 to compute Gröbner basis effectively using some linear algebra techniques [17, 18].

3.2 Gröbner Basis Attacks

Algebraic attack is a type of cryptographic attack that exploits the algebraic structure of the cipher to recover the secrets. This class of attacks deduce the secret key by solving multivariate polynomial system of equations which consists of key, plaintext and ciphertext bits. Gröbner basis attack is an example for algebraic attacks.

The first step of the attack is to represent the cipher as a system of polynomial equations. Then, the attacker computes the Gröbner basis for the ideal generated by corresponding equations and finally solve the system for unknown variables. The phases of Gröbner basis attack are detailed below.

1. Set up a multivariate polynomial system of equations that describe the cipher. Note that, one can always find a polynomial representation of a function over finite fields, but the crucial point is to find the simplest description due to the complexity of algebraic attacks.

2. Compute a Gröbner basis for the polynomial system, which forms an ideal, in degree reverse lexicografic order (mostly preferred for performance reasons) using Gröbner basis algorithms such as Buchberger's, F4, F5 or Macaulay matrices. In general, this is the most expensive step.

3. Change the ordering in Gröbner basis from degrevlex order to the lex order via Gröbner basis conversion algorithms like FGLM [19], which works only for zero dimensional ideals, or Gröbner Walk algorithm [14]. Lex ordered coefficient matrix of Gröbner basis is in triangular shape and the last row gives the solution for univariate equation, that's why the lex order is used to eliminate variables.

4. Factorize the last element in lexicographic Gröbner basis (lex ordered Gröbner basis guarantees there is at least one univariate polynomial) using polynomial factorization algorithms such as Berlekamp algorithm [20]. Finally, compute the full solution of the system by back substituting roots of the univariate polynomial.

A general algorithm for key recovery using Gröbner bases [12] is provided below:

Algorithm 2 Gröbner basis attack [12]

1. Set up a polynomial system of equations $P = \{p_i = 0\}$ for the cipher in question which consists of both cipher and key schedule equations.

2. Request a plaintext/ciphertext pair $((P_0, \cdots, P_{t-1}), (C_0, \cdots, C_{t-1}))$. This gives rise to the following additional system of linear equations $G = \{g_i = 0\}$:

$$\begin{array}{ccc} x_0^{(0)} + P_0 = 0 & \dots & x_0^{(r)} + C_0 = 0 \\ x_1^{(0)} + P_1 = 0 & \dots & x_1^{(r)} + C_1 = 0 \\ \vdots & & \vdots \\ x_{t-1}^{(0)} + P_{t-1} = 0 & \dots & x_{t-1}^{(r)} + C_{t-1} = 0 \end{array}$$

   Let I be the ideal generated by the set of polynomials $J = (\bigcup_i \{p_i\}) \cup (\bigcup_i \{g_i\})$. We call this ideal as key recovery ideal.

3. Compute a degree reverse lexicographic ordered Gröbner basis $G'_{degrevlex}$ of I. For ciphers using a multiplicative inverse as S-box function, the system may be inconsistent, resulting in $G'_{degrevlex} = 1$.

4. If $G'_{degrevlex} = 1$ go to step 2, otherwise continue.

5. Use a Gröbner basis order conversion algorithm to obtain a lexicographical Gröbner basis G_{lex} from $G'_{degrevlex}$. The variable ordering should be such that the key variables of the first round are the least elements.

6. Compute the variety Z of I using the Gröbner basis G_{lex}.

7. Request another plaintext/ciphertext pair (P, C).

8. Try all elements $k \in Z$ as key candidates to encrypt P. If k does not encrypt P to C, remove k from Z, otherwise retain.

9. If Z contains more than one element, go to step 7.

10. Terminate

Note that the above algorithm is very general, many changes are possible such as computing Gröbner basis with a different monomial ordering rather than *degrevlex* or lex. Observe that in step 6, to compute variety Z of I, variety of an ideal is the set of all common solutions of the elements in ideal, one needs to factor univariate polynomials and substitute the roots into other equations to check if that root is a solution for whole system.

In the following sections, we will discuss the complexity of each step.

3.2.1 Complexity of Gröbner Basis Computation

For a generic system of m equations in k variables

$$f_1(x_1, \cdots, x_k) = \cdots = f_m(x_1, \cdots, x_k) = 0$$

the complexity of computing Gröbner basis [10] is

$$\mathcal{O}\left(\binom{k + d_{reg}}{d_{reg}}^{\omega}\right), \tag{3.1}$$

operations over the field $\mathbb{F}$, where $2 \leq \omega < 3$ is the exponent for the complexity of matrix multiplication and d_{reg} is the degree of regularity [9]. The degree of regularity is informally the highest degree reached during Gröbner basis computation and therefore is the key concept to analyze the complexity of polynomial solving algorithms. There is a common belief that this degree determines when the solving algorithm will terminate, that's why it is used to parametrize the complexity [27]. In general, computing degree of regularity for the overdetermined systems ($m > k$) is a hard problem and still an active research area [3]. Notice that the complexity does not contain the number of equations m explicitly but, the degree of regularity depends on the number of equations.

For the *regular systems*, where the number of equations is equal to the number of variables, $m = k$, we can calculate this degree by using the formula:

$$d_{reg} = 1 + \sum_{i=1}^{m}(d_i - 1), \tag{3.2}$$

where d_i is the degree of f_i, see [8]. In general, for the semi-regular (random) systems with the number of equations greater than the number of variables, over-determined systems ($m > k$), the degree of regularity can be computed using Hilbert series expansion of the ideal generated by the polynomials $f_1, \cdots, f_m$. In this case, d_{reg} is defined [8] as the first non-positive coefficient in

$$H(t) = \frac{1}{(1-t)^k} \times \prod_{i=1}^{m} (1-t)^{d_i}.$$

3.2.2 Complexity of Change of Term Ordering

The input of the FGLM algorithm is the Gröbner basis (degrevlex ordered in our case) of a zero-dimensional ideal I, having finitely many solutions, and it returns the Gröbner basis with respect to the lex order.

The complexity of the FGLM algorithm [19] is

$$\mathcal{O}(k \cdot D^3), \tag{3.3}$$

where k is the number of variables and D is the degree of the ideal I which is the vector space dimension of the quotient ring $\mathbb{F}[x_1, \cdots, x_k]/I$. In general, we know that FGLM algorithm is faster than the Gröbner Walk algorithm [12].

3.2.3 Complexity of Factorization

Finally, we need to factorize the last univariate polynomial and find its roots in lex ordered Gröbner basis we discovered. A polynomial of degree d over a finite field $\mathbb{F}_{2^n}$ can be factorized using the improved version of Berklekamp algorithm [20]. The complexity of the algorithm is

$$\mathcal{O}(d^3 n^2 + d n^3). \tag{3.4}$$

In the following chapters, we will describe three block ciphers, JARVIS, MiMC and GMiMC$_{erf}$. We will present Gröbner basis attacks for each cipher, analyze the complexity of the attack for variants of JARVIS, our experimental results for key recovery attack on MiMC and our attack strategy on GMiMC$_{erf}$.

CHAPTER 4

THE BLOCK CIPHER JARVIS

Dhooghe and Ashur proposed JARVIS as a STARK-friendly block cipher in 2018 [7]. Its design inspired by the design of the AES with the aim to gain resistance against differential and linear cryptanalysis. They instantiate JARVIS to offer 128, 160, 192 and 256-bit security levels.

4.1 Description of JARVIS

JARVIS is a family of SPN block ciphers designed for STARK-applications. It uses *wide-trail strategy* as in the case AES which allows to be secure againsts differential and linear cryptanalysis. JARVIS works on an entire n-bit state and an n-bit key over the finite field $\mathbb{F}_{2^n}$. The non-linear layer in JARVIS uses a single S-box over F_{2^n} and defined as a multiplicative inverse function

$$\begin{aligned} S : \mathbb{F}_{2^n} &\longrightarrow \mathbb{F}_{2^n} \\ x &\longrightarrow x^{2^n-2}, \end{aligned}$$

or in rational form

$$S(x) := \begin{cases} \frac{1}{x}, \ x \neq 0 \\ 0, \ x = 0. \end{cases}$$

The linear part in JARVIS is defined as the composition of two affine polynomials. These affine polynomials are created by adding a constant value to a linearized polynomial. Remember that an $\mathbb{F}_2$ linearized permutation polynomial is defined as

$$L(x) = \sum_{i=0}^{n-1} l_i x^{2^i} \in \mathbb{F}_{2^n}[x].$$

And the affine polynomial obtained from $L(x)$ is

$$A(x) = l_{-1} + \sum_{i=0}^{n-1} l_i x^{2^i} \in \mathbb{F}_{2^n}[x].$$

In JARVIS, two monic affine polynomials B and C of degree 4 are chosen in the form

$$B(x) = x^4 + b_2x^2 + b_1x + b_0 \text{ and } C(x) = x^4 + c_2x^2 + c_1x + c_0,$$

so that the linear layer $A(x)$ is splitted as $A(x) = C \circ B^{-1}(x)$, where B^{-1} is the compositional inverse satisfying $B^{-1}(B(x)) = x$. Note that the compositional inverse of B is still an affine polynomial but it has much more high degree. The round function of JARVIS is depicted below in Figure 4.1.

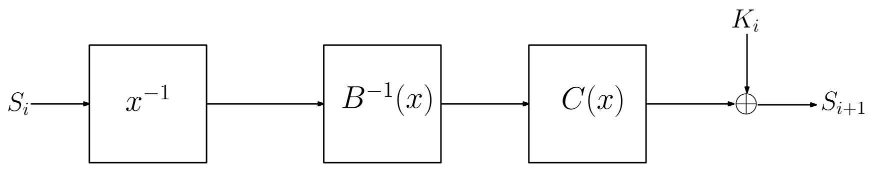

Figure 4.1: One round of the JARVIS block cipher

Key Schedule The key schedule in JARVIS is similar to the round function. It uses the same S-box as in the round function whereas the affine part omitted. The first key k_0 is the master key and the round keys are generated by adding a round constant c_i to the output of the S-box in the key schedule. One round of the key schedule is shown in Figure 4.2.

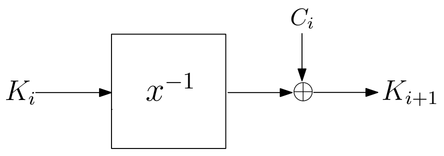

Figure 4.2: One round of the key schedule used in JARVIS block cipher

The designers of JARVIS propose the security levels for four different block sizes and different number of rounds $r = 10, 11, 12, 14$ for a chosen polynomials B and C with fix round constants, see in Table 4.1 [7].

Table 4.1: Instances of JARVIS

Instances	n	number of rounds r
JARVIS-128	128	10
JARVIS-160	160	11
JARVIS-192	192	12
JARVIS-256	256	14

However, it has been shown that the specified number of rounds for JARVIS does not provide above security levels as claimed. In the following section, we will give successful Gröbner basis attack on JARVIS by Albrecht et al. [3].

4.2 Gröbner basis attack

The authors of [3] showed that the JARVIS is not secure as claimed since the certain characteristics of JARVIS makes the cipher vulnerable to Gröbner basis attacks. The one is that the S-box of JARVIS, $S(x) = x^{2^n-2}$, can be written as a degree-2 polynomial

$$S(x) = x^{-1} = y,$$

where $x \cdot y = 1$ for any non zero element $x \in \mathbf{F}_{2^n}$. For a sufficiently large n, it is claimed that x is not equal to zero with a high probability. The other is that whereas the affine polynomial A has high degree, it is a decomposition of two low degree polynomials, see (4.1), and setting up equations by avoiding the inverse computation of high degree B^{-1} makes the system vulnerable to the attack.

4.2.1 Gröbner basis attack on Reduced Round JARVIS

In the original proposal, the authors of [3] first present the Gröbner basis attack approach on reduced round JARVIS and then they improve the attack to apply the full round of JARVIS.

They describe the primitive by introducing an intermediate variable x_i for the i-th round where $1 \leq i \leq r$, see in Figure 4.3.

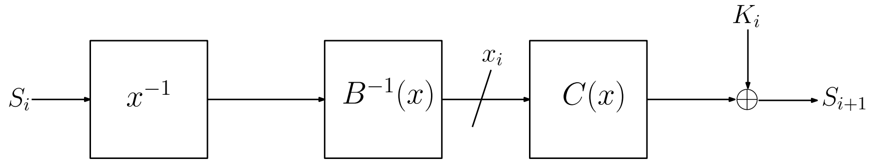

Figure 4.3: Introducing new intermediate variable x_i for the one round of the JARVIS block cipher

The two consecutive rounds of JARVIS is expressed by the equation

$$(C(x_i) + k_i) \cdot B(x_{i+1}) = 1 \tag{4.1}$$

for $1 \leq i \leq r-1$, where r is the number of rounds and the equations for the plaintext p and the ciphertext c described as

$$(p + k_0) \cdot B(x_1) = 1, \tag{4.2}$$

$$C(x_r) + k_r = c. \tag{4.3}$$

The two consecutive round keys in JARVIS are defined by the equation

$$k_{i+1} = \frac{1}{k_i} + c_i$$

which can be written as

$$(k_{i+1} + c_i) \cdot k_i = 1,\ 0 \leq i \leq r-1. \tag{4.4}$$

Since B and C are both degree 4 polynomials, the equations in (4.1), (4.2), (4.3), (4.4) respectively result in:

- $(r-1)$ equations of degree 8 with $(2\cdot r-1)$ variables, $x_1 \cdots x_r$ and $k_1, \cdots k_{r-1}$,
- one equation of degree 5 in two variables k_0 and x_1,
- one degree-4 equation with two variables x_r and k_r,
- r equations having degree 2.

Overall, the above polynomial system of equations that describes the primitive has $2 \cdot r + 1$ equations in $2 \cdot r + 1$ variables $x_1, \cdots, x_r$ and $k_0, \cdots, k_r$. Since the number of equations and the number of variables are equal and assuming system behaves like *regular sequences*, one may calculate the degree of regularity using (3.2) and estimate the complexity of computing Gröbner basis (3.1). Even for the number of rounds $r = 6$, this complexity is almost 120 bits and 85 bits for $\omega = 2.8$ and $\omega = 2$, respectively. However, it is shown in [3] that these theoretical estimations are too pessimistic. In practice, the authors compute the Gröbner basis for the above polynomial system and apply the attack to full-round of JARVIS by improving the attack.

4.2.2 Improved attack: A more efficient description of JARVIS

The authors of [3] improved the attack described in previous section by reducing the number of equations and the number of variables. In order to reduce the number of variables for round equations, they fix the intermediate variables x_i for the even number of rounds and express them using previous x_{i-1} and next following intermediate variables x_{i+1}. For each intermediate variable x_i

$$B(x_i) = \frac{1}{C(x_{i-1}) + k_{i-1}}, \text{ and } C(x_i) = \frac{1}{B(x_{i+1})} + k_i \tag{4.5}$$

where $2 \leq i \leq r-1$. In order to skip intermediate variables x_i, they define monic degree 4 affine polynomials D and E of the form

$$D(x) = x^4 + d_2x^2 + d_1x + d_0, \text{ and } E(x) = x^4 + e_2x^2 + e_1x + e_0$$

satisfying the equation

$$D(B) = E(C). \tag{4.6}$$

It has been shown that the above equation (4.6) can be solved by equalizing the coefficients of polynomials, see [3]. After finding such suitable polynomials D and E, they apply these polynomials to B and C as expressed in (4.5) which yields the polynomial system:

$$D\left(\frac{1}{C(x_{i-1}) + k_{i-1}}\right) = E\left(\frac{1}{B(x_{i+1})} + k_i\right) \text{ for } 2 \leq i \leq r-1, \tag{4.7}$$

$$D\left(\frac{1}{p + k_0}\right) = E\left(\frac{1}{B(x_2)} + k_1\right), \tag{4.8}$$

$$C(x_r) + k_r = c. \tag{4.9}$$

The degrees of each equations are as follows

- For the intermediate round equations in (4.7), the left hand side is of degree 16, since D and C are degree 4 polynomials, and the right hand side is of degree 20, after equalizing denominators degree 36 polynomials obtained.
- The degree of Equation (4.8) is 24, degree 4 from left and degree 20 from the right hand side.
- Equation (4.9) is of degree 4.

Assuming the number of rounds r to be even, above polynomial system gives:

- $\frac{r}{2} - 1$ equations of degree 36,
- one equation of degree 24,
- one equation of degree 4.

In total, above system expressed in $\frac{r}{2} + 1$ equations with variables $x_2, x_4, \cdots, x_r$ and $k_0, \cdots, k_r$. They also reduce the number of key variables by connecting each round key to the master key k_0

$$k_{i+1} = \frac{\alpha_i \cdot k_0 + \beta_i}{\gamma_i \cdot k_0 + \delta_i} \tag{4.10}$$

where $\alpha_i, \beta_i, \gamma_i$ and δ_i are constants and can be found explicitly by solving recursive relation. Final improvement results in:

- $\frac{r}{2} - 1$ equations of degree 40,
- one equation of degree 24,
- one equation of degree 5.

Overall, the improved attack strategy on JARVIS halves the number of equations and variables needed to describe cipher. Hence, it yields a polynomial system with $\frac{r}{2} + 1$ equations in $\frac{r}{2} + 1$ variables $x_2, \cdots, x_r$ and k_0.

Table 4.2: Experimental results of the improved attack on JARVIS using Sage [3]

r	k	d_{reg}	$2\log_2\binom{k+d_{reg}}{d_{reg}}$	d	$2\log_2\binom{k+d}{d}$	d_u	Time
3	2	47	20	26	17	256	0.3s
4	3	67	31	40	27	1280	9.4s
5	3	86	34	40	27	6144	891.4s
6	4	106	45	41	34	28672	99989.0s

In Table 4.2, r denotes the number of rounds, k is the number of variables and d_{reg} is the degree of regularity calculated assuming the system behaves like regular (3.2). The estimated complexity in bits is $2\log_2\binom{k+d_{reg}}{d_{reg}}$, for $\omega = 2$, d is the highest degree reached during the basis computation and the expected security based on the experiment in [3] is $2\log_2\binom{k+d}{d}$. The degree of the univariate polynomial obtained in the last step to solve the system is denoted by d_u.

4.3 Complexity Estimates of Gröbner Basis Computation for the variants of JARVIS

The improved attack given in [3], as described in Section 4.2.2, motivated us to formulate the attack for the block ciphers having affine polynomial like JARVIS. Since the affine layer of JARVIS is the composition of two low degree (degree-4) polynomials B and C, we mentioned that one can find two low degree polynomials D, E which makes cipher vulnerable to Gröbner basis attack. The question is what if one replaces B and C with higher degree polynomials. In order to determine whether the higher degree polynomials choice makes the cipher resistant against Gröbner basis attacks or not, in this section, we try to generalize the complexity of the improved attack on JARVIS. We show that JARVIS with degree 8 affine polynomials is still vulnerable to Gröbner basis attack.

Proposition 4.3.1. Let B and C be arbitrary affine polynomials used in JARVIS. Let D and E be the monic affine polynomials satisfying the equation $D(B) = E(C)$. Let d_b, d_c, d_d, d_e be the degrees of B, C, D, E respectively. Then the complexity of computing Gröbner basis with the improved attack on r rounds JARVIS in bits is

$$\omega \log_2 \binom{((\frac{r}{2}-1)(d_d(d_c+1)+d_e(d_b+1)-1)+(d_d+d_e(d_b+1)+d_c))+\frac{r}{2}+1}{(\frac{r}{2}-1)(d_d(d_c+1)+d_e(d_b+1)-1)+d_d+d_e(d_b+1)+d_c} \tag{4.11}$$

where $(\frac{r}{2}-1)(d_d(d_c+1)+d_e(d_b+1)-1)+(d_d+d_e(d_b+1)+d_c)$ is the degree of regularity.

Proof. Assume that the degrees of the monic affine polynomials B, C, D, E be d_b, d_c, d_d and d_e respectively. The improved attack strategy yields the below equations

- $\frac{r}{2}-1$ equations of degree $d_d(d_c+1)+d_e(d_b+1)$ (from (4.7)),
- one equation of degree $d_d+d_e(d_b+1)$ (from (4.8)),
- one equation of degree d_c+1 (from (4.9)).

We know that the complexity of the Gröbner basis computation in bits is $\omega \log_2 \binom{k+d_{reg}}{d_{reg}}$, see (3.1). Since the number of equations and the number of variables are the same ($m = k = \frac{r}{2}+1$), assuming the system behaves like a regular system, we can estimate the degree of regularity using the closed formula (3.2). The result follows from putting the values we obtained from above system. □

Example 4.3.1. Let's choose B and C as degree-4 polynomials as in original JARVIS. Consider the polynomials D and E have degree 8. The improved attack results in:

- $\frac{r}{2} - 1$ equations of degree 80,
- one equation of degree 48,
- one equation of degree 5.

By using the general formula we found in Proposition 4.3.1, we can estimate the complexity for a different number of rounds r. In Table 4.3, complexities estimated by setting $\omega = 2.8$ and for $\omega = 2$ in parenthesis as in [3], where k is the number of variables and d_{reg} is the estimated degree of regularity (3.2).

Table 4.3: Complexity estimates for B, C are of degree 4 and corresponding D, E are of degree 8 polynomials.

r	k	d_{reg}	Complexity in bits
6	4	210	74 (53)
8	5	299	96 (69)
10	6	368	117(83)
12	7	447	138 (99)
14	8	526	160 (114)

We estimate the complexity of computing Gröbner basis for the affine polynomials B and C of degree 8 and corresponding polynomials D and E of degrees 2, 4 and 8 using (4.11), without regarding if there is a solution for the system ($D(B) = E(C)$) or not. The results can be seen below in Table 4.4.

Table 4.4: Complexity estimates for degree-8 polynomials B and C

		D, E are degree-2		D, E are degree-4		D, E are degree-8	
r	k	d_{reg}	C_{GB}	d_{reg}	C_{GB}	d_{reg}	C_{GB}
6	4	98	62 (44)	190	72 (52)	316	80 (57)
8	5	133	80 (57)	261	93 (67)	430	103 (74)
10	6	168	98 (70)	332	114 (82)	544	126 (90)
12	7	203	116 (83)	403	135 (97)	658	149(107)
14	8	238	135 (96)	474	157 (112)	772	172 (123)

In the table, expected bit security and the degree of regularity denoted by C_{GB} and d_{reg} respectively.

Remark 4.3.1. The complexity of the improved attack on JARVIS increases when the degrees of the polynomials increase. For example, when the number of rounds $r = 6$ estimated complexity is ≈ 45 bits (see (4.2)) for the polynomials B, C, D, E are all degree 4 (in original JARVIS), and complexity is ≈ 57 bits (see (4.4)) for degree-8 polynomials.

4.3.1 Comparison with the S-box of the AES and Decomposing AES S-box

The non-linear part in JARVIS applies the same idea with the S-box of the AES, $S\text{-box}_{AES}(z)$. In this section, we try to decompose $S\text{-box}_{AES}(z)$ for different degree affine polynomials. We provide some lemmas to decide appropriate degrees of the decomposition polynomials of AES S-box.

We know that AES S-box is the composition of an affine function $A_{AES}(z)$ over $\mathbb{F}_2$ and the multiplicative inverse of the input over $\mathbb{F}_{2^8}$. In particular

$$S\text{-box}_{AES}(z) = A_{AES}(z^{254}).$$

The multiplicative inverse is defined by the function F over $\mathbb{F}_{2^8}$

$$\begin{aligned} F : \mathbb{F}_{2^8} &\longrightarrow \mathbb{F}_{2^8} \\ x &\longrightarrow x^{254}, \end{aligned}$$

where zero is mapped to zero. The affine function in AES can be expressed as a degree 128 polynomial over $\mathbb{F}_{2^8}$:

$$\begin{aligned} A_{AES}(z) = {} & \texttt{0x8F} \cdot z^{128} + \texttt{0xB5} \cdot z^{64} + \texttt{0x01} \cdot z^{32} + \texttt{0xF4} \cdot z^{16} + \texttt{0x25} \cdot z^{8} + \\ & \texttt{0xF9} \cdot z^{4} + \texttt{0x09} \cdot z^{2} + \texttt{0x05} \cdot z + \texttt{0x63}. \end{aligned}$$

Then, the S-box of AES is represented as

$$\begin{aligned} S\text{-box}_{AES}(z) = {} & \texttt{0x05} \cdot z^{254} + \texttt{0x09} \cdot z^{253} + \texttt{0xF9} \cdot z^{251} + \texttt{0x25} \cdot z^{247} + \texttt{0xF4} \cdot z^{239} + \\ & \texttt{0x01} \cdot z^{223} + \texttt{0xB5} \cdot z^{191} + \texttt{0x8F} \cdot z^{127} + \texttt{0x63}. \end{aligned}$$

Since JARVIS is also composition of the inverse multiplication and the affine function, S-box of JARVIS $S(z)$ can be written as

$$S(z) = A(z^{254}),$$

and the affine function $A(z)$ is

$$A(z) = (C \circ B^{-1})(z),$$

where B and C are both monic permutation polynomials of degree 4. In the original paper [3], it is shown that the $A_{AES}(z)$ can not be viewed as a decomposition of the polynomials such that

$$A_{AES}(z) = (\hat{C} \circ \hat{B}^{-1})(z),$$

both $\hat{B}$ and $\hat{C}$ have degree 4. The above equation implies

$$A_{AES}^{-1}(z) = (\hat{B} \circ \hat{C}^{-1})(z),$$
$$A_{AES}^{-1}(\hat{C}(z)) = \hat{B}(z),$$

where

$$A_{AES}^{-1}(z) = \texttt{0x6E} \cdot z^{128} + \texttt{0xDB} \cdot z^{64} + \texttt{0x59} \cdot z^{32} + \texttt{0x78} \cdot z^{16} + \texttt{0x5A} \cdot z^{8} + \texttt{0x7F} \cdot z^{4} + \texttt{0xFE} \cdot z^{2} + \texttt{0x5} \cdot z + \texttt{0x5}$$

is the compositional inverse polynomial of A_{AES} which satisfies $A_{AES}^{-1}(A_{AES}(z)) = z$ for every $z \in \mathbb{F}_{2^8}$.

Lemma 4.3.1. There are no two affine polynomials $\hat{B}$ and $\hat{C}$ of degree-4

$$\hat{B}(z) := \hat{b}_4 z^4 + \hat{b}_2 z^2 + \hat{b}_1 z + \hat{b}_0,\ \hat{C}(z) := \hat{c}_4 z^4 + \hat{c}_2 z^2 + \hat{c}_1 z + \hat{c}_0. \tag{4.12}$$

such that $A_{AES}^{-1}(\hat{C}(z))$ is equal to $\hat{B}(z)$.

Proof. Assume the equality holds for the polynomials of both degree 4, then we must have zero coefficients in resulting polynomial $A_{AES}^{-1}(\hat{C}(z))$ for the degrees $8, 16, 32, 64, 128$. That means, we need to solve the following multivariate polynomial system with 5 equations in 3 variables $\hat{c}_4, \hat{c}_2, \hat{c}_1$:

$$\begin{aligned}
\texttt{0xFE} \cdot \hat{c}_4^2 + \texttt{0x7F} \cdot \hat{c}_2^4 + \texttt{0x5A} \cdot \hat{c}_1^8 &= 0,\\
\texttt{0x7F} \cdot \hat{c}_4^4 + \texttt{0x5A} \cdot \hat{c}_2^8 + \texttt{0x78} \cdot \hat{c}_1^{16} &= 0,\\
\texttt{0x5A} \cdot \hat{c}_4^8 + \texttt{0x78} \cdot \hat{c}_2^{16} + \texttt{0x59} \cdot \hat{c}_1^{32} &= 0,\\
\texttt{0x78} \cdot \hat{c}_4^{16} + \texttt{0x59} \cdot \hat{c}_2^{32} + \texttt{0xDB} \cdot \hat{c}_1^{64} &= 0,\\
\texttt{0x59} \cdot \hat{c}_4^{32} + \texttt{0xDB} \cdot \hat{c}_2^{64} + \texttt{0x6E} \cdot \hat{c}_1^{128} &= 0.
\end{aligned}$$

In practice, we have obtained that the only solution satisfies the above system is the trivial solution, $\hat{c}_4 = \hat{c}_2 = \hat{c}_1 = 0$ as shown in [3]. Therefore, affine function of AES can not be decomposed by two degree 4 polynomials. □

Example 4.3.2. Assume $\hat{C}$ as degree 4 affine polynomial. Now, we want to determine if there is an affine polynomial $\hat{B}$ of degree 8 which satisfies $A_{AES}^{-1}(\hat{C}(z)) = \hat{B}(z)$.

Since we want $\hat{B}$ to be a polynomial of degree 8, we equalize the coefficients of the resulting polynomial $A_{AES}^{-1}(\hat{C}(z))$ for the degrees 16, 32, 64, 128 to zero and hence, we obtain 4 have equations in 3 unknowns $\hat{c}_1, \hat{c}_2, \hat{c}_4$:

$$\begin{aligned}
\texttt{0x7F} \cdot \hat{c}_4^4 + \texttt{0x5A} \cdot \hat{c}_2^8 + \texttt{0x78} \cdot \hat{c}_1^{16} &= 0,\\
\texttt{0x5A} \cdot \hat{c}_4^8 + \texttt{0x78} \cdot \hat{c}_2^{16} + \texttt{0x59} \cdot \hat{c}_1^{32} &= 0,\\
\texttt{0x78} \cdot \hat{c}_4^{16} + \texttt{0x59} \cdot \hat{c}_2^{32} + \texttt{0xDB} \cdot \hat{c}_1^{64} &= 0,\\
\texttt{0x59} \cdot \hat{c}_4^{32} + \texttt{0xDB} \cdot \hat{c}_2^{64} + \texttt{0x6E} \cdot \hat{c}_1^{128} &= 0.
\end{aligned}$$

We tried to solve this system and observe that there is no solution different than 0.

In the following Lemma 4.3.2, we will show that $A_{AES}(z)$ can be decomposed as $A_{AES}(z) = (\hat{C} \circ \hat{B}^{-1})(z)$ if the degree of the product of the polynomials $\hat{B}$ and $\hat{C}$ is at least 128.

Lemma 4.3.2. Let $\hat{B}$ and $\hat{C}$ be two affine polynomials of degree 2^b and 2^c, respectively such that

$$\hat{B}(z) = \hat{b}_{2^b} z^{2^b} + \hat{b}_{2^{(b-1)}} z^{2^{(b-1)}} \cdots + \hat{b}_2 z^2 + \hat{b}_1 z + \hat{b}_0$$

and

$$\hat{C}(z) = \hat{c}_{2^c} z^{2^c} + \hat{c}_{2^{(c-1)}} z^{2^{(c-1)}} \cdots + \hat{c}_2 z^2 + \hat{c}_1 z + \hat{c}_0,\ b, c \in \{0, \cdots, 7\}.$$

Then, $A_{AES}(z)$ results in $\hat{C}(\hat{B}^{-1}(z))$ provided that $6 < (b + c) \leq 14$.

Proof. Assume that the degree of $\hat{C}$ is 2^c and the polynomial $A^{-1}_{AES}(\hat{C}(z))$ is equal to $\hat{B}$ having degree 2^b, which implies we need to have zero coefficients for the degrees $2^{(b+1)}, 2^{(b+2)}, \cdots, 2^7$. This results in a polynomial system of $(7-b)$ equations with $(c+1)$ variables $\hat{c}_{2^c}, \cdots, \hat{c}_4, \hat{c}_2, \hat{c}_1$. In order to find a non-zero solution for this system, we need to have more unknowns than the equations. Therefore, b and c must satisfy, $6 < (b+c) \leq 14$. $\square$

We have used the above lemma and decomposed the affine function of AES in practice for the following pairs of the degrees of $\hat{B}$ and $\hat{C}$:

- degree of $\hat{C} = 4, \hat{B} = 32$,
- degree of $\hat{C} = 8, \hat{B} = 16, 32$,
- degree of $\hat{C} = 16, \hat{B} = 8, 16, 32$,
- degree of $\hat{C} = 32, \hat{B} = 4, 8, 16, 32$.

Example 4.3.3. Let $\hat{B}$ and $\hat{C}$ be two affine polynomials of degree 16 and 8 respectively. Then, we must have zero coefficients in the resulting polynomial $A^{-1}_{AES}(\hat{C}(z))$ for the degrees 32, 64, 128 which yields a multivariate polynomial system with 3 equations in 4 unknowns $\hat{c}_8, \hat{c}_4, \hat{c}_2, \hat{c}_1$:

$$\texttt{0x7F} \cdot \hat{c}_8^4 + \texttt{0x5A} \cdot \hat{c}_4^8 + \texttt{0x78} \cdot \hat{c}_2^{16} + \texttt{0x59} \cdot \hat{c}_1^{32} = 0,$$
$$\texttt{0x5A} \cdot \hat{c}_8^8 + \texttt{0x78} \cdot \hat{c}_4^{16} + \texttt{0x59} \cdot \hat{c}_2^{32} + \texttt{0xDB} \cdot \hat{c}_1^{64} = 0,$$
$$\texttt{0x78} \cdot \hat{c}_8^{16} + \texttt{0x59} \cdot \hat{c}_4^{32} + \texttt{0xDB} \cdot \hat{c}_2^{64} + \texttt{0x6E} \cdot \hat{c}_1^{128} = 0.$$

Observe that the dimension of the ideal corresponding to above equations is 1 (4-3=1). We assign a random value to the free variable $\hat{c}_1 \in \mathbb{F}_{2^8}$ to make the ideal to be zero-dimensional and then solve the system. For example, one can easily check that the following polynomials $\hat{B}$ and $\hat{C}$ satisfy the equality $A^{-1}_{AES}(\hat{C}(z)) = \hat{B}(z)$

$$\hat{B}(z) = \texttt{0xE4} \cdot z^{16} + \texttt{0xA0} \cdot z^8 + \texttt{0x9A} \cdot z^4 + \texttt{0x2D} \cdot z^2 + \texttt{0xDA} \cdot z + \texttt{0x83}, \quad (4.13)$$
$$\hat{C}(z) = \texttt{0xAF} \cdot z^8 + \texttt{0x37} \cdot z^4 + \texttt{0xD8} \cdot z^2 + \texttt{0xE7} \cdot z + \texttt{0x48}. \quad (4.14)$$

Sage code is provided in Appendix A.1 to illustrate how we solve such a system using Gröbner basis method. After finding such decomposing polynomials of the AES S-box, we may apply the appropriate affine polynomials D and E to the S-box of AES as in the improved JARVIS attack and estimate the complexity of Gröbner basis attack on the JARVIS with AES S-box.

4.3.2 Gröbner basis attack on JARVIS with AES S-box

In the previous section, we show that how the S-boxes of AES and JARVIS are similar and decompose the S-box of AES. In this section, we will replace the non-linear operation in JARVIS with $S\text{-box}_{AES}(z)$ and estimate the complexity of improved attack strategy given in [3]. Assume we have

$$S\text{-box}_{AES}(z) = (C \circ B^{-1})(z^{254}), z \in \mathbb{F}_{2^8} \tag{4.15}$$

for a known affine polynomials B and C

$$\begin{aligned}
B(z) &= b_{2^b} z^{2^b} + b_{2^{(b-1)}} z^{2^{(b-1)}} + \cdots + b_2 z^2 + b_1 z + b_0, \\
C(z) &= c_{2^c} z^{2^c} + c_{2^{(c-1)}} z^{2^{(c-1)}} + \cdots + c_2 z^2 + c_1 z + c_0,\ b, c \in \{0, \cdots, 7\}
\end{aligned}$$

where $(b+c) > 6$ (see Lemma 4.3.2). The polynomial equations defining the JARVIS with AES S-box can be viewed as a system of equations such that the equality

$$D(B) = E(C) \tag{4.16}$$

is satisfied for the affine polynomials D and E of the form

$$\begin{aligned}
D(z) &= d_{2^d} z^{2^d} + d_{2^{(d-1)}} z^{2^{(d-1)}} \cdots + d_2 z^2 + d_1 z + d_0,\ \text{and} \\
E(z) &= e_{2^e} z^{2^e} + e_{2^{(e-1)}} z^{2^{(e-1)}} \cdots + e_2 z^2 + e_1 z + e_0,\ d, e \in \{0, \cdots, 7\}.
\end{aligned}$$

We will consider two cases to estimate the complexity of the improved attack on JARVIS with $S\text{-box}_{AES}(z)$:

1. The key schedule is the same as in (4.2).
2. The key schedule in AES is used and all subkeys are captured by the attacker, but not the master key.

Before moving on we first need to find suitable D and E such that the system (4.16) has a solution. Let's see the following lemma to decide degrees of the polynomials D and E.

Lemma 4.3.3. Let B and C be given decomposition polynomials of the AES S-box as in (4.15) having degree d_b and d_c respectively where $(b+c) > 6$ and $d_e d_c \geq d_d d_b$. Then, one can find two non-zero affine polynomials D and E of degrees d_d and d_e respectively satisfying the system (4.16) provided that $d + 2 \geq c$.

Proof. Write the polynomial system for $D(B) = E(C)$ by comparing the coefficients of $D(B)$ and $E(C)$ and assume that $d_e d_c \geq d_d d_b$. This system results in $e + c + 2$ equations, since the number of equations determined by the highest degree, with $d + e + 4$ variables $d_{2^d}, d_{2^{(d-1)}}, \cdots, d_2, d_1, d_0$ and $e_{2^e}, e_{2^{(e-1)}}, \cdots, e_2, e_1, e_0$. In order to find non-zero solutions to recover the polynomials D and E, we must have at least as many variables as the number of equations, which implies $d + e + 4 \geq e + c + 2$. $\square$

Example 4.3.4. Given two affine polynomials B degree-16 and C degree-8 of the forms:

$$\begin{aligned} B(x) &= b_{16} \cdot x^{16} + b_8 \cdot x^8 + b_4 \cdot x^4 + b_2 \cdot x^2 + b_1 \cdot x + b_0, \text{ and} \\ C(x) &= c_8 \cdot x^8 + c_4 \cdot x^4 + c_2 \cdot x^2 + c_1 \cdot x + c_0. \end{aligned}$$

Our aim is to find affine polynomials D and E such that the equality $D(B) = E(C)$ is holds. Consider D and E as degree 4 and degree 8 polynomials respectively where

$$\begin{aligned} D(x) &= d_4 \cdot x^4 + d_2 \cdot x^2 + d_1 \cdot x + d_0, \text{ and} \\ E(x) &= e_8 \cdot x^8 + e_4 \cdot x^4 + e_2 \cdot x^2 + e_1 \cdot x + e_0. \end{aligned}$$

We obtain a linear polynomial system of 8 equations in 9 variables $d_4, d_2, d_1, d_0, e_8, e_4, e_2, e_1, e_0$ by comparing coefficients of $D(B)$ and $E(C)$:

$$\begin{aligned} d_4 \cdot b_{16}^4 + e_8 \cdot c_8^8 &= 0, \\ d_4 \cdot b_8^4 + d_2 \cdot b_{16}^2 + e_8 \cdot c_4^8 + e_4 \cdot c_8^4 &= 0, \\ d_4 \cdot b_4^4 + d_2 \cdot b_8^2 + d_1 \cdot b_{16} + e_8 \cdot c_2^8 + e_4 \cdot c_4^4 + e_2 \cdot c_8^2 &= 0, \\ d_4 \cdot b_2^4 + d_2 \cdot b_4^2 + d_1 \cdot b_8 + e_8 \cdot c_1^8 + e_4 \cdot c_2^4 + e_2 \cdot c_4^2 + e_1 \cdot c_8 &= 0, \\ d_4 \cdot b_1^4 + d_2 \cdot b_2^2 + d_1 \cdot b_4 + e_4 \cdot c_1^4 + e_2 \cdot c_2^2 + e_1 \cdot c_4 &= 0, \\ d_2 \cdot b_1^2 + d_1 \cdot b_2 + e_2 \cdot c_1^2 + e_1 \cdot c_2 &= 0, \\ d_1 \cdot b_1 + e_1 \cdot c_1 &= 0, \\ d_4 \cdot b_0^4 + d_2 \cdot b_0^2 + d_1 \cdot b_0 + d_0 + e_8 \cdot c_0^8 + e_4 \cdot c_0^4 + e_2 \cdot c_0^2 + e_1 \cdot c_0 + e_0 &= 0. \end{aligned}$$

We solve this system for given polynomials B and C in (4.13) using Gröbner basis method and get one of the following solutions:

$$D(x) = \texttt{0xB4} \cdot x^4 + \texttt{0x3B} \cdot x^2 + \texttt{0x56} \cdot x + \texttt{0x30}$$

and

$$E(x) = \texttt{0xC5} \cdot x^8 + \texttt{0xE2} \cdot x^4 + \texttt{0x73} \cdot x^2 + \texttt{0x98} \cdot x + \texttt{0xCC}.$$

We apply suitable polynomials D and E which satisfy the above Lemma 4.3.3 and estimate the complexity of improved attack for both two cases, see in Tables 4.5 and 4.6, respectively.

Table 4.5: Complexity estimates of the improved attack on JARVIS with S-box$_{AES}(z)$ and the same key schedule described as in 4.10.

r	k	d_b	d_c	d_d	d_e	d_{reg}	Complexity in bits
6	4	16	8	4	8	490	62
8	5	16	8	4	8	661	80
10	6	16	8	4	8	832	97
12	7	16	8	4	8	1003	115

In the table, r denotes the number of rounds and k is the number of variables. The degrees of the decomposition polynomials B and C of S-box$_{AES}(z)$ and the degrees of the corresponding polynomials D and C are denoted by d_b, d_c, d_d, d_e respectively. The expected degree of regularity d_{reg} and complexity estimation in bits are computed, assuming the system behaves like regular sequences, via the formula we give in 4.3.1 for $\omega = 2$.

Table 4.6: Complexity estimates of the improved attack on JARVIS with S-box$_{AES}(z)$ and AES key schedule in the case of all subkeys are captured by the attacker, but not the master key.

r	k	d_b	d_c	d_d	d_e	d_{reg}	Complexity in bits $2\log_2\binom{k+d_{reg}}{d_{reg}}$
6	4	16	8	4	8	457	62
8	5	16	8	4	8	616	79
10	6	16	8	4	8	775	96
12	7	16	8	4	8	934	114

In Table 4.6, for the number of rounds r, the attacker obtain all the key variables $k_1, \cdots, k_r$. The improved attack for the polynomials B, C, D, E having degree $16, 8, 4, 8$

denoted as d_b, d_c, d_d, d_e yields, $\frac{r}{2} - 1$ equations of degrees 160 (from (4.7)), one equation having degree 132 (from (4.8)), one equation having degree 8 (from (4.9)). Since the number of equations is same as the number of variables we estimate d_{reg} using (3.2), and the expected the bit security computed for $\omega = 2$.

Remark 4.3.2. We note that while the estimated complexity for JARVIS is ≈ 45 bits, for the number of rounds $r = 6$, this complexity becomes ≈ 62 bits, see Table 4.5, when JARVIS using the S-box of AES, with an input 8 bits. If we use AES key schedule and S-box of AES in JARVIS and assume the attacker captures all the subkeys, except the master key, the improved attack complexity is ≈ 96 bits for 10 rounds, see Table 4.6.

CHAPTER 5

THE BLOCK CIPHER MiMC

The block cipher MiMC *"Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity"* [2], with its variants, published in 2016 and designed to provide high performance for the applications of secure multi-party computation (MPC), fully homomorphic encryption (FHE), zero knowledge proofs (ZK) and the other popular proof systems like SNARKs, STARKs. It minimizes multiplicative complexity to be efficient over larger fields. In this chapter, we will describe the block cipher MiMC-n/n and present our experimental results from running the Gröbner basis attack on reduced rounds of MiMC. We will discuss why cipher is secure against the attack.

5.1 MiMC-n/n

MiMC is an arithmetic-oriented block cipher works over a finite field $\mathbb{F}_q$, where q is either a prime number or a power of 2. We will mainly consider MiMC over $\mathbb{F}_{2^n}$, same description of the cipher is used for prime fields. The round function of MiMC-n/n is described by a non-linear cubic function $x \longmapsto x^3$ where $x \in \mathbb{F}_{2^n}$. At each round, the same key k and the randomly chosen round constants $c_i \in \mathbb{F}_{2^n}$ are added to the output of the function. The round function of MiMC can be found in Figure 5.1. Note that the cube function is a permutation in $\mathbb{F}_{2^n}$ only if n is an odd number or if gcd$(3, p-1) = 1$ when operate over prime field $\mathbb{F}_p$. The decryption in MiMC is done using the round constants in reverse order and inverting the non-linear function x^3 ($S^{-1}(x) := x^s$ where $s = (2^{n+1} - 1)/3$) for odd n [2]. Because of the high degree

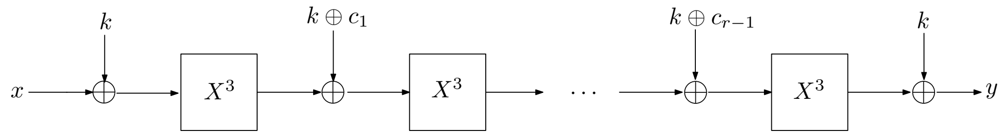

Figure 5.1: r rounds of the MiMC-n/n block cipher

of inverse cubing function, decryption part is more computationally expensive than the encryption however, the target applications of MiMC, like cryptographic hash functions, not usually require to perform decryption. The designers give the security analysis for various algebraic attacks and the number of rounds r for MiMC-n/n is decided by the interpolation attack as $r = \left\lceil \frac{n}{\log_2 3} \right\rceil$. It is claimed that 82 rounds is enough for MiMC-$129/129$ to be secure against GCD, interpolation and the other attacks.

5.2 Gröbner Basis Attack

Gröbner basis attacks, as detailed in Chapter 3, have mainly three steps:

1. Compute a Gröbner basis in degrevlex order for the polynomial system describing the primitive

2. Perform a change of term ordering from the degrevlex order to the lex order

3. Factorize the univariate polynomial for the last variable and solve the system by substituting back its roots

Since the MiMC- n/n has a simple algebraic expression, several algebraic attacks performed in literature [16, 3]. The authors of [3] state that the equations describing MiMC are already form a Gröbner basis, therefore the first step of the attack (computing basis) is free but the recovered univariate polynomial has degree $\approx 3^r$ for r rounds. Because of the cost of the factorization algorithm, they conclude that Gröbner basis attack has no thread on the security of MiMC.

The graphical representation of introducing new variables for MiMC-n/n is given in 5.2.

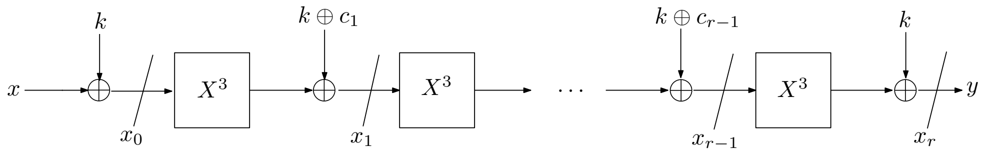

Figure 5.2: Introducing new intermediate variable x_i for r rounds of MiMC-n/n

As stated in [3], we express the intermediate rounds of MiMC as follows:

$$x_{i-1}^3 + x_i + c_i + k_0 = 0, \tag{5.1}$$

$$x_{r-1}^3 + x_r + k_0 = 0, \tag{5.2}$$

for $1 \leq i \leq r$ where k_0 is the key variable. In order to make the polynomial system dependent on plaintext p and ciphertext c, we write

$$p + k_0 + x_0 = 0, \tag{5.3}$$

$$c + x_r = 0. \tag{5.4}$$

Since the above system already forms a Gröbner basis, we skip the first step of the attack and try to recover the key for the reduced rounds of MiMC-$129/129$ in practice, see Table 5.1.

Table 5.1: The number of rounds and the degree of the univariate equation after applying r rounds MiMC denoted as r and d_u respectively. FGLM and FACT times represents the time, in seconds, needed to compute FGLM and Factorization algorithms for the corresponding number of rounds.

r	FGLM time	FACT time	d_u
3	0.4s	0.2s	27
4	8.8s	2.2s	81
5	266.0s	31.6s	243
6	11462.0s	248.0s	729

Although the equations for MiMC-n/n form a Gröbner basis, times needed to compute FGLM and Factorization algorithms increase exponentially when the number of rounds increase. Therefore, we conclude that Gröbner basis attack has no threat on MiMC with 82 rounds.

CHAPTER 6

THE BLOCK CIPHER GMiMC

The block cipher GMiMC *"Generalized Feistel MiMC"*, proposed in 2019, with its variants is the more efficient generalized version of MiMC and designed to benefit MPC, SNARK applications and PQ-secure signature schemes [4]. In this chapter, we will briefly describe GMiMC$_{\text{erf}}$, a variant of GMiMC using expanding round function, and then give our Gröbner basis attack strategy. In the original proposal [4], the security analysis of the cipher against Gröbner basis attack is based on the difficulty of computing Gröbner basis. However, we discover a recursion in Gröbner basis of GMiMC$_{\text{erf}}$ with four branches for the univariate case and that enables us to skip the first step of the attack, see 5.2 to remember the steps of the attack. We will show that cipher secure against Gröbner basis attack not because of the complexity of computing Gröbner basis but for a different reason.

6.1 Description of GMiMC$_{\text{erf}}$

GMiMC-with an expanding round function (erf) is an unbalanced Feistel cipher. One round of an unbalanced Feistel Network with an expanding round function can be written as [4]:

$$(X_{t-1}, X_{t-2}, \cdots, X_0) \leftarrow (X_{t-2} \oplus F(X_{t-1}), \cdots, X_0 \oplus F(X_{t-1}), X_{t-1})$$

where $X_j \in \mathbb{F}_{2^n}$ is an input to the jth branch, $1 \leq j \leq t-1$, of the Feistel network and F is the round function similar to MiMC defined as

$$F(x) := (x \oplus k_i \oplus C_i)^3,$$

where k_i is the round key and C_i is the randomly chosen and fixed round constant for the ith round, $1 \leq i \leq r$. The graphical representation of the cipher can be found below in Figure 6.1.

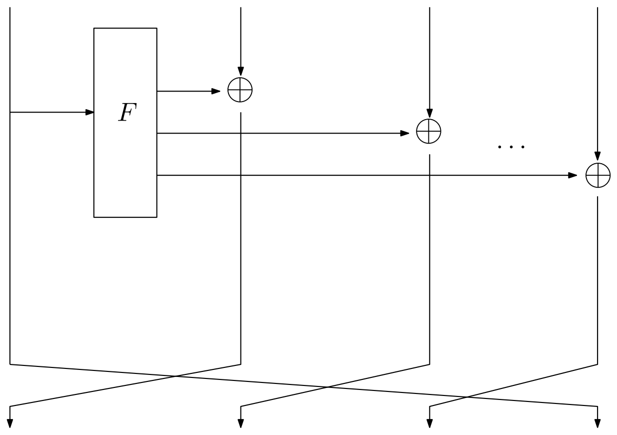

Figure 6.1: One round of an unbalanced Feistel Network GMiMC with an expanding round function

The description of the cipher over the prime finite field $\mathbb{F}_p$ with order p is obtained by replacing XOR operation with the sum in modulo p. Throughout this paper, we consider the *univariate case* $\kappa = n$ (or equivalently for the $\mathbb{F}_p$ case, $2^\kappa \simeq p$) where the key size, denoted by κ, is equal to the branch size $n = \lceil \log_2 |\mathbb{F}| \rceil$ in bits. Key schedule for the univarite case in GMiMC$_{\text{erf}}$, also for the other variants, is linear, $k_i = k$ for any i.

6.2 Gröbner Basis Attack

The authors of [4] give a detailed security analysis of GMiMC over $\mathbb{F}_p$ and discuss the minimum number of rounds that guarantees the security of the cipher for several attacks. They state that most of the attack techniques over $\mathbb{F}_p$ can be performed similarly in $\mathbb{F}_{2^n}$. The minimum number of rounds required to prevent the corresponding possible attacks towards GMiMC$_{\text{erf}}$ can be found below in Table 6.1 [4]. They propose the minimum number of rounds r to protect the cipher against Gröbner basis attacks only for the multivariate case (when the key size κ is equal to the block size N, $N = n \cdot t$ or equivalently, $2^\kappa \simeq 2^N \simeq p^t$ for the $\mathbb{F}_p$ case). They claim that the attack is the same as GCD attack for the univariate case. The minimum required number of

Table 6.1: The minimum number of rounds r to provide the security of GMiMC$_{\text{erf}}$ against the corresponding attacks over $\mathbb{F}_p$ for the univariate case ($\kappa = n$), where $t > 2$ is the number of branches and $2 \cdot \log_3(2) = 1.262$.

	Number of rounds r
GCD	$\lceil 1.262 \cdot \log_2(p) - 4 \cdot \log_3(\log_2(p)) \rceil + 2t - 2$
Interpolation	$\lceil 1.262 \cdot \log_2(p) \rceil + 2t$
Higher Order	$2 + 2t + \lceil 2\log_3(t) \rceil$
Truncated Differential	$2 + \left\lceil (t^2 + t) \cdot \frac{\log_2(p)}{2(\log_2(p) - 1)} \right\rceil$
Impossible Differential	$2t$

rounds to be resistant against Gröbner basis attacks is given as

$$r = \lceil 0.631 \cdot \log_2(p) + 2\log_3(t) \rceil + 4t - 5.$$

They obtain this value by first observing the degrees of the polynomial equations describing cipher after r rounds, and then estimating the complexity of computing Gröbner basis for this degree. It is claimed that introducing new intermediate variables does not decrease the complexity of the attack since it causes to increase in number of variables.

6.2.1 Our attack strategy

In contrast to the block cipher MiMC-n/n, our polynomial equations describing GMiMC$_{\text{erf}}$ do not form a Gröbner basis. Therefore, to perform the attack, one first needs to compute the Gröbner basis which is the most expensive step of the attack. Our idea was to find -if any- a recursion or a path in basis in order to skip basis computation. We discovered the recursion in degrevlex ordered Gröbner basis of GMiMC$_{\text{erf}}$ with four branches for the univariate case ($2^\kappa \simeq p$) over arbitrary prime fields $\mathbb{F}_p$ and so able to write a general recursive formula of the basis for r rounds of the cipher.

We describe the primitive by setting four intermediate variables $x_{4(i-1)}$, $x_{4(i-1)+1}$, $x_{4(i-1)+2}$, $x_{4(i-1)+3}$ for each round from leftmost to the rightmost branch where $1 \leq i \leq r$ as depicted below in Figure 6.2.

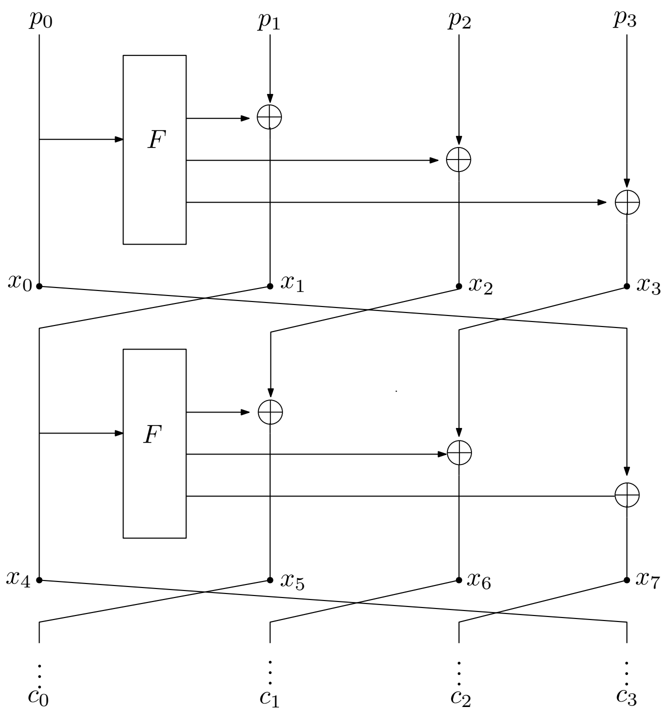

Figure 6.2: Introducing new intermediate variables $x_{4(i-1)}$, $x_{4(i-1)+1}$, $x_{4(i-1)+2}$, $x_{4(i-1)+3}$ for r rounds of GMiMC$_{\text{erf}}$ where $1 \leq i \leq r$ with branch number $t = 4$.

Two consecutive rounds of GMiMC$_{\text{erf}}$ with 4 branches can be related as follows

$$x_{4i} - x_{4(i-1)+1} = 0, \tag{6.1}$$

$$x_{4i+1} - x_{4(i-1)+2} - F(x_{4(i-1)+1}, k_0, C_i) = 0, \tag{6.2}$$

$$x_{4i+2} - x_{4(i-1)+3} - F(x_{4(i-1)+1}, k_0, C_i) = 0, \tag{6.3}$$

$$x_{4i+3} - x_{4(i-1)} - F(x_{4(i-1)+1}, k_0, C_i) = 0, \tag{6.4}$$

for $1 \leq i \leq r-1$ where k_0 and C_i's are key and constant variables respectively. To make the system dependent on the plaintext p and the ciphertext $c \in (\mathbb{F}_p)^t$ where, $p = (p_0, p_1, p_2, p_3)$ and $c = (c_0, c_1, c_2, c_3)$, we add 4 plaintext equations

$$x_0 - p_0 = 0, \tag{6.5}$$

$$x_1 - F(p_0, k_0, C_0) - p_1 = 0, \tag{6.6}$$

$$x_2 - F(p_0, k_0, C_0) - p_2 = 0, \tag{6.7}$$

$$x_3 - F(p_0, k_0, C_0) - p_3 = 0, \tag{6.8}$$

and the 4 ciphertext equations

$$x_{4(r-1)} - c_3 = 0, \tag{6.9}$$

$$x_{4(r-1)+1} - c_0 = 0, \tag{6.10}$$

$$x_{4(r-1)+2} - c_1 = 0, \tag{6.11}$$

$$x_{4(r-1)+3} - c_2 = 0. \tag{6.12}$$

Notice that the above system has six polynomial equations of degree 3 and two equations of degree 1 for the intermediate and the plaintext equations, and has four linear ciphertext equations. In practice, we observe that this system do not form a Gröbner basis for the primitive unlike to MiMC-n/n.

In order to recover a recursion or a structure in Gröbner basis, for the above polynomial equations describing the cipher, we compute the basis for reduced rounds of GMiMC$_{\text{erf}}$ over prime fields $\mathbb{F}_p$ having different prime orders $p > 11$ and see that the basis is independent from the order choice. We were able to compute the Gröbner basis in degrevlex order until 13 rounds using SageMath, which was enough to discover the pattern in the basis. The below Sage code block illustrates how we compute the basis for 2 rounds equations and fixed same round constants C_0.

```
sage: P.<x7,x6,x5,x4,x3,x2,x1,x0,p3,p2,p1,p0,k0,C0>=
      PolynomialRing(GF(307))
sage: eqn1=p1+(p0+k0+C0)^3-x4
sage: eqn2=p2+(p0+k0+C0)^3+(x4+k0+C0)^3-x5
sage: eqn3=p3+(p0+k0+C0)^3+(x4+k0+C0)^3+-x6
sage: eqn4=p0+(x4+k0+C0)^3-x7
sage: I=Ideal([eqn1,eqn2,eqn3,eqn4])
sage: I.basis_is_groebner()
False
sage: gb =I.groebner_basis()
sage: gb
[x4^3+3*x4^2*k0+3*x4*k0^2+k0^3+3*x4^2*C0+6*x4*k0*C0+3*k0^2*C0+
3*x4*C0^2+3*k0*C0^2+C0^3-x5+x4+p2-p1,
p0^3+3*p0^2*k0+3*p0*k0^2+k0^3+ 3*p0^2*C0+6*p0*k0*C0+3*k0^2*C0+
3*p0*C0^2+3*k0*C0^2+C0^3-x4+p1,
x7-x5+x4+p2-p1-p0,x6-x5-p3+p2]
```

6.2.2 Observation

Gröbner basis G for GMiMC$_{\text{erf}}$ with 4 branches over arbitrary prime field $\mathbb{F}_p$, with respect to the degrevlex order, has the following forms:

when the number of rounds $r = 5 + 3k, k \in \mathbb{N}$

$G = \{(x_{4(r-1)} + k_0 + C_0)^3 - x_{4(r-1)+1} + x_{4(r-1)} - x_{4(r-3)} + 2x_{4(r-4)} - x_{4(r-5)} - x_{4(r-6)} + 2x_{4(r-7)} - x_{4(r-8)} - x_{4(r-9)} + \cdots + 2x_4 + p_2 - p_1 - p_0,$
$(x_{4(r-2)} + k_0 + C_0)^3 - x_{4(r-1)} + x_{4(r-2)} - x_{4(r-4)} + 2x_{4(r-5)} - x_{4(r-6)} - x_{4(r-7)} + \cdots + 2x_{12} - x_8 - x_4 - p_3 + p_1 + p_0,$
$(x_{4(r-3)} + k_0 + C_0)^3 - x_{4(r-2)} + x_{4(r-3)} - x_{4(r-5)} + 2x_{4(r-6)} - x_{4(r-7)} - x_{4(r-8)} + \cdots + 2x_8 - x_4 + p_3 - p_2,$
$(x_{4(r-4)} + k_0 + C_0)^3 - x_{4(r-3)} + x_{4(r-4)} - x_{4(r-6)} + 2x_{4(r-7)} - x_{4(r-8)} - x_{4(r-9)} + \cdots + 2x_4 + p_2 - p_1 - p_0,$
$\vdots$
$(x_{12} + k_0 + C_0)^3 - x_{16} + x_{12} - x_4 - p_3 + p_1 + p_0,$
$(x_8 + k_0 + C_0)^3 - x_{12} + x_8 + p_3 - p_2,$
$(x_4 + k_0 + C_0)^3 - x_8 + x_4 + p_2 - p_1,$
$(p_0 + k_0 + C_0)^3 - x_4 + p_1,$
$x_{4(r-1)+3} - x_{4(r-1)+1} + x_{4(r-1)} - x_{4(r-2)} - x_{4(r-3)} + 2x_{4(r-4)} - x_{4(r-5)} - x_{4(r-6)} + 2x_{4(r-7)} - \cdots + 2x_4 + p_2 - p_1 - p_0,$
$x_{4(r-1)+2} - x_{4(r-1)+1} + x_{4(r-2)} - 2x_{4(r-3)} + x_{4(r-4)} + x_{4(r-5)} - 2x_{4(r-6)} + \cdots + x_4 - p_3 + p_2\},$

when the number of rounds $r = 6 + 3k, k \in \mathbb{N}$

$G = \{(x_{4(r-1)} + k_0 + C_0)^3 - x_{4(r-1)+1} + x_{4(r-1)} - x_{4(r-3)} + 2x_{4(r-4)} - x_{4(r-5)} - x_{4(r-6)} + 2x_{4(r-7)} - x_{4(r-8)} - x_{4(r-9)} + \cdots - x_4 + p_3 - p_2,$
$(x_{4(r-2)} + k_0 + C_0)^3 - x_{4(r-1)} + x_{4(r-2)} - x_{4(r-4)} + 2x_{4(r-5)} - x_{4(r-6)} - x_{4(r-7)} + \cdots + 2x_4 + p_2 - p_1 - p_0,$
$(x_{4(r-3)} + k_0 + C_0)^3 - x_{4(r-2)} + x_{4(r-3)} - x_{4(r-5)} + 2x_{4(r-6)} - x_{4(r-7)} - x_{4(r-8)} + \cdots - x_4 - p_3 + p_1 + p_0,$
$(x_{4(r-4)} + k_0 + C_0)^3 - x_{4(r-3)} + x_{4(r-4)} - x_{4(r-6)} + 2x_{4(r-7)} - x_{4(r-8)} - x_{4(r-9)} + \cdots - x_4 + p_3 - p_2,$

$\vdots$

$(x_{12} + k_0 + C_0)^3 - x_{16} + x_{12} - x_4 - p_3 + p_1 + p_0,$

$(x_8 + k_0 + C_0)^3 - x_{12} + x_8 + p_3 - p_2,$

$(x_4 + k_0 + C_0)^3 - x_8 + x_4 + p_2 - p_1,$

$(p_0 + k_0 + C_0)^3 - x_4 + p_1,$

$x_{4(r-1)+3} - x_{4(r-1)+1} + x_{4(r-1)} - x_{4(r-2)} - x_{4(r-3)} + 2x_{4(r-4)} - x_{4(r-5)} - x_{4(r-6)} + 2x_{4(r-7)} - \cdots + 2x_8 - x_4 + p_3 - p_2,$

$x_{4(r-1)+2} - x_{4(r-1)+1} + x_{4(r-2)} - 2x_{4(r-3)} + x_{4(r-4)} + x_{4(r-5)} - 2x_{4(r-6)} + \cdots + x_4 + p_3 - p_1 - p_0\},$

when the number of rounds $r = 7 + 3k, k \in \mathbb{N}$

$G = \{(x_{4(r-1)} + k_0 + C_0)^3 - x_{4(r-1)+1} + x_{4(r-1)} - x_{4(r-3)} + 2x_{4(r-4)} - x_{4(r-5)} - x_{4(r-6)} + 2x_{4(r-7)} - x_{4(r-8)} - x_{4(r-9)} + \cdots - x_8 - x_4 - p_3 + p_1 + p_0,$

$(x_{4(r-2)} + k_0 + C_0)^3 - x_{4(r-1)} + x_{4(r-2)} - x_{4(r-4)} + 2x_{4(r-5)} - x_{4(r-6)} - x_{4(r-7)} + \cdots + 2x_8 - x_4 + p_3 - p_2,$

$(x_{4(r-3)} + k_0 + C_0)^3 - x_{4(r-2)} + x_{4(r-3)} - x_{4(r-5)} + 2x_{4(r-6)} - x_{4(r-7)} - x_{4(r-8)} + \cdots + 2x_4 + p_2 - p_1 - p_0,$

$(x_{4(r-4)} + k_0 + C_0)^3 - x_{4(r-3)} + x_{4(r-4)} - x_{4(r-6)} + 2x_{4(r-7)} - x_{4(r-8)} - x_{4(r-9)} + \cdots - x_4 - p_3 + p_1 + p_0,$

$\vdots$

$(x_{12} + k_0 + C_0)^3 - x_{16} + x_{12} - x_4 - p_3 + p_1 + p_0,$

$(x_8 + k_0 + C_0)^3 - x_{12} + x_8 + p_3 - p_2,$

$(x_4 + k_0 + C_0)^3 - x_8 + x_4 + p_2 - p_1,$

$(p_0 + k_0 + C_0)^3 - x_4 + p_1,$

$x_{4(r-1)+3} - x_{4(r-1)+1} + x_{4(r-1)} - x_{4(r-2)} - x_{4(r-3)} + 2x_{4(r-4)} - x_{4(r-5)} - x_{4(r-6)} + 2x_{4(r-7)} - \cdots - x_4 - p_3 + p_1 + p_0,$

$x_{4(r-1)+2} - x_{4(r-1)+1} + x_{4(r-2)} - 2x_{4(r-3)} + x_{4(r-4)} + x_{4(r-5)} - 2x_{4(r-6)} + \cdots - 2x_4 - p_2 + p_1 + p_0\},$

where $p = (p_0, p_1, p_2, p_3)$ is the plaintext and C_0 is the round constant variable.

We note that we only consider the case where the same key and round constant are used in each round and the number of branches is four. We conclude that the Gröbner basis of GMiMC$_{\text{erf}}$ has the above structures which makes the first step of the Gröbner basis attack for free. That means, one can always compute the Gröbner basis for any number of rounds and therefore there is no complexity of computing Gröbner basis for $\text{GMiMC}_{\text{erf}}$. In order to recover the key, we used the above Gröbner basis elements as our polynomial equations, with a single known p/c pair, and tried to change the term ordering in basis from *degrevlex* order to *lex* order via the FGLM algorithm (second step of the attack). We observed that the ideal generated by those equations is not zero-dimensional, so used the Gröbner Walk algorithm and recovered the key until 13 rounds by solving the last univariate basis equation. However, the Gröbner Walk algorithm was slower than the FGLM algorithm and hence, our attack strategy did not speed up the Gröbner basis attack on $\text{GMiMC}_{\text{erf}}$. We give the attack code in Appendix A.2. The natural question is to ask what happens if one makes the ideal zero-dimensional.

CHAPTER 7

CONCLUSION

In this master thesis we focus on the Gröbner basis attack on three different symmetric-key primitives JARVIS-like ciphers, MiMC and GMiMC$_{\text{erf}}$ which are designed to offer efficient solution in applications of advanced cryptographic protocols. We give some mathematical background required to understand the concept of Gröbner basis attacks in Chapter 2 and describe the Gröbner basis and the phases of the Gröbner basis attacks in Chapter 3.

We study the successful Gröbner basis attack against JARVIS by Albrecht et al. [3] and later present our general formula to estimate the complexity of the attack on variants of JARVIS in Section 4.3. We use this formula to analyze the security of JARVIS-like ciphers with higher degree polynomials. We choose the affine polynomials in JARVIS-128 as degree 8 polynomials rather than 4 and observe that although the expected bit security increases (≈ 90 bits for 10 rounds), the cipher still does not provide the claimed security level in the original proposal [7]. Since the block cipher JARVIS is very similar to the AES S-box, we write the S-box of AES as a decomposition of two affine polynomials in Section 4.3.1. We decompose the S-box of AES for different degree of polynomials satisfying Lemma 4.3.2.

Next, we replace the JARVIS round function with the AES S-box which operates on inputs of 8-bits in Section 4.3.2. We estimate the complexity of the improved attack in two cases, the first one is that we use the key schedule of JARVIS, and the other is that we regard the AES key schedule but the attacker obtained all subkeys. For both cases, we see that the improved attack complexity is around ≈ 97 bits when number of rounds is 10, see in Tables 4.5, 4.6.

Also, we apply a Gröbner basis attack to MiMC block cipher, see in Section 5.2. The first step of the attack is free because of the equations describing MiMC already forms a Gröbner basis as emphasized in [3]. We recovered the secret key until 6 rounds using SageMath. Our equations for MiMC result in a univariate polynomial of degree $\approx 3^r$. We conclude that our Gröbner basis attack strategy has no thread on MiMC due to the complexities of FGLM and factorization algorithms.

The polynomial equations we construct for the other block cipher GMiMC$_{\text{erf}}$ do not form a Gröbner basis unlike to MiMC. We consider the cipher with 4 branches and compute degrevlex ordered Gröbner basis until 13 rounds. However, the specified number of rounds for GMiMC is much higher. Therefore, our attack strategy is to make the first step of the attack free. In order to avoid basis computation, we try to find a structure in basis which leads us to obtain the Gröbner basis for any number of rounds. We find Gröbner bases in degrevlex in the Section 6.2. However, we couldn't change order of the terms to lexicographic order via FGLM algorithm since the dimension of the ideal was not zero. We use Gröbner Walk algorithm to recover univariate equation with single plaintext/ciphertext pair and solve for the key until 13 rounds but attack is still not applicable to the full round of GMiMC due to the performance reasons of Gröbner Walk algorithm.

7.1 Discussion and Future Work

The symmetric-key primitives become more algebraically simple to provide efficient solution in applications of advanced cryptographic protocols in recent years. Security of these designs usually assured by the number of rounds to avoid corresponding algebraic attacks. Gröbner basis is one of those attacks should be regarded especially due to the recent success of the attack on primitives JARVIS and FRIDAY. However, there is no generic systematic security argument for deciding resistance towards Gröbner basis attacks without experimentally running the flavor of the attack we give in Chapter 3. A systematic way to describe complexity of the attack should be a future work to be investigated.

The paper [6] provides a novel framework to determine the security of the cipher against Gröbner basis attack. For most of the new designs, it becomes a standard that *the cipher's resistance against the Gröbner basis attack should based on infeasible complexity of computing Gröbner basis in degrevlex order.* Note that, the first step of the attack is free for the MiMC case. But, it turns out that the cipher secure against the attack because of the following steps, complexity of order conversion and factorization algorithms.

We provide a complexity estimation for the JARVIS-like ciphers according to the attack strategy called *bridging equations over two rounds* in the original paper [3]. We apply this attack to AES S-box but that is not a comprehensive work due to the lack of time. Bridging more than two equations to reduce number of variables or generalization of the improved attack strategy on JARVIS and apply to AES or new arithmetization-oriented ciphers such as *GMiMC, Starkad/Poseidon* is a subject for the future wok.

REFERENCES

[1] Starkware industries: Stark-friendly hash challenge.

[2] M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity, in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 191–219, Springer, 2016.

[3] M. R. Albrecht, C. Cid, L. Grassi, D. Khovratovich, R. Lüftenegger, C. Rechberger, and M. Schofnegger, Algebraic cryptanalysis of stark-friendly designs: application to marvellous and mimc, in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 371–397, Springer, 2019.

[4] M. R. Albrecht, L. Grassi, L. Perrin, S. Ramacher, C. Rechberger, D. Rotaru, A. Roy, and M. Schofnegger, Feistel structures for mpc, and more, in *European Symposium on Research in Computer Security*, pp. 151–171, Springer, 2019.

[5] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner, Ciphers for mpc and fhe, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 430–454, Springer, 2015.

[6] A. Aly, T. Ashur, E. Ben-Sasson, S. Dhooghe, and A. Szepieniec, Design of symmetric-key primitives for advanced cryptographic protocols, IACR Transactions on Symmetric Cryptology, pp. 1–45, 2020.

[7] T. Ashur and S. Dhooghe, Marvellous: a stark-friendly family of cryptographic primitives., IACR Cryptol. ePrint Arch., 2018, p. 1098, 2018.

[8] M. Bardet, J. Faugere, B. Salvy, and B. Yang, Asymptotic behaviour of the index of regularity of quadratic semi-regular polynomial systems, in *8th International Symposium on Effective Methods in Algebraic Geometry*, Citeseer, 2005.

[9] 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, 2003.

[10] L. Bettale, J.-C. Faugère, and L. Perret, Solving polynomial systems over finite fields: improved analysis of the hybrid approach, in *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*, pp. 67–74, 2012.

[11] B. Buchberger, Bruno buchberger's phd thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal, Journal of symbolic computation, 41(3-4), pp. 475–511, 2006.

[12] J. Buchmann, A. Pyshkin, and R.-P. Weinmann, Block ciphers sensitive to gröbner basis attacks, in *Cryptographers' Track at the RSA Conference*, pp. 313–331, Springer, 2006.

[13] A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, M. Naya-Plasencia, P. Paillier, and R. Sirdey, Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression, Journal of Cryptology, 31(3), pp. 885–916, 2018.

[14] S. Collart, M. Kalkbrener, and D. Mall, Converting bases with the gröbner walk, Journal of Symbolic Computation, 24(3-4), pp. 465–469, 1997.

[15] D. Cox, J. Little, and D. OShea, *Ideals, varieties, and algorithms: an introduction to computational algebraic geometry and commutative algebra*, Springer Science & Business Media, 2013.

[16] M. Eichlseder, L. Grassi, R. Lüftenegger, M. Øygarden, C. Rechberger, M. Schofnegger, and Q. Wang, An algebraic attack on ciphers with low-degree round functions: Application to full mimc., IACR Cryptol. ePrint Arch., 2020, p. 182, 2020.

[17] J.-C. Faugere, A new efficient algorithm for computing gröbner bases (f4), Journal of pure and applied algebra, 139(1-3), pp. 61–88, 1999.

[18] J. C. Faugère, A new efficient algorithm for computing gröbner bases without reduction to zero (f 5), in *Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, pp. 75–83, 2002.

[19] J.-C. Faugere, P. Gianni, D. Lazard, and T. Mora, Efficient computation of zero-dimensional gröbner bases by change of ordering, Journal of Symbolic Computation, 16(4), pp. 329–344, 1993.

[20] G. Genovese, Improving the algorithms of berlekamp and niederreiter for factoring polynomials over finite fields, Journal of Symbolic Computation, 42(1-2), pp. 159–177, 2007.

[21] L. Grassi, D. Kales, D. Khovratovich, A. Roy, C. Rechberger, and M. Schofnegger, Starkad and poseidon: New hash functions for zero knowledge proof systems, 2019.

[22] L. Grassi, R. Lüftenegger, C. Rechberger, D. Rotaru, and M. Schofnegger, On a generalization of substitution-permutation networks: The hades design strategy, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 674–704, Springer, 2020.

[23] P. Karn, P. Metzger, and W. Simpson, The esp triple des transform, RFC1851, 1995.

[24] P. Méaux, A. Journault, F.-X. Standaert, and C. Carlet, Towards stream ciphers for efficient fhe with low-noise ciphertexts, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 311–343, Springer, 2016.

[25] V. Rijmen and J. Daemen, Advanced encryption standard, Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology, pp. 19–22, 2001.

[26] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, Zerocash: Decentralized anonymous payments from bitcoin, in *2014 IEEE Symposium on Security and Privacy*, pp. 459–474, IEEE, 2014.

[27] I. Semaev and A. Tenti, Probabilistic analysis on macaulay matrices over finite fields and complexity of constructing gröbner bases., IACR Cryptol. ePrint Arch., 2019, p. 903, 2019.

[28] W. Stein and D. Joyner, Sage: System for algebra and geometry experimentation, Acm Sigsam Bulletin, 39(2), pp. 61–64, 2005.

APPENDIX A

SAGE CODE LISTING

This appendix will provide Sage code to decompose inverse affine function in AES, A_{AES}^{-1}, with affine polynomials C and B of degrees 8 and 16 respectively in A.1. In A.2, we will give Sage code for the Gröbner basis attack on the reduced rounds of MiMC, GMiMC$_{\text{erf}}$ and implementation of the attack on GMiMC$_{\text{erf}}$ without computing Gröbner basis.

A.1 Solving Multivariate Polynomial Equations from Section 4.3.1

```
#!/usr/bin/env python
# coding: utf-8


"""
Solving the multivariate polynomial system satisfying the equation

A^-1_AES (C)= B^-1 using GB method.

AUTHOR: Gizem Kara <kara.gizem@metu.edu.tr>

"""


from sage.rings.polynomial.multi_polynomial_ideal import MPolynomialIdeal
from sage.rings.polynomial.polynomial_gf2x import GF2X_BuildIrred_list

import sage.libs.singular.function_factory
eliminate = sage.libs.singular.function_factory.ff.eliminate


K.<a>= GF(2**8)

L.<c8,c4,c2>=PolynomialRing(K,order='lex')

"""
Coefficients of degree 128 polynomial representation of
inverse affine function of AES
"""
s128=K._cache.fetch_int(Integer(0x6E))
s64=K._cache.fetch_int(Integer(0xDB))
s32=K._cache.fetch_int(Integer(0x59))
s16=K._cache.fetch_int(Integer(0x78))
s8=K._cache.fetch_int(Integer(0x5A))
s4=K._cache.fetch_int(Integer(0x7F))
s2=K._cache.fetch_int(Integer(0xFE))
s=K._cache.fetch_int(Integer(0x5))

"""The variables c1,c2,c4,c8 are coefficients of C where
c1 is free"""

```

```
c1=K.random_element()


eqn1=s4*(c8^4)+s8*(c4)^8+s16*(c2^16)+s32*(c1^32)
eqn2=s8*(c8^8)+s16*(c4)^16+s32*(c2^32)+s64*(c1^64)
eqn3=s16*(c8^16)+s32*(c4)^32+s64*(c2^64)+s128*(c1^128)


Id= ideal(eqn1,eqn2,eqn3) # ideal is zero-dimensional
Gb=Id.groebner_basis()


#print("C= degree 8, B= degree 16, dimension:", Id.dimension())

print(Id.variety(),"c1:", c1)
```

A.2 Attacks

A.2.1 MiMC

```
#!/usr/bin/env python
# coding: utf-8


"""
GB Attack on reduced rounds of MiMC
AUTHOR: Gizem Kara <kara.gizem@metu.edu.tr>

NOTE: This code is based on the sage implementations of the
attacks on Jarvis and Friday by Albrecht et al.
at https://github.com/IAIK/marvellous-attacks
"""


from sage.rings.polynomial.multi_polynomial_ideal import MPolynomialIdeal
from sage.rings.polynomial.polynomial_gf2x import GF2X_BuildIrred_list

import sage.libs.singular.function_factory


def define_constants():
    array=[]
    for i in range(0,num_rounds-1):
        array.append(K.random_element())
    return array


def mimc_encryption(plaintext,key,num_rounds,constants):
    cipher=(plaintext+key)^3
    for i in range(0,num_rounds-1):
        cipher=(cipher+key+constants[i])^3
    ciphertext=(cipher+key)
    return ciphertext

def mimc_decryption(ciphertext,key,num_rounds,constants):
    alpha_inverse = (2**(n+1)-1)/3
    plain=(ciphertext-key)
    for i in range(0,num_rounds-1):
        y=num_rounds-2-i
        plain = (plain^alpha_inverse - key - constants[y])
    plaintext=(plain^alpha_inverse-key)
    return plaintext


#decrypted_plaintext= mimc_decryption(ciphertext,key,num_rounds,constants)
#print "plaintext: ",decrypted_plaintext

def get_elimination_ideal(F, variables, deg_bound=None, algorithm="singular:slimgb", *args, **kwds):
    P = PolynomialRing(K, variables, order="degrevlex")
    F = Sequence([P(f) for f in F])
    print("n_v: {n_v:3d}, n_e: {n_e:3d}, max_deg: {d:3d}".format(n_v=F.nvariables(), n_e=len(F),
                                                                d=F.maximal_degree()))

    gb = None
    if deg_bound:
       ideal = Ideal(F)
       print ("Dimension of R/I as vector space:", len(ideal.normal_basis()))
       H = ideal.change_ring(P.change_ring(order="degrevlex")).gens()
       print ("n: {n:3d}, m: {m:3d}, max(deg(f)): {d:3d},".format(n=F.nvariables(), m=len(F),
                                                                  d=H.maximal_degree()))
       if deg_bound is True:
          deg_bound = 1 + sum(f.degree() - 1 for f in H)
       if deg_bound < H.maximal_degree():
          raise ValueError("Degree bound %d is smaller than input degrees %d."%(deg_bound,
                                                                                H.maximal_degree()))
       t = walltime()
       gb = H.groebner_basis(deg_bound=deg_bound, algorithm=algorithm, *args, **kwds)
       print ("GB time: {t:5.1f}s".format(t=walltime(t)))
       print ("deg_bound: {deg_bound:3d}, is GB: {is_gb}".format(deg_bound=deg_bound,
                                                                 is_gb=gb.is_groebner()))
       if not gb.is_groebner():
          raise ValueError("Degree bound %d too low, output is not a Gröbner basis."%deg_bound)

    if gb == None:
       ideal = Ideal(F)
       print ("Dimension of R/I as vector space:", len(ideal.normal_basis()))
       t = walltime()
       gb = F.groebner_basis(algorithm=algorithm, *args, **kwds)
       print ("GB time: {t:5.1f}s".format(t=walltime(t)))

    t = walltime()
    gb_lex = Ideal(gb).transformed_basis('fglm')
    print ("FGLM time: {t:5.1f}s".format(t=walltime(t)))

    univariate = Sequence([f for f in gb_lex if f.is_univariate()])
    return univariate

def solve(eqns, vars, rem_var, deg_bound, *args, **kwds):

    P = PolynomialRing(K, vars)

  # Print degrees
    if debug:
        for eq in eqns:
            print ("Degree:", P(eq).degree())

  # Get elimination ideal
    elGB = get_elimination_ideal(eqns, vars, deg_bound=deg_bound, *args, **kwds)
    if debug:
        print("Length of elimination ideal:", len(elGB))

  # Solve univariate equation
    Q = PolynomialRing(K, rem_var)
    elim = Q(elGB[elGB.nvariables()-1])
    t = walltime()
    sols = [el[0] for el in set(elim.roots(ring=K))]
    print ("FACT time: {t:5.1f}s".format(t=walltime(t)))
    print ("Degree of univariate equation:", elim.degree())

    return sols


def mimc_attack(plaintext,key,num_rounds, constants, deg_bound=None, *args, **kwds):

    ciphertext = mimc_encryption(plaintext,key,num_rounds,constants)

    variables = []
    for i in range(0,num_rounds+1):
        variables.append("x_"+str(i))
    variables.append("k_0")
    P = PolynomialRing(K, variables, order="degrevlex")
    P.inject_variables()

    variables = [P(v) for v in variables]
    equations = []

    for i in range(0,num_rounds+1):
        if i==0:
            equations.append(variables[i]-k_0-plaintext)
        elif i<num_rounds:
             equations.append(variables[i]-k_0-constants[i-1]-(variables[i-1])^3)
        else:
            equations.append(variables[i]-k_0-(variables[i-1])^3)
            equations.append(ciphertext-variables[i])


    remaining_variable = "k_0"
    print ("Solutions:")
    for s in solve(equations, variables, remaining_variable, deg_bound, *args, **kwds):
        print ("K: ",s)

def run_mimc_attack(r=3, deg_bound=None, optimized=True, *args, **kwds):
    k = K.random_element()
    p = K.random_element()
    constants = define_constants()
    print ("key is :", k)
    mimc_attack(p,k,r,constants,deg_bound=deg_bound, *args, **kwds)
```

```python


n = 129
num_rounds=3
testing_polys = False
debug = False

K = GF(2**n,"a")
K.inject_variables()

run_mimc_attack(3,deg_bound=None)
```

A.2.2 GMiMC$_{\mathrm{erf}}$

```python
#!/usr/bin/env python
# coding: utf-8


"""
GB Attack on GMiMCerf,t=4, prime field=307
AUTHOR: Gizem Kara <kara.gizem@metu.edu.tr>

NOTE: This code is based on the sage implementations of the
attacks on Jarvis and Friday by Albrecht et al.
at https://github.com/IAIK/marvellous-attacks
"""

from sage.rings.polynomial.multi_polynomial_ideal import MPolynomialIdeal
from sage.rings.polynomial.polynomial_gf2x import GF2X_BuildIrred_list

import sage.libs.singular.function_factory
eliminate = sage.libs.singular.function_factory.ff.eliminate
import copy
import time

def define_constants():
    array=[]
    for i in range(0,num_rounds):
        array.append(K.random_element())
    return array

def F_func(x,k,c):
    return (x+k+c)**3

def define_plaintext():
    array=[0,0,0,0]
    for i in range(4):
        array[i]= K.random_element()
    return array

def encr_gmimc_erf(plaintext,key,num_rounds,constants):
    cipher=[0,0,0,0]
    plaintext_copy=copy.deepcopy(plaintext)
    for i in range(num_rounds):
        F_i =F_func(plaintext_copy[0],key,constants[i])
        cipher[0]=F_i+plaintext_copy[1]
        cipher[1]=F_i+plaintext_copy[2]
        cipher[2]=F_i+plaintext_copy[3]
        cipher[3]=plaintext_copy[0]
        for j in range(4):
            plaintext_copy[j]=cipher[j]
    return cipher

def decr_gmimc_erf(ciphertext,key,num_rounds,constants):
    plain=[0,0,0,0]
    ciphertext_copy=copy.deepcopy(ciphertext)
    constants=constants[::-1]
    for i in range(num_rounds):
        plain[0]=ciphertext_copy[3]
        plain[1]=ciphertext_copy[0]
        plain[2]=ciphertext_copy[1]
        plain[3]=ciphertext_copy[2]
        F_i =F_func(plain[0],key,constants[i])
        plain[1]-=F_i
        plain[2]-=F_i
        plain[3]-=F_i
        for j in range(4):
            ciphertext_copy[j]=plain[j]
    return plain


def get_elimination_ideal(F, variables, deg_bound=None, algorithm="singular:slimgb", *args, **kwds):

    P = PolynomialRing(K, variables, order="degrevlex")
    F = Sequence([P(f) for f in F])
    print( "n_v: {n_v:3d}, n_e: {n_e:3d}, max_deg: {d:3d}".format(n_v=F.nvariables(), n_e=len(F),
                                                                  d=F.maximal_degree()))

    gb = None
    if deg_bound:
        ideal = Ideal(F)
        print(ideal)
        print( "1 Dimension of R/I as vector space:", len(ideal.normal_basis()))
        H = ideal.change_ring(P.change_ring(order="degrevlex")).gens()
        print("n: {n:3d}, m: {m:3d}, max(deg(f)): {d:3d},".format(n=F.nvariables(), m=len(F),
                                                                  d=H.maximal_degree()))
        if deg_bound is True:
            deg_bound = 1 + sum(f.degree() - 1 for f in H)
        if deg_bound < H.maximal_degree():
            raise ValueError("Degree bound %d is smaller than input degrees %d."%(deg_bound,
                                                                                  H.maximal_degree()))
        t = walltime()
        gb = H.groebner_basis(deg_bound=deg_bound, algorithm=algorithm, *args, **kwds)
        print( "GB time: {t:5.1f}s". format(t=walltime(t)))
        print( "deg_bound: {deg_bound:3d}, is GB: {is_gb}".format(deg_bound=deg_bound,
                                                                  is_gb=gb.is_groebner()))
        if not gb.is_groebner():
            raise ValueError("Degree bound %d too low, output is not a Grobner basis."%deg_bound)

    if gb == None:
        ideal = Ideal(F)
        print( "2 Dimension of R/I as vector space:", len(ideal.normal_basis()))
        t = walltime()
        gb = F.groebner_basis(algorithm=algorithm, *args, **kwds)
        print( "GB time: {t:5.1f}s".format(t=walltime(t)))

    t = walltime()
    gb_lex = Ideal(gb).transformed_basis('fglm')
    print( "FGLM time: {t:5.1f}s".format(t=walltime(t)))
    univariate = Sequence([f for f in gb_lex if f.is_univariate()])
    return univariate

t = walltime()
print( "FACT time: {t:5.1f}s".format(t=walltime(t)))

def solve(eqns, vars, rem_var, deg_bound, *args, **kwds):
    P = PolynomialRing(K, vars)
    if debug:
        for eq in eqns:
            print( "Degree:", P(eq).degree())
    #print("eqns",eqns)
    elGB = get_elimination_ideal(eqns, vars, deg_bound=deg_bound, *args, **kwds)
    if debug:
        print( "Length of elimination ideal:", len(elGB))
    Q = PolynomialRing(K, rem_var)
    print("elGB",elGB)
    #print("elGB0",elGB[0])
    #print("elGB-1",elGB[-1])

    #print( "elGB elements",[elt for elt in elGB])
    elim = Q(elGB[-1])
    sols = [el[0] for el in set(elim.roots(ring=K))]
    print( "Degree of univariate equation:", elim.degree())
    print("roots",elim.roots(ring=K))
    return sols

def gmimc_attack(plaintext,key,num_rounds, constants, deg_bound=None, *args, **kwds):
    ciphertext=encr_gmimc_erf(plaintext,key,num_rounds,constants)
    print( "Ciphertext ", ciphertext)
    variables = []
    for i in range(0, num_rounds):
        variables.append("x_"+str(4*i))
        variables.append("x_"+str(4*i+1))
        variables.append("x_"+str(4*i+2))
        variables.append("x_"+str(4*i+3))
    variables.append("k_0")
    print(variables)
    P = PolynomialRing(K, variables, order="degrevlex")
    P.inject_variables()
    variables = [P(v) for v in variables]
    equations = []
    elGB = None
    for i in range(0,num_rounds+1):
        if i==0:
            equations.append(variables[4*i+1]-F_func(plaintext[0],k_0,constants[i])-plaintext[1])
            equations.append(variables[4*i+2]-F_func(plaintext[0],k_0,constants[i])-plaintext[2])
            equations.append(variables[4*i+3]-F_func(plaintext[0],k_0,constants[i])-plaintext[3])
            equations.append(variables[4*i]-plaintext[0])

        elif i<num_rounds:
            equations.append(variables[4*i]-variables[4*(i-1)+1])
            equations.append(variables[4*i+1]-(variables[4*(i-1)+2])-F_func(variables[4*(i-1)+1],k_0,
                                                                                  constants[i]))
            equations.append(variables[4*i+2]-(variables[4*(i-1)+3])-F_func(variables[4*(i-1)+1],k_0,
                                                                                  constants[i]))
            equations.append(variables[4*i+3]-(variables[4*(i-1)])-F_func(variables[4*(i-1)+1],k_0,
                                                                                constants[i]))
        else:
            equations.append(variables[4*(i-1)]-ciphertext[3])
            equations.append(variables[4*(i-1)+1]-ciphertext[0])
            equations.append(variables[4*(i-1)+2]-ciphertext[1])
            equations.append(variables[4*(i-1)+3]-ciphertext[2])

    remaining_variable = "k_0"


    print( "Solutions:")
    for s in solve(equations, variables, remaining_variable, deg_bound, *args, **kwds):
        print( "K: ",s)

def run_gmimc_attack(r, deg_bound=None, *args, **kwds):
    K= GF(307)
    k= K.random_element()
    constants = define_constants()
    print("key", k)
    print("constants", constants)
    p=define_plaintext()
    print( "Plaintext: ", p)
    #ciphertext=encr_gmimc_erf(p,k,r,constants)
    #print( "Ciphertext:", ciphertext)
    gmimc_attack(p,k,r,constants,deg_bound=deg_bound, *args, **kwds)

num_rounds=10
K= GF(307)
run_gmimc_attack(r=10, deg_bound=None)
```

```
#!/usr/bin/env python
# coding: utf-8


"""
Implementing Gröbner basis elements as polynomial equations
until 12 rounds for GMiMCerf,t=4, field= GF(307)"

AUTHOR: Gizem Kara <kara.gizem@metu.edu.tr>

"""

from sage.rings.polynomial.multi_polynomial_ideal import MPolynomialIdeal
from sage.rings.polynomial.polynomial_gf2x import GF2X_BuildIrred_list

import sage.libs.singular.function_factory
eliminate = sage.libs.singular.function_factory.ff.eliminate
import copy
import time

def define_constants():
    array=[]
    a=K.random_element()
    for i in range(0,num_rounds):
        array.append(a)
    return array

def F_func(x,k,c):
    return (x+k+c)**3

def define_plaintext():
    array=[0,0,0,0]
    for i in range(4):
        array[i]= K.random_element()
    return array

def encr_gmimc_erf(plaintext,key,num_rounds,constants):
    cipher=[0,0,0,0]
    plaintext_copy=copy.deepcopy(plaintext)
    for i in range(num_rounds):
        F_i =F_func(plaintext_copy[0],key,constants[i])
        cipher[0]=F_i+plaintext_copy[1]
        cipher[1]=F_i+plaintext_copy[2]
        cipher[2]=F_i+plaintext_copy[3]
        cipher[3]=plaintext_copy[0]
        for j in range(4):
            plaintext_copy[j]=cipher[j]
    return cipher

def decr_gmimc_erf(ciphertext,key,num_rounds,constants):
    plain=[0,0,0,0]
    ciphertext_copy=copy.deepcopy(ciphertext)
```

```python
    constants=constants[::-1]
    for i in range(num_rounds):
        plain[0]=ciphertext_copy[3]
        plain[1]=ciphertext_copy[0]
        plain[2]=ciphertext_copy[1]
        plain[3]=ciphertext_copy[2]
        F_i =F_func(plain[0],key,constants[i])
        plain[1]-=F_i
        plain[2]-=F_i
        plain[3]-=F_i
        for j in range(4):
            ciphertext_copy[j]=plain[j]
    return plain


def get_elimination_ideal(F, variables, deg_bound=None, algorithm="singular:slimgb", *args, **kwds):
    P = PolynomialRing(K, variables, order="degrevlex")
    F = Sequence([P(f) for f in F])
    print( "n_v: {n_v:3d}, n_e: {n_e:3d}, max_deg: {d:3d}".format(n_v=F.nvariables(), n_e=len(F),
                                                                  d=F.maximal_degree()))
    gb = None
    if deg_bound:
        ideal = Ideal(F)
        print( "1 Dimension of R/I as vector space:", len(ideal.normal_basis()))
        H = ideal.change_ring(P.change_ring(order="lex")).gens()
        print("n: {n:3d}, m: {m:3d}, max(deg(f)): {d:3d},".format(n=F.nvariables(), m=len(F),
                                                                  d=H.maximal_degree()))
        if deg_bound is True:
            deg_bound = 1 + sum(f.degree() - 1 for f in H)
        if deg_bound < H.maximal_degree():
            raise ValueError("Degree bound %d is smaller than input degrees %d."%(deg_bound,
                                                                  H.maximal_degree()))
        t = walltime()
        gb = H.groebner_basis(deg_bound=deg_bound, algorithm=algorithm, *args, **kwds)


        print( "GB time: {t:5.1f}s". format(t=walltime(t)))
        print( "deg_bound: {deg_bound:3d}, is GB: {is_gb}".format(deg_bound=deg_bound,
                                                                  is_gb=gb.is_groebner()))
        if not gb.is_groebner():
            raise ValueError("Degree bound %d too low, output is not a Grobner basis."%deg_bound)

    if gb == None:
        ideal = Ideal(F)
        print( "2 Dimension of R/I as vector space:", len(ideal.normal_basis()))
        t = walltime()
        gb = ideal
        print( "GB time: {t:5.1f}s".format(t=walltime(t)))

    t = walltime()
    gb_lex = Ideal(gb).transformed_basis('gwalk')
    print( "gwalk time: {t:5.1f}s".format(t=walltime(t)))
    #print( "gröbner basis: ",gb)
    univariate = Sequence([f for f in gb_lex if f.is_univariate()])
    return univariate

t = walltime()
print( "FACT time: {t:5.1f}s".format(t=walltime(t)))

def solve(eqns, vars, rem_var, deg_bound, *args, **kwds):
    P = PolynomialRing(K, vars)
    if debug:
        for eq in eqns:
            print( "Degree:", P(eq).degree())
    print("eqns",eqns)
    elGB = get_elimination_ideal(eqns, vars, deg_bound=deg_bound, *args, **kwds)
    if debug:
        print( "Length of elimination ideal:", len(elGB))
    Q = PolynomialRing(K, rem_var)
    print("elGB",elGB)
    print( "elGB elements",[elt for elt in elGB])
    elim = Q(elGB[-1])

    sols = [el[0] for el in set(elim.roots(ring=K))]
    print( "Degree of univariate equation:", elim.degree())
    print("roots",elim.roots(ring=K))
    return sols

def gmimc_attack(plaintext,key,num_rounds, constants, deg_bound=None, *args, **kwds):
    ciphertext=encr_gmimc_erf(plaintext,key,num_rounds,constants)
    print( "Ciphertext ", ciphertext)
    variables = []

    for i in range(0, num_rounds):
        variables.append("x_"+str(4*i))
        variables.append("x_"+str(4*i+1))
        variables.append("x_"+str(4*i+2))
        variables.append("x_"+str(4*i+3))

    variables.append("k_0")
    print("variables:",variables)
    P = PolynomialRing(K, variables, order="degrevlex")
    P.inject_variables()
    variables = [P(v) for v in variables]
    equations = []
    #comment
    elGB = None

    if num_rounds==5:
        equations.append(F_func(plaintext[0],k_0,constants[0])-variables[4]+plaintext[1])
        equations.append(F_func(variables[4],k_0,constants[0])-variables[8]+variables[4]
                         +plaintext[2]-plaintext[1])
        equations.append(F_func(variables[8],k_0,constants[0])-variables[12]+variables[8]
                         +plaintext[3]-plaintext[2])
        equations.append(F_func(variables[12],k_0,constants[0])-variables[16]+
                         variables[12]-variables[4]-plaintext[3]+plaintext[1]+plaintext[0])
        equations.append(F_func(variables[16],k_0,constants[0])-variables[17]+variables[16]
                         -variables[8]+2*variables[4]+plaintext[2]-plaintext[1]-plaintext[0])
        equations.append(variables[19]-variables[17]+variables[16]-variables[12]-variables[8]
                         +2*variables[4]+plaintext[2]-plaintext[1]-plaintext[0])
        equations.append(variables[18]-variables[17]+variables[12]-2*variables[8]
                         +variables[4]-plaintext[3]+plaintext[2])
        equations.append(variables[16]-ciphertext[3])
        equations.append(variables[17]-ciphertext[0])
        equations.append(variables[18]-ciphertext[1])
        equations.append(variables[19]-ciphertext[2])

    if num_rounds==6:
        equations.append(variables[0]-plaintext[0])
        equations.append(variables[1]-plaintext[1])
        equations.append(variables[2]-plaintext[2])
        equations.append(variables[3]-plaintext[3])
        equations.append(F_func(variables[0],k_0,constants[0])-variables[4]+variables[1])
        equations.append(F_func(variables[4],k_0,constants[0])-variables[8]+variables[4]
                         +variables[2]-variables[1])
        equations.append(F_func(variables[8],k_0,constants[0])-variables[12]+variables[8]
                         +variables[3]-variables[2])
        equations.append(F_func(variables[12],k_0,constants[0])-variables[16]+variables[12]
                         -variables[4]-variables[3]+variables[1]+variables[0])
        equations.append(F_func(variables[16],k_0,constants[0])-variables[20]+variables[16]
                         -variables[8]+2*variables[4]+variables[2]-variables[1]-variables[0])
        equations.append(F_func(variables[20],k_0,constants[0])-variables[21]+variables[20]
                         -variables[12]+2*variables[8]-variables[4]+variables[3]-variables[2])
        equations.append(variables[23]-variables[21]+variables[20]-variables[16]-variables[12]
                         +2*variables[8]-variables[4]+variables[3]-variables[2])
        equations.append(variables[22]-variables[21]+variables[16]-2*variables[12]
                        +variables[8]+variables[4]+variables[3]-variables[1]-variables[0])
        equations.append(variables[20]-ciphertext[3])
        equations.append(variables[21]-ciphertext[0])
        equations.append(variables[22]-ciphertext[1])
        equations.append(variables[23]-ciphertext[2])

    if num_rounds==7:
        equations.append(F_func(plaintext[0],k_0,constants[0])-variables[4]+plaintext[1])
        equations.append(F_func(variables[4],k_0,constants[0])-variables[8]+variables[4]
                         +plaintext[2]-plaintext[1])
        equations.append(F_func(variables[8],k_0,constants[0])-variables[12]+variables[8]
                         +plaintext[3]-plaintext[2])
        equations.append(F_func(variables[12],k_0,constants[0])-variables[16]+variables[12]
                         -variables[4]-plaintext[3]+plaintext[1]+plaintext[0])
        equations.append(F_func(variables[16],k_0,constants[0])-variables[20]+variables[16]
                         -variables[8]+2*variables[4]+plaintext[2]-plaintext[1]-plaintext[0])
        equations.append(F_func(variables[20],k_0,constants[0])-variables[24]+variables[20]
                         -variables[12]+2*variables[8]-variables[4]+plaintext[3]-plaintext[2])
        equations.append(F_func(variables[24],k_0,constants[0])-variables[25]+variables[24]
                         -variables[16]+2*variables[12]-variables[8]-variables[4]-plaintext[3]
                         +plaintext[1]+plaintext[0])
        equations.append(variables[27]-variables[25]+variables[24]-variables[20]-variables[16]
                         +2*variables[12]-variables[8]-variables[4]-plaintext[3]+plaintext[1]
                         +plaintext[0])
        equations.append(variables[26]-variables[25]+variables[20]-2*variables[16]+variables[12]
                         +variables[8]-2*variables[4]-plaintext[2]+plaintext[1]+plaintext[0])
        equations.append(variables[24]-ciphertext[3])
        equations.append(variables[25]-ciphertext[0])
        equations.append(variables[26]-ciphertext[1])
        equations.append(variables[27]-ciphertext[2])

    if num_rounds==8:
        equations.append(F_func(plaintext[0],k_0,constants[0])-variables[4]+plaintext[1])
        equations.append(F_func(variables[4],k_0,constants[0])-variables[8]+variables[4]
                         +plaintext[2]-plaintext[1])
        equations.append(F_func(variables[8],k_0,constants[0])-variables[12]+variables[8]
                         +plaintext[3]-plaintext[2])
        equations.append(F_func(variables[12],k_0,constants[0])-variables[16]+variables[12]
                         -variables[4]-plaintext[3]+plaintext[1]+plaintext[0])
        equations.append(F_func(variables[16],k_0,constants[0])-variables[20]+variables[16]
                         -variables[8]+2*variables[4]+plaintext[2]-plaintext[1]-plaintext[0])
        equations.append(F_func(variables[20],k_0,constants[0])-variables[24]+variables[20]
                         -variables[12]+2*variables[8]-variables[4]+plaintext[3]-plaintext[2])
        equations.append(F_func(variables[24],k_0,constants[0])-variables[28]+variables[24]
                         -variables[16]+2*variables[12]-variables[8]-variables[4]-plaintext[3]
                         +plaintext[1]+plaintext[0])
        equations.append(F_func(variables[28],k_0,constants[0])-variables[29]+variables[28]
                         -variables[20]+2*variables[16]-variables[12]-variables[8]
                         +2*variables[4]+plaintext[2]-plaintext[1]-plaintext[0])
        equations.append(variables[31]-variables[29]+variables[28]-variables[24]
                         -variables[20]+2*variables[16]-variables[12]-variables[8]
                         +2*variables[4]+plaintext[2]-plaintext[1]-plaintext[0])
        equations.append(variables[30]-variables[29]+variables[24]-2*variables[20]+
                         variables[16]+variables[12]-2*variables[8]+variables[4]
                         -plaintext[3]+plaintext[2])
        equations.append(variables[28]-ciphertext[3])
        equations.append(variables[29]-ciphertext[0])
        equations.append(variables[30]-ciphertext[1])
        equations.append(variables[31]-ciphertext[2])

    if num_rounds==9:
        equations.append(F_func(plaintext[0],k_0,constants[0])-variables[4]+plaintext[1])
        equations.append(F_func(variables[4],k_0,constants[0])-variables[8]+variables[4]
                         +plaintext[2]-plaintext[1])
        equations.append(F_func(variables[8],k_0,constants[0])-variables[12]+variables[8]
                         +plaintext[3]-plaintext[2])
        equations.append(F_func(variables[12],k_0,constants[0])-variables[16]+variables[12]
                         -variables[4]-plaintext[3]+plaintext[1]+plaintext[0])
        equations.append(F_func(variables[16],k_0,constants[0])-variables[20]+variables[16]
                         -variables[8]+2*variables[4]+plaintext[2]-plaintext[1]-plaintext[0])
        equations.append(F_func(variables[20],k_0,constants[0])-variables[24]+variables[20]
                         -variables[12]+2*variables[8]-variables[4]+plaintext[3]-plaintext[2])
        equations.append(F_func(variables[24],k_0,constants[0])-variables[28]+variables[24]
                         -variables[16]+2*variables[12]-variables[8]-variables[4]
                         -plaintext[3]+plaintext[1]+plaintext[0])
        equations.append(F_func(variables[28],k_0,constants[0])-variables[32]+variables[28]
                         -variables[20]+2*variables[16]-variables[12]-variables[8]
                         +2*variables[4]+plaintext[2]-plaintext[1]-plaintext[0])
        equations.append(F_func(variables[32],k_0,constants[0])-variables[33]+variables[32]
                         -variables[24]+2*variables[20]-variables[16]-variables[12]
                         +2*variables[8]-variables[4]+plaintext[3]-plaintext[2])
        equations.append(variables[35]-variables[33]+variables[32]-variables[28]-variables[24]
                         +2*variables[20]-variables[16]-variables[12]+2*variables[8]
                         -variables[4]+plaintext[3]-plaintext[2])
        equations.append(variables[34]-variables[33]+variables[28]-2*variables[24]+variables[20]
                         +variables[16]-2*variables[12]+variables[8]+variables[4]+plaintext[3]
                         -plaintext[1]-plaintext[0])
        equations.append(variables[32]-ciphertext[3])
        equations.append(variables[33]-ciphertext[0])
        equations.append(variables[34]-ciphertext[1])
        equations.append(variables[35]-ciphertext[2])

    if num_rounds==10:
        equations.append(F_func(plaintext[0],k_0,constants[0])-variables[4]+plaintext[1])
        equations.append(F_func(variables[4],k_0,constants[0])-variables[8]+variables[4]
                         +plaintext[2]-plaintext[1])
        equations.append(F_func(variables[8],k_0,constants[0])-variables[12]+variables[8]
                         +plaintext[3]-plaintext[2])
        equations.append(F_func(variables[12],k_0,constants[0])-variables[16]+variables[12]
                         -variables[4]-plaintext[3]+plaintext[1]+plaintext[0])
        equations.append(F_func(variables[16],k_0,constants[0])-variables[20]+variables[16]
                         -variables[8]+2*variables[4]+plaintext[2]-plaintext[1]-plaintext[0])
        equations.append(F_func(variables[20],k_0,constants[0])-variables[24]+variables[20]
                         -variables[12]+2*variables[8]-variables[4]+plaintext[3]-plaintext[2])
        equations.append(F_func(variables[24],k_0,constants[0])-variables[28]+variables[24]
                         -variables[16]+2*variables[12]-variables[8]-variables[4]
                         -plaintext[3]+plaintext[1]+plaintext[0])
        equations.append(F_func(variables[28],k_0,constants[0])-variables[32]+variables[28]
                         -variables[20]+2*variables[16]-variables[12]-variables[8]
                         +2*variables[4]+plaintext[2]-plaintext[1]-plaintext[0])
        equations.append(F_func(variables[32],k_0,constants[0])-variables[36]+variables[32]
                         -variables[24]+2*variables[20]-variables[16]-variables[12]
                         +2*variables[8]-variables[4]+plaintext[3]-plaintext[2])
        equations.append(F_func(variables[36],k_0,constants[0])-variables[37]+variables[36]
                         -variables[28]+2*variables[24]-variables[20]-variables[16]
                         +2*variables[12]-variables[8]-variables[4]-plaintext[3]+plaintext[1]
                         +plaintext[0])
        equations.append(variables[39]-variables[37]+variables[36]-variables[32]-variables[28]
                         +2*variables[24]-variables[20]-variables[16]+2*variables[12]
                         -variables[8]-variables[4]-plaintext[3]+plaintext[1]+plaintext[0])
        equations.append(variables[38]-variables[37]+variables[32]-2*variables[28]+variables[24]
                         +variables[20]-2*variables[16]+variables[12]+variables[8]
                         -2*variables[4]-plaintext[2]+plaintext[1]+plaintext[0])
        equations.append(variables[36]-ciphertext[3])
        equations.append(variables[37]-ciphertext[0])
        equations.append(variables[38]-ciphertext[1])
        equations.append(variables[39]-ciphertext[2])

    if num_rounds==11:
        equations.append(F_func(plaintext[0],k_0,constants[0])-variables[4]+plaintext[1])
        equations.append(F_func(variables[4],k_0,constants[0])-variables[8]+variables[4]
                         +plaintext[2]-plaintext[1])
        equations.append(F_func(variables[8],k_0,constants[0])-variables[12]+variables[8]
                         +plaintext[3]-plaintext[2])
        equations.append(F_func(variables[12],k_0,constants[0])-variables[16]+variables[12]
                         -variables[4]-plaintext[3]+plaintext[1]+plaintext[0])
        equations.append(F_func(variables[16],k_0,constants[0])-variables[20]+variables[16]
                         -variables[8]+2*variables[4]+plaintext[2]-plaintext[1]-plaintext[0])
        equations.append(F_func(variables[20],k_0,constants[0])-variables[24]+variables[20]
                         -variables[12]+2*variables[8]-variables[4]+plaintext[3]-plaintext[2])
        equations.append(F_func(variables[24],k_0,constants[0])-variables[28]+variables[24]
                         -variables[16]+2*variables[12]-variables[8]-variables[4]-plaintext[3]
                         +plaintext[1]+plaintext[0])
        equations.append(F_func(variables[28],k_0,constants[0])-variables[32]+variables[28]
                         -variables[20]+2*variables[16]-variables[12]-variables[8]+2*variables[4]
                         +plaintext[2]-plaintext[1]-plaintext[0])
        equations.append(F_func(variables[32],k_0,constants[0])-variables[36]+variables[32]
                         -variables[24]+2*variables[20]-variables[16]-variables[12]
                         +2*variables[8]-variables[4]+plaintext[3]-plaintext[2])
        equations.append(F_func(variables[36],k_0,constants[0])-variables[40]+variables[36]
                         -variables[28]+2*variables[24]-variables[20]-variables[16]+2*variables[12]
                         -variables[8]-variables[4]-plaintext[3]+
                         plaintext[1]+plaintext[0])
        equations.append(F_func(variables[40],k_0,constants[0])-variables[41]+variables[40]-
                         variables[32]+2*variables[28]-variables[24]-variables[20]+2*variables[16]
                         -variables[12]-variables[8]+2*variables[4]+
                         plaintext[2]-plaintext[1]-plaintext[0])
        equations.append(variables[43]-variables[41]+variables[40]-variables[36]-variables[32]
                         +2*variables[28]-variables[24]-variables[20]+2*variables[16]-
                         variables[12]-variables[8]+2*variables[4]+plaintext[2]-plaintext[1]-
                         plaintext[0])
        equations.append(variables[42]-variables[41]+variables[36]-2*variables[32]+variables[28]
                         +variables[24]-2*variables[20]+variables[16]+variables[12]
                         -2*variables[8]+variables[4]-plaintext[3]+plaintext[2])
        equations.append(variables[40]-ciphertext[3])
        equations.append(variables[41]-ciphertext[0])
        equations.append(variables[42]-ciphertext[1])
        equations.append(variables[43]-ciphertext[2])

    if num_rounds==12:
        equations.append(F_func(plaintext[0],k_0,constants[0])-variables[4]+plaintext[1])
        equations.append(F_func(variables[4],k_0,constants[0])-variables[8]+variables[4]
                         +plaintext[2]-plaintext[1])
        equations.append(F_func(variables[8],k_0,constants[0])-variables[12]+variables[8]
                         +plaintext[3]-plaintext[2])
        equations.append(F_func(variables[12],k_0,constants[0])-variables[16]+variables[12]
                         -variables[4]-plaintext[3]+plaintext[1]+plaintext[0])
        equations.append(F_func(variables[16],k_0,constants[0])-variables[20]+variables[16]
                         -variables[8]+2*variables[4]+plaintext[2]-plaintext[1]-plaintext[0])
        equations.append(F_func(variables[20],k_0,constants[0])-variables[24]+variables[20]
                         -variables[12]+2*variables[8]-variables[4]+plaintext[3]-plaintext[2])
        equations.append(F_func(variables[24],k_0,constants[0])-variables[28]+variables[24]
                         -variables[16]+2*variables[12]-variables[8]-variables[4]
                         -plaintext[3]+plaintext[1]+plaintext[0])
        equations.append(F_func(variables[28],k_0,constants[0])-variables[32]+variables[28]
                         -variables[20]+2*variables[16]-variables[12]-variables[8]
                         +2*variables[4]+plaintext[2]-plaintext[1]-plaintext[0])
        equations.append(F_func(variables[32],k_0,constants[0])-variables[36]+variables[32]
                         -variables[24]+2*variables[20]-variables[16]-variables[12]
                         +2*variables[8]-variables[4]+plaintext[3]-plaintext[2])
        equations.append(F_func(variables[36],k_0,constants[0])-variables[40]+variables[36]
                         -variables[28]+2*variables[24]-variables[20]-variables[16]
                         +2*variables[12]-variables[8]-variables[4]-plaintext[3]+
                         plaintext[1]+plaintext[0])
        equations.append(F_func(variables[40],k_0,constants[0])-variables[44]+variables[40]
                         -variables[32]+2*variables[28]-variables[24]-variables[20]
                         +2*variables[16]-variables[12]-variables[8]+2*variables[4]+
                         plaintext[2]-plaintext[1]-plaintext[0])

        equations.append(F_func(variables[44],k_0,constants[0])-variables[45]+variables[44]
                         -variables[36]+2*variables[32]-variables[28]-variables[24]
                         +2*variables[20]-variables[16]-variables[12]+2*variables[8]
                         -variables[4]+plaintext[3]-plaintext[2])
        equations.append(variables[47]-variables[45]+variables[44]-variables[40]-variables[36]
                         +2*variables[32]-variables[28]-variables[24]+2*variables[20]
                         -variables[16]-variables[12]+2*variables[8]-variables[4]
                         +plaintext[3]-plaintext[2])
        equations.append(variables[46]-variables[45]+variables[40]-2*variables[36]+variables[32]
                         +variables[28]-2*variables[24]+variables[20]+variables[16]
                         -2*variables[12]+variables[8]+variables[4]+plaintext[3]-plaintext[1]
                         -plaintext[0])
        equations.append(variables[44]-ciphertext[3])
        equations.append(variables[45]-ciphertext[0])
        equations.append(variables[46]-ciphertext[1])
        equations.append(variables[47]-ciphertext[2])


    remaining_variable = "k_0"

    #print( "equations: ")
    #[show(eq) for eq in equations]
    print( "Solutions:")
    for s in solve(equations, variables, remaining_variable, deg_bound, *args, **kwds):
        print( "K: ",s)

def run_gmimc_attack(r, deg_bound=None, *args, **kwds):
    K= GF(307,modulus="primitive")
    k=K.random_element()
    constants = define_constants()
    print("key", k)
    print("constants", constants)
    p= define_plaintext()
    print( "Plaintext: ", p)
    ciphertext=encr_gmimc_erf(p,k,r,constants)
    #print( "Ciphertext:", ciphertext)
    #print( "key is :", key)
    gmimc_attack(p,k,r,constants,deg_bound=deg_bound, *args, **kwds)


num_rounds=9
```

```
K= GF(307,modulus="primitive")
run_gmimc_attack(r=9, deg_bound=None)
```